Dynamic Optimisation of Road Freight Transport Operations in Competitive Market Environments

Hendrik Braun, B.Sc., M.Sc.

Faculty of Arts, Business, Law, and Education - Business School
Management and Organisations

This thesis is presented for the degree of Doctor of Philosophy
at The University of Western Australia

August 2018
Thesis declaration

I, Hendrik Braun, certify that:

This thesis has been substantially accomplished during enrolment in the degree.

This thesis does not contain material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution.

No part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of The University of Western Australia and where applicable, any partner institution responsible for the joint-award of this degree.

This thesis does not contain any material previously published or written by another person, except where due reference has been made in the text.

The work(s) are not in any way a violation or infringement of any copyright, trademark, patent, or other rights whatsoever of any person.

No human data is used or reported in this thesis; therefore no approval by The University of Western Australia Human Research Ethics Committee has been required.

The work described in this thesis was funded by the IPRS Scholarship and APA Award (which were converted to the RTP Stipend during the candidature), the UWA Top-up Scholarship and the Planning and Transport Research Centre (PATREC) Scholarship.

Editorial assistance was kindly provided by Cate Pattison.

This thesis does not contain work that I have published, nor work under review for publication.

Signature: ____________________________

Date: Monday, 6 August 2018
Abstract

In most industrialised countries, road freight transport operators with small market share, collectively account for a large segment of the sector. However, they deal with strong competition, low profit margins and uncertainty about future customer demand. Despite the wide availability of highly developed IT-systems, GPS tracking and internet connectivity; operative processes in RFT still involve a large amount of manual planning and lack in integration and analysis of the available data. This thesis presents a profit optimising method for point-to-point RFT tasks fulfilled by small road freight carriers, featuring the examination of dynamically incoming transport requests. Marginal cost and prices are determined in terms of present and future demand with the objective of maximising profits for the current time-period.

The approach is divided into two sub-problems: (1) a cost determination problem; and (2) the price determination and request selection problem. Marginal costs for incoming transport requests are calculated by solving a dynamic pick-up and delivery problem with time windows (DPDPTW). Then, the bid-price is maximised under consideration of the current competitive situation and expected future demand. An Adaptive Tabu Search heuristic (ATS) has been proposed to solve the problem.

The ATS has been validated on a number of static benchmark cases with reasonable results in terms of objective value and computational time. Dynamic experiments have been conducted on artificial and real-world cases. The former have been created on the basis of the widely used Solomon cases, the latter are based on real-world data from the Australian market (www.truckit.net). The experiments for the profit optimisation problem show that the use of marginal cost for price determination leads to better outcomes for the carriers compared to myopic approaches, based on distance or time and volume/density. Knowledge of incremental costs for the current set of requests enhances the freedom of operators to offer competitive prices without loss of profit. Further, marginal costs tend to decrease with higher utilisation, where especially first requests in new routes are comparatively ‘expensive’. A number of basic discount policies and a forecasting method are proposed in this research to counteract this problem. Results show that including the proposed system helps operators to build sets of transport assignments that fit particularly well together, and even owner operators benefit by finding their ‘niche market’ and generating economies of scale and scope. On average, the combined routing and dynamic price determination offers profit margins of 16-20%, with even
higher values on O-D routes with lower demand. Implementing such a system in reality could provide a significant advantage, especially to small operators whose profits are expected to be considerably below the industry average of 9.3%.
# Table of contents

Thesis declaration ........................................................................................................... ii  
Abstract .......................................................................................................................... iii  
Table of contents ............................................................................................................. v  
List of Figures ................................................................................................................ x  
List of Tables .................................................................................................................. xii  
Abbreviations .................................................................................................................. xiv  
Acronyms for Australian states ....................................................................................... xvi  
List of Notations (Sets) ................................................................................................. xvii  
List of Notations (Variables and Parameters) ................................................................. xviii  
Acknowledgements ........................................................................................................ xx  

1 Introduction .................................................................................................................. 22  
  1.1 Research motivation .................................................................................................. 22  
      1.1.1 Features of Australian road freight transport .............................................. 22  
      1.1.2 Problem description ..................................................................................... 23  
  1.2 Research objectives and contributions ..................................................................... 23  
      1.2.1 Research objective ....................................................................................... 23  
      1.2.2 Research questions ...................................................................................... 24  
      1.2.3 Contributions ............................................................................................... 25  
  1.3 Thesis organisation .................................................................................................. 26  

2 Literature review and market conditions .................................................................... 28  
  2.1 Market description .................................................................................................. 28  
      2.1.1 Industry description and structure ............................................................... 29  
      2.1.2 Competitive analysis and market segments ............................................... 29  
      2.1.3 Bid price generation and decision support systems ................................... 30  
  2.2 The cost minimisation problem .............................................................................. 32  
      2.2.1 Introduction to VRPs ................................................................................... 32
2.2.2 Dynamic vehicle routing ................................................................. 33
2.2.3 Vehicle routing problems dealing with profits ........................................ 36
2.3 The price determination problem .......................................................... 40
  2.3.1 Capacity control ................................................................................ 41
  2.3.2 Forecasting ....................................................................................... 43
  2.3.3 Dynamic pricing ................................................................................ 43
  2.3.4 Overbooking ...................................................................................... 44
  2.3.5 Applied revenue management in freight transport .................................. 46
2.4 Requirements for advanced decision support in the road freight market ............ 47

3 Optimisation System .................................................................................. 53
  3.1 Problem formulation .............................................................................. 53
  3.2 Problem approach .................................................................................. 55
    3.2.1 The pick-up and delivery problem with time windows (PDPTW) ............ 55
      3.2.1.1 Network construction .................................................................. 56
      3.2.1.2 Notation ..................................................................................... 56
      3.2.1.3 Mathematical formulation for the static PDPTW .............................. 58
    3.2.2 The dynamic PDPTW ....................................................................... 60
      3.2.2.1 Additional notation ..................................................................... 60
    3.2.3 The price determination and request selection problem ......................... 62
      3.2.3.1 Dynamic Pricing ........................................................................ 62
      3.2.3.2 Capacity Control ........................................................................ 64
      3.2.3.3 Forecasting ................................................................................. 65

4 Heuristics for solving optimisation problems ................................................. 69
  4.1 Introduction ............................................................................................ 69
  4.2 Adaptive Tabu Search description ............................................................ 72
  4.3 Heuristic move operators ......................................................................... 76
    4.3.1 Removal heuristics .......................................................................... 78
4.3.1.1 Remove single customer based on large distance ........................................... 78
4.3.1.2 Remove single customer based on time window violation .............................. 78
4.3.1.3 Move customer with ‘acute angle’ .................................................................. 78
4.3.1.4 Shaw removal heuristic .................................................................................. 79
4.3.1.5 Random removal heuristic ........................................................................... 80
4.3.1.6 Worst removal heuristic .................................................................................. 80
4.3.1.7 Route elimination heuristic ............................................................................ 81
4.3.2 Insertion Heuristics ........................................................................................... 81
  4.3.2.1 Insert single request based on short distance ................................................ 82
  4.3.2.2 Insert single request based on reduced time window (TW) discrepancy ... 82
  4.3.2.3 Insert single request based on distance and TW ........................................ 82
  4.3.2.4 Basic ‘greedy’ insertion ................................................................................ 83
  4.3.2.5 Regret heuristic ........................................................................................... 83
4.3.3 Case-based heuristics that combine removal and insertion ................................. 84
  4.3.3.1 Postpone single customer by one position .................................................... 85
  4.3.3.2 Swap customer with flexible TW-end to a late position in the route .......... 85
  4.3.3.3 Cross exchange ............................................................................................ 86
  4.3.3.4 Swap pair to closest route ............................................................................ 87
4.3.4 Summary of heuristic operators ....................................................................... 87
4.3.5 Usage of move operators - Probability distribution ............................................ 88
4.3.6 Adaptation of weight adjustment ...................................................................... 89
4.4 Heuristic evaluation .............................................................................................. 90
  4.4.1 Metaheuristic set up and settings ..................................................................... 90
  4.4.2 Benchmark test cases ...................................................................................... 91
  4.4.3 Experimental results ....................................................................................... 91
5 Experiments ............................................................................................................ 94
  5.1 Test instances .................................................................................................... 94
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>Experiment 1: Comparison of RM approach with random pricing</td>
<td>96</td>
</tr>
<tr>
<td>5.3</td>
<td>Experiment 2: Costs with increasing numbers of assigned transport jobs</td>
<td>101</td>
</tr>
<tr>
<td>5.4</td>
<td>Experiment 3: Knowledge about marginal costs in a competitive environment</td>
<td>103</td>
</tr>
<tr>
<td>5.5</td>
<td>Experiment 4: Can knowledge of the market give competitive advantage?</td>
<td>105</td>
</tr>
<tr>
<td>5.6</td>
<td>Experiment 5: Competitor with regional advantage</td>
<td>106</td>
</tr>
<tr>
<td>5.7</td>
<td>Experiment 6: Substitution of early requests for expected future profits</td>
<td>109</td>
</tr>
<tr>
<td>5.8</td>
<td>Experiment 7: Request forecasting and price substitution</td>
<td>111</td>
</tr>
<tr>
<td>6</td>
<td>Case study</td>
<td>119</td>
</tr>
<tr>
<td>6.1</td>
<td>Data collection</td>
<td>119</td>
</tr>
<tr>
<td>6.2</td>
<td>Data description</td>
<td>120</td>
</tr>
<tr>
<td>6.3</td>
<td>Test case generation</td>
<td>122</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Business operating cost</td>
<td>124</td>
</tr>
<tr>
<td>6.4</td>
<td>Experiments</td>
<td>126</td>
</tr>
<tr>
<td>6.4.1</td>
<td>System performance on real-world data</td>
<td>127</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Request forecasting from historical data</td>
<td>131</td>
</tr>
<tr>
<td>6.5</td>
<td>Additional case study observations</td>
<td>137</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Bid-price estimation</td>
<td>138</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Other observations</td>
<td>138</td>
</tr>
<tr>
<td>7</td>
<td>Conclusions and further research</td>
<td>140</td>
</tr>
<tr>
<td>7.1</td>
<td>Focus of the work</td>
<td>140</td>
</tr>
<tr>
<td>7.2</td>
<td>Limitations and possible future extensions</td>
<td>144</td>
</tr>
<tr>
<td>8</td>
<td>Bibliography</td>
<td>147</td>
</tr>
<tr>
<td>9</td>
<td>Appendices</td>
<td>159</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Flowchart of the optimisation system</td>
<td>159</td>
</tr>
<tr>
<td>Appendix B</td>
<td>Heuristic benchmark route sequences</td>
<td>160</td>
</tr>
<tr>
<td>Appendix C</td>
<td>Case study file description</td>
<td>170</td>
</tr>
<tr>
<td>Appendix D</td>
<td>Case study route example</td>
<td>173</td>
</tr>
</tbody>
</table>
Source code .................................................................................................................. 174

E.1. MyRMcontrol class ............................................................................................... 174
E.2. Main class ............................................................................................................. 196

E.3. DemandGenerator class ....................................................................................... 203
E.4. DrawingComponent class .................................................................................... 207
E.5. ExcelFiller class .................................................................................................. 211
E.6. InputDataReader class ......................................................................................... 212
E.7. MyCanvas class ................................................................................................... 217
E.8. MyData class ....................................................................................................... 225
E.9. MyDynamicPricing class ..................................................................................... 226
E.10. MyMoveManager class ....................................................................................... 233
E.11. MyObjectiveFunction class .............................................................................. 256
E.12. MySolution class ............................................................................................... 272
E.13. MySwapMove class ........................................................................................... 285
E.14. MyTabuList class .............................................................................................. 287
E.15. Operator class ................................................................................................... 291
E.16. PositionValues class ........................................................................................ 331
E.17. ReadSINTEFwriteH class ................................................................................. 332
E.18. SinglePositionValues class ............................................................................... 347
E.19. WriteExcel class ............................................................................................... 348
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Factors influencing price determination in road freight transport (Author’s own elaboration)</td>
</tr>
<tr>
<td>3.1</td>
<td>Scheme of a second-price auction</td>
</tr>
<tr>
<td>3.2</td>
<td>Modelling approach for solving the optimisation problem</td>
</tr>
<tr>
<td>3.3</td>
<td>The calculation of marginal cost</td>
</tr>
<tr>
<td>3.4</td>
<td>Request cost and pricing example</td>
</tr>
<tr>
<td>3.5</td>
<td>Expected incoming requests in sections of serviced area</td>
</tr>
<tr>
<td>4.1</td>
<td>Tabu Search Algorithm</td>
</tr>
<tr>
<td>4.2</td>
<td>Starting Solution for Static Problems or Initial Solutions for Dynamic Problems</td>
</tr>
<tr>
<td>4.3</td>
<td>Starting Solution for Dynamic Problems when a Previous Solution Exists</td>
</tr>
<tr>
<td>4.4</td>
<td>Simple Swap Move</td>
</tr>
<tr>
<td>4.5</td>
<td>Swap Pair Move</td>
</tr>
<tr>
<td>4.6</td>
<td>Multi swap operator</td>
</tr>
<tr>
<td>5.1</td>
<td>Uniform request arrival over considered time-period</td>
</tr>
<tr>
<td>5.2</td>
<td>Marginal cost for the insertion of a single request</td>
</tr>
<tr>
<td>5.3</td>
<td>Example for a cost overestimation</td>
</tr>
<tr>
<td>5.4</td>
<td>Example for a cost underestimation</td>
</tr>
<tr>
<td>5.5</td>
<td>Marginal costs depending on number of requests included in routing (only feasible solutions)</td>
</tr>
<tr>
<td>5.6</td>
<td>Unitary marginal costs DD depending on number of requests included in routing (feasible solutions)</td>
</tr>
<tr>
<td>5.7</td>
<td>Unitary marginal costs RTD depending on number of requests included in routing (feasible solutions)</td>
</tr>
<tr>
<td>5.8</td>
<td>Set of customer locations and routing without regional competition</td>
</tr>
<tr>
<td>5.9</td>
<td>Set of customer locations and routing with regional competition</td>
</tr>
<tr>
<td>5.10</td>
<td>Average profits of simulated random competition experiments with different discount rates</td>
</tr>
<tr>
<td>5.11</td>
<td>Expected incoming requests in sections of serviced area (Figure 3.5)</td>
</tr>
<tr>
<td>5.12</td>
<td>Number of admitted discounts for numbers of requests included in vehicle routing</td>
</tr>
<tr>
<td>5.13</td>
<td>Granted discount in relation to customer requests included in routing</td>
</tr>
</tbody>
</table>
Figure 5.14: Granted discount in relation to customer requests included in routing (proportional) ........................................................................................................................................117
Figure 6.1: Origins of the requests presented in www.truckit.net ..........................................................120
Figure 6.2: Destinations of the requests presented in www.truckit.net ..................................................121
Figure 6.3: Distribution of profits per request ($).................................................................................137
Figure 9.1: Flowchart of the optimising system .....................................................................................159
Figure 9.2: Vehicle route of case study instance AU_NSW_R100_109 ..................................................173
List of Tables

Table 2.1: Characteristics and constraints of vehicle routing problems ..................................33
Table 2.2: Taxonomy of dynamic vehicle routing problems (Pillac et al., 2013) ..................36
Table 2.3: Vehicle routing with profits and/or competition ..................................................38
Table 2.4: Requirements and conducive conditions for Revenue Management .....................48
Table 2.5: Differences between traditional Revenue Management and Operations in Road Freight Transport .........................................................................................................................49
Table 4.1: Heuristic operators with removal and insertion combination ..................................88
Table 4.2: Pre-processing distribution .....................................................................................88
Table 4.3: Standard distribution ..............................................................................................89
Table 4.4: Description of scoring criteria ................................................................................89
Table 4.5: Results from computational experiments on the PDPTW benchmark instances ...91
Table 5.1: Overview of experiments and test files used ..........................................................96
Table 5.2: Excerpt from the results of instance lc201RM (Incoming requests 1 – 18) ...........98
Table 5.3: Results including ‘incorrect’ cost estimations ..........................................................100
Table 5.4: Results from experiment 3 using all test instances and three criteria ....................104
Table 5.5: Results of the simulations of market price ..............................................................106
Table 5.6: Computational results for experiments with regional competition .......................107
Table 5.7: Average profits of simulated random competition experiments with different subsidy rates ..........................................................................................................................110
Table 5.8: Profit results from forecasting experiments ............................................................112
Table 5.9: Concluded results for discount values and number of grids ..................................114
Table 6.1: Freight flows within and between states for the collected data ............................121
Table 6.2: Overview of case instance setup ............................................................................124
Table 6.3: Detailed cost specification (1) ..............................................................................125
Table 6.4: Detailed cost specification (2) ..............................................................................126
Table 6.5: Cost summary ........................................................................................................126
Table 6.6: Computational results for cases with focus around major cities (100 km radius) 127
Table 6.7: Computational results for cases with focus around major cities (250 km radius) 129
Table 6.8: Average result values for cases with focus around single metropolitan areas (300 km radius) ...............................................................................................................130
Table 6.9: Profit margin overview for markets and experimental setup ..................................131
Table 6.10: Experimental results for R100 cases when forecasting is included .....................132
Table 6.11: Experimental results for R250 cases under application of forecasting............133
Table 6.12: Experimental results for M300 cases under application of forecasting............134
Table 6.13: Profit margin overview for markets and experimental setup when forecasting is applied..................................................................................................................................................136
Table 6.14: Relation between profit and activity, expressed as distance, number of requests, and location........................................................................................................................................................................137
Table 9.1: Route sequences of the benchmark experiments.............................................160
Table 9.2: R100 case description......................................................................................170
Table 9.3: R250 case description......................................................................................171
Table 9.4: M300 metro case description..........................................................................172
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>American Airlines</td>
</tr>
<tr>
<td>APT</td>
<td>Airline Passenger Transport</td>
</tr>
<tr>
<td>ATS</td>
<td>Adaptive Tabu Search</td>
</tr>
<tr>
<td>BGP</td>
<td>Bid Generation Problem</td>
</tr>
<tr>
<td>CPP</td>
<td>Chinese Postman Problem</td>
</tr>
<tr>
<td>DARP</td>
<td>Dial A Ride Problem</td>
</tr>
<tr>
<td>DD</td>
<td>Direct Distance</td>
</tr>
<tr>
<td>DINAMO</td>
<td>Dynamic Inventory Allocation and Maintenance Optimizer System</td>
</tr>
<tr>
<td>DLP</td>
<td>Deterministic Linear Programming</td>
</tr>
<tr>
<td>DP</td>
<td>Dynamic Pricing</td>
</tr>
<tr>
<td>DPM</td>
<td>Dynamic Pricing Model</td>
</tr>
<tr>
<td>FCFS</td>
<td>First-Come-First-Served</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>LS</td>
<td>Local Search</td>
</tr>
<tr>
<td>LTL</td>
<td>Less-Than-Truckload</td>
</tr>
<tr>
<td>OD</td>
<td>Origin-Destination</td>
</tr>
<tr>
<td>PD</td>
<td>Pick-up and Delivery</td>
</tr>
<tr>
<td>PDPTW</td>
<td>Pick-up and Delivery Problem with Time Windows</td>
</tr>
<tr>
<td>PTP</td>
<td>Profitable Tour Problem</td>
</tr>
<tr>
<td>RFT</td>
<td>Road Freight Transport</td>
</tr>
<tr>
<td>RM</td>
<td>Revenue Management</td>
</tr>
<tr>
<td>RTD</td>
<td>Roundtrip Distance</td>
</tr>
<tr>
<td>TOP</td>
<td>Team Operating Problem</td>
</tr>
<tr>
<td>TSP</td>
<td>Travelling Salesman Problem</td>
</tr>
<tr>
<td>VNS</td>
<td>Variable Neighbourhood Search</td>
</tr>
<tr>
<td>VPG</td>
<td>Vehicle Pricing Game</td>
</tr>
<tr>
<td>VRP</td>
<td>Vehicle Routing Problem</td>
</tr>
<tr>
<td>VRPTW</td>
<td>Vehicle Routing Problem with Time Windows</td>
</tr>
<tr>
<td>VRPTWOV</td>
<td>Vehicle Routing Problem with Time Windows with overtime and outsourcing vehicles</td>
</tr>
</tbody>
</table>
VRPTWPD  Vehicle Routing Problem with Time Windows and Pick-up and Deliveries
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT</td>
<td>Australian Capital Territory</td>
</tr>
<tr>
<td>NSW</td>
<td>New South Wales</td>
</tr>
<tr>
<td>NT</td>
<td>Northern Territory</td>
</tr>
<tr>
<td>QLD</td>
<td>Queensland</td>
</tr>
<tr>
<td>SA</td>
<td>South Australia</td>
</tr>
<tr>
<td>TAS</td>
<td>Tasmania</td>
</tr>
<tr>
<td>VIC</td>
<td>Victoria</td>
</tr>
<tr>
<td>WA</td>
<td>Western Australia</td>
</tr>
</tbody>
</table>
List of Notations (Sets)

\[ V: \] Set of all locations \( i \in \{0, \ldots, n+1\} \) such that \( i=0 \): depot, and \( i=1, \ldots, 2n \): customer nodes; \( V_k \): Subset of locations visited by vehicle \( k \) with \( V_k \in V \).

\[ N: \] Set of all customer locations \( i \in \{P, D\} \); \( N_k \): Subset of customer locations visited by vehicle \( k \) with \( N_k \in N \).

\[ P \] Set of all pick-up locations \( i \in \{1, \ldots, n\} \); \( P_k \): Subset of pick-up locations visited by vehicle \( k \) with \( P_k \in P \).

\[ D \] Set of all delivery locations \( i \in \{n+1, \ldots, 2n\} \); \( D_k \): Subset of delivery locations visited by vehicle \( k \) with \( D_k \in D \).

\[ A: \] Set of all arcs with \( A = V \times V \); \( A_k \): Subset of arcs visited by vehicle \( k \) with \( A_k \in A \).

\[ G: \] Set of the complete network with \( G = \{V, A\} \); \( G_k \): Subset of the network visited by vehicle \( k \) with \( G_k \in G \).

\[ Q: \] Set of positions a vehicle can visit in a route with the depot always being at position 0.

\[ Q(i, j): \] Set of all positions in which node \( j \) can be visited from node \( i \), with \( Q(0,j) = \{1\} \). Hence, the first customer in a route is always at position 1 of a route. Also the depot can never be at position 1, consequently \( 1 \notin Q(i, 0) \).

\[ K: \] Set of all vehicles with \( K = \{1, \ldots, k^{\text{max}}\} \).

\[ X: \] Set of horizontal subarea identifiers for forecasting with \( X = \{1, \ldots, \text{gridnumber}\} \).

\[ Y: \] Set of vertical subarea identifiers for forecasting with \( Y = \{1, \ldots, \text{gridnumber}\} \).

\[ S: \] Set of all subareas in serving area for forecasting with \( S = X \times Y \).
**List of Notations (Variables and Parameters)**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_{ijk}$</td>
<td>Decision variable with $w_{ijk} = 1$ if vehicle $k$ drives from node $i \in V$ to node $j \in V$ at position $k \in K$, and $w_{ijk} = 0$ otherwise.</td>
</tr>
<tr>
<td>$c_{ijk}$</td>
<td>Cost for travelling from node $i \in V$ to node $j \in V$.</td>
</tr>
<tr>
<td>$T_{ik}$</td>
<td>Time when vehicle $k \in K$ begins service at node $i \in V$.</td>
</tr>
<tr>
<td>$L_{ik}$</td>
<td>Total load of vehicle $k \in K$ at location $i \in V$.</td>
</tr>
<tr>
<td>$o(k)$</td>
<td>Origin of vehicle $k \in K$ at the start of the driving period.</td>
</tr>
<tr>
<td>$d(k)$</td>
<td>Destination of vehicle $k \in K$ at the end of the driving period.</td>
</tr>
<tr>
<td>$a_i$</td>
<td>Beginning of service time window of node $i \in V$.</td>
</tr>
<tr>
<td>$b_i$</td>
<td>End of service time window of node $i \in V$.</td>
</tr>
<tr>
<td>$s_i$</td>
<td>Service time required to serve customer $i \in V$.</td>
</tr>
<tr>
<td>$tt_{ijk}$</td>
<td>Travel time of vehicle $k$ between node $i \in V$ to node $j \in V$.</td>
</tr>
<tr>
<td>$d_i$</td>
<td>Load for request $i \in V$.</td>
</tr>
<tr>
<td>$l_i$</td>
<td>Load to be transported at location $i$ with $l_i = d$ and $l_{i+1} = -d$.</td>
</tr>
<tr>
<td>$C$</td>
<td>Vehicle capacity.</td>
</tr>
<tr>
<td>$k_{max}$</td>
<td>Maximum number of vehicles available.</td>
</tr>
<tr>
<td>$s_{i,n+i}$</td>
<td>Identifier for request $s$ from pick-up location $i \in P$ to delivery location $n + i \in D$.</td>
</tr>
<tr>
<td>$z_{i,n+i}^a$</td>
<td>Fleet status for request $s_{i,n+i}$ from $i$ to $n + i$ with superscript $a \in {0,1}$ and $a = 1$ if $s_{i,n+i}$ is included in the routing and $a = 0$ otherwise.</td>
</tr>
<tr>
<td>$t$</td>
<td>Current time within regarded period.</td>
</tr>
<tr>
<td>$c(s_{i,n+i})$</td>
<td>Marginal costs of the operator for serving request $s_{i,n+i}$.</td>
</tr>
<tr>
<td>$d(z_{i,n+i}^a,t)$</td>
<td>Total distance for routing with fleet status $z_{i,n+i}^a$ at time of calculation $t$.</td>
</tr>
<tr>
<td>$ACC(s_{i,n+i})$</td>
<td>Customer’s acceptance of transport bid price</td>
</tr>
<tr>
<td>$p(s_{i,n+i})$</td>
<td>Price/bid for transport request $s_{i,n+i}$.</td>
</tr>
<tr>
<td>$p_c(s_{i,n+i})$</td>
<td>Customer’s expected maximum price/upper bound for $s_{i,n+i}/$ shadow price for $s_{i,n+i}$.</td>
</tr>
<tr>
<td>$DD(s_{i,n+i})$</td>
<td>Direct distance for request $s_{i,n+i}$.</td>
</tr>
<tr>
<td>$C_t$</td>
<td>Capacity level at time $t$.</td>
</tr>
</tbody>
</table>
\textbf{DTC} \quad \text{Demand to come (forecast)} \\
\textbf{T} \quad \text{Time of service execution} \\
\textbf{t} \quad \text{Current time} \\
\textbf{p}_B(s_{i,n+i}) \quad \text{Competitor’s lowest price for transport } s_{i,n+i} \\
\textbf{LB} \quad \text{Lower bound to } p(s_{i,n+i}); \text{ determined by capacity control} \\
\textbf{subarea}_{xy} \quad \text{Regarded } \text{subarea}_{xy} \text{ with } x,y \in S. \\
\textbf{demExp}_{xy} \quad \text{Expected demand for } \text{subarea}_{xy} \text{ with } x,y \in S. \\
\textbf{\delta} \quad \text{Discount parameter.}
Acknowledgements

First of all, I do not know how to enough express my gratitude to my coordinating supervisor, Associate Professor Doina Olaru; the completion of this project would not have been possible without her guidance and friendship. Doina has been an inspirational example of a mentor and scholar to me, our discussions as well as her ideas, enthusiasm, curiosity and rigour motivated and kept me on the right path throughout the past 3.5 years. Thank you for giving me a good feeling about this, from the first contact emails to the thesis submissions, and for so many working hours and meetings we had to discuss, improve and finalise this project.

Secondly, I would like to thank my co-supervisor Dr Brett Smith, who has always been encouraging and supportive during my stay in Perth. He only had a small share in my supervision but was always there to give me guidance and help if I needed to talk to someone.

Thirdly, I would like to thank Emeritus Professor John Taplin, who was willing to share his wealth of experience and knowledge. He provided valuable comments and helped me to regard my research from a different angle.

I am profoundly grateful to Professor Teodor Gabriel Crainic for enabling me to do my external stay at the Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport (CIRRELT) in Montréal, Canada. Our discussions truly helped me to improve my thesis and gave me meaningful inspiration for my work. During this visit I met a number of peers and who were very welcoming, interested in my work and willing to discuss ideas.

My sincere thanks go to Professor Sharon Biermann and the people from the Planning and Transport Research Centre for including me in their group, introducing me to regional planning problems and giving me the opportunity to work on them.

The administrative team of the UWA Business School and the Graduate Research School were an enormous help with many administrative and organisational issues; especially Mei Han and Adam Hearman who put so much effort in organising students’ events or putting in applications for travel award applications.

I would like to acknowledge that my research was funded and supported by the PATREC scholarship, the International Postgraduate Research Scholarship (IPRS), the UWA Top-up scholarship and the Australian Postgraduate Award (APA). During the candidature, IPRS and APA were converted into the Australian Research Training Program Stipend (RTP).
I would also like to thank all the colleagues and friends in Perth who helped me to have such a wonderful time during the past years. It was great having lunch, playing football or meeting up with you and discussing issues that come up on this journey, from small to big problems, from a short chat on the hallway to evenings spent at the pub.

Finally, I would like to thank my sister Kathrin and my parents Regina and Klaus for their endless love and support, not only throughout my PhD years, but for my whole life. Without their encouragement and understanding, this would not have been possible. You helped me to feel close, even when being on the other side of the planet. Thank you for your understanding, your advice and for talking me strong when I was in doubt of my work.
1 Introduction

This thesis investigates the challenge faced by road freight carriers in acquiring transport jobs and minimising costs in an increasingly competitive environment. A decision support system is developed, implemented and evaluated, adopting a profit maximisation perspective.

This chapter provides an introduction to the problem that is investigated. First, it describes the challenges of small to medium sized operators in road freight transport (with a focus on Australia) in a research motivation section (1.1). Afterwards, the scope of the thesis is elaborated in the problem description (1.1.2). Section 1.2 presents the objective of the thesis and the specific research questions to be answered. Finally, in section 1.3, the structure of the thesis is described in detail.

1.1 Research motivation

Large parts of the economy of industrialised countries rely on road freight transport. In Australia, thousands of trucks navigate the road system every day, serving primary, secondary and tertiary economic sectors. While for most cases the transport cost only accounts for a fraction of the price of the final product, drop outs or delays are a threat to supply chains and can result in companies being forced to stop their production completely until certain goods arrive (Chopra and Sodhi, 2004).

Consequently, a reliable and healthy road freight industry is an important factor to account for. In Australia, more than 80% of freight carriers are single truck operators (Bureau of Infrastructure Transport and Regional Economics (BITRE), 2014). Despite their considerable share of the market in moving goods, these operators are under enormous pressure due to the operational costs and price competition created by a few big players. Understanding the operation of road freight transport and how it can be improved is a vital concern, given the importance of a reliable and healthy road freight industry to the economy.

1.1.1 Features of Australian road freight transport

In Australia, the majority of non-bulk freight is transported by road, with the industry contributing almost 2% to the country’s GDP (BITRE, 2016; Ghaderi et al., 2015). This indicates a large demand for reliable (on-time) transport services with competitive rates and
extended coverage. Historically, this service level has been maintained by market regulations and fixed transport rates. However, these regulations have been slowly removed (to open the market, increase competition and hence efficiency) in the past decades (Australia: from 1954; Europe: from 1988; USA: from 1980). The deregulation has brought many positive effects, such as a reduction in freight rates and greater efficiency (Cooper, 1991). Nevertheless, small transport carriers commonly struggle to compete with larger firms, who can afford to operate hub-and-spoke networks and may offer more frequent services.

1.1.2 Problem description

This thesis aims to investigate operations and costing/pricing in the non-bulk freight transport sector with the main focus on the Australian market. As indicated, the main interest lies in the operation of smaller companies using up to 25 vehicles, examining trips from origin to destination. Further, the focus is on palletised transport items due to the high degree of industry usage and increasing standardisation. Transport is assumed to be regional and vehicle routes cover time-frames with a maximum of one week.

Transport requests are characterised by a pick-up and a delivery location, visitation time windows for both locations, a request load size and service times for loading and unloading. Transport requests are received continuously and may arrive before, as well as during the weekly planning and driving time. It is assumed that the planner can change the vehicle schedules as long as the goods that have already been picked-up get delivered to the right location by the pick-up vehicle. As transport requests are of varying sizes, multiple customer consignments can be combined in the same vehicle (this is also referred to as ‘less-than-truckload’ transport or abbreviated LTL).

1.2 Research objectives and contributions

1.2.1 Research objective

This thesis answers the call for a better understanding of road freight transport requests under competitive market conditions when information on already acquired jobs is known. The aim is to give small transport operators a simple tool that helps them to analyse incoming requests (of pick-up and delivery type) with respect to their current situation and the market conditions. The study investigates whether the analysis of existing data can help transport operators to make better demand management decisions and improve their competitive situation and profits,
despite being restricted by limited business capital. Hence, the research objective and approach can be formulated as follows:

*This thesis adopts a profit maximisation approach to determine transport costs and set prices for road freight carriers in an integral manner, considering the competitive market environment and potential future requests.*

1.2.2 Research questions

A number of specific research questions were proposed to guide the investigation and lead the reader through the problem, towards the realisation of the objective of this thesis.

- **What is the current competitive market situation on the LTL road freight market in Australia and other industrialised countries and how does the literature address this topic?**
  Before defining the optimisation problem, it is important to understand the road freight transport market’s competitive environment, structure and current pricing practices. This was achieved through analysis of industry reports and scholarly work that has been undertaken on the topic to date.

- **What are the determining factors necessary to define transport prices in practice and are there decision support tools available at present?**
  Chapter 2 of this thesis provides a comprehensive overview of the RFT pricing process and the available decision support systems.

- **Based on the data that is available to the industry, which methods and techniques identified in the literature can bring benefits to the industry and what are the limitations?**
  The overall optimisation problem was split into two sub-problems: (1) the cost determination problem; and (2) the pricing and request selection problem. Chapter 3 discusses both sub-problems and the methods that can be implemented.

- **What are the solving methods available for this type of problem and how can it be addressed efficiently?**
  Chapter 4 gives an overview of the mathematical methods for solving this type of problem and describes in detail the heuristic implemented for this research. Experimental results for the evaluation and validation of the heuristic are also provided.
What is the advantage to freight operators of having better knowledge of their costs and the competitive market situation?

Based on the algorithms applied to the two optimisation sub-problems (Chapter 3), a variety of experiments have been conducted, with the results presented in Chapter 5.

Are the findings transferable to the real-world, and what are the limitations and important factors to consider for optimising operation and pricing in road freight transport?

Chapter 6 presents a case study that applies the optimisation system presented in Chapter 3 to real-world data. In order to realise this, approximately 9,000 transport requests, recorded from an online marketplace for road freight transport requests in Australia were analysed. This chapter also highlights the limitations of the application.

1.2.3 Contributions

This thesis makes a number of contributions to the operations research (OR) literature and future developments on freight carrying decision support tools. This includes modelling, methodology and application.

Despite the high number of publications on vehicle routing, profit optimisation for road freight carriers has rarely been investigated in real-world conditions. To date, there is no integrated profit optimising system for pick-up and delivery type requests that solves large-scale problems in adequate time. Figliozzi et al. (2007) investigated a similar problem, but they considered VRP requests for truck-load size shipments, instead of pick-up and delivery (PDP) requests for less-than-truckload requests, and they applied a distinct methodology to solve the problem (backwards induction rather than Tabu Search). Most other research only addressed parts of the problem, for example profit optimisation, but not market dynamics. In this research, the problem was divided into two components, for which a comprehensive literature review has been undertaken. Then, an algorithm that accommodates both the cost minimisation and pricing problem has been developed. The approach used here applies a dynamic pick-up and delivery problem with time windows (DPDPTW), including request forecasting based on historic

1 The Vehicle Routing Problem (VRP) considers the visitation of single customer locations with a fleet of vehicles.
demand and dynamic pricing and capacity control methods. An adaptive Tabu Search heuristic has been implemented, validated and applied to solve the two sub-problems.

A variety of computational experiments have been undertaken in order to verify the optimising system in a holistic manner, as well as the implemented methods separately. The results show that cost estimations for transport requests based on myopic approaches (e.g. distance or time) are not a good reflection of actual costs. Better price estimations enable operators to be more competitive, avoid accepting less profitable requests and consequently increase their profits. An analysis of the marginal/incremental costs shows that costs per kilometre tend to decrease with rising numbers of assigned transport requests. Consequently, a number of discount policies and a forecasting method are proposed to subsidise early requests. While the policy settings are case-dependent, the experiments demonstrate superiority to the benchmark solutions in all cases, leading to higher total profits.

Finally, to apply the findings to a real-world scenario, a case study on the Australian road freight market has been conducted. Approximately 9,000 transport requests have been collected from an online transport marketplace and transformed into ‘cases’, representing realistic market scenarios for road freight operators. Experiments verify the results found in the artificial test cases.

1.3 Thesis organisation

The thesis is organised as follows. In Chapter 2, the market situation and existing research on profit optimisation in RFT are reviewed. First, a comprehensive overview of the road freight transport industry and its structure is provided, with a focus on the competitive aspects of the market. Then, the carrier’s cost structure and bid-price generation are considered. As indicated, the overall optimisation problem was separated into two sub-problems: (1) the cost determination problem, which is mainly related to vehicle routing; and (2) the pricing and request selection problem, which has many similarities to the revenue management field. At the end of Chapter 2, requirements for decision support systems in road freight transport are also discussed.

Chapter 3 describes the optimisation system and the models that have been considered for this thesis. The chapter starts with the problem formulation and then presents the two sub-problems.
Finally, the practical applicability of the proposed methods for road freight transport is discussed.

Chapter 4 deals with the mataheuristic that has been developed to solve the problem described in Chapter 3. First, an introduction to different solving approaches is given. Then, the applied heuristics, including construction heuristics and improvement heuristics, are described in detail. Higher level heuristic procedures, such as the tabu criteria and adaptation of heuristic selection probabilities are considered comprehensively. Finally, results of the heuristic evaluation are provided.

Chapter 5 of the thesis presents the computational experiments and their results. First, the test cases that were used are described, then, the experimental setup, with discussion of the results follows. The experiments can be classified as: (1) Comparison of myopic cost values to the marginal costs; (2) Benefits of marginal cost knowledge; (3) Effects of applying discounts aimed at increasing competitiveness; and (4) Advantages of forecasting.

In Chapter 6, a case study for the Australian road freight market is provided. Data collection and test case generation are described. Results for the application of the optimisation system to the experiments with real-world data are then shown and discussed.

Chapter 7 concludes the thesis, summarising the investigated problem, applied methodology as well as the results found and highlighting the innovations brought by this research. Limitations of the study and future extensions are also pointed out.
2 Literature review and market conditions

This section reviews the literature and applications regarding profit optimisation with relevance to road freight transport (RFT). An investigation of the topic reveals that job acquisition, cost optimisation, competitive market situation and revenue/profit maximisation are key for understanding decisions in RFT. Then, applied methods are reviewed and separated into two areas:

(1) The majority of operations-based cost in road freight transport is related to the time or distance travelled, or the selection of routes and visitation schedule. This group of problems has been widely studied in the field of vehicle routing and is grouped here as the cost minimisation problem;

(2) Non-operational factors are customer behaviour, job acquisition and the competitive market situation. These factors are dealt with in the price determination and request selection problem, which has been researched in the field of revenue management (RM).

Both research areas have numerous commonalities even though topics have been investigated separately.

The chapter is structured as follows.: The road freight transport market, including industry description and structure, competitive analysis and price generation, is described first; subsequently, the cost minimisation problem and price determination problem are reviewed. The chapter closes with a discussion about the significance and applicability of the RM aspects and models for road freight transport.

2.1 Market description

For a better understanding of the overall problem addressed in this research, it is important to delineate the characteristics of the RFT market. The focus of this research is on freight transport in industrialised countries (including countries in Europe and North America), which have characteristics in common with the case study country (Australia) where the work was undertaken. While the emphasis is on small and medium sized LTL (less-than-truck load) carriers, the findings may be ‘exported’ to other operators. For example, many of the problems described here, also apply to freight forwarders.
2.1.1 Industry description and structure

Road freight transport dominates the non-bulk freight market in most industrialised countries in the world (Whytcross, 2015). Supplying services to other industries, such as manufacturing, construction, wholesale and retail distribution, RFT is of major importance to maintain business flow in the regions. Given its relevance to economic growth and considering the importance and dependence of the Australian economy on transport (Commonwealth of Australia, 2015), the performance and service quality of RFT is of interest to many stakeholders (freight professionals, government organisations, planning agencies and operators). In Australia, the industry’s annual revenue over the years 2015 and 2016 was estimated at 52.8 billion AU$ with an annual growth rate of approximately 2.6% over the previous 5 years (Whytcross, 2015). While the revenue and profits only represent a small proportion of the GDP, RFT has the highest employment rank in its industry sector and the transport, postal and warehousing sector ranks 35/839 in revenue contribution to the Australian economy (Ledovskikh, 2016). Similar values can be found for other regions, with almost 2.9 million jobs in RFT in the EU-28 (European Commision, 2017) and over 2.5 million truck operator jobs in the USA (U.S. Department of Labor (Bureau of Labor Statistics), 2016).

The RFT market can be described as heterogeneous with many different services and varying information availability, time and spatial constraints (Mouchart and Vandresse, 2007). In most industrialised countries, this industry structure is characterised by a large number of small companies and only very few large operators with more than 50 trucks. For example, in Australia, more than 80% of companies are owner-operators (BITRE, 2014). The large number of small companies can also be found elsewhere: 90% of operators in the USA have 10 or less trucks and more than half (57%) of companies in Germany\(^2\) have five or less trucks (Bundesamt für Güterverkehr, 2012; Owner-Operator Independent Drivers Association, 2016).

2.1.2 Competitive analysis and market segments

In many countries, prices for road freight transport have been regulated for economic or public policy reasons for several decades (Bayliss, 1998). However, the situation has changed. The deregulation of the RFT market started in Australia in 1954, 1980 in the USA and 1988 in the European Community (Cooper, 1991). Today, due to the large number of operators, the market

\(^2\) Values for the whole European Union could not be found.
is very competitive. In Australia, the prices are usually determined by the larger companies. They have a competitive advantage as they serve larger areas, have a larger vehicle fleet and sometimes operate hub-and-spoke systems (Taha and Taylor, 1994). Due to the high competition, smaller companies have to follow the prices of the big players, unless they operate in niche markets (e.g. markets that require special equipment or markets in remote areas). The profit margins are traditionally low and the operation is sensitive to economic conditions (e.g., recent savings from the decreasing fuel price being transferred to customers (Whytcross, 2015)).

The road freight transport market can be separated into two broad segments: Contract logistics and the single shipment market. The term ‘contract logistics’ describes a long-term transport agreement between shipper and carrier. This can include a certain time period and shipment volume. On the other hand, the ‘single shipment market’ represents the trade of single transports that are not recurring. Whereas traditionally many agreements were made via telephone or facsimile, online marketplaces have become increasingly popular (Nandiraju and Regan, 2005). Trading on online marketplaces is usually designed as a ‘reverse auction’, where carriers can submit their ‘bid-price’ for a certain transport request. The shipper can then choose between the given bid-prices. Lowest bid-price is a major criterion for choosing an offer, however there may be other elements to consider, such as service level or prior experience with the carrier. Trading shipments on an online marketplace gives both shippers and carriers the advantage of reaching a higher number of market participants. By doing so, shippers can minimise the transport price whereas carriers can find the potentially best request matching their operating area. Examples of online marketplaces are Uship in North America (Uship Inc., 2014), Truckit.net in Australia (Arcube Pty Ltd, 2015) or Teleroute in Europe (Wolters Kluwer Transport Services, 2015).

2.1.3 Bid price generation and decision support systems

Freight transport requests reach the carrier from many different sources as described in section 2.1.2. The scheduler/dispatcher assesses the transport requests and ideally relies on available decision support systems. Partyka and Hall (2014) presented a survey of 17 different vehicle routing software-tools currently on the market in the USA, Europe and Asia. They stated that interface capability to other planning tools is an ongoing trend. Other than widely advertised, most planning software is not as flexible but rather specialised to industries to some extent.
Drexl (2012) investigated vehicle routing applications in practice. According to him, exact solutions for vehicle routing are not yet achievable for real-world problems as the problem size is too large compared to the existing computational capabilities. Besides, he states that vehicle routing tools will not become fully automated, but rather remain as informative decision support systems. This supports the observation of a large amount of manual planning in the industry as described in Jurczyk et al. (2006); Krajewska and Kopfer (2009).

Prices for freight carrying services can be set in various ways (Song and Regan, 2003; Triki et al., 2014). For single transport requests, the price finding consists of two steps: (1) determination of a base rate; and (2) a negotiation process (Shanahan, 2003). The carrier’s most influential factors for the determination of the base rate are weight, distance, goods classification and accessories/surcharges (Marks, 2014). After finding the base rate, the price is further negotiated around attributes on both sides: ease of servicing and shipping volumes on the shipper’s side, and capacity utilisation and business risks on the carrier’s side. Besides, the competitive market situation, existing customer requests and the corporate relationship between both parties play an important role (Combes and Lafourcade, 2005; Robinson, 2013; Smith et al., 2007). An overview of the pricing process is presented in Figure 2.1.

Another type of forwarding agreement is contract logistics, where freight carriers usually place bids for a bundle of consignments over a certain period of time. This problem is described as combinatorial auction, where the shipper takes the role of the auctioneer and the carriers face
the Bid Generation Problem (BGP) with the aim of submitting a competitive offer (Triki et al., 2014). The shipper can make various requests (lanes) available at simultaneously and carriers can bid on combinations of individual lanes (Song and Regan, 2003). Consequently, carriers are looking at a bundle of requests from which they can generate a cluster that can be served effectively. This combination lets them submit competitive offers depending on their availability and fit of certain requests, while shippers benefit from attractive bids. Today, there are software packages available that allows for selling bundles of shipping lanes and a variety of shippers, applied by large companies such as Procter & Gamble Co., Ford Motor Company or Staples Inc. (Caplice and Sheffi, 2006). While the important topic of combinatorial auctions is related to the topic presented in this thesis, it will not be further regarded here, as this thesis focuses on single transport requests. However, this could be a natural direction of investigation for future research.

2.2 The cost minimisation problem

Vehicle routing problems (VRPs) have been widely studied and various extensions are available in the literature. In this thesis, an existing model is applied and adopted to minimise the cost of a job resulting from the vehicle routing and scheduling. This section provides an introduction to vehicle routing and describes the most common models and relevant research.

2.2.1 Introduction to VRPs

For this thesis, vehicle routing is regarded as a group of problems of combinatorial optimisation and integer programming. Examples of problems are the ‘Travelling Salesman Problem’ (TSP), the ‘Vehicle Routing Problem’ (VRP) or the ‘Chinese Postman Problem’ (CPP). The TSP describes the problem of a salesman starting at a central depot, visiting a fixed number of customer positions exactly once and returning to the depot. The VRP is an extension of the TSP, characterised by multiple salesmen or vehicles. Here, the total number of location visits is split between the vehicles. Both problems can be extended by adding visiting time windows (TW) for each location; accordingly, they are becoming the TSPTW and VRPTW. In the context of this thesis, the relevant problem is the pick-up and delivery problem with time windows, which again, it is an extension of the VRP. In addition to the added time windows, the optimisation problem also includes linked pick-up and delivery locations (PD). This means that each PD set must be served by the same vehicle and the pick-up location must be visited prior to the delivery location. In addition, the vehicle load must not exceed the vehicle capacity.
at any time. The CCP on the other side has the objective of finding the shortest route of a graph while visiting all arcs.

Vehicle routing problems are widely studied and there are numerous extensions and combinations of those. As an overview, a number of common characteristics and constraints are listed in the following table as described by Eksioglu et al. (2009). Routing problems can be categorised by characteristics (Table 2.1):

**Table 2.1: Characteristics and constraints of vehicle routing problems**

<table>
<thead>
<tr>
<th>Scenario characteristics</th>
<th>Physical characteristics</th>
<th>Information availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Number of stops on route</td>
<td>1. Design of the transport network</td>
<td>1. Evolution of information</td>
</tr>
<tr>
<td>2. Load splitting constraints</td>
<td>2. Location of addresses</td>
<td>2. Quality of information</td>
</tr>
<tr>
<td>3. Customer service demand</td>
<td>3. Geographical location of</td>
<td></td>
</tr>
<tr>
<td>quality</td>
<td>customers</td>
<td></td>
</tr>
<tr>
<td>4. Request service times of</td>
<td>4. Number of points of origin</td>
<td></td>
</tr>
<tr>
<td>new customers</td>
<td>5. Number of points of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>loading/unloading facilities</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(depot)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. On-site service/Waiting times</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6. Time window structure</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7. Time horizon</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8. Backhauls</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9. Node/Arc covering constraints</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10. Type of time window</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11. Fleet size (Number of vehicles)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12. Capacity consideration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>13. Vehicle homogeneity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>14. Travel time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15. Transport cost</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16. Availability of information</td>
<td></td>
</tr>
<tr>
<td></td>
<td>17. Processing of information</td>
<td></td>
</tr>
</tbody>
</table>

Further information on vehicle routing problem definitions and notations can be found in Eiselt et al. (1995); Gutin and Punnen (2002); Kopfer et al. (2016); Toth and Vigo (2001), among others.

### 2.2.2 Dynamic vehicle routing

The problem addressed in this thesis regards the ‘evolution’ of available information, consisting of incoming transport requests over a certain planning and operation period. There are dynamic extensions for many types of vehicle routing problems. For further information, the reader is referred to Bektaş et al. (2014) and Pillac et al. (2013). To include additional information in the cost minimisation problem, the routing needs to be dynamically adapted. Psaraftis (1988, 1995) summarises the differences of dynamic vehicle routing problems and static problems as follows:
1. **Time dimension is essential**
In static problems, the time dimension is only important if time windows are considered. In dynamic problems, the time dimension is always important. This incorporates time windows, but also knowledge about current positions of the vehicles, in order to be able to include new customer requests into the routing and scheduling.

2. **The problem may be open ended**
Static routing problems mostly deal with certain time horizons and all vehicles start at and return to the depot. On the contrary, dynamic routing problems can feature both finite and infinite time horizons. If a problem is unbounded, the vehicle tours are described as (open) paths.

3. **Future information may be imprecise or unknown**
In static vehicle routing problems, all information about the customer’s request is known simultaneously, before the route execution starts. Dynamic routing problems do not provide all information at once. This is consistent with real-world problems where certainty almost never exists. Instead, dynamic problems may include the following features: subsequent arrival of transport requests, imprecise or incomplete information about requests or uncertainty about travel times or load sizes. Probabilistic information may be available.

4. **Near-term events are more important than longer-term events**
Near-term requests can be more important than requests that involve commitments that lie in the future. In competitive environments, carriers need to able to make ad-hoc decisions, anticipating potential underutilisation of the vehicles in the near future. Making commitments for future activities may cause limitations in accepting other immediate future requests.

5. **Information update mechanisms are essential**
Other than static problems, dynamic problems may feature additional sources of uncertainty that require adjustment of routes and schedule. This can be caused by changes of travel times, delayed loading and unloading processes (due to high utilisation of customer facilities), cancellations or no-shows of requests, or vehicle breakdowns. For these cases, update mechanisms are essential in solving the routing problem.

6. **Re-sequencing and reassignment decisions may be warranted**
Due to the dynamic changes of the problem, previously made decisions may become suboptimal. As a result, the customer vehicle assignments, routing and scheduling decisions can be modified during the time lapse, as long as no constraints are violated.

7. **Faster computation times are necessary**
The improvements in CPU times of heuristic algorithms for static vehicle routing problems have been tremendous over recent years. Yet, the problems do not require instantaneous solutions. For dynamic problems, however, the solutions must be found within a limited time, usually considered as seconds or minutes, which may present a challenge.

8. **Indefinite deferment mechanisms are essential**

Indefinite deferment means that serving certain customers can be delayed indefinitely. This becomes essential when a customer’s location is far from other customer locations and possible future requests may be in the same region. The deferment mechanism then delays this request to the end of the route, in order to reduce driving costs - if the shipping contract allows for this delay.

9. **Objective function may be different**

Most static problems aim to minimise driving distance or driving time. In dynamic cases, especially if the problem is open-ended, this measure is unbounded too. For that reason, the objective of a dynamic problem may be to maximise productivity or throughput. Other options are the optimisation of currently relevant assignments or the inclusion of predicted future workload.

10. **Time constraints may be different**

As information changes, and given the high degree of uncertainty in dynamic problems, time windows may be treated as ‘soft constraints’ (as opposed to ‘hard constraints’). Soft time windows can be applied through penalties for delayed visits. This is distinct from hard time window constraints where violations result in infeasibility of the overall problem.

11. **Lower flexibility to vary vehicle fleet**

In static problems, the time between routing and scheduling and tour execution is often undetermined and sometimes it allows the scheduler to change the size of the vehicle fleet. As dynamic problems require real-time decisions, it is unlikely that the size of the fleets would change (in most cases). This assumption mimics current operation conditions (thus presents a high degree of realism), where carriers do not change their fleet size on a daily basis.

12. **Queueing considerations may become important**

As dynamic vehicle routing problems require subsequent decisions, there may be times when the system gets saturated or congested and the routing algorithm loses the ability to produce good results. This can be caused by multiple requests arriving at the same time or within a short time interval. In this case, pre-evaluations, using queueing techniques, may be useful (future research may be undertaken in this direction).
Pillac et al. (2013) provided a taxonomy of dynamic vehicle routing problems and differentiated four cases in terms of *quality* and *evolution* of information which are presented in Table 2.2.

Table 2.2: Taxonomy of dynamic vehicle routing problems (Pillac et al., 2013)

<table>
<thead>
<tr>
<th>Information evolution</th>
<th>Information quality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Deterministic input</td>
</tr>
<tr>
<td>Input known beforehand</td>
<td>Static and deterministic</td>
</tr>
<tr>
<td></td>
<td>Stochastic input</td>
</tr>
<tr>
<td>Input changes over time</td>
<td>Dynamic and deterministic</td>
</tr>
<tr>
<td></td>
<td>Dynamic and stochastic</td>
</tr>
</tbody>
</table>

The *evolution* describes whether the available information for the problem changes over time (e.g. new customer arrivals). Information *quality* covers possible uncertainties that are not foreseeable, for instance, varying driving times as a result of traffic. This results in four problem cases (static and deterministic, static and stochastic, dynamic and deterministic, dynamic and stochastic).

For more information about dynamic VRPs, the reader is referred to Bektaş et al. (2014), Psaraftis et al. (2016) and Ritzinger et al. (2016), who presented comprehensive overviews on the DVRP types and their development.

### 2.2.3 Vehicle routing problems dealing with profits

Besides the fundamental problem formulations, the vehicle routing literature covers important knowledge relevant to the topic researched in this project, referring to *cost minimisation*, *revenue collection* or *competition*. A summary of these studies is presented in Table 2.3.

Feillet et al. (2005b) provided an overview of variations of the travelling salesman problem with profits. In this problem, customer visits are not mandatory, as the operator’s goal is to select the most profitable customers under given constraints. Feillet et al. (2005b) also offered linear programming formulations and surveyed common solving algorithms and heuristics. They showed that different solving approaches are needed depending on various criteria that are applied (solution quality vs. solving time) and called for more practical cases to be undertaken in the future. Archetti et al. (2014) presented a similar study for the VRP with profits. They included both TSP and VRP problems and attested that most research in this area still focuses on the TSP-type problems. Consequently, they still see open space in the research area of VRPs with profits. Feillet et al. (2005a) introduced the profitable arc routing problem,
which has the objective of maximising profits resulting from revenues associated with the visitation of arcs minus the costs that occur from traversing them. A branch-and-price algorithm was presented to solve the problem. Baykasoğlu and Kaplanoğlu (2008) proposed an activity-based costing approach for a transport operator. They contended that the real costs per transport request are often unknown and presented a real-life example of a transport carrier based in Turkey. Their process modelling and costing approach was compared to commonly used cost accounting approaches and identified substantial differences. Gribkovskaia et al. (2008) introduced the single VRP with deliveries and selective pick-ups. The uniqueness of this problem is that pick-up locations do not have to be visited if it is not profitable to do so. Construction and improvement heuristics, as well as a Tabu Search are presented and compared to benchmark results with near-optimal results. Jozefowiez et al. (2008) presented a bi-level approach for the TSP with profits. The problem has the conflicting objectives of minimising tour length and maximising collected revenues. While other approaches combine the two objectives or set one of them as a constraint, the research by Jozefowiez and colleagues aimed for Pareto optimality by searching for an efficiency frontier. An ejection chain local search\(^3\) was developed and combined with a multi-objective evolutionary algorithm. A benchmark with a successful metaheuristic for this problem showed advantages of the proposed algorithm with increasing problem sizes. Archetti et al. (2009) studied the capacitated versions of the ‘Team Operating Problem’ (TOP) and the ‘Profitable Tour Problem’ (PTP). These problems deal with the selection of a subset of customers in order to maximise profits. A branch-and-price algorithm and heuristic procedures were proposed with good results for the heuristics (less than 2% deviation from best known results on average). Gutiérrez-Jarpa et al. (2010) also used a branch-and-price algorithm for the vehicle routing problem with deliveries, selected pick-ups and time windows. They investigated five different problems with optimally solved problems for cases with up to 50 customers. In their paper, Aras et al. (2011) provided a model for vehicle routing with selective pick-ups and profits. The problem describes a scenario where a company has to collect certain goods from different pick-up locations and to pay a reservation price for the item in return. The problem for the trucking company is to maximise profits of the collected items, while considering the reservation price and operational costs. Chbichib et al. (2012) examined the profitable VRP with multiple trips. In this problem, the operator has to select the most profitable requests from a set of customers and the vehicles are allowed to visit the depot

\(^3\) A heuristic procedure where certain arcs in a cycle route are ‘ejected’ and new connection introduced after. For additional information, please refer to Glover (1992).
multiple times. Four different mathematical models were compared, with a commercial solver (for small test cases) and two heuristics (for large test cases) being applied. The heuristics showed satisfactory results.

Albareda-Sambola et al. (2014) investigated a dynamic multi-period vehicle routing problem with probabilistic information. They provided different policies, which solve prize collecting vehicle routing problems (PCVRP) for each time period and determine which customer to serve and which ones to postpone. They provided experimental results for many small to large problems and applied exact algorithms and a Variable Neighbourhood Search (VNS) heuristic. Their results show that adaptive policies result in considerable cost savings and that the effectiveness depends on the accurate definition of selection indicators.

Table 2.3: Vehicle routing with profits and/or competition

<table>
<thead>
<tr>
<th>Reference (sorted in alphabetical order)</th>
<th>Research focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Albareda-Sambola et al. (2014)</td>
<td>Dynamic multi-period VRP with prize collection and stochastic information</td>
</tr>
<tr>
<td>Aras et al. (2011)</td>
<td>Vehicle routing with selective pick-ups and profits</td>
</tr>
<tr>
<td>Archetti et al. (2009)</td>
<td>Team Operating Problem’ (TOP) and the ‘Profitable Tour Problem’ (PTP)</td>
</tr>
<tr>
<td>Archetti et al. (2014)</td>
<td>Survey on vehicle routing problems with profits</td>
</tr>
<tr>
<td>Baykasoğlu and Kaplanoğlu (2008)</td>
<td>Activity-based costing</td>
</tr>
<tr>
<td>Brotcorne et al. (2000)</td>
<td>A bi-level problem for settling freight tariffs</td>
</tr>
<tr>
<td>Chbichib et al. (2012)</td>
<td>A profitable vehicle routing problem with multiple trips</td>
</tr>
<tr>
<td>Feillet et al. (2005a)</td>
<td>A profitable arc-routing problem</td>
</tr>
<tr>
<td>(Feillet et al., 2005b)</td>
<td>Survey on TSP variants, considering profits</td>
</tr>
<tr>
<td>Figliozzi et al. (2007)</td>
<td>Pricing in a dynamic vehicle routing problem</td>
</tr>
<tr>
<td>Gribkovskaia et al. (2008)</td>
<td>Single vehicle routing problem with deliveries and selective pick-ups</td>
</tr>
<tr>
<td>Gutiérrez-Jarpa et al. (2010)</td>
<td>A branch-and-price algorithm for the VRP with deliveries, selected pick-ups and time windows</td>
</tr>
<tr>
<td>Hyytiä et al. (2012)</td>
<td>Dynamic dial-a-ride problem with travel time and workload objectives</td>
</tr>
<tr>
<td>Jozefowiez et al. (2008)</td>
<td>TSP with profits</td>
</tr>
<tr>
<td>Mes et al. (2009)</td>
<td>Sequential auctions for full truckload allocation</td>
</tr>
<tr>
<td>Sayarshad and Chow (2015)</td>
<td>A non-myopic dynamic dial-a-ride and pricing problem</td>
</tr>
<tr>
<td>Schwarze (2016)</td>
<td>The vehicle pricing game</td>
</tr>
</tbody>
</table>

---

4 In the PCVRP, not all locations have to be visited, but a prize can be collected by visiting (Tang and Wang, 2006)

5 For more information on VNS, the author refers to Mladenović and Hansen (1997).
Another research stream on road freight transport focuses on capacity utilisation. Van de Klundert and Otten (2011) were concerned with the improvement of capacity utilisation of LTL (less-than-truckload) operators. While for most carriers, full capacity utilisation has a very low likelihood, an improvement can help to increase profits without extra investments. Van de Klundert and Otten investigated three types of problems: i) the offline problem; ii) the online problem; and iii) the on-the-line problem. The ‘offline problem’ represents the case where all requests are available before starting the tour execution. For the ‘online problem’, the requests are available before execution, but arrive subsequently. In the ‘on-the-line problem’, the requests arrive subsequently and during the tour execution. Van de Klundert and Otten’s work does not include vehicle routing, but instead considers additional requests on already scheduled routes. They used an approximation algorithm and provided good computational results.

Another related research area including RFT is the pricing of transport services. Setting transport prices strongly depends on the market form, market rules and competition (for a discussion of these elements please refer to section 2.1). Brotcorne et al. (2000) were concerned with a bi-level problem for settling freight tariffs. They considered a group of carriers and a single shipper and modelled the price settling in a non-cooperative and sequential way. They put their focus on both exact problem solving (with a commercial solver) and approximate solving (for large instances, using a heuristic). The results obtained by the heuristic were near optimal, with reasonable solving times. Figliozi et al. (2007) investigated pricing in a dynamic vehicle routing problem. They extended the TSP with profits by adding a market dimension. Their model calculated marginal costs for additional transport assignments, with pricing based on the marginal costs and the opportunity costs for future requests. Their simulation-based experiments showed that static pricing was outperformed by the proposed model. Hyytiä et al. (2012) presented a non-myopic vehicle and route selection model for the dynamic dial-a-ride problem with travel time and workload objectives. The dial-a-ride-problem (DARP) is a subclass of the pick-up and delivery problem that describes the problem of collecting single customers at origin locations and dropping them off at specific delivery locations (e.g.: a taxi-cab delivery). Three control policies for non-myopic price determination were provided to

<table>
<thead>
<tr>
<th>Reference (sorted in alphabetical order)</th>
<th>Research focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vidal et al. (2013)</td>
<td>A heuristic comparison for multi-attribute problems</td>
</tr>
<tr>
<td>Vidal et al. (2016)</td>
<td>A new neighbourhood search heuristic for the vehicle routing problem with profits</td>
</tr>
</tbody>
</table>


evaluate the applied test cases and showed that the policies work well in practice. Sayarshad and Chow (2015) also presented a non-myopic dynamic dial-a-ride and pricing problem, regarding a case of public service transit and a flexible, dynamic model. While most other models apply a myopic pricing approach, Sayarshad and Chow (2015)’s research aimed to reach socially optimal tolling by regarding request queues. The results showed that this approach can outperform the policies of Hyttia et al. (2012) in social welfare. They justified their approach of non-myopic policies with a potentially increased ridership.

Schwarze (2016) introduced the ‘Vehicle Pricing Game’ (VPG) that describes vehicles as decentralised decision-making units in a transport system. For each request, separate vehicles have to determine prices resulting from driving costs with the aim to maximise profits in the competitive system. Experimental results were provided for multiple scenarios, showing that penalty options have positive influence on load imbalances within the network. Vidal et al. (2013) offered a heuristic comparison for multi-attribute problems, which partially includes profits. Vidal et al. (2016) provided a new neighbourhood search heuristic for the VRP with profits. This heuristic was compared to benchmark results and 52 new best solutions have been found.

To summarise, given that road freight transport requests are traded in different forms, using various means of communication (contract logistics, traditional job negotiations where customers ask carriers or forwarders for quotes via telephone or facsimile, or online marketplaces for the so-called ‘spot market’) and often seeking various objectives; distinct solutions are required. In his thesis, Mes (2008) investigated sequential auctions for full truckload allocation. He focused on the market side and proposed a policy for transport carriers, which takes the direct transport costs and opportunity costs for future requests into account. For the shippers, he also proposed a policy that includes ‘de-commitment’ from transport assignments, which allows the shipper to step back from a planned transport. Besides the obvious operational costs minimisation on both sides, the reduction of greenhouse gas (GHG) emissions resulting from an overall optimised system is explicitly presented.

2.3 The price determination problem

This section gives an overview of the price determination problem. While the previous two sections also included pricing considerations, they were from the point of view of vehicle routing. This section focuses on research that deals with demand management and pricing
decisions, often covered in the research area of ‘revenue management’ (RM). Besides price determination, RM covers a number of other methods that are relevant here.

The concept of revenue management (also called yield management) emerged in the airline industry in the late 1950s. Beckmann and Bobkowski (1958) first introduced stochastic process models, which led to controlled overbooking of capacity. Rothstein (1968, 1971, 1974) presented stochastic models for booking policies and capacity overbooking. As early as 1972, Littlewood presented a paper which features short-term forecasting and seat inventory management (capacity control) for airlines. This gained recognition and was later known as ‘Littlewood’s rule’. By the late 1970s, revenue management already found wide application in the airline industry and when American Airlines (AA) introduced their ‘Dynamic Inventory Allocation and Maintenance Optimizer System’ (DINAMO), RM provided AA with an immense competitive advantage (Smith et al., 1992). According to McGill and van Ryzin (1999), this was also the time when the application of RM rapidly increased and led to its present widespread application.

Today, beside its classical field of airline passenger transport, it is also applied in the car rental, hospitality and airline cargo industries. RM consists of four main elements, including demand forecasting, dynamic pricing, overbooking of capacity and capacity (or inventory) control, which describes the capacity allocation. For most RM applications, the elements complement each other in system and exchange information. However, there are some cases where only single elements are used. In the following section, the four main elements of RM are discussed in detail.

2.3.1 Capacity control

The term capacity control (also called inventory control) describes the capacity management of a sold service/good over the time horizon. More precisely, it decides whether a certain service/good should be sold at a specific price or not, while considering the capacity available and whether it should be reserved for a later (more profitable) request. McGill and van Ryzin (1999) distinguished between single-leg inventory control and segment or origin-destination control (network capacity control). The single-leg inventory control is concerned with single services, i.e. a seat on an airplane flight. The network capacity control, on the other side, decides on capacity allocation over the complete network, e.g. for a multi-leg flight. For both types, there are controls that affect the decision processes. These lie in booking limits,
protection levels and bid-price controls for the different booking classes (Talluri and Van Ryzin, 2004). Schönberger and Kopfer published a number of papers on capacity control for road freight transport (2011a, 2011b, 2012). First, they investigated subcontracting and the application boundaries of network capacity control (Schönberger and Kopfer, 2011a). They proposed a revenue management capacity control model and two control policies (booking limits and bid price controls). Their experiments proved both policies advantageous, and they were further examined in their research. In a subsequent paper, Schönberger and Kopfer (2011b) investigated the impacts of imprecise demand forecasts. In their experiments, transport requests arrive subsequently (online problem). The highest revenues are collected by the two bid-price strategies, followed by booking limits and ‘first-come-first-serve’. In their third paper, Schönberger and Kopfer (2012) investigated capacity control and demand uncertainty for RM in RFT. Their work aimed to optimise capacity utilisation, while dealing with inaccurate demand forecasts, especially about the occurrence of future requests. Their model takes into account the knowledge about already sold capacity for newly incoming requests and their bid prices.

Another way to cope with capacity control in terms of demand uncertainty is the use of flexible products. Flexible products allow the selling company the right to specify or modify some features of the products or services after selling the product (e.g., a cargo airline specifies the exact time of delivery only after the service is booked, in order to reserve capacity for urgent loads). Gallego and Phillips (2004) introduced the concept of flexible products and provided algorithms and conditions for a flexible product with two alternatives. Petrick et al. (2012) provided a number of models and control mechanisms to deal with demand uncertainty by offering flexible products. Their models included extensions of the capacity control concepts of booking limits and bid-price controls. Their numerical studies showed that flexibility brings the highest advantage to companies when demand uncertainty is high. Gönsch et al. (2014) showed that traditional Deterministic Linear Programming (DLP) approaches do not recognise the value of flexible products. They proposed a new model with an integration of flexible products in DLP and confirmed its value numerically.

Considering RFT, there is a variety of different systems and ways in which road haulage companies operate. There are companies which operate without any fixed routes, then there are companies which have scheduled line services, companies which operate a hub-and-spoke system and all kinds of blends in-between. Also, for dynamic routing, the existing capacity
gains in complexity, such as available capacity over time and position. This is one of the main reasons why RM is much more complicated in RFT than in its traditional fields.

2.3.2 Forecasting

Demand forecasting helps companies to obtain estimations about future demand. There are two major sources of customer request uncertainty: the distribution of request arrival and the distribution of customer reservation price (Lin, 2006). The request arrival distribution describes the arrival rate and amount over a selling period. The customer reservation price is the least favourable price a customer is willing to pay for a service. Accurate demand and customer behaviour forecasting are of major importance for revenue management. Still, the level of accuracy and reliability varies across industries and applications. Weatherford and Kimes (2003) gave an overview and compared different forecasting methods in the context of hotel revenue management. They distinguished between three types of forecasting: i) historical booking models; ii) advanced booking models, and iii) combined models. Historical models consider the final number of bookings on a particular night of stay, advanced models consider the distribution of bookings for a particular stay night and combined models include both. Weatherford and Kimes (2003) applied their models on the data of two hotel chains and suggested distinct best fitting approaches. Given the fact that their recommendations were different, it shows that forecasting models and their results vary depending on the applications. Recently, Weatherford (2016) offered an overview of the history of revenue management forecasting. He surveyed over 80 articles and discussed the various types of forecasting, attesting a large research potential but also stating that human assessment will remain necessary for forecasting systems’ capabilities.

Other studies such as Chow et al. (2010) and Tavasszy et al. (2012) reviewed demand forecasting from a broader perspective in terms of traffic outcome and flows. Regan and Garrido (2001) compared various freight demand and shipper behaviour models, however to date there is no research on single shipper requests.

2.3.3 Dynamic pricing

The aim of dynamic pricing (DP) is to set the price of the good or service at a level that maximises revenues, while considering a variable demand level. Many DP models require demand forecasting. Most dynamic pricing models in the literature rely on a fixed capacity,
for an exactly known time of request fulfilment. They also calculate solutions for demand that is known well in advance and results from sales estimations and strategic forecasting. For many cases, long-term forecasting in road freight transport is inaccurate. This is especially true in remote areas, such as regional Western Australia, for example. That is why Lin (2006) provided a dynamic pricing model (DPM) for service industries that does not require accurate demand forecast. The DPM includes a learning algorithm and promotes sales if the service time is approaching and the inventory level is still high. Computational experiments showed near optimality and also a high robustness in cases where forecasting from the learning algorithm failed. Van den Boer (2015) also considered learning within dynamic pricing. In addition to Lin (2006), he considered a changing market environment. The pricing policy used two estimation functions, which describe the difference between possible wrong price estimation and the actual market price. Using these functions, the model sets the price in order to minimise the estimation error over time. Two different cases are used; numerical experiments show the efficiency of the model.

Klein (2006) presented a model for network capacity control and self-adjusting bid prices. The bid prices were calculated based on the already sold capacity and the expected demand to come. The coefficients for the calibration of the prices were calculated by simulation-based optimisation. Other researchers, such as Martínez-de-Albéniz and Talluri (2011) or Gallego and Hu (2014) focused on dynamic pricing and market dynamics applying game theory. Whereas their research is of high importance for many fields of application, this study assumes a heterogeneous market with limited accessibility and thus limits the applicability of game theory. Along with different prices to their customers, companies also offer slightly different products or services. This is called diversification and enables companies to vary the price of a product by offering extras, like flexibility or additional services. For example, airline tickets can be bought with a change date or with additional luggage options.

2.3.4 Overbooking

Revenue management also features an element described as overbooking of capacity. This concept emerged from the fact that for a certain range of industries, customers purchase a service, but may not ever claim or execute it (McGill and van Ryzin, 1999). Out of this operating experience, airlines started overbooking their seat capacity with the risk of rejecting customers to check-in, should every customer show up. When a critical number of customers
are arriving, those missing out are usually offered a ‘make-up’ compensation, i.e. a better seat on a later flight, monetary rewards etc. (Talluri and Van Ryzin, 2004). The concept of overbooking is well studied in the traditional RM fields (i.e., McGill and van Ryzin, 1999; Talluri and Van Ryzin, 2004), but less in freight transport. Luo et al. (2009) investigated cargo overbooking models considering two dimensions (weight and volume). They compared their approach with a one-dimensional model and highlighted significant differences in the decision-making process. Using two dimensions allowed them to calculate more accurately the amount of overbooking, compared to one-dimensional models, which usually set limits for combined weight and volume. They provided a numerical solution with near optimality.

For the road haulage industry, there are only rare cases of customers booking a service without claiming it. Most shippers would call the carrier and cancel the service or another deal could be negotiated. Still, accepting very profitable transport requests without having the required capacity is tempting for some carriers. There are three options on how to handle overbooked transport jobs: (1) cancelling; (2) postponing; (3) outsourcing.

The first two options depend on the problem formulation and can be dealt with by penalising the carrier. The third option has gained little attention in the literature. Lee et al. (2003) proposed a system for selecting loads to be outsourced to subcontractors, presenting a real-world application in Singapore. First, a number of requests are directly selected. In a following step, remaining requests are assigned to vehicles using a greedy criterion and requests that could not be assigned are added to the set of jobs to be outsourced. Lee et al. (2003) applied a Tabu Search heuristic to solve the problem. Their results showed an increase of efficiency of 8% when compared to the existing results. A similar problem was examined by Zäpfel and Bögl (2008), who considered a parcel service company that has to decide on its operations depending on a fluctuating demand volume. Outsourcing options were included in the mathematical formulation. They applied different metaheuristics with superior results being obtained, using their Tabu Search. Moon et al. (2012) also presented a model with outsourcing options for the VRPTW (VRPTWOV). They proposed two metaheuristics and showed that the problem can be solved efficiently for realistic problem cases.

On the shippers’ side, issues and implications of breaking commitments were analysed by Mes et al. (2009), who investigated a dynamic threshold policy for delaying and breaking commitments in transport auctions. They proposed two policies: (1) shippers can postpone commitments for which they expect better commitments in the future; (2) shippers can de-
commit from a transport agreement against a penalty. The simulation-based experiments showed that both policies reduce shippers’ total costs. However, from the experiments, it appeared that de-commitment strategy results in higher transport cost for individual shippers.

2.3.5 Applied revenue management in freight transport

Despite a vast amount of revenue management applications in certain industries, the freight transport industry is almost ‘untouched’. Most applications are focused on air-cargo, container-cargo, rail-cargo and parcel transport. All these industries feature fixed transport routes and networks. Even the hub-to-customer delivery process in parcel transport follows a delivery area-vehicle-assignment. This explains why road freight is niche in the RM area. Still, research undertaken in these areas contributes specific findings to the development of freight RM and highlights the distinct features of cargo RM.

Amaruchkul et al. (2007) provided a model for single-leg air-cargo revenue management. Their model was formulated as a Markov decision process and included uncertainty about volume and weight of the shipped items until shortly before the flight. They provided six heuristics, comparing different approaches. The results obtained from their simulations suggest that it is better to use a value function including weight and volume instead of standard policies such as first-come-first-served. There are also some publications addressing revenue management implementation at large airline companies, such as KLM or Lufthansa. Billings et al. (2003) provided an overview of the implementation of revenue management in air cargo transport. They compared air cargo and air passenger services and pointed out that the success of such systems is strongly associated with an effective implementation, with its settings and the managerial willingness to adapt. Slager and Kapteijns (2004) argued that RM is underdeveloped in air cargo transport, compared to the wide applications in passenger air transport. They described the RM freight processes at KLM and the critical factors they found affecting the implementation. According to them, the combination of simple tools, management commitment and real-world applicability (in terms of simplicity) were the main reasons for the program’s success at their company. Becker and Dill (2007) further analysed the complexity of air cargo revenue management at Lufthansa. They discussed the different complexity drivers in the industry, clustered them and developed appropriate managing approaches. Their comprehensive list of complexity drivers contributes to the investigation of RM in RFT. Besides specific capacity issues considering passenger and cargo mix, they
highlighted the following features: multi-dimensionality, fleet heterogeneity, number of routing possibilities, larger number of restrictions, multiple flight-segment combinations, air cargo business structure and unequal trade lanes, stowage loss from uncertainty, short booking periods, uncertain arrival times, continuous show-up rates, market structure and insufficient data. However, the characteristics and complexities of air cargo revenue management were pointed out much earlier by Kasilingam (1997). Their problem was concerned with available load space, payload and the baggage load resulting from the passengers’ luggage. He highlighted the major differences between passenger RM and cargo RM. Recently, Wang et al. (2016) presented another real-world application that applies network capacity allocation of an intermodal barge transport system. They separated customers into three groups and differentiated the prices accordingly. Experiments showed that their capacity control outperforms FCFS (first-come-first-serve), even under inaccurate demand forecast.

2.4 Requirements for advanced decision support in the road freight market

The previous sections showed that research in the fields of vehicle routing and revenue management have substantial overlaps. Given the fact that freight carriers also aim for revenue and profit maximisation, it is only reasonable to combine and extend some of these methods in a decision support system for road freight transport. This section examines the requirements for the application of revenue management in RFT.

As already indicated, RM is widely applied in a variety of industries. Over the past 60 years, it has found its way from passenger airline, car rental and hospitality industry to insurances, air- and rail cargo. While these industries have many differences, there are also a number of commonalities. This section summarises the requirements for RM applications in general and identifies the relevant features for road freight transport. Table 2.4 concludes with the requirements for the application of RM as described by various authors.
Table 2.4: Requirements and conducive conditions for Revenue Management

<table>
<thead>
<tr>
<th>Author and Year</th>
<th>Requirements</th>
<th>Customer segmentation and product differentiation</th>
<th>Demand variability and uncertainty</th>
<th>Capacity inflexibility, high change costs and perishability</th>
<th>Data and information system infrastructure</th>
<th>Management culture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kimes (1989)</td>
<td>Ability to segment markets</td>
<td>Product sold in advance</td>
<td>Fluctuating demand</td>
<td>Relative fixed capacity</td>
<td>Perishable inventory</td>
<td>Low marginal sales costs / high marginal capacity change costs</td>
</tr>
<tr>
<td>Weatherford and Bodily (1992, pp.831)</td>
<td>Fixed capacity</td>
<td>Product is perishable</td>
<td>Low marginal sales costs / high marginal capacity change costs</td>
<td>Choice of stimulating demand (i.e. advertisement)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Harris and Pinder (1995)</td>
<td>Segmentable demand</td>
<td>Advance sale / bookings</td>
<td>Stochastic demand</td>
<td>Fixed capacity</td>
<td>Perishable inventory</td>
<td>High capacity change costs</td>
</tr>
<tr>
<td>Bobb and Veral (2008)</td>
<td>Segmentation</td>
<td>Time varied demand</td>
<td>Fixed capacity</td>
<td>Perishable inventory</td>
<td>High fixed costs</td>
<td></td>
</tr>
<tr>
<td>Talluri and Van Ryzin (2004, pp.13)</td>
<td>Customer heterogeneity</td>
<td>Price as signal of quality</td>
<td>Demand variability and uncertainty</td>
<td>Production inflexibility</td>
<td>Data and information system infrastructure</td>
<td>Management culture</td>
</tr>
<tr>
<td>Synopsis</td>
<td>Customer segmentation and product differentiation</td>
<td>Demand variability and uncertainty</td>
<td>Capacity inflexibility, high change costs and perishability</td>
<td>Data and information system infrastructure</td>
<td>Management culture</td>
<td></td>
</tr>
</tbody>
</table>

Although different authors use various concepts, the meanings overlap significantly. For further discussion, terms presented in the last row (called Synopsis) are used with the following connotations: 1) Customer segmentation and product differentiation; 2) Demand variability and uncertainty; 3) Capacity inflexibility, high change costs and perishability; 4) Data and information system infrastructure; and 5) Management culture. A reflection on these characteristics for RFT is provided next.

- In RFT, customers have different needs and urgencies for their transport requests. For this reason, customers can be segmented and the services provided can be differentiated according to their needs.

- Demand variability and uncertainty are typical characteristics of road freight transport. In general, the request arrival rate is stochastically distributed and includes seasonality (such as time of the year or harvest season, and weekly and daily variation), economic cycles, and randomness.

- Capacity inflexibility, high change cost and request perishability can also be found in RFT. In general, a transport carrier has a fixed number of vehicles (inflexibility), which can only be extended or reduced by a vehicle purchase or sale (high change costs).
Request perishability is present due to the market structure and the economic behaviour of customers. A certain request has a determined service time-window and some requests may be more urgent than others. Consequently, if a request cannot be served on time, it perishes or is served by a competitor.

- Data and information system infrastructure are not necessarily present in every business. However, IT systems are now widely available and can be purchased at reasonable costs.

- Management culture is strongly associated with the employees in every business. In the case of RFT, a successful application of RM depends not only on the policy makers, but also on the company size and the strategic and technical objectives and knowledge of the human resources.

This brief discussion shows that the basic requirements of revenue management are met in road freight transport. Nevertheless, there are many differences in the way it is applied. For this reason, a comparison of a traditional RM field (in this case ‘airline passenger transport’, APT) and road freight transport is necessary to delineate the characteristics and understand the motivation for using an application in RFT. Table 2.5 shows the various dimensions and differences between APT and road freight transport. Seven dimensions are compared across the two industries. For RFT, again, the focus is on the case study presented in this thesis.

### Table 2.5: Differences between traditional Revenue Management and Operations in Road Freight Transport
(adapted after Bartodziej et al., 2007; Billings et al., 2003; Schönberger and Kopfer, 2012)

<table>
<thead>
<tr>
<th></th>
<th>(Airline) Passenger transport</th>
<th>Road Freight Transport</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Industry profile</strong></td>
<td>Monopoly/oligopoly/poly-poly (depending on O-D pair), many customers, high value services, high accessibility</td>
<td>Oligopoly/polypoly (depending on O-D pair), low profit margins, varying market accessibility (e-commerce increases market size and accessibility)</td>
</tr>
<tr>
<td><strong>Demand variation</strong></td>
<td>1-dimension: 1 seat</td>
<td>3-dimensions plus additional characteristics: i.e. weight, handling needs, etc.</td>
</tr>
<tr>
<td><strong>Unloaded trips</strong></td>
<td>Most customers book return flights, varying capacity utilisation for airlines</td>
<td>One-way shipping is the norm, but request combinations reduce unloaded drives</td>
</tr>
<tr>
<td><strong>Service distribution</strong></td>
<td>Fixed hub-and-spoke system, multi-leg flights, fixed routes</td>
<td>Round trip routing, degrees of freedom with customer time windows, re-routing (after acceptance)</td>
</tr>
</tbody>
</table>
The airline industry market is very diverse depending on region and origin-destination pairing. For example, in Australia, there are examples of monopoly (Perth – Exmouth), oligopoly (Perth – Brisbane), and polypoly (Sydney – Melbourne) in the airline market. A relatively large number of customers are able to access the market over different platforms (online, travel agency, call centre). The prices are traditionally high. However, nowadays there are some low cost carriers (Jetstar Airways, Tigerair Australia), operating on many connections. For RFT, it is quite similar. Depending on the shipper location and the OD-pairing, a carrier monopoly, oligopoly or polypoly situation may apply. Compared to passenger air transport, freight carriers’ profit margins are relatively low and there is high competition (Gargano, 2014a). Market accessibility depends on the processes of shippers and carriers: beside traditional distribution channels, such as telephone or facsimile, an increasing number of transport services are traded on online marketplaces. Accordingly, market accessibility can vary strongly, between low and high. For the demand variation and product/service characteristic, there are significant differences between the two markets. In airline passenger transport, the product is always one-dimensional (sales item: one seat), whereas the majority of transported goods in RFT are three dimensional (length x width x height) and have additional characteristics such as weight or special handling needs etc. Another difference between the two industries is ‘unloaded trips/drives’ (shown in row three of Table 2.4) and is described by the balance of trips by direction. For APT, the majority of customers book return flights. Hence, airlines can balance the used capacity in both directions. For RFT, return trips are an exception. Shippers usually send out goods from a production site to a purchaser’s location or relocate certain goods. For this reason, balancing capacity utilisation on vehicle trips is very difficult unless there is a natural balance occurring from multiple shippers’ requests. Otherwise, carriers have to deal with completely or partly empty trucks on certain route sections, which is usually the case. The service distribution of both industries is in most cases different again. Whereas

<table>
<thead>
<tr>
<th>Request distribution and characteristics</th>
<th>(Airline) Passenger transport</th>
<th>Road Freight Transport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spare capacity</td>
<td>High number of similar requests, except class differentiation and upgrades</td>
<td>Requests vary, many origins/many destinations, by time of day, day of the week, etc.</td>
</tr>
<tr>
<td>Revenue variation</td>
<td>No (ad-hoc discounts are never agreed)</td>
<td>Yes (ad-hoc discounts are often agreed)</td>
</tr>
</tbody>
</table>

Spare capacity can be filled as a number of possible customers willing to take a particular flight does not show-up for boarding; price adaption possible. Spare capacity cannot always be filled. Routing can be adapted to fill spare capacity, price and route adaptation are possible.
airlines operate a strategically planned route network of single OD-pairs, RFT is concerned with an operating region and adjusts the vehicle routing depending on transport requests for each planning period. This is a major distinction between the markets, and the incremental costs for each RFT service are different. There are a number of operators offering liner transport or operating hub-and spoke systems. However, these are not regarded in this thesis - as already indicated. The fifth row of Table 2.5 shows the distribution and characteristics of customers’ requests. In the airline industry, a large number of similar requests appear for the different OD-pairs. Customers choose between the available categories or classes (i.e. economy, premium economy, business, first) and can purchase additional services, such as extra luggage or flexibility of changing the date of the trip. In RFT, the transport OD-pairs are of various properties in terms of location and time-window. Both industries are affected by seasonality and unused (spare) capacity. As for APT, the booking period starts well in advance and the airlines can react to unreached booking limits. They can offer discounts and try to fill the capacity. For road freight transport operators, high discounts are rarely an option, as the profit margins are generally quite low and operators are not working on a fixed network. Instead, they can try filling their spare capacity on the spot market. Revenue variation, as listed in the last row of Table 2.5 does not apply for APT; the sales price is set according to a sales scheme. This is different in RFT, where negotiations between shipper and carrier are the norm.

The list of requirements shows that an application of RM in RFT is generally possible, although, the comparison of the industries reveals that RFT is much more complex than APT in this context. In addition, the market structure is very different to traditional RM industries. In particular, the market accessibility and booking procedure, as well as the average fleet size of freight carriers and their capabilities to introduce IT varies significantly across the sector.

As described in the market analysis of section 2.1, the RFT market is characterised by a high number of small companies. Consequently, customer interaction, negotiation, routing and scheduling include a large amount of manual labour. Traditional revenue management industries on the contrary (airlines, car rental companies and hotel chains) have special departments for operations planning. Considering this and the higher complexity of the RM related market characteristics, the missing applications of RM for RFT are not surprising. However, recent changes in the market dynamics and computational capabilities bring some ‘wind of change’ into the RFT industry: 1) The trade volume on online freight marketplaces is increasing, resulting in a better market accessibility for shippers and carriers. 2) IT-systems are
widely available, the computational capacity is at a high level and the costs are reasonable; therefore, systems can potentially be operated by every agent. 3) The software market is increasing, including internet and mobile phone applications that can be run on smartphones. Platforms like ‘eBay’ or ‘Uber’ revolutionised their industries with this capability.
3 Optimisation System

This chapter describes the profit optimisation system that represents the central part of this research. It is based on findings from previous scholarly work and can be applied as a stand-alone framework or as decision support system for the scheduler.

The chapter develops as follows: First, the optimisation problem is specified and formulated; then, the solution approach and the architecture of the applied model are presented; and finally, the individual elements of the system are described in detail.

3.1 Problem formulation

The thesis is regarding a dynamic version of the pick-up and delivery problem with time windows, where a road freight carrier is aiming to optimise profits by evaluating transport requests and setting bid-prices in a competitive market environment.

The carrier operates a central depot and has a homogeneous fleet of vehicles with restricted capacity. Customer transport requests are characterised by pick-up and delivery locations, load volumes, associated costs for the travel between locations, time windows for the pick-up and delivery locations, and service times for loading/unloading operations. For simplicity, travel costs are considered proportional to travel distances and/or travel times.

The planning period covers a pre-driving (one-week) and a driving (one-week) period. The requests are arriving continuously, starting at the beginning of the pre-driving period throughout the execution period. The carrier is expected to have some probabilistic knowledge about future requests for the different sub-regions of their overall service area.

The requests are traded in a second price auction which is illustrated in Figure 3.1: The illustration shows a number of bids \((b_1, \ldots, b_n)\) over the auction period. The carrier with the lowest bid \((b_n)\) gets awarded the request assignment for the price of the second lowest bidder \((b_{n-1})\). The bidding process is transparent, therefore the carrier knows the current lowest bid. The carrier is not required to participate in all auctions, however, the decision on whether to make a bid and the bid-price must be made before the auction ends. For the experiments in this thesis, decisions have to be made immediately and knowledge of the market price is assumed.
The carrier’s objective is to make bids on the transport requests so that the overall profit for the planning period is maximised. She/he faces the problem of selecting the most profitable requests considering a combination of current and future requests, thus accounting for uncertainty. The problem could also be formulated as PDPTW with Profits (with profits being equal to the difference between market prices and costs), but here the algorithm is applied to costs and the profit optimisation follows in a separate, second step.

All awarded requests must be assigned to a vehicle and served on feasible routes that start and end at the depot. Not all vehicles have to be incorporated in the routing. Participating vehicles can start at the depot any time after the beginning of the driving period and should return no later than the end of the driving period. Each awarded request must be visited only once. The pick-up & delivery (PD) customer pairs have to be visited in a certain order, where the pick-up location precedes the delivery location. Service at the customer locations can only be started in the admissible time windows and has the duration of the corresponding service time. If a vehicle arrives before the start of the time window, it is allowed to wait. Late arrivals are penalised with a value equal to the delay time. The vehicle capacity must not be exceeded with respect to the loads of all PD pairs. At any time during the planning period, PD pairs can be reassigned to other vehicles, unless the pick-up location has already been visited.

The proposed system combines vehicle routing and a number of RM methods to optimise the yield from accepting new transport requests, given the market situation and offering a transport bid that is competitive and at the same time maximises revenues. Although not included in this research, the inventory of existing and incoming requests may be examined for ‘unfavourable requests’ which could be outsourced if necessary. This could be an aspect for future research.
3.2 Problem approach

The sub-problems of cost determination and pricing/request selection, combined in this research, are presented in Figure 3.2:

![Figure 3.2: Modelling approach for solving the optimisation problem](image)

As described in Chapter 2, the overall optimisation problem was separated into: (1) the cost determination sub-problem; and (2) the pricing/request selection sub-problem. A dynamic pick-up and delivery method with time windows is chosen for solving sub-problem (1). For (2) a number of methods related to RM are embedded.

Once the marginal cost value for request $s_{ij}$ is determined, the second sub-problem is solved considering a number of assumptions. As in the spot market auctions, shippers ask carriers to submit bid-price proposals for their shipments. Carriers submit their prices and select the most attractive proposal. While selecting the most preferred proposal should perhaps consider multiple attributes (i.e. price, service ratings, experience), here the price is deemed as the decisive factor. Hence, the price setting may result in approval or rejection of a request. The decision for sub-problem (2) relies on dynamic pricing and capacity control (given the cost $c(s_{ij})$) and a forecasting technique. Details are presented in section 3.2.3.

3.2.1 The pick-up and delivery problem with time windows (PDPTW)

In this thesis, a dynamic version of the pick-up and delivery problem with time windows (PDPTW) is applied to solve the cost optimisation sub-problem (1). The formulation is based on Dumas et al. (1991) with position-based identifiers (e.g. Dethloff, 1994).
The PDPTW refers to an optimisation problem with a number of transport requests, each characterised by a pick-up and a delivery location, load size, service time and time windows for both locations. The objective is to visit all customers with a number of vehicles \( k \), while taking account of the PD sequence and minimising the travel time or distance. Further, the capacity of the vehicles used must not be exceeded and there is a central depot from which the \( k \) vehicles leave and where the vehicles return after visiting the customers. The problem formulation is presented next.

### 3.2.1.1 Network construction

All customer requests are identified by two nodes, \( i \) and \( n + i \), which are the pick-up and delivery locations. The set of pick-up locations is denoted as \( P = \{1, \ldots, n\} \) and the set of delivery locations as \( D = \{n + 1, \ldots, 2n\} \). The set of all customer locations is denoted as \( N = P \cup D \). Requests consist of a demand \( d_i \) that has to be transported from location \( i \) to \( n + i \). The related vehicle loads at \( i \) and \( n + i \) are denoted as \( l_i = d_i \) and \( l_{n+i} = -d_i \), respectively.

Let us identify \( K \) as the set of vehicles. As each vehicle \( k \) serves a certain subset of customers, \( N_k = P_k \cup D_k \) denotes the set of customers served by vehicle \( k \) with the corresponding subsets \( N_k \subseteq N \), \( P_k \subseteq P \) and \( D_k \subseteq D \). For each vehicle \( k \), a network \( G_k = (V_k, A_k) \) has to be defined with the subset of nodes \( V_k = N_k \cup \{o(k), d(k)\} \) and all feasible arcs \( A_k = V_k \times V_k \). The origin of \( k \) is \( o(k) \) and the destination \( d(k) \). The vehicle capacity is denoted with \( C_k \), the travel time between two distinct nodes \( i, j \in V_k \) by \( t_{ijk} \) and the travel costs by \( c_{ijk} \).

All vehicles are assumed to leave unloaded from their origin depot no earlier than time \( a_{o(k)} \) and return no later than \( b_{o(k)} \). Admissible pick-up and delivery routes follow a feasible path in \( G_k \), starting at \( o(k) \) and ending at \( d(k) \). Each node must be visited exactly once and reached within its admissible time window \( [a_i, b_i] \), when the service with the corresponding time \( s_i \) begins. If a vehicle arrives too early, it is allowed to wait until the time window opens at \( a_i \).

### 3.2.1.2 Notation

For the mathematical formulation of the problem, the following sets are required:

\[ V: \quad \text{Set of all locations } i \in \{0, \ldots, n+1\} \text{ such that } i=0: \text{ depot, and } i=1, \ldots, 2n: \text{ customer nodes}; \ V_k: \text{ Subset of locations visited by vehicle } k \text{ with } V_k \subseteq V. \]
\(N:\) Set of all customer locations \(i \in \{P, D\}\); \(N_k:\) Subset of customer locations visited by vehicle \(k\) with \(N_k \subseteq N\).

\(P:\) Set of all pick-up locations \(i \in \{1, \ldots, n\}\); \(P_k:\) Subset of pick-up locations visited by vehicle \(k\) with \(P_k \subseteq P\).

\(D:\) Set of all delivery locations \(i \in \{n + 1, \ldots, 2n\}\); \(D_k:\) Subset of delivery locations visited by vehicle \(k\) with \(D_k \subseteq D\).

\(A:\) Set of all arcs with \(A = V \times V\); \(A_k:\) Subset of arcs visited by vehicle \(k\) with \(A_k \subseteq A\).

\(G:\) Set of the complete network with \(G = \{V, A\}\); \(G_k:\) Subset of the network visited by vehicle \(k\) with \(G_k \subseteq G\).

\(Q:\) Set of positions a vehicle can visit in a route with the depot always being at position 0.

\(Q(i, j):\) Set of all positions in which node \(j\) can be visited from node \(i\), with \(Q(0, i) = \{1\}\). Hence, the first customer in a route is always at position 1 of a route. Also, the depot can never be at position 1, consequently \(1 \notin Q(i, 0)\).

\(K:\) Set of all vehicles with \(K = \{1, \ldots, k_{\text{max}}\}\).

The following variables are used:

\(w_{ijk}:\) Decision variable with \(w_{ijk} = 1\) if vehicle \(k\) drives from node \(i \in V\) to node \(j \in V\) at position \(k \in K\), and \(w_{ijk} = 0\) otherwise.

\(c_{ijk}:\) Cost for travelling from node \(i \in V\) to node \(j \in V\).

\(T_{ik}:\) Time when vehicle \(k \in K\) begins service at node \(i \in V\).

\(L_{ik}:\) Total load of vehicle \(k \in K\) at location \(i \in V\).

\(o(k):\) Origin of vehicle \(k \in K\) at the start of the driving period.

\(d(k):\) Destination of vehicle \(k \in K\) at the end of the driving period.

The following parameters are defined:

\(a_i:\) Beginning of service time window of node \(i \in V\).

\(b_i:\) End of service time window of node \(i \in V\).

\(s_i:\) Service time required to serve customer \(i \in V\).

\(tt_{ijk}:\) Travel time of vehicle \(k\) between node \(i \in V\) to node \(j \in V\).
\(d_i\)  
\(l_i\)  
\(C\)  
\(k^{\text{max}}\)

- Load for request \(i \in V\).
- Load to be transported at location \(i\) with \(l_i = d\) and \(l_{i+1} = -d\).
- Vehicle capacity.
- Maximum number of vehicles available.

### 3.2.1.3 Mathematical formulation for the static PDPTW

The mathematical formulation of the PDPTW involves the binary flow variable \(w_{ijqk}\), which determines whether an arc \((i,j) \in A_k\) from node \(i\) to node \(j\) is traversed by vehicle \(k\) (\(w_{ijqk} = 1\) if the arc is used and 0 otherwise) on position \(q \in Q\). Variable \(L_{ik}\) denotes the load of vehicle \(k\) after servicing node \(i \in V_k\). The objective function is:

\[
\text{minimise} \quad \sum_{k \in K} \sum_{q \in Q} \sum_{i \in I} \sum_{j \in J} c_{ijqk} w_{ijqk}
\]  
(3.1)

subject to

\[
\sum_{j \in N_k} \sum_{q \in Q} w_{ijqk} = 1 \quad \forall i \in P_k
\]  
(3.2)

\[
\sum_{j \in N_k} \sum_{q \in Q} w_{ijqk} - \sum_{j \in N_k} \sum_{q \in Q} w_{j,n+i,q,k} = 0 \quad \forall k \in K, \ i \in P_k
\]  
(3.3)

\[
\sum_{j \in P_2 \cup (d(k))} w_{o(k),j,1,k} = 1 \quad \forall k \in K
\]  
(3.4)

\[
\sum_{i \in N_k \cup \{o(k)\}} \sum_{q \in Q} w_{ijqk} - \sum_{i \in N_k \cup \{d(k)\}} \sum_{q \in Q} w_{j,1,q,k} = 0 \quad \forall \ k \in K, \ j \in N_k
\]  
(3.5)

\[
\sum_{i \in D_k \cup \{o(k)\}} \sum_{q \in Q} w_{i,d(k),q,k} = 1 \quad \forall k \in K
\]  
(3.6)

The objective function (3.1) aims to minimise the total distance driven by the vehicles in order to serve all customers and then return to the depot. Equation (3.2) indicates that every pick-up location \(i\) has to be left exactly once towards any other location including the depot. Restriction (3.3) requires that if a pick-up location has been visited by vehicle \(k\), the same vehicle must visit the associated delivery location. Constraints (3.4)-(3.6) describe the multi-commodity
flow constraints and ensure that every starting location is left exactly once, that every customer location is reached and visited only once and that all vehicles return to the depot. In addition, the following time window and capacity constraints are defined:

\[
\left( \sum_{q \in Q} w_{ijqk} \right) \left( T_{ik} + s_k + t_{t_{ijk}} - T_{jk} \right) \leq 0 \quad \forall \ k \in K, (i, j) \in A_k \tag{3.7}
\]

\[a_i \leq T_{ik} \leq b_i \quad \forall \ k \in K, i \in V_k \tag{3.8}\]

\[T_{ik} + t_{t_{i,n+i,k}} \leq T_{n+i,k} \quad \forall \ k \in K, i \in P_k \tag{3.9}\]

\[
\left( \sum_{q \in Q} w_{ijqk} \right) \left( L_{ik} + l_j - L_{jk} \right) = 0 \quad \forall \ k \in K, (i, j) \in A_k \tag{3.10}
\]

\[l_i \leq L_{ik} \leq C_k \quad \forall \ k \in K, i \in P_k \tag{3.11}\]

Constraint (3.7) describes the arrival time \( T_{jk} \) at customer \( j \) and ensures that vehicle \( k \) cannot arrive earlier than the sum of arrival time at the previous visited customer \( i \), service time at customer \( i \), and the travel time from \( i \) to \( j \). Relation (3.8) ensures, that vehicle \( k \) does not commence its service before the node’s time window begins or after it ends. Relation (3.9) specifies that the time when vehicle \( k \) starts serving the delivery node \( n+i \) cannot be earlier than the service commencing time at the corresponding pick-up node \( i \) plus the driving time from \( i \) to \( n+i \). Equation (3.10) ensures that the load of vehicle \( k \) at customer \( j \) has to be equal to the load before visiting the previous customer \( i \) plus the additional load that has been collected/delivered at \( i \). Inequation (3.11) describes the capacity constraints for the vehicle and implies that the total vehicle load \( L_{ik} \) has to be smaller than the service load at customer \( i \).

Further, constraint (3.12) ensures that a shipment \( l_i \), that has been loaded by vehicle \( k \) at pick-up location \( i \), is unloaded at delivery location \( n+i \) by the same vehicle. Relation (3.13) identifies the starting load of vehicle \( k \). For the cases regarded here, every vehicle that starts at the depot is unloaded. Constraints (3.14) and (3.15) show that the decision variable \( w_{ijqk} \) is nonnegative and binary.

\[0 \leq L_{n+i,k} \leq C_k - l_i \quad \forall \ k \in K, n + i \in D_k \tag{3.12}\]

\[L_{o(k), k} = \text{starting load } k \quad \forall \ k \in K \tag{3.13}\]
\[ w_{ijqk} \geq 0 \quad \forall \, k \in K, q \in Q, (i, j) \in A_k \quad (3.14) \]
\[ w_{ijqk} \in \{0, 1\} \quad \forall \, k \in K, q \in Q, (i, j) \in A_k \quad (3.15) \]

### 3.2.2 The dynamic PDPTW

The previous section described the static version of the PDPTW. However, the problem investigated in this thesis requires dynamic vehicle routing, hence, a number of additional assumptions must be made. Drawing on the comparison undertaken by Psaraftis (1988, 1995), this research considers that there is no complete information before the operation starts, and subsequent requests are considered in the routing problem. Also, it assumes a fixed fleet size and soft window constraints with penalties proportional to the delay in time units. In addition, the near-term events are considered more important than events distant in the future and resequencing and reassignment may occur. Finally, based on the taxonomy of Pillac et al. (2013), the problem regarded here can be described as ‘dynamic in evolution’ and ‘deterministic in information quality’: customer requests arrive over time and are not known in advance; vehicle locations are updated during the dynamic optimisation; customer-vehicle assignments can be changed unless a customer’s load has already been collected, but no queueing mechanisms are applied.

#### 3.2.2.1 Additional notation

For the dynamic PDPTW, the objective function is constantly re-optimised as more information becomes available. Some parts of the solution are frozen as soon as customer-vehicle assignments are fixed (i.e. by reaching the pickup location), while others can be changed under the condition that no constraints are violated. The following additional notation, based on Figliozzi et al. (2007), is applied:

- \( s_{i, n+i} \) Identifier for request \( s \) from pick-up location \( i \in P \) to delivery location \( n + i \in D \).
- \( z_{i,n+i}^a \) Fleet status for request \( s_{i,n+i} \) from \( i \) to \( n + i \) with superscript \( a \in \{0,1\} \) and \( a = 1 \) if \( s_{i,n+i} \) is included in the routing and \( a = 0 \) otherwise.
- \( t \) Current time within regarded period.
- \( c(s_{i,n+i}) \) Marginal costs of the operator for serving request \( s_{i,n+i} \).
\[ d(z_{i,n+i}^a, t) \] Total distance for routing with fleet status \( z_{i,n+i}^a \) at time of calculation \( t \).

Each transport request \( s_{i,n+i} \) is evaluated by looking at marginal/incremental costs for adding the requested transport to the routing problem at time \( t \). Marginal cost is calculated as illustrated in the following (Figure 3.3):

![Figure 3.3: The calculation of marginal cost](image)

In particular, the costs \( c(s_{i,n+i}) \) are calculated by discriminating the costs from before the insertion \( d(z_{i,n+i}^0, t) \) from the new costs \( d(z_{i,n+i}^1, t) \) after inserting the transport request. The variable \( z_{i,n+i}^a \) indicates whether \( s_{i,n+i} \) is included or not with the superscript \( a = 1 \) (included) and \( a = 0 \) (not included).

\[
c(s_{i,n+i}) = d(z_{i,n+i}^1, t) - d(z_{i,n+i}^0, t)
\]

(3.16)

The relation (3.16) represents the input for the price determination and request selection problem (2) as shown in Section 3.2.3. However, the final routing for time period \( t \) depends on the outcome of sub-problem (2) with superscript \( a=1/0 \) and will lead to \( z_{i,n+i}^a \).
3.2.3 The price determination and request selection problem

For the price determination problem, it is important to note that three components are embedded (capacity control, dynamic pricing and forecasting), but they work in conjunction. The literature review showed that research in vehicle routing, as well as revenue management, are also concerned with requests that potentially exceed the available transport capacity (overbooking/outsourcing). While this topic is relevant, it is omitted here because it is beyond the scope of this thesis. This will remain a direction for future research.

3.2.3.1 Dynamic Pricing

The dynamic pricing algorithm aims to maximise revenues, while generating a competitive price in terms of the current market situation. However, the problem here is not knowing the current competitive market situation. As described in Chapter 2, for most countries, the RFT market can be described as heterogeneous, with use of various communication platforms for shippers and carriers. They include the traditional communication via telephone or facsimile, as well as online marketplaces where the market accessibility and competition is expected to be much higher. The latter ensures greater knowledge about the competitive market situation. Online marketplaces are generally designed as reverse auctions with visible bid structures. In comparison, traditional communication does not usually offer to the carrier any information about the shipper’s communication with competitors, unless this is intended. To address these differences, two scenarios are regarded in the thesis:

1. A second-price auction scenario where competitor prices are visible;
2. A hidden-price scenario, where the carrier has to make a bid without information on the competition.

Both scenarios result in an anticipated market price \( p_B(s_{i,n+i}) \) for the dynamic pricing model. Available price information as assumed in case (1) is applied in experiments as described in Chapter 5 (Experiments), while a hidden price scenario is used in Chapter 6 (Case study). In the following, the notation for the dynamic pricing model is presented:

\[
ACC(s_{i,n+i}) \quad \text{Decision variable for the customer’s acceptance of transport bid price with}
\]

\[
ACC(s_{i,n+i}) = 1, \text{ if the bid is accepted and}
\]

\[
ACC(s_{i,n+i}) = 0, \text{ otherwise.}
\]
\[ p(s_{i,n+i}) \] Price/bid for transport request \( s_{i,n+i} \).

\[ p_c(s_{i,n+i}) \] Customer’s expected maximum price/upper bound for \( s_{i,n+i} \) / shadow price for \( s_{i,n+i} \).

\[ DD(s_{i,n+i}) \] Direct distance for request \( s_{i,n+i} \).

\[ C_t \] Capacity level at time \( t \).

\[ DTC \] Demand to come (forecast).

\[ T \] Time of service execution.

\[ t \] Current time.

\[ p_B(s_{i,n+i}) \] Competitor’s lowest price for transport \( s_{i,n+i} \).

\[ LB \] Lower bound to \( p(s_{i,n+i}) \); determined by capacity control.

The objective of the dynamic pricing model is to maximise the overall profits (3.17). These are determined by the sum of all profits of auctions that have been won as determined by decision variable \( ACC(s_{i,n+i}) \). A number of constraints have been noted: (3.18) – (3.20).

\[
\text{Maximise} \quad \sum_{i=1}^{n} \left( p(s_{i,n+i}) - c(s_{i,n+i}) \right) * ACC(s_{i,n+i}) \tag{3.17}
\]

\[
\text{Subject to:} \quad p(s_{i,n+i}) \leq f(p_c(s_{i,n+i}), C_t, DTC, T - t) \tag{3.18}
\]

\[
p(s_{i,n+i}) < p_B(s_{i,n+i}) \tag{3.19}
\]

\[
p(s_{i,n+i}) \geq LB \tag{3.20}
\]

Inequations (3.18) and (3.19) ensure the competitiveness of the bid considering the market situation: a) (3.18) considers the case where a market price is unknown and it states that the offered bid has to be at most equal to the estimated market price; b) (3.19) considers the case of high market transparency, where the competitors prices are known. The constraint states that the bid \( p(s_{i,n+i}) \) has to be lower than the best competitor’s bid. Inequation (3.20) ensures that the bid-price stays above a lower bound \( LB \) which can be cost or a strategic value set above or below the costs \( c(s_{i,n+i}) \).
3.2.3.2 Capacity Control

A number of possible capacity control policies are discussed in the literature. These include: booking limits, protection limits, bid-price control, and first-come-first-serve policy (e.g. Schönberger and Kopfer, 2011a; Talluri and Van Ryzin, 2004). All of them have different advantages and limitations. Traditional RM policies rely on quality forecasting, knowledge about capacity, cost structure, and resulting profits, as well as standardisation of units sold. However, in RFT each freight carrier has a different routing, therefore different costs, revenues and profits are associated with the requests. Also, there is no fixed number of capacity units, but rather capacity depending on the type of transport requested. Yet estimating a capacity-space-time-function is possible. This would require re-estimation after every new assignment and may require a large amount of computational effort, depending on the map size, number of requests, and accuracy of estimation.

An additional point to consider here is that there is no single ‘best’ control policy, but rather a ‘best’ option depending on the local/regional market of operation. The analysis of the truckit.net data for the case study shows that the markets can be very different in terms of requests (their characteristics and arrival rate), number of competitors, etc.

Few studies acknowledge the complexity of the capacity problem (e.g., Schönberger and Kopfer, 2011a, 2011b, 2012) in RFT in comparison the traditional RM fields (Klein, 2006; Talluri et al., 2008)). For example, (Schönberger and Kopfer, 2011a, 2011b, 2012) investigated single connections (lanes) where requests do not generate additional costs in the route execution. Network capacity control as applied in airline RM, regards the combination of lanes (combination of connecting flights), but relies on an existing service network as well. Because the problem regarded here implies request-vehicle-reassignments and cost changes during the complete planning and driving period, other policies must be applied. The concept of capacity control/inventory control is regarded as the determination of a ‘lower bound’ (LB) variable in the pricing and request selection problem (2). As a result, the applied capacity control policies can be formulated as follows. Whereas conditions (3.21) and (3.22) are mandatory, (3.23)-(3.27) are optional and depend on the specific policy selected:

\[
\text{Vehicle routing} = \text{feasible,} \quad (3.21)
\]

meaning that the DPDPTW solved for the cost determination problem must have a feasible solution.
Further, (3.22) determines that the bid price for request $s_{i,n+i}$ has to be below the current market price.

\[ p(s_{i,n+i}) < p_B(s_{i,n+i}) \]  \hspace{1cm} (3.22)

The subsequent relations (3.23)-(3.27) are policy related and set a lower bound $LB$ for the dynamic pricing problem. Policy (3.23) determines the $LB$ based on marginal cost for request $s_{i,n+i}$ and adds a percentage rate of minimum profits. Policy (3.24) and (3.25) set $LB$ equal the marginal cost (3.24) and the direct distance of the request (3.25). The last two policies (3.26) and (3.27) apply a discount on the marginal cost $c(s_{i,n+i})$. While (3.26) sets a fix value, (3.27) uses a discount that results from forecasting\(^6\).

It is important to note that the policies presented here are only a few of the possible control policies, out of a large number of combinations, applicable for the DPDPTW problem. Due to space limitation, this thesis exemplifies a sample of them.

### 3.2.3.3 Forecasting

Information about future requests can play an important role in dynamic decision-making problems. It helps companies to estimate rate/frequency and time of arrival of requests as well as market prices, and can improve decisions about capacity allocation for different booking classes and thus prices over the booking period.

As described in Chapter 2, RFT has many differences from the traditional RM fields and this is especially the case for capacity and spatial capacity availability. The outcome and applicability of forecasting in this context strongly depends on the case and quality of the data available. Whereas extensive databases are widely available in many traditional RM areas, this is often not the case for road freight transport. This is also reflected in the scarce scientific

\(^6\) The applied forecasting methodology is presented in the forthcoming section 3.2.3.3.
literature on this topic. An exception is the work by Schönberger and Kopfer (2011b) using forecasting of freight requests, which is still distinct from this work, as it regards liner transport rather than dynamically routed vehicles.

In this thesis, forecasting is used to predict whether future requests could justify a ‘subsidisation’ of current requests. The idea behind this is to count on ‘future’ profits to subsidise and increase competitiveness at the present time. The method forecasts future requests and returns a discount percentage depending on customer location and current time. By doing so, the carrier may offer attractive prices for first requests of a cluster that is profitable as a bundle. An example is presented in Figure 3.4.

![Figure 3.4: Request cost and pricing example](image)

The example in Figure 3.4 shows an existing vehicle route (0, A, B, 0 at the top), then the same route with a future request $\overrightarrow{CD}$ (in the middle) and the future routing including requests $\overrightarrow{AB}$ and $\overrightarrow{CD}$ (0, A, B, D, C, 0, at the bottom). Additional information on the right represents the cost based on distance. While only serving $\overrightarrow{AB}$ costs 12.05 units, the marginal costs for inserting $\overrightarrow{CD}$ is at only 1.01 units. This example shows that a large part of the cost is the return trip between the depot (0) and the cluster {A, B, C, D}. Given this knowledge, the cost of requests $\overrightarrow{AB}$ and $\overrightarrow{CD}$ could be split equally to 6.53 cost units, much lower than the cost of satisfying only the $\overrightarrow{AB}$ request. This would be more satisfying for both shippers of the demands $\overrightarrow{AB}$ and $\overrightarrow{CD}$, than if they were treated separately.
For this ‘subsidisation’, historical demand is assumed to be available. The demand is cumulated and presented in a grid laid over the service area. Each field of the grid is issued with a number of requests expected to be generated in the service area and destined to other grids in the area, which represents the expected (demand) matrix of incoming requests (Figure 3.5).

\[
\begin{array}{ccccccccccc}
0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 2 & 0 & 3 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 2 \\
2 & 0 & 1 & 3 & 1 & 0 & 0 & 0 & 1 \\
0 & 3 & 0 & 1 & 0 & 1 & 2 & 1 & 0 \\
0 & 1 & 1 & 0 & 2 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

**Figure 3.5**: Expected incoming requests in sections of serviced area

Additional variables and parameters are denoted as follows:

- **X**: Set of horizontal subarea identifiers \(X = \{1, \ldots, \text{gridnumber}\}\).
- **Y**: Set of vertical subarea identifiers \(Y = \{1, \ldots, \text{gridnumber}\}\).
- **S**: Set of all subareas in serving area with \(S = X \times Y\).
- **subarea\(_{xy}\)**: Regarded subarea\(_{xy}\) with \(x, y \in S\).
- **demExp\(_{xy}\)**: Expected demand for subarea\(_{xy}\) with \(x, y \in S\).
- **\(\delta\)**: Discount parameter.

The number of expected requests in each grid subarea is determined as:

\[
demExp_{xy} = \sum s_{ij} \forall \text{subarea}_{xy} \in S \quad (3.28)
\]

Based on the demand matrix, incoming requests in certain grid areas can be discounted. Parameters such as grid size and discount settings have to be determined. Several experiments have been conducted to validate the functionality of this module and they are described in Chapter 5. On the other hand, no historical data is available for the experimental test instances,
hence the expected demand matrices have been generated by randomising the applied data sets. This has been done by simulating values following a normal distribution with the mean equal to the ‘real’ demand and different standard deviation values.

Prices were calculated based on the notation in section 3.2.3.3. Additionally, discounted prices were applied if the market price was above the operator’s price. The discounted prices were calculated based on the expected demand and the discount parameter $\delta$ as follows:

$$\text{discount} = \text{demExp}_{xy} \times \delta$$

(3.29)

Where parameter $\delta$ determines how much discount is admitted for each expected request. The value choice of $\delta$ is context dependent. Experimental results for values between 0.05-0.5 are provided in Chapter 5 of this thesis. After each evaluation of the requests in a grid cell, the expected demand for grid section $xy$ has to be adjusted to $\text{demExp}_{xy} = \text{demExp}_{xy} - 1$ in order to take into account the reduction in expected further demand as defined in this section.
4 Heuristics for solving optimisation problems

Vehicle routing is an essential element of the examined problem as the operational costs including driving time and distance represent a large share of the overall cost incurred by logistics operators. Hence, route optimisation, as an integral part of cost minimisation, is a critical process. This chapter describes the Adaptive Tabu Search heuristic that has been designed and implemented to optimise the vehicle routing for the problem examined.

4.1 Introduction

As described in Section 3.2, the routing problem observed in this thesis is a (Dynamic) Pick-up and Delivery Problem with Time Windows (DPDPTW).

Problems can be solved in different ways. Eksioglu et al. (2009) separated the methods dealing with mathematical optimisation into the following four categories:

1. Exact methods
2. Heuristics
3. Simulation
4. Real time solution methods

For small problem instances, this can be solved quickly, with optimal results. However, for large size cases, DPDPTW is considered as a non-deterministic polynomial (NP) hard problem. The class NP is a subclass of computational problems, which are considered not solvable optimally in polynomial time (Gendreau, 2003; Johnson, 1990). In these cases, heuristic algorithms are used to solve problems with a near optimal values in reasonable time. Heuristics aim to manipulate potential solutions in smart ways and progressively improve them, in order to aim towards the optimum. Many of them are based on local search (LS) improvement techniques, that is, iteratively applying local modifications (‘moves’) to advance the solution towards the optimum.

The advantage of heuristics, compared to the exact methods, is their rapid solving time, which can often be reduced to a fraction of that of an exact solver, while reaching the optimal solution or a result very close to it. However, while exact algorithms (e.g. simplex algorithm) guarantee finding the best solution, this is not the case for heuristics (Barr et al., 1995; Gendreau, 2003).

A special class of heuristics, so called metaheuristics, embed higher search procedures to guide the search mechanisms. Metaheuristics are often inspired by certain procedure or natural phenomena. Examples for metaheuristics are Variable Neighbourhood Search (VNS) (Mladenović and Hansen, 1997), Simulated Annealing (SA) (Henderson et al., 2003), Genetic
Algorithms (GA) (Holland, 1975; Taplin et al., 2005), Ant colony optimisation (Dorigo & Stützle, 2003), or Tabu Search (TS) (Gendreau, 2003; Glover, 1986; Glover and Laguna, 2013). Some metaheuristics have been proven useful for certain problem types (e.g., ant colony optimisation), while others appear to be highly versatile (e.g., Genetic Algorithms, Simulated Annealing). For further information on metaheuristics, the reader is referred to Glover and Kochenberger (2006).

While many publications in the field use the same vocabulary or technical jargon, to better understand the context and application described in this thesis, some concepts need a clearer definition:

**Customer location:** A (customer) location that is part of a pick-up and delivery request. Hence, both locations of a request can be referred to as customer locations in a vehicle route. Belonging pick-up and delivery locations shall be referred to as ‘customer pairs’ or ‘PD pair’ (pick-up & delivery pair).

**Objective function (OF):** It takes into account the overall costs for the vehicle routing while meeting all given constraints. The resulting objective value can be measured in different units, depending on the requirements (e.g. distance, time, monetary units, combinations). Here the objective value is measured in cost units.

**Move:** A manipulation of the solution. For vehicle routing, variation of the customer visitation sequence is a move. Often, different moves are used, i.e. first-fit (selection of the first customer that fits in a certain position) or best-fit (selection of the customer that best fits into a certain position while considering all existing customers).

**Heuristic (search) operator:** A search operator accommodates a certain type of move. Operators select customers to move and find the position where they are moved to. Accordingly, there are as many operators as there are move types. There are local and global search procedures. While local operators aim for an intensification of the search in a certain region, global operators aim for a diversification and to escape the region (Rochat and Taillard, 1995).

**Neighbourhood:** A neighbourhood $N(s)$ describes the search space that can be reached from the current solution $s$ by using a heuristic operator. $N(s)$ is a subset of the solution space which depends on the search operator that is used. Consequently, the size of the
neighbourhood could be small or large. Different operators can reach varying neighbourhoods of $s$.

**Local and global optima:** The best solution of a combinatorial problem is called global optimum. As heuristics iteratively move through the search space, they continuously find new current best solutions. If a solution is the best solution in a neighbourhood $N(s)$, it is called local optimum. Simple heuristic search procedures tend to ‘get stuck’ in local optima locations, when they change the solution, but return to this local improvement. This behaviour is called cycling and can be avoided by including diversification methods in the applied neighbourhoods. Examples for diversification are randomisation or procedures used in metaheuristics, such as mutation in GAs and the use of a tabu list in Tabu Search.

**Deterministic search/randomisation:** A search procedure is called deterministic if it is value orientated. This could be a search for the first fitting position or the overall best fitting position. Deterministic search procedures are often myopic and thus reduced to moves that include best positions or worst customers. To prevent the algorithm from cycling, a certain degree of randomisation can help to avoid this.

**Intensification/diversification:** The intensification of search procedures describes focusing on certain regions of the solution space. An example would be the move of single customers in a given solution. Intensification is necessary to improve the solution without destroying parts of it that may already be optimal. However, in some cases, the move of a single customer in a deterministic search can only result in a limited improvement of the solution. For that reason, a diversification allows for reaching other areas in the solution space while tolerating the destruction of parts of the current solution.

**Termination criteria:** As heuristic algorithms search iteratively, they end when reaching a stopping criterion. Stopping criteria can be: A fixed number of iterations, a fixed amount of CPU time, a fixed number of iterations without improvement in the objective value or solution specific, e.g. reaching a certain threshold for the objective value.

The algorithm is applied to optimise the vehicle routing and solve the problems as stated in Sections 3.2.1 and 3.2.2. and is described in the following.
4.2 Adaptive Tabu Search description

Tabu Search is a metaheuristic that was formally introduced by Fred Glover in 1986, drawing on his previous work (Glover, 1977). The heuristic executes a number of changes (moves) of the solution with the aim of global optimisation, and the algorithm allows for searching and worsening of the current solution, while storing the current best solution. “The basic principle of TS is to pursue [local search] LS whenever it encounters a local optimum by allowing non-improving moves; cycling back to previously visited solutions is prevented by the use of memories, called tabu lists that record the recent history of the search, a key idea that can be linked to Artificial Intelligence concepts.” (Gendreau, 2003, p.41). The short-term memory characteristic, that prevents the reversal of recent moves, and the longer-term frequency memory to reinforce attractive components, were present in the early work of Glover (1977).

The special feature is the tabu list, which stores and prohibits recent moves to allow escaping local optima. The tabu list prevents underperforming of the heuristic and guides the search to better neighbourhoods. In the following, the tabu list and aspiration criteria are explained in more detail.

**Tabu list:** The tabu list stores recently executed moves and prohibits their re-execution for a certain time (number of moves). As moves often follow a ‘greedy criterion’, banning the re-execution of a move avoids being trapped in a local optimum and expands the area of search/exploration. During every iteration of the algorithm, the move that has been on the list the longest gets erased and the most recent move is added to the list. The number of moves stored on a tabu list is called ‘tenure’. A longer tabu list avoids cycling and aims for diversification, as stored moves cannot be repeated. A shorter tabu list allows the algorithm to search a certain region of the solution more thoroughly (intensification). The tabu list length can also be adjusted dynamically according to the search results. This is called ‘reactive tabu list’.

**Aspiration criteria:** The aspiration criterion is part of a Tabu Search heuristic. “[…] tabus are sometimes too powerful: they may prohibit attractive moves, even when there is no danger of cycling, or they may lead to an overall stagnation of the searching process. It is thus necessary to use algorithmic devices that will allow one to revoke (cancel) tabus.” (Gendreau, 2003, p.44). The aspiration criterion checks if a solution is a new best solution.
and if this is the case, allows an exception for the move on the tabu list, which can therefore be executed.

Tabu Search has been selected as a solving method as it has been widely applied in combinatorial optimisation problems and demonstrated its capability to deliver high quality results under decent efficiency (Gendreau and Potvin, 2014). Typical applications are scheduling, routing, multi-criteria optimisation in resource industries, portfolio selection. For vehicle routing problems, a large amount of research with use of TS has been published, examples are: Cordeau and Laporte (2005); Jin et al. (2012); Vidal et al. (2013). For a review, the reader is directed to Gendreau (2003). Yet, as Gendreau (2003, p.50) indicated, “in spite of this abundant literature, there still seem to be many researchers who, while they are eager to apply [tabu search] TS to new problem settings, find it difficult to properly grasp the fundamental concepts of the method, its strengths and its limitations, and to come up with effective implementations.”

As an extension of the standard Tabu Search, an adaption criteria for the dynamic adaption of heuristic usage has been implemented. The adaption criteria allows the algorithm to increase the search in certain neighbourhoods depending on the search performance. ALNS was applied successfully for a variety of optimisation problems including pickup and delivery problems as regarded in this thesis (Ghilas et al., 2016; Mattos Ribeiro and Laporte, 2012; Ropke and Pisinger, 2006). A thorough explanation is provided in section 4.3.6.

Figure 4.1 presents a flowchart of the Tabu Search algorithm that has been implemented for this thesis.
Adaptive Tabu Search Algorithm:

1: begin
2: Create starting solution
3: initialise Tabu Search
4: Set neighbourhood selection probabilities to default values
5: while (iteration < iteration_{\text{max}}) do
6: select neighbourhood/operator
7: select customer(s) to move and remove from current positions
8: reininsert customer(s) according to operator procedure
9: evaluate solution
10: if solution improved
11: decrease tabu list tenure
12: adapt neighbourhood probabilities in favour of current neighbourhood
13: if solution = new best solution: Accept solution and store as best solution
14: else if move is tabu: restore previous solution
15: else:
16: else if solution worsened:
17: increase tabu list tenure
18: adapt neighbourhood probabilities against the current neighbourhood
19: if move is tabu: restore previous solution
20: else: continue with solution
21: else
22: if move is tabu: restore previous solution
23: else: continue with solution
24: update tabu list
25: if iteration is divisible by $\xi^7$: neighbourhood selection probabilities $\leftarrow$ default
26: iteration $\leftarrow$ iteration +1
27: return best solution
28: end

Figure 4.1: Tabu Search Algorithm

The algorithm is applied in two distinct ways, depending on if a static or dynamic problem is regarded. In static cases, problems are regarded altogether and the algorithm performs one optimisation process. For dynamic cases, the algorithm performs multiple re-optimisation processes over the regarded time horizon. The number of re-optimisations depends on the number of incoming requests and certain parts of the solution are ‘frozen’ and hence protected from any changes. Frozen parts are firstly, all parts of the routing that have already been executed, and secondly customer-vehicle assignments, if a customer is already bound to a specific vehicle (e.g. a delivery location is allocated to a vehicle if the pickup location has already been visited).

$^7$ Selection probability reset parameter; default value = 100.
For static cases and the first initial solution of dynamic cases (Figure 4.2) TS applies a different approach than for dynamic cases, if a previous solution is already available (Figure 4.3). The latter case, where a previous solution is available can also be described as a continuation from the available solution.

**Starting Solution (1):**

1: Initialise variables
2: for each vehicle \( k \): Create an empty route
3: Separate serving area into \( k \) polar-sectors, originating from the depot
4: Allocate Pick-up and Delivery pairs \( c_1-c_2 \) to the closest sector
5: Allocate each sector and its customers to one vehicle route, without inserting them
6: for each route \( k \):
8: while number of customers not inserted > 0:
9: Find current route centre
10: From the allocated customers, find the customer pair \( c_1-c_2 \) with furthest distance from centre of the route
11: Insert \( c_1 \) and \( c_2 \) individually, by finding positions with minimum incremental distance to predecessor and successor
12: end
13: end
14: return solution

**Figure 4.2: Starting Solution for Static Problems or Initial Solutions for Dynamic Problems**

As shown in the two algorithms, the starting solutions aim for a partial optimisation by route, to enable TS to focus search on areas with some potential for improvement.

**Starting Solution (2):**

1: if the vehicles \( k \) have not started yet \((t<0)\):
2: recall previous solution
3: insert incoming requests by finding minimum incremental distance positions in the routes
4: else:
5: recall previous solution
6: determine current vehicle locations and remove all visited customers from all routes \( k \)
7: insert incoming requests at time \( t \) by finding minimum incremental distance positions in the routes
8: return solution

**Figure 4.3: Starting Solution for Dynamic Problems when a Previous Solution Exists**
4.3 Heuristic move operators

The heuristic operators are a central aspect in the generated solution quality of the Tabu Search. There are three types of operators, which are presented next: 1) a ‘simple swap’ move changes the position of customers within the route sequence; 2) a ‘swap pair’ move, which moves a PD-pair to a different route; 3) a ‘multi swap procedure’, which removes a set of multiple customer pairs from the solution and reinserts them. The visualisation of these moves can be found in Figure 4.4, Figure 4.5 and Figure 4.6.

**Figure 4.4: Simple Swap Move**

Figure 4.4 shows that in Route 1, delivery location D and 2 were swapped, and the order of visiting customers changed from 1-P-2-3-D-4 to 1-P-D-2-3-4.

**Figure 4.5: Swap Pair Move**

Figure 4.5 displays a change from Route 1 to Route 2, with the pick-up and delivery locations P1 and D1 being visited after locations P2 and D2. Thus, the order of visiting customers changed for vehicle 1 from 1-P1-2-3-D1-4 to 1-2-3-4 and for vehicle 2 from 5-P2-6-7-D2-8 to 5-P2-P1-6-7-D2-D1-8.
In the multi swap operator as shown in Figure 4.6, a number of locations are removed first. Then, routes are reoptimized and the previously removed locations are reinserted. Every one of these move types embeds removal and insertion of one or more customers. This means that a single customer, a customer PD pair or a set of customers is first removed, and then reinserted into the solution after a certain procedure.

Overall, there are 21 heuristic operators embedded in the Tabu Search that change the solution in various ways, according to their different purposes. Each operator implements a method to source customers to be ‘removed’ from the solution and a method to ‘insert’ them in their prospective new position. In many cases, the proposed operators use identical methods, but in various combinations. The reason for this is that in combinatorial problems encountered in real-life, there are few deterministic ways to improve the solution (i.e. reduce time-window violations, minimise distance), but only a variation of solving operators can solve the ‘puzzle’.

Although all moves have the capability to improve the solution, observations have shown that some moves perform better than others and lead to a higher number of improvements. Yet,
moves with few numbers of solution improvements are sometimes indispensable to leave a certain neighbourhood in the search space and further improve the solution. This necessity of combining operators makes it hard to evaluate the individual performance of each operator. The observations show that a mix of deterministic and random search, with operators including problem restrictions, produces the best results (Gendreau, 2003).

Although not exhaustive, the set of operators is considered sufficient for the DPDPTW problem analysed here. The set of operators was refined in an iterative process programming – testing, with some of them emerging as a solution for a specific local problem.

The heuristic operators that have been tested for this project can be separated into two groups: 1) ‘Generic’ operators that combine a removal and an insertion heuristic; and 2) ‘Special’ operators where removal and insertion are related and case specific.

4.3.1 Removal heuristics
Removal heuristics describe a method to select and remove one or a set of customers from a current solution. For this thesis, the following seven removal heuristics have been implemented and tested:

4.3.1.1 Remove single customer based on large distance
This operator removes a single customer with a large distance from the predecessor and successor customers from its current position. For the selection of the customer, a candidate list, storing high distance customers, is used and a customer is randomly selected. The random choice from a range of high distance customers prevents the algorithm from cycling and allows for diversification.

4.3.1.2 Remove single customer based on time window violation
In each iteration, the algorithm evaluates the current solution and stores customers whose service time window is violated, as well as their predecessor, in a list. The time window removal of this operator randomly selects one customer from this list and removes it from its current route. The reasoning for the random selection is analogous to the previous operator.

4.3.1.3 Move customer with ‘acute angle’
This operator uses a geometric measure to improve the solution. If the angle formed by lines \( c_{i-1} - c_i \) and \( c_i - c_{i+1} \) (when visiting the customers \( c_{i-1}, c_i, \) and \( c_{i+1} \)) is very acute (\( \leq 15 \) degrees), customer \( c_i \) is removed from its position and inserted at another position, by minimising the
incremental distance change. Although the operator applies geometric measures based on straight distance (the angle between the two lines), the insertion considers the network distance.

**Algorithm: Steep angle**

1: Iterate through all routes
2: for every three subsequent customers:
3: if the angle that occurs when connecting customers $i$-$l$ with $i$ and $i$ with $i+l \leq 15$ degrees: Add customer $i$ to a new list $l$
4: end for
5: Randomly choose customer $c_i$ from list $l$
6: Remove customer $c_i$ from its route $k$
7: Determine possible positions for $c_i$ in $k$ considering pick-up or delivery locations, as appropriate
8: Find new position for $c_i$ in $k$, by minimising the incremental distance for inserting $c_i$ in route sequence
9: return solution

Note: The value of 15 degrees was chosen through trial-and-error, to capture steep-angle customers while still limiting the number of customers on list $l$.

**4.3.1.4 Shaw removal heuristic**

The Shaw removal heuristic has been introduced by Shaw (1997, 1998). It aims to remove a number $q$ of requests with high similarity in their characteristics at the same time. The heuristic first selects one request $i$ (i.e. randomly, as applied for the experiments here) and then compares all remaining requests with $i$. For the comparison of requests, a relatedness measurement $R(i,j)$, that measures the similarities of requests $i$ and $j$, is introduced:

$$R(i,j) = \phi(d_{A(i),A(j)} + d_{B(i),B(j)}) + \chi(|T_{A(i)} - T_{A(j)}| + |T_{B(i)} - T_{B(j)}|)$$

$$+ \psi(|l_i - l_j|)$$

(4.1)

In this thesis, $R(i,j)$ combines three features: The differences in distance, time windows and load size of the request pairs. The first term of the equation describes the difference in distance with $d_{A(i),A(j)}$ being the distance between the pick-up locations of $i$ and $j$ and $d_{B(i),B(j)}$ the distance between the delivery locations, respectively. The second part of the equation describes temporal connectedness with the arrival time for the pick-up location $T_A$ and delivery location $T_B$. The last term measures the load size difference with the requested load $l_i$ for request $i$ and $l_j$ for request $j$. Each term is weighted with a parameter $(\phi, \chi, \psi)$ to factorise the terms with different weight according to their importance in the problem. Given all $R(i,j)$ values, they can
then be ranked to find the most similar requests. The following pseudo-code describes the algorithm:

---

**Algorithm: Shaw removal**

1. **request**: \( r = \) randomly select a request from existing requests \( s \)
2. **create** a list of requests to be removed \( D = \{ r \} \)
3. **Array**: \( L = \) an array containing all requests from \( s \) not in \( D \)
4. **for** all elements \( i \) in \( L \) **do**:
   5. calculate \( R(r, i) \)
   6. choose a random number \( y \) from the interval \([0,1)\)
   7. randomise \( R(r, i) \) value by setting \( R(r, i) = R(r, i) \ast (1 - y^p) \)
5. **end for**
7. sort \( L \) such that \( i < j \Rightarrow R(r, L[i]) < R(r, L[j]) \)
8. **remove** the first \( q \) requests in \( D \) from \( s \)

---

1) a single reference request is randomly selected from the set of existing requests; 2) the relatedness \( R(i,j) \) is calculated for all remaining requests; 3) the relatedness measures are multiplied with randomness factor \( 1 - y^p \), with \( y \) being a random number \( \in [0,1) \) and \( p \) a parameter \( \geq 1 \), that determines the spread of the random factor; 4) requests are then sorted by descending \( R(i,j) \). Finally, the reference request and the first \( q-1 \) requests are removed from the current solution.

The Shaw removal heuristic allows for the removal of a number of requests with high similarity. These requests can come from the same or different routes, which enables the algorithm to re-optimise certain parts of the solution that are very dense or ‘crowded’.

### 4.3.1.5 Random removal heuristic

This removal heuristic randomly selects \( q \) request pairs and removes them from the solution \( s \). It can also be regarded as the previous removal heuristic with \( p=1 \) (absolute randomness). Other than the removal heuristics that have been described before, this operator does not incorporate any costs, time, or relatedness measures and may therefore remove requests that would not have been considered otherwise.

### 4.3.1.6 Worst removal heuristic

The aim of this heuristic is to select a set of \( q \) requests with the highest incremental costs and to remove them from the solution. If request \( i \) is served by a vehicle in a solution \( s \), the costs for executing request \( i \) can be calculated as \( \text{costs}(i, s) = f(s) - f_i(s) \) where \( f_i(s) \) represent the costs of the solution excluding request \( i \). In other words, the heuristic calculates the current
costs and the costs if request $i$ would be removed from the routing, then returns the
discriminative value as savings. The pseudo code is presented in the following:

**Algorithm:** Worst removal

1: Array: $L$ = an array containing all requests from solution $s$
2: for all elements $i$ in $L$ do:
3: calculate costs$(i,s)$
4: choose a random number $y$ from the interval $[0,1)$
5: randomise costs$(i,s)$ value by setting costs$(i,s) = \text{costs}(i,s) \ast (1 - y^p)$
6: end for
7: sort $L$ by descending costs$(i,s)$-value
8: remove the first $q$ requests in D from $s$

The algorithm iterates through all requests and calculates the $\text{costs}(i,s)$. Again, a randomising
factor $(1 - y^p)$ is used as described in the Shaw removal heuristic.

**4.3.1.7 Route elimination heuristic**

This heuristic aims to reduce the overall number of routes in the solution. Single routes are
completely eliminated.

**Algorithm:** Route elimination heuristic

1: for all routes in $s$ do:
2: Count number of constraint violations in routes
3: end for
4: select route with a selection probability equal to the distribution of constraint variations
5: remove all requests from selected route from the solution

For the selection of the route to be eliminated, the number of constraint violations in each route
are measured. The probability for a route to be selected is set equal to the share of constraint
violations. Once a route to be eliminated is selected, all customers are removed from it.

**4.3.2 Insertion Heuristics**

Insertion heuristics have the aim to repair or construct a solution from a number of requests
that have to be inserted in existent or non-existent routes. Insertion heuristics can be separated
into two classes: a) sequential insertion heuristics, where routes are built subsequently; and b)
parallel insertion heuristics, where multiple routes are constructed simultaneously. The
heuristics presented here are parallel insertion heuristics; for further information, the author
refers to Potvin and Rousseau (1993).
4.3.2.1 *Insert single request based on short distance*

This heuristic aims to find the position that minimises the incremental distance by inserting a customer in a route. The customer has to be inserted in the route it originated from to maintain the pick-up and delivery pair coupling. Moreover, the operator needs to take account of the correct order of pick-up and delivery location.

**Algorithm:** Insert based on distance

1: Determine possible positions for $c$ in $k$ respecting pick-up & delivery succession  
2: for all possible positions:  
3: Find position for $c$ in $k$ by minimising the incremental distance for inserting $c$ in route sequence  
4: end for  
5: insert $c$ in position  
6: return solution

4.3.2.2 *Insert single request based on reduced time window (TW) discrepancy*

This insertion heuristic inserts by minimising the discrepancy of the TW end. As before, the operator has to take account of the pick-up delivery sequence and route affiliation.

**Algorithm:** Low TW-discrepancy

1: Determine possible positions for $c$ in $k$ respecting pick-up & delivery succession  
2: for all subsequent customers $i$-$j$ in route $k$:  
3: find pair with minimum TW end discrepancy between $i$-$c$ and $c$-$j$ and store position  
4: end for  
5: insert $c$ in position  
6: return solution

4.3.2.3 *Insert single request based on distance and TW*

This operator combines the previous two insertion heuristics and finds a position by considering the lowest incremental distance while regarding TW restrictions.
Algorithm: Distance and TW-discrepancy

1: Determine possible positions for \( c \) in \( k \) respecting pick-up & delivery succession

2: \textbf{for} all possible positions:

3: \quad Find position for \( c \) in \( k \) with minimum incremental distance for inserting \( c \) in route sequence, while ensuring \( TW \) start of \( c < TW \) end of successor

4: \textbf{end for}

5: \textbf{insert} \( c \) in position

6: \textbf{return} solution

4.3.2.4 Basic ‘greedy’ insertion

The basic ‘greedy’ heuristic (i.e., Ropke and Pisinger, 2006) aims to construct a solution by inserting customers at their cheapest position. For every customer pair that is currently not part of the solution, the algorithm iterates through all possible positions and calculates the change in the objective function \( \Delta f_{i,k} \) resulting from the insertion of the request pair. The overall lowest costs for inserting the request pair at a position is denoted as \( c_i \) with:

\[
  c_i = \min_{k \in K} \{ \Delta f_{i,k} \} \tag{4.2}
\]

Finally, the request that minimises the overall insertion costs is chosen:

\[
  \min_{i \in U} c_i \tag{4.3}
\]

The set \( U \) represents all un-inserted requests. The process repeats until all requests have been inserted. Despite the advantage of being a simple insertion procedure, problems occur as requests with low \( c_i \) values are often easy to insert, while expensive and thus hard to incorporate requests remain until the routes are filled up. This problem is considered in the next set of insertion heuristics.

4.3.2.5 Regret heuristic

Regret heuristics aim to improve results from simple greedy heuristics by incorporating a look-ahead feature (Balas and Saltzman, 1991; Gutin et al., 2007). Following this, the regret value can be described as the difference in costs for not inserting the customer pair \( i \) in the best, but in the second-best fitting route. Consequently, when having a set of customer pairs \( U \) that have to be inserted, the pair with the biggest regret value \( c_i^* \) is inserted first at their minimum cost position:
\[
\max_{i \in U} c_i^*
\] (4.4)

Let us consider \(x_{ik} \in \{1, \ldots, m\}\) as the variable indicating which route has the \(k^{th}\) lowest insertion cost for request pair \(i\), route \(k\) and \(m\) being equal to the total number of routes. Additional cost for inserting a customer pair in a route \(k\) are denoted by \(\Delta f_{i,x_{ik}}\). These values are sorted for each customer pair:

\[
\Delta f_{i,x_{ik}} \leq \Delta f_{i,x_{ik}'} \text{ for } k \leq k'
\] (4.5)

Finally, the regret value \(c_i^*\) can be denoted as (the difference between the two cheapest insertion positions):

\[
c_i^* = \Delta f_{i,x_{i2}} - \Delta f_{i,x_{i1}}
\] (4.6)

The standard regret value for insertion can be extended by incorporating not only the first two best route positions, but up to \(k\) positions with \(k\) equal to the number of existing routes. This insertion heuristic becomes very successful if single routes or a set of routes become ‘crowded’. The heuristic then decides which requests have high importance (in terms of the number of fitting routes) and which ones can also be served in routes that are not the best fitting ones on a myopic view. This can be denoted as follows:

\[
\max_{i \in U} \left\{ \sum_{j=1}^{k} (\Delta f_{i,x_{ij}} - \Delta f_{i,x_{i1}}) \right\}
\] (4.7)

The regret value for insertion at the beginning of this section considers the two best positions \((k=2)\) and is hence called a ‘regret-2 insertion’. Since regret values are calculated after every insertion, the values always depend on the current situation and multiple pairs can be inserted in the same route.

4.3.3 Case-based heuristics that combine removal and insertion

As described earlier, this set of heuristics combines removal and insertion of customers. It covers both local heuristics that move single customers with a route, and global heuristics that move a customer pair to another route.
4.3.3.1 Postpone single customer by one position

This operator iterates through all positions of all routes and checks whether postponing a customer by one position would improve the overall solution. This simple operator has proved its efficiency in substantially improving the solution right at the beginning, after the initial solution $s_0$ has been created.

**Algorithm:** Postpone customer by one position

1: Randomly choose route $k$
2: Iterate through route positions
3: **for** all positions, postpone customer $c$ in the visiting sequence by one position
4: **if:** the resulting sequence leads to a shorter total distance (objective function) than the previous solution: store the move and break iteration
5: **else:** continue with the next position
6: **if:** the route end is reached: return to step 1 until all routes are checked
7: **end for**
8: Execute move from step 4
9: **return** solution

4.3.3.2 Swap customer with flexible TW-end to a late position in the route

This operator aims to reduce the concentration of customer visits in the first half of each route to allow for customers with rigid constraints (or time sensitive customers) to be scheduled first. The operator *moves* customers with flexible (late) service times to be visited later in the route giving ‘priority’ to rigid constraints in the visiting sequence. The selected customer ideally finds a position geographically close, but in the return section of a route.
Algorithm: Delay flexible TW customer

1: Randomly select a route $k$
2: for all positions $i$ in $I$ within the first two thirds ($2/3$) of the route:
3:     for all positions $j$ in $J$ within the last third ($1/3$) of the route:
4:         Find the customer in $I$ which has a flexible time window and the lowest incremental distance to any position $j$ in $J$.
5:     end for
6: end for
7: Remove customer $c$ from its route $k$
8: Insert customer $c$ at position $j$ in route $k$
9: return solution

4.3.3.3 Cross exchange

This operator searches for possible crossings in a vehicle route and tries to eliminate them. The operator iterates through all direct connections between two sequential customers $c_{i-1}$ and $c_i$, starting from the beginning of a route and from its end. If an intersection of pairs is found, the customers involved are removed and reinserted, by looking at the simplest link, with the fewest customers. Similarly, when multiple intersections are found, the one with the fewest customers involved is considered for exchange. Customers from the crossing link are removed from the route and reinserted by performing a local optimisation.

Algorithm: Cross exchange: Remove path crossways

1: Iterate through all routes by randomly selecting a route $k$
2: for all arcs created between two successive customers in a route $k$:
3:     if arcs intersect: store involved positions
4:     end for
5: Select the intersection with the lowest number of customers involved
6: Remove all involved customers from their route $k$ and remember their immediate connections before removal (e.g., $c_1$ and $c_2$)
7: for all removed customers:
8:     find the customer furthest away from $c_1$ and $c_2$ and insert it in-between
9: end for
10: Insert all remaining customers in-between $c_1$, $c_3$ and $c_2$ by minimising incremental distance changes
11: Check for possible pick-up and delivery sequence and repair if necessary
12: return solution
4.3.3.4 Swap pair to closest route

This operator swaps customers between routes. It first determines the geographic centre location of each route (centre of gravity) by considering all customer locations of each route. Then, the customer pair furthest apart from its route centre is removed from their route and inserted into their closest route. The pick-up and delivery pair is inserted by minimising the distance to their predecessor and successor.

Algorithm: Swap pair to closest route

1: Determine location of the centre of gravity of every route
2: for all routes:
3:   for all customers:
4:     find customer c with largest distance to its route centre and shorter distance to another route k’s centre
5:   end for
6: end for
7: Remove c and its partner c2 from its route
8: Find positions for c and c2 in k by minimising distance
9: Insert c and c2 in route k
10: return solution

4.3.4 Summary of heuristic operators

Table 4.1 shows the combinations of removal and insertion heuristics in the heuristic operators. The table groups the heuristics by single customer swaps and pair swaps. The columns Removal and Insertion indicate the methods applied as described previously.
<table>
<thead>
<tr>
<th>Number</th>
<th>Removal criteria</th>
<th>Insertion criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Large distance removal (4.3.1.1)</td>
<td>Insert based on distance (4.3.2.1)</td>
</tr>
<tr>
<td>2</td>
<td>Distance and TW-discrepancy (4.3.2.3)</td>
<td>Distance and TW-discrepancy (4.3.2.3)</td>
</tr>
<tr>
<td>3</td>
<td>Time window (TW) violation (4.3.1.2)</td>
<td>Insert based on distance (4.3.2.1)</td>
</tr>
<tr>
<td>4</td>
<td>Low TW-discrepancy (4.3.2.2)</td>
<td>Insert based on distance (4.3.2.1)</td>
</tr>
<tr>
<td>5</td>
<td>Distance and TW-discrepancy (4.3.2.3)</td>
<td>Insert based on distance (4.3.2.1)</td>
</tr>
<tr>
<td>6</td>
<td>Steep angle (4.3.1.3)</td>
<td>Insert based on distance (4.3.2.1)</td>
</tr>
<tr>
<td>7</td>
<td>Shaw removal (4.3.1.4)</td>
<td>greedy insertion (4.3.2.4)</td>
</tr>
<tr>
<td>8</td>
<td>Steep angle (4.3.1.3)</td>
<td>regret-2-insertion (4.3.2.5)</td>
</tr>
<tr>
<td>9</td>
<td>Steep angle (4.3.1.3)</td>
<td>regret-k-insertion (4.3.2.5)</td>
</tr>
<tr>
<td>10</td>
<td>Random removal (4.3.1.5)</td>
<td>Insert based on distance (4.3.2.1)</td>
</tr>
<tr>
<td>11</td>
<td>Random removal (4.3.1.5)</td>
<td>greedy insertion (4.3.2.4)</td>
</tr>
<tr>
<td>12</td>
<td>Worst removal (4.3.1.6)</td>
<td>regret-2-insertion (4.3.2.5)</td>
</tr>
<tr>
<td>13</td>
<td>Worst removal (4.3.1.6)</td>
<td>regret-k-insertion (4.3.2.5)</td>
</tr>
<tr>
<td>14</td>
<td>Worst removal (4.3.1.6)</td>
<td>greedy insertion (4.3.2.4)</td>
</tr>
<tr>
<td>15</td>
<td>Worst removal (4.3.1.6)</td>
<td>regret-2-insertion (4.3.2.5)</td>
</tr>
<tr>
<td>16</td>
<td>Worst removal (4.3.1.6)</td>
<td>regret-k-insertion (4.3.2.5)</td>
</tr>
<tr>
<td>17</td>
<td>Route elimination heuristic (4.3.1.7)</td>
<td>regrekt-insertion (4.3.2.5)</td>
</tr>
<tr>
<td>18</td>
<td>Postpone customer by one position (4.3.3.1)</td>
<td>greedy insertion (4.3.2.4)</td>
</tr>
<tr>
<td>19</td>
<td>Delay flexible TW customer (4.3.3.2)</td>
<td>regret-2-insertion (4.3.2.5)</td>
</tr>
<tr>
<td>20</td>
<td>Cross exchange: Remove path crossways (4.3.3.3)</td>
<td>regret-k-insertion (4.3.2.5)</td>
</tr>
<tr>
<td>21</td>
<td>Swap pair to closest route (4.3.3.4)</td>
<td>greedy insertion (4.3.2.4)</td>
</tr>
</tbody>
</table>

### 4.3.5 Usage of move operators - Probability distribution

In the Tabu Search algorithm, the selection of move operators follows certain probability distributions. Observations during the setup of the heuristics showed that different combinations of probabilities for the operators worked in favour of certain test cases. A detailed comparison of operator performances is not within the scope of this thesis. Instead, the proposed set of operators is applied using discrete probability distributions obtained from extensive testing. Although probability distributions have been previously used by (e.g., Jin et al., 2012), they seem to be context-specific, rather than a ‘universal recipe’. For future research, the sensitivity of applying operators to the probability distribution and their performance in attaining good results over a wide range of problems may be needed.

The proposed heuristic uses two combinations of operator probabilities: Pre-processing and standard.

### Table 4.2: Pre-processing distribution

<table>
<thead>
<tr>
<th>Operator</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probability in %</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
The pre-processing distribution is used after the algorithm has started and the initial solution \( s_0 \) has been selected. The aim is to improve the customer route allocation while simultaneously repairing and adjusting the sequence of routes. Table 4.3 shows the standard operator distribution, which is applied for the majority of algorithm iterations.

Table 4.3: Standard distribution

<table>
<thead>
<tr>
<th>Operator</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probability in %</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11.5</td>
<td>11.5</td>
<td>34.5</td>
<td>34.5</td>
<td>7.9</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.3 presents the standard distribution of heuristic operator usage. Finally, not all heuristics as presented in this chapter are applied in the last version of the Tabu Search. Testing has shown that certain operators perform best and are therefore selected to be used with the probabilities shown.

Besides the two probability distributions shown above, many combinations for post-processing, at both the end of the algorithm (by stopping criteria) and after finding a new best solution, have been tested, without improvement. This led to the conclusion that post-processing is not required for this heuristic setup, under the use of the implemented operators.

### 4.3.6 Adaptation of weight adjustment

To take account of different problem cases and the performance of the variety of heuristics, a machine learning element has been implemented for the automatic adjustment of heuristic operator selection. This adaptation element measures the performance of the heuristics over a number of iterations and adjusts the selection probabilities accordingly. Similar adaptation elements have been implemented before, especially in ALNS\(^8\) heuristics (e.g. Cortés et al., 2009; Ghilas et al., 2016; Mattos Ribeiro and Laporte, 2012). The criterion that is applied in the proposed Tabu Search is based on the notation of Ropke and Pisinger (2006). The operator performance is measured by scores \( \sigma_1, \sigma_2, \sigma_3 \) which can be achieved as described in Table 4.4:

Table 4.4: Description of scoring criteria

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_1 )</td>
<td>The application of this operator leads to a new global best solution</td>
</tr>
<tr>
<td>( \sigma_2 )</td>
<td>The application of this operator leads to a solution that has never been considered (i.e., locations visited) before and improves the current solution value</td>
</tr>
<tr>
<td>( \sigma_3 )</td>
<td>The application of this operator leads to a solution that has never been considered (i.e., locations visited) before and worsens the current solution value</td>
</tr>
</tbody>
</table>

---

\(^8\) Adaptive Large Neighborhood Search
Based on this description, the Tabu Search runs for a period of iterations \( j \) and the operators \( i \in \{1, \ldots, n\} \) accumulate scores based on their performance. Then, the operator probabilities \( w \) are adjusted for the next segment \( j+1 \) as follows:

\[
w_{i,j+1} = w_{i,j}(1 - r) + r \frac{\pi_i}{\theta_i}
\]

(4.8)

The reaction factor \( r \) determines how quickly the probability adjustment of the heuristics is undertaken. \( \pi_i \) is the performance of operator \( i \) in the previous set of iterations \( j \) and is calculated as a sum of \( \sigma \)-values. \( \theta_i \) is the total number of iterations in segment \( j \).

The advantage of using probability adjustment is that it accounts for the variance of performance of different operators for diverse problems. Moreover, certain operators may be superior in different phases of the overall heuristic run. The scheme of operator scores could also be extended by operator runtime or negative scores for solutions that were not accepted.

### 4.4 Heuristic evaluation

As mentioned in the introduction, a vital element for the credibility of the solution and the use of optimisation results in practice is the quality of the proposed heuristic. This section describes the experiments undertaken to validate the Tabu Search and its computational results. Special emphasis is laid on the type of experiment, the selection of test instances and heuristics’ strengths and weaknesses.

#### 4.4.1 Metaheuristic set up and settings

The Tabu Search has been implemented in Java programming language. It uses two external open source software libraries: The OpenTS Tabu Search framework and the JXL Microsoft excel interface. The objective of the program is to minimise time, distance or generalised cost. Here, for simplicity, cost and time are equal to distance, as the vehicle travel speed is equal to 1 in all benchmark instances. Following the description of the tabu list in Section 4.2, the number of moves stored in the list (or its size) is changed dynamically.

As indicated in Section 4.2, if the algorithm does not find new best solutions, the size of the tabu list increases. This helps the algorithm to escape local optima as the neighbourhood \( N(s) \) changes and previous ‘greedy’ moves are prohibited. This is also known as diversification of the solution and helps to reach other areas of the solution space. On the other side, if the
algorithm finds new best solutions, the size of the tabu list decreases and the algorithm has more options for local improvement.

### 4.4.2 Benchmark test cases

For the validation of the heuristic, the test instances from (Li and Lim, 2003), which are based on the (Solomon, 1987) instances, are used\(^9\). The Solomon test cases are widely used in vehicle routing and the Li and Lim instances are an extension for the PDPTW which is examined in this thesis. In contrast to the Solomon instances, the Li and Lim instances are extended by an allocation of Pick-up and Delivery pairs. There are three types of problems\(^10\) considered in the cases: a) clustered; b) random; and c) random-clustered, which refer to the geographical distribution of the customers.

It has to be noted, that only a limited comparison is possible as the problem objective with the corresponding published results is slightly different to the problem regarded in this thesis. While the objective of the benchmark problem lies in a minimisation of vehicles used and subsequently costs, the problem regarded in this thesis only focuses on costs minimisation (which may reduce the number of vehicles used as a side effect). Further, only the static cases can be compared as there are no dynamic cases available in the literature.

### 4.4.3 Experimental results

Table 4.5 shows the results of the computational experiments of the benchmark instances of Li and Lim (2003). It is important to acknowledge that the experiments have been conducted with soft TW settings and that late arrivals were penalised with a penalty equal to the delay time.

For the 56 instances calculated, a better solution could be found in 11 cases (negative values, highlighted in green), although in 15 cases the objective function is larger (in red). Overall, the heuristics provided an improvement of 0.345, with a number of instances showing a significant improvement (e.g. lc103, lc109). The runs were undertaken on an Intel® Core™ i5-3320 CPU with 2.60GHz and 8.00 GB RAM and a 64-bit operating system.

Table 4.5: Results from computational experiments on the PDPTW benchmark instances

<table>
<thead>
<tr>
<th>Instance</th>
<th>Number of Vehicles</th>
<th>CPU time (s)</th>
<th>Best published result (BPR)</th>
<th>Objective function value</th>
<th>Deviation of objective value from BPR in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>lc101</td>
<td>10</td>
<td>434.647</td>
<td>828.94</td>
<td>828.9</td>
<td>0.00</td>
</tr>
<tr>
<td>lc102</td>
<td>10</td>
<td>653.564</td>
<td>828.94</td>
<td>828.9</td>
<td>0.00</td>
</tr>
<tr>
<td>lc103</td>
<td>9</td>
<td>578.828</td>
<td>1,035.35</td>
<td>827.9</td>
<td>-20.04</td>
</tr>
</tbody>
</table>

\(^9\) The instances are publicly accessible (SINTEF, 2008)

\(^10\) Abbreviations in file names are: ‘lc’ – clustered; ‘lr’ – random; ‘lrc’ – random-clustered
<table>
<thead>
<tr>
<th>Instance</th>
<th>Number of Vehicles</th>
<th>CPU time (s)</th>
<th>Best published result (BPR)</th>
<th>Objective function value</th>
<th>Deviation of objective value from BPR in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>lc104</td>
<td>9</td>
<td>864.304</td>
<td>860.01</td>
<td>819.4</td>
<td>-4.72</td>
</tr>
<tr>
<td>lc105</td>
<td>10</td>
<td>428.519</td>
<td>828.94</td>
<td>828.9</td>
<td>0.00</td>
</tr>
<tr>
<td>lc106</td>
<td>10</td>
<td>291.997</td>
<td>828.94</td>
<td>828.9</td>
<td>0.00</td>
</tr>
<tr>
<td>lc107</td>
<td>10</td>
<td>506.818</td>
<td>828.94</td>
<td>828.9</td>
<td>0.00</td>
</tr>
<tr>
<td>lc108</td>
<td>10</td>
<td>515.137</td>
<td>826.44</td>
<td>826.4</td>
<td>0.00</td>
</tr>
<tr>
<td>lc109</td>
<td>9</td>
<td>1,561.661</td>
<td>1,000.6</td>
<td>827.8</td>
<td>-17.27</td>
</tr>
<tr>
<td>lc201</td>
<td>3</td>
<td>739.922</td>
<td>591.56</td>
<td>591.6</td>
<td>0.00</td>
</tr>
<tr>
<td>lc202</td>
<td>3</td>
<td>718.567</td>
<td>591.56</td>
<td>591.6</td>
<td>0.00</td>
</tr>
<tr>
<td>lc203</td>
<td>3</td>
<td>721.504</td>
<td>591.17</td>
<td>591.2</td>
<td>0.00</td>
</tr>
<tr>
<td>lc204</td>
<td>3</td>
<td>1,071.268</td>
<td>590.60</td>
<td>600.4</td>
<td>1.66</td>
</tr>
<tr>
<td>lc205</td>
<td>3</td>
<td>692.243</td>
<td>588.88</td>
<td>588.9</td>
<td>0.00</td>
</tr>
<tr>
<td>lc206</td>
<td>3</td>
<td>698.903</td>
<td>589.49</td>
<td>590.1</td>
<td>0.28</td>
</tr>
<tr>
<td>lc207</td>
<td>3</td>
<td>880.441</td>
<td>588.29</td>
<td>589.2</td>
<td>0.15</td>
</tr>
<tr>
<td>lc208</td>
<td>3</td>
<td>2061.727</td>
<td>588.32</td>
<td>591.6</td>
<td>0.56</td>
</tr>
<tr>
<td>lr101</td>
<td>19</td>
<td>303.374</td>
<td>1650.8</td>
<td>1,683.5</td>
<td>1.98</td>
</tr>
<tr>
<td>lr102</td>
<td>17</td>
<td>1,574.234</td>
<td>1,487.57</td>
<td>1,509.8</td>
<td>1.49</td>
</tr>
<tr>
<td>lr103</td>
<td>13</td>
<td>1,792.452</td>
<td>1,292.68</td>
<td>1,292.7</td>
<td>0.00</td>
</tr>
<tr>
<td>lr104</td>
<td>9</td>
<td>2,045.684</td>
<td>1,013.39</td>
<td>1,057.4</td>
<td>4.34</td>
</tr>
<tr>
<td>lr105</td>
<td>14</td>
<td>1,486.261</td>
<td>1,377.11</td>
<td>1,376.0</td>
<td>-0.08</td>
</tr>
<tr>
<td>lr106</td>
<td>12</td>
<td>2,366.411</td>
<td>1,252.62</td>
<td>1,253.1</td>
<td>0.03</td>
</tr>
<tr>
<td>lr107</td>
<td>10</td>
<td>2,132.448</td>
<td>1,111.31</td>
<td>1,111.3</td>
<td>0.00</td>
</tr>
<tr>
<td>lr108</td>
<td>9</td>
<td>2,247.162</td>
<td>968.97</td>
<td>961.6</td>
<td>-0.76</td>
</tr>
<tr>
<td>lr109</td>
<td>11</td>
<td>2,187.862</td>
<td>1,208.96</td>
<td>1,218.9</td>
<td>0.82</td>
</tr>
<tr>
<td>lr110</td>
<td>10</td>
<td>1,982.951</td>
<td>1,159.35</td>
<td>1,181.9</td>
<td>-2.24</td>
</tr>
<tr>
<td>lr111</td>
<td>10</td>
<td>3,107.175</td>
<td>1,108.9</td>
<td>1,108.9</td>
<td>0.00</td>
</tr>
<tr>
<td>lr112</td>
<td>9</td>
<td>2,379.281</td>
<td>1,003.77</td>
<td>1,033.9</td>
<td>3.00</td>
</tr>
<tr>
<td>lr201</td>
<td>4</td>
<td>1,515.632</td>
<td>1,253.23</td>
<td>1,259.7</td>
<td>0.51</td>
</tr>
<tr>
<td>lr202</td>
<td>3</td>
<td>2,152.457</td>
<td>1,197.67</td>
<td>1,224.1</td>
<td>2.21</td>
</tr>
<tr>
<td>lr203</td>
<td>3</td>
<td>1,813.419</td>
<td>949.4</td>
<td>949.4</td>
<td>0.00</td>
</tr>
<tr>
<td>lr204</td>
<td>2</td>
<td>1,773.415</td>
<td>849.05</td>
<td>849.1</td>
<td>0.00</td>
</tr>
<tr>
<td>lr205</td>
<td>3</td>
<td>2,072.381</td>
<td>1,054.02</td>
<td>1,069.8</td>
<td>1.50</td>
</tr>
<tr>
<td>lr206</td>
<td>3</td>
<td>1,968.158</td>
<td>931.63</td>
<td>932.7</td>
<td>0.11</td>
</tr>
<tr>
<td>lr207</td>
<td>2</td>
<td>1,543.695</td>
<td>903.06</td>
<td>906.0</td>
<td>0.33</td>
</tr>
<tr>
<td>lr208</td>
<td>2</td>
<td>2,012.402</td>
<td>734.85</td>
<td>739.3</td>
<td>0.61</td>
</tr>
<tr>
<td>lr209</td>
<td>3</td>
<td>1,451.73</td>
<td>930.59</td>
<td>932.4</td>
<td>0.20</td>
</tr>
<tr>
<td>lr210</td>
<td>3</td>
<td>1,957.864</td>
<td>964.22</td>
<td>964.2</td>
<td>0.00</td>
</tr>
<tr>
<td>lr211</td>
<td>2</td>
<td>1,499.034</td>
<td>911.52</td>
<td>892.5</td>
<td>-2.09</td>
</tr>
<tr>
<td>lc101</td>
<td>14</td>
<td>1,593.972</td>
<td>1,708.8</td>
<td>1,705.3</td>
<td>-0.21</td>
</tr>
<tr>
<td>lc102</td>
<td>12</td>
<td>1,155.868</td>
<td>1,558.07</td>
<td>1,573.6</td>
<td>1.00</td>
</tr>
<tr>
<td>lc103</td>
<td>11</td>
<td>3,072.837</td>
<td>1,258.74</td>
<td>1,264.0</td>
<td>0.42</td>
</tr>
<tr>
<td>lc104</td>
<td>10</td>
<td>1,876.97</td>
<td>1,128.4</td>
<td>1,132.4</td>
<td>0.36</td>
</tr>
<tr>
<td>lc105</td>
<td>13</td>
<td>2,733.515</td>
<td>1,637.62</td>
<td>1,691.6</td>
<td>3.30</td>
</tr>
<tr>
<td>lc106</td>
<td>11</td>
<td>1,008.64</td>
<td>1,424.73</td>
<td>1,435.9</td>
<td>0.78</td>
</tr>
<tr>
<td>lc107</td>
<td>11</td>
<td>2,511.118</td>
<td>1,230.14</td>
<td>1,231.5</td>
<td>0.11</td>
</tr>
<tr>
<td>lc108</td>
<td>10</td>
<td>1,884.021</td>
<td>1,147.43</td>
<td>1,148.0</td>
<td>0.05</td>
</tr>
<tr>
<td>lc201</td>
<td>4</td>
<td>1,198.205</td>
<td>1,406.94</td>
<td>1,413.2</td>
<td>0.45</td>
</tr>
<tr>
<td>lc202</td>
<td>3</td>
<td>1,953.93</td>
<td>1,374.27</td>
<td>1,360.2</td>
<td>-1.02</td>
</tr>
<tr>
<td>lc203</td>
<td>3</td>
<td>1,230.591</td>
<td>1,089.07</td>
<td>1,097.9</td>
<td>0.81</td>
</tr>
<tr>
<td>lc204</td>
<td>3</td>
<td>1,933.744</td>
<td>818.66</td>
<td>830.5</td>
<td>1.45</td>
</tr>
<tr>
<td>lc205</td>
<td>4</td>
<td>1,346.003</td>
<td>1,302.2</td>
<td>1,305.6</td>
<td>0.26</td>
</tr>
<tr>
<td>lc206</td>
<td>3</td>
<td>1,334.63</td>
<td>1,159.03</td>
<td>1,164.7</td>
<td>0.49</td>
</tr>
<tr>
<td>Instance</td>
<td>Number of Vehicles</td>
<td>CPU time (s)</td>
<td>Best published result (BPR)</td>
<td>Objective function value</td>
<td>Deviation of objective value from BPR in %</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
<td>----------------------------</td>
<td>-------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>lrc207</td>
<td>3</td>
<td>2,110.769</td>
<td>1,062.05</td>
<td>1,057.8</td>
<td>-0.40</td>
</tr>
<tr>
<td>lrc208</td>
<td>3</td>
<td>1,560.738</td>
<td>852.76</td>
<td>855.3</td>
<td>0.30</td>
</tr>
</tbody>
</table>

Note: The documentation of best published results can be found in SINTEF (2008) and includes published results of Bent and Van Hentenryck (2004); Hasle and Kloster (2007); Li and Lim (2003) among others. Route sequences for the benchmark results can be found in Appendix B.
5 Experiments

Previous chapters of this thesis introduced the problem, described the industry and scholarly work undertaken to date, and presented the proposed optimisation system and the heuristic solver. This chapter aims to verify the performance of the overall optimisation and its separate components experimentally.

5.1 Test instances

Mathematical experiments in this research field are usually conducted on benchmark problems, which allow repetitive comparisons of the experiments under similar conditions. For the verification of the heuristic solver, the ‘Solomon benchmark instances’ for the PDPTW that were used for the heuristic verification in the previous section (Li and Lim, 2001) were applied. The main problem investigated in this thesis is a dynamic case of the PDPTW and to the best of the author’s knowledge, there are currently no comprehensive benchmark instances for this problem available. Hence, the Li and Lim (2001) test instances for the PDPTW have been adjusted and extended for request arrival time for the ‘dynamic’ problem. The request arrival time refers to the moment when the customer contacts the carrier. For the selected experiments, revenues/market prices have been preassigned.

**Benchmark problem information for each customer:** Customer-ID, x-y-coordinates, load-size, start and end of the visit TW, service time, corresponding pick-up and delivery locations

**Additional information generated to extend the existing problem cases:** Request arrival time, request associated revenues (for certain experiments)

As already mentioned in 4.4.2., the test cases consist of three types of problems: 1) clustered problems - where customer locations are geographically distributed in groups; 2) random problems with randomly distributed customer locations; and 3) semi-clustered problems, which include partial grouping. For the adjustment to the dynamic attributes, the transport request pairs have been grouped and sorted by time window. The request arrival times have been set as described in the following and illustrated in Figure 5.1:
The transport requests arrive continuously during the time period considered for optimisation. This includes the pre-driving and the driving periods. One to two thirds of the requests arrive before \( t=0 \) (the beginning of the driving period) and the remaining requests after \( t=0 \). There is one request arrival at every 1 time unit. For the problem calculations, the fleet size has been limited. The maximum number of available vehicles equals the number used for the currently best-found solutions of the static experiments (SINTEF, 2008), unless stated otherwise. All other characteristics have been adopted from Li and Lim (2001), considering that the cases are widely applied to benchmark this type of problem and anticipated to be comprehensive in terms of request density and time window variance (Bent and Van Hentenryck, 2004; Hasle and Kloster, 2007; Ropke and Pisinger, 2006).

Table 5.1 summarises the created test files and their differences:
Table 5.1: Overview of experiments and test files used

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Request arrival</th>
<th>Properties on generation of related market price</th>
<th>Deterministic</th>
<th>Stochastic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heuristic benchmark (Chapter 4)</td>
<td>At the beginning of the planning period (at once)</td>
<td>No market prices included</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 Comparison of cost estimations</td>
<td>Subsequent arrivals</td>
<td>No market prices included</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Costs with increasing numbers of assigned transport jobs</td>
<td>Subsequent arrivals</td>
<td>No market prices included</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Marginal cost vs. myopic cost(^{11})</td>
<td>Subsequent arrivals</td>
<td>Simulated as normal distribution with ( \mu = DD + (RTD-DD)/2 ) and ( \sigma = (RTD-DD)/2 )</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>4 Marginal cost vs. myopic cost (simulation)</td>
<td>Subsequent arrivals</td>
<td>As in 3, but with additional strong regional competition simulated considering an additional 40% lower market price in the sector.</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5 Regional competition</td>
<td>Subsequent arrivals</td>
<td>As in 3</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6 Substitution</td>
<td>Subsequent arrivals</td>
<td>As in 3</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>7 Forecasting</td>
<td>Subsequent arrivals</td>
<td>As in 3</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

5.2 Experiment 1: Comparison of RM approach with random pricing

Section 2 of this thesis (the literature review) described the road freight market situation and carrier bid-price generation. Carriers’ operational expenses represent a significant part of the transport price. For a better understanding of the carrier cost situation, the marginal transport costs are initially compared with two reference values. These values are the ‘direct distance’ (DD) costs and the ‘round-trip distance’ (RTD) costs. The former refers to the costs for travelling between the pick-up and delivery locations, whereas the latter refers to the total cost of traversing from the carrier’s depot to the pick-up location, then to the delivery location and back to the depot. These two key figures give the scheduler the required information on the costs for executing a customer’s request; often there is no other information available (KordaMentha, 2012). Hence, these two figures represent potential boundaries for minimum

\(^{11}\) Costs that are determined by only looking at a single request but disregarding other circumstances
and maximum costs for requests. Table 5.2 shows an excerpt from experimental results of instance lc201RM. The third value and outcome of critical information is the ‘marginal cost’ which describes the incremental or additional cost for adding a new request to an existing route set. An example is provided in Figure 5.2. Here, part a) represents an existing route; part b) represents the existing route with an incoming request as indicated with dotted nodes and line; part c) shows the vehicle route after insertion of the new request. The marginal costs are represented by the blue dashed lines and replicate costs that occur additionally, compared to the costs when using route (a).

![Diagram](image)

Figure 5.2: Marginal cost for the insertion of a single request

The table includes detailed information on the insertion of requests 1-18 into the vehicle route from the scheduler’s point of view. Each row gives information about a single request. As indicated, besides information such as request number, request arrival time and runtime, there is data on DD, RTD, solution feasibility and marginal request costs. The marginal request costs represent the additional costs for including the request in the vehicle routing based on the previously existing request set. Marginal costs of infeasible solutions are shown in light grey. All requests that result in feasible solutions are included in the routing. When comparing the marginal values with the reference values, it becomes apparent that some values are below the DD (italics) and others are above the RTD (bold). As a result, the use of DD and RTD values for cost estimation and price setting can lead to over and underestimations of the expected costs related to a transport request.
Table 5.2: Excerpt from the results of instance lc201RM (Incoming requests 1 – 18)

<table>
<thead>
<tr>
<th>Request number</th>
<th>Request arrival time</th>
<th>Runtime (ms)</th>
<th>DD distance (cost units)</th>
<th>RTD (cost units)</th>
<th>Insertion is feasible</th>
<th>Marginal costs for inserting request (cost units)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-12</td>
<td>99</td>
<td>11.2</td>
<td>43.5</td>
<td>Yes</td>
<td>43.8</td>
</tr>
<tr>
<td>2</td>
<td>-11</td>
<td>170</td>
<td>3.6</td>
<td>30.8</td>
<td>Yes</td>
<td>16.4</td>
</tr>
<tr>
<td>3</td>
<td>-10</td>
<td>214</td>
<td>36.1</td>
<td>83.6</td>
<td>No</td>
<td>79.6</td>
</tr>
<tr>
<td>4</td>
<td>-9</td>
<td>206</td>
<td>5.8</td>
<td>36.5</td>
<td>No</td>
<td>40.5</td>
</tr>
<tr>
<td>5</td>
<td>-8</td>
<td>169</td>
<td>41.4</td>
<td>96.5</td>
<td>Yes</td>
<td>56.6</td>
</tr>
<tr>
<td>6</td>
<td>-7</td>
<td>220</td>
<td>27.7</td>
<td>85.8</td>
<td>No</td>
<td>134.9</td>
</tr>
<tr>
<td>7</td>
<td>-6</td>
<td>215</td>
<td>23.3</td>
<td>84.5</td>
<td>Yes</td>
<td>11.5</td>
</tr>
<tr>
<td>8</td>
<td>-5</td>
<td>292</td>
<td>28.4</td>
<td>73.3</td>
<td>No</td>
<td>111.6</td>
</tr>
<tr>
<td>9</td>
<td>-4</td>
<td>330</td>
<td>30.1</td>
<td>90.0</td>
<td>Yes</td>
<td>56.5</td>
</tr>
<tr>
<td>10</td>
<td>-3</td>
<td>497</td>
<td>22.8</td>
<td>86.7</td>
<td>No</td>
<td>166.1</td>
</tr>
<tr>
<td>11</td>
<td>-2</td>
<td>526</td>
<td>59.9</td>
<td>139.1</td>
<td>Yes</td>
<td>23.5</td>
</tr>
<tr>
<td>12</td>
<td>-1</td>
<td>570</td>
<td>25.1</td>
<td>75.9</td>
<td>Yes</td>
<td>11.4</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>766</td>
<td>20.0</td>
<td>64.7</td>
<td>No</td>
<td>268.1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>980</td>
<td>27.3</td>
<td>87.3</td>
<td>Yes</td>
<td>27.3</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>1189</td>
<td>18.0</td>
<td>74.6</td>
<td>Yes</td>
<td>87.9</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>1261</td>
<td>6.3</td>
<td>63.2</td>
<td>No</td>
<td>176.0</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>1434</td>
<td>19.0</td>
<td>84.9</td>
<td>No</td>
<td>26.1</td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>1253</td>
<td>15.3</td>
<td>62.2</td>
<td>Yes</td>
<td>27.2</td>
</tr>
</tbody>
</table>

Note: The number of vehicles used for the experiments presented equals 2.

Reasons why marginal transport costs cannot be bounded by DD and RTD costs are illustrated in Figure 5.3 and Figure 5.4.

Figure 5.3: Example for a cost overestimation

(a) 

(b) 

(c)

The illustrations are to be interpreted as follows: the rectangle represents the depot, which is the origin and final destination of each route; solid circle nodes are customer locations already included in the route; existing routes are marked with directional arrows with solid lines; incoming requests are illustrated with ‘empty’ (hollow) nodes and arrows (both dotted). Both illustrations are separated into three parts (a, b, and c). Diagram a) presents the vehicle route before request insertion (solid lines) and the incoming request (dotted nodes and lines); b) presents the cost calculations (DD in Figure 5.3 and RTD in Figure 5.4); c) displays the vehicle
routes after insertion of the incoming request under optimisation. Both examples show that the additional distance for incorporating a request in a vehicle route strongly depends on the conditions for the existing vehicle route and customer requests. Figure 5.3 gives an example of cost overestimation where the inclusion of the incoming request leads to additional costs lower than the DD-costs. Figure 5.4 gives an example of cost underestimation, where the existing vehicle route does not allow for a simple insertion of the incoming request and thus the additional costs are higher than the corresponding RTD-costs.\(^{12}\)

![Figure 5.4: Example for a cost underestimation](image)

These two cases exemplify how the use of DD and RTD for price calculations can be misleading, with negative impacts on the carrier’s operations in different ways. The literature review highlighted that comprehensive decision support systems (DSS) that provide marginal request costs are not available at the moment. As a result, freight carriers may be tempted to consider DD and RTD values and key figures as a base for their costing. Although these values may be good starting points for feasible minimum and maximum costs, Table 5.2 showed that there are cases where DD costs are higher than the actual costs and RTD costs are lower than marginal costs. Hence, operators who assume that DD costs are competitive may still overestimate their costs, whereas the use of RTD costs may underestimate costs. These ‘miscalibrations’ may result in reducing profits or failing to consider advantageous requests. Specifically, overestimations can result in reduced competitiveness as an operator may decide not to reduce a transport price further. Underestimations may cause profit losses without the carrier realising it.

To better assess the implications of these miscalculations, the problem is investigated over the complete set of test cases. Table 5.3 provides results for all test instances.

\(^{12}\) The route choice may also depend on time-window restrictions and not only shortest distance.
It can be easily seen that there are a large number of incorrect estimations over all test cases, which result in an average absolute deviation of about 34% of the real costs. The high proportion of overestimations originates from the experimental setup, which uses the number of vehicles from their best ever solutions on each test case (SINTEF, 2008). This results in a high vehicle utilisation and low marginal costs for each request. Even though over- and underestimations may compensate for each other, they represent a knowledge gap for the operator. Underestimations reduce the overall profits, while overestimations reduce the operator’s competitiveness and ability to act on the market. Consequently, more precise knowledge about costs and revenues would allow the carrier to select the most profitable requests based on her/his current situation.
5.3 Experiment 2: Costs with increasing numbers of assigned transport jobs

As the previous results show, knowledge about marginal costs can be advantageous to freight carriers. However, marginal transport costs highly depend on the requests already accepted by the carrier. For a more accurate evaluation of the marginal transport costs, the following three figures plot the marginal cost (Figure 5.5) and unitary marginal cost per km of DD (Figure 5.6) and RTD (Figure 5.7), in relation to the number of requests accepted by the carrier.

All data entries from Figure 5.5, Figure 5.6 and Figure 5.7 result from the same experiments\textsuperscript{13}. They show feasible solutions, but not necessarily all requests shown were included in the routing (for example, if auctions were not won by the carrier).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.5}
\caption{Marginal costs depending on number of requests included in routing (only feasible solutions)}
\end{figure}

The boxplot shows a weakly decreasing trend of marginal costs and dispersion of these costs with number of requests. While with zero requests included, the minimum marginal cost value for a request lies at ($\approx$) 25 and the maximum value at ($\approx$) 192, these values steadily decrease as more requests are accepted by the carrier. This indicates that, at the beginning of their planning

\textsuperscript{13} The experiments include calculations on files lc201DVRPPDCE_t - lc208DVRPPDCE_t, lr201DVRPPDCE_t - lr211DVRPPDCE_t, lrc201DVRPPDCE_t - lrc208DVRPPDCE_t.
periods, freight carriers may have to accept high costs requests, before benefitting from decreasing marginal costs as they accept more jobs.

When comparing unitary marginal costs per km for DD (Figure 5.6) and RTD (Figure 5.7) for feasible requests, the linear negative trend may be less visible, due to a number of outliers, which show a relation of high costs and short driving distances. These outliers are expected and do not necessarily have odd characteristics; they rather appear expensive as they do not ‘fit’ well with other currently served requests. Hence, the high cost/distance relationship is an indication that a request should be rejected unless, for example it generates a high profit ratio, or the carrier expects other benefits from it.

![Figure 5.6: Unitary marginal costs DD depending on number of requests included in routing (feasible solutions)](image)

In an ideal situation, an operator should be able to identify the outlier cases and disregard them. Nevertheless, as indicated in Figure 5.5, at the beginning of each planning period, there are no economies of scale and carriers may need to drive roundtrip distances (e.g. every first request in a new route implies RTD cost). After 10-15 requests are incorporated, marginal costs diminish considerably.

Next, the unitary marginal costs are considered in relation to the RTD (Figure 5.7). Values of y=1 represent marginal costs that are equal to RTD costs. Similarly to Figure 5.6, the plot shows an accumulation of those requests along the horizontal axis, but values are decreasing.
when higher numbers of requests are included. Entries with y-values above 1 indicate cases where marginal costs are above the RTD. These cases were earlier described as ‘underestimations’. Overall, the trend is decreasing, not in a linear fashion, but rather polynomial and with substantial heteroscedasticity ($R^2 = 35.6\%$). Consistently with the previous observations, the marginal costs for the very first request is always equal to RTD. This can be observed at the coordinates: $x=0$, $y=1$, where the first included requests of all test cases are stacked on top of each other, appearing as a single entry. The chart also emphasises that there are only a few marginal costs greater than RTD and that most costs are lower than half of the RTD.

![Chart showing marginal costs RTD depending on number of requests included in routing](image)

Figure 5.7: Unitary marginal costs RTD depending on number of requests included in routing (feasible solutions)

In sum, there is a decreasing cost trend when the number of customers is increasing, although there are numerous outliers, as shown in all three charts.

### 5.4 Experiment 3: Knowledge about marginal costs in a competitive environment

Based on the findings from experiments 1 and 2 (Sections 5.2 and 5.3), the advantage of having knowledge about marginal costs is investigated from a profit point of view. Here, associated revenues are set beforehand for each transport request. Then, the proposed system is run three times with different request acceptance criteria. The first two runs follow a first-come-first-
serve (FCFS) request selection, with a bid-price that has to cover the DD costs (1) and RTD costs (2). Then, in the third run, it is assumed that the operator has knowledge about marginal costs (MC) and the acceptance criterion is to break-even. By applying these settings together with the set revenue values, the tests run under the same circumstances. The market prices (lowest bids) for the requests are simulated as follows:

- For every transport request, the market price is simulated with a Gaussian probability distribution with its average between DD and RTD.
- Two standard deviations of the distribution cover the span between DD and RTD, which means that 95% of the time, the market price will take a value between those limits. Any non-positive values were ‘re-simulated’ to eliminate infeasible price values.

Table 5.4 presents the complete results for all test instances. The scenarios for this experiment are based on the previous test cases and extended by considering revenue values for each transport request.

Table 5.4: Results from experiment 3 using all test instances and three criteria

<table>
<thead>
<tr>
<th>Filename</th>
<th>Based on FCFS (DD)</th>
<th>Based on FCFS (RTD)</th>
<th>Profit based selection with marginal costs (min. 7.5 %)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total DD costs</td>
<td>Request assignments</td>
<td>Profits</td>
</tr>
<tr>
<td>lc201DCE</td>
<td>782.8</td>
<td>46</td>
<td>1,782.6</td>
</tr>
<tr>
<td>lc202DCE</td>
<td>796.0</td>
<td>38</td>
<td>1,630.2</td>
</tr>
<tr>
<td>lc203DCE</td>
<td>647.9</td>
<td>34</td>
<td>1,503.9</td>
</tr>
<tr>
<td>lc204DCE</td>
<td>718.6</td>
<td>43</td>
<td>1,578.8</td>
</tr>
<tr>
<td>lc205DCE</td>
<td>652.8</td>
<td>40</td>
<td>1,633.0</td>
</tr>
<tr>
<td>lc206DCE</td>
<td>761.4</td>
<td>39</td>
<td>1,712.1</td>
</tr>
<tr>
<td>lc207DCE</td>
<td>671.2</td>
<td>42</td>
<td>1,710.5</td>
</tr>
<tr>
<td>lc208DCE</td>
<td>698.1</td>
<td>41</td>
<td>1,745.8</td>
</tr>
<tr>
<td>lr201DCE</td>
<td>139.4</td>
<td>10</td>
<td>-6.0</td>
</tr>
<tr>
<td>lr202DCE</td>
<td>732.1</td>
<td>29</td>
<td>726.7</td>
</tr>
<tr>
<td>lr203DCE</td>
<td>351.5</td>
<td>15</td>
<td>64.0</td>
</tr>
<tr>
<td>lr204DCE</td>
<td>841.8</td>
<td>28</td>
<td>926.5</td>
</tr>
<tr>
<td>lr205DCE</td>
<td>692.3</td>
<td>29</td>
<td>736.4</td>
</tr>
<tr>
<td>lr206DCE</td>
<td>420.9</td>
<td>22</td>
<td>642.6</td>
</tr>
<tr>
<td>lr207DCE</td>
<td>751.9</td>
<td>26</td>
<td>783.2</td>
</tr>
<tr>
<td>lr208DCE</td>
<td>807.1</td>
<td>28</td>
<td>1,105.2</td>
</tr>
<tr>
<td>lr209DCE</td>
<td>503.5</td>
<td>25</td>
<td>747.8</td>
</tr>
<tr>
<td>lr210DCE</td>
<td>578.5</td>
<td>22</td>
<td>443.2</td>
</tr>
<tr>
<td>lr211DCE</td>
<td>628.5</td>
<td>30</td>
<td>1,127.3</td>
</tr>
<tr>
<td>lrc201DCE</td>
<td>510.8</td>
<td>20</td>
<td>151.5</td>
</tr>
<tr>
<td>lrc202DCE</td>
<td>488.7</td>
<td>20</td>
<td>406.0</td>
</tr>
<tr>
<td>lrc204DCE</td>
<td>417.5</td>
<td>19</td>
<td>560.2</td>
</tr>
</tbody>
</table>
Based on the findings from experiments 1 and 2, the knowledge advantage about marginal costs is investigated in a competitive environment. For this experiment, assignment of transport requests/jobs follows the principle of a second price auction. In second price auctions, the bidder with the best bid wins the assignment for the price of the second lowest bid. Market prices are simulated equal to the previously simulated prices in Experiment 3. For the carrier’s transport price determination, a transparent market is assumed. In order to maximise profits, the carrier’s bid-price is increased to the market price, if the marginal costs are lower than the current market price (Table 5.5).

### 5.5 Experiment 4: Can knowledge of the market give competitive advantage?

<table>
<thead>
<tr>
<th>Filename</th>
<th>Based on FCFS (DD)</th>
<th>Based on FCFS (RTD)</th>
<th>Profit based selection with marginal costs (min. 7.5 %)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total DD costs</td>
<td>Request assignments</td>
<td>Profits</td>
</tr>
<tr>
<td>lrc205DCE</td>
<td>666.9</td>
<td>22</td>
<td>287.0</td>
</tr>
<tr>
<td>lrc206DCE</td>
<td>879.5</td>
<td>32</td>
<td>853.7</td>
</tr>
<tr>
<td>lrc207DCE</td>
<td>102.8</td>
<td>2</td>
<td>-63.3</td>
</tr>
<tr>
<td>lrc208DCE</td>
<td>592.9</td>
<td>21</td>
<td>612.4</td>
</tr>
<tr>
<td>Total</td>
<td>723</td>
<td>23,401.5</td>
<td>588.3</td>
</tr>
</tbody>
</table>

Note: The table shows the results for three different test setups. Each setup resembles a freight carrier who has to decide whether to accept or reject transport requests based on the price. Revenues are included in the test files to guarantee the same problem for each of the three cases. For each test set, the decision is based on DD costs, RTD costs, or ‘marginal costs’. For the first two cases, the carrier has no knowledge about the marginal costs. The acceptance of transport requests is based on the cost value results in different request sets for each test case. The table shows totals of costs, accepted transport assignments and profits for each setup.

The results show that the MC approach has the highest profits with about 7% more than the DD approach while requiring 2.5% less accepted requests. The RTD approach has the least profits with a total of less than half of the other two criteria. When only considering the DD and MC setup, the lower standard deviation of the marginal cost approach (540.6) compared to the DD approach (588.3) indicates a lower profit variance and thus reduces cash flow risks for the operator.
The previous four experiments have shown that knowledge about marginal costs can be beneficial for freight carriers in terms of request selection and in a competitive environment. Marginal costs and the resulting profit margins always depend on the situation of the operator.
in terms of depot and vehicle location, as well as the set of existing and expected future requests. This means that a certain request can have a significant profit for one operator, while it means a negative profit to another. Based on this, the following experiments investigate the proposed model in terms of two aspects:

1. Regional disadvantage and the ability to find market niches (Experiment 5)
2. Cost substitution for expected future profits (Experiment 6)

With knowledge of marginal costs, a DSS could help to identify profitable requests. Next, it is shown how this system may assist operators to find niche markets in terms of geographical distribution. This is especially significant with the growth of transport auctions and online marketplaces where shipper-carrier relationships may be replaced by service rating systems.

For this experiment, the test files including fixed revenues as presented in experiment 3 are used. To simulate a strong spatial competition, the geographical service area is split into four sectors (quadrants) with the carrier’s depot in the centre. The sectors are numbered similarly to a Cartesian coordinate system, anti-clockwise, with 1 being the top-right quadrant. Sector 3 is arbitrarily chosen here as a sector with high competition. This may simulate a strong competitor with a depot or numerous requests in this area. If both locations of a transport request fall in sector 3, the market price is reduced by an additional 40% to simulate this strong competition.

Table 5.6 shows the results for all test cases and the accumulated number of accepted transport requests in each of the four sectors.

Table 5.6: Computational results for experiments with regional competition

<table>
<thead>
<tr>
<th>File name</th>
<th>Total costs</th>
<th>Revenues</th>
<th>Profits</th>
<th>Accepted number of requests in sectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>lC201DCE</td>
<td>383.6</td>
<td>1,336.4</td>
<td>952.8</td>
<td>7  7  1  11</td>
</tr>
<tr>
<td>lC202DCE</td>
<td>570.0</td>
<td>1,852.4</td>
<td>1,282.4</td>
<td>11 9 11 9</td>
</tr>
<tr>
<td>lC203DCE</td>
<td>468.2</td>
<td>1,363.4</td>
<td>895.2</td>
<td>9  8  3  8</td>
</tr>
<tr>
<td>lC204DCE</td>
<td>577.7</td>
<td>1,725.4</td>
<td>1,147.7</td>
<td>10 7 2  9</td>
</tr>
<tr>
<td>lC205DCE</td>
<td>566.8</td>
<td>2,203.8</td>
<td>1,637.0</td>
<td>14 9 9 10</td>
</tr>
<tr>
<td>lC206DCE</td>
<td>461.4</td>
<td>1,641.2</td>
<td>1,179.8</td>
<td>6  12 7  5</td>
</tr>
<tr>
<td>lC207DCE</td>
<td>568.6</td>
<td>2,100.8</td>
<td>1,532.2</td>
<td>15 11 10 8</td>
</tr>
<tr>
<td>lC208DCE</td>
<td>567.0</td>
<td>2,227.4</td>
<td>1,660.4</td>
<td>15 10 9 10</td>
</tr>
<tr>
<td>lR201DCE</td>
<td>627.3</td>
<td>1,240.0</td>
<td>612.7</td>
<td>5  7  0  7</td>
</tr>
<tr>
<td>lR202DCE</td>
<td>743.9</td>
<td>1,423.2</td>
<td>679.3</td>
<td>7  9  2  5</td>
</tr>
<tr>
<td>lR203DCE</td>
<td>833.1</td>
<td>1,038.4</td>
<td>205.3</td>
<td>6  6  5  3</td>
</tr>
<tr>
<td>lR204DCE</td>
<td>663.4</td>
<td>1,246.0</td>
<td>582.6</td>
<td>5  6  3  10</td>
</tr>
<tr>
<td>lR205DCE</td>
<td>455.6</td>
<td>998.8</td>
<td>543.2</td>
<td>2  7  4  6</td>
</tr>
<tr>
<td>lR206DCE</td>
<td>720.0</td>
<td>1,290.6</td>
<td>570.6</td>
<td>6  5  3  6</td>
</tr>
<tr>
<td>lR207DCE</td>
<td>760.6</td>
<td>1,334.4</td>
<td>573.8</td>
<td>5  9  7  7</td>
</tr>
<tr>
<td>lR208DCE</td>
<td>427.3</td>
<td>971.8</td>
<td>544.5</td>
<td>1  5  4  5</td>
</tr>
<tr>
<td>File name</td>
<td>Total costs</td>
<td>Revenues</td>
<td>Profits</td>
<td>Accepted number of requests in sectors</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>----------</td>
<td>---------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sector 1</td>
</tr>
<tr>
<td>lr209DCE</td>
<td>620.8</td>
<td>1,343.0</td>
<td>722.2</td>
<td>10</td>
</tr>
<tr>
<td>lr210DCE</td>
<td>816.6</td>
<td>1,426.4</td>
<td>609.8</td>
<td>10</td>
</tr>
<tr>
<td>lr211DCE</td>
<td>728.9</td>
<td>1,340.8</td>
<td>611.9</td>
<td>3</td>
</tr>
<tr>
<td>lr201DCE</td>
<td>674.8</td>
<td>1,060.6</td>
<td>385.8</td>
<td>6</td>
</tr>
<tr>
<td>lr202DCE</td>
<td>620.1</td>
<td>772.4</td>
<td>152.3</td>
<td>4</td>
</tr>
<tr>
<td>lr204DCE</td>
<td>610.0</td>
<td>1,438.0</td>
<td>828.0</td>
<td>8</td>
</tr>
<tr>
<td>lr205DCE</td>
<td>652.2</td>
<td>1,073.2</td>
<td>421.0</td>
<td>8</td>
</tr>
<tr>
<td>lr206DCE</td>
<td>797.2</td>
<td>1,354.8</td>
<td>557.6</td>
<td>6</td>
</tr>
<tr>
<td>lr207DCE</td>
<td>260.3</td>
<td>178.0</td>
<td>-82.3</td>
<td>2</td>
</tr>
<tr>
<td>lr208DCE</td>
<td>633.1</td>
<td>1,522.0</td>
<td>888.9</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>186</strong></td>
</tr>
</tbody>
</table>

The results show that the share of requests collected in the sector with high competition (sector 3) is significantly smaller. According to the experimental setup, market prices for requests in this sector are on average lower than others and thus less likely to be profitable. As the algorithm (for this setting) only collects profitable requests, it avoids this sector and instead focuses on collecting requests in higher profit regions. Hence, the algorithm detects areas where operations are profitable and finds a niche market which is specific to the carrier’s current transport assignments and the vehicle and depot locations. As a result of the focus on these regions, future demand can be met with more competitive prices. The effect of regional request selection as a result of competition is illustrated in Figure 5.8 and Figure 5.9.

![Figure 5.8: Set of customer locations and routing without regional competition](image1)

![Figure 5.9: Set of customer locations and routing with regional competition](image2)

Note: Both figures show vehicle routes with the rectangular shaped depot in the centre. Pick-up locations are indicated by the hollow circles, while delivery locations are indicated by solid circles. Each location is identified by a dynamic ID and a standard ID. The different routes are discriminated by colours. The routes start in the open field after the vehicles left the depot, but they all terminate at the depot. The grey rectangle in Figure 5.9 represents the ‘high competition’ sector.
Both examples (Figure 5.8 and Figure 5.9) show how the algorithm automatically selects profitable requests, avoids high competition regions and identifies niche areas. A real-world example of this scenario could be a freight carrier that operates across many different regions, but gets around a region being served by a large competitor, as the transport prices in this area are not profitable.

While these results give an indication of customer selection by profits, a freight carrier may have different intentions than just maximising profits. These may be maximising vehicle utilisation or maintaining regional availability to their customers. Means to achieve this can be the substitution of transport requests. This is investigated in Experiment 6, which is presented subsequently.

5.7  **Experiment 6: Substitution of early requests for expected future profits**

Because of limited knowledge about future demand, at the beginning of the scheduling period, marginal costs for new requests are often high and tend to equal RTD costs (the first transport request of every acquisition period is always equal to the RTD). Consequently, the proposed approach is not competitive at the beginning of each planning period. This also means that potentially good auctions are lost at this time. On the contrary, late arriving requests in regions that are well-visited profit from the ‘cheap’ integration into existing routes and their resulting low marginal costs.

Therefore, the system does not provide a high degree of competitiveness at the starting time, due to the fact that first requests are not ‘aware’, nor ‘take advantage’ of the expected future utilisation of the vehicles in the fleet. In other words, early customers have to pay a higher price, whereas, as indicated, late customers benefit from high vehicle utilisation, from dispersion of activities in various parts of the area of operation, and therefore gain from prices that much lower.

To overcome this problem, the dynamic pricing policy has been changed in a way that allows discounts to go below the marginal routing costs. The overall profits per test case for five discount scenarios are presented in Table 5.7. The standard setting without any discount where the price equals the marginal costs (Case 1) is presented along with a number of further tests with 20-35% discounts (Cases 2 to 5). Cases 2-5 allow the carrier the possibility to offer these discounts over the whole scheduling period.
Table 5.7: Average profits of simulated random competition experiments with different subsidy rates

<table>
<thead>
<tr>
<th>Filename</th>
<th>Case 1: marginal costs</th>
<th>Case 2: marginal costs - 20%</th>
<th>Case 3: marginal costs - 25%</th>
<th>Case 4: marginal costs - 30%</th>
<th>Case 5: marginal costs - 35%</th>
</tr>
</thead>
<tbody>
<tr>
<td>lc201DVRPPDCE</td>
<td>679.1</td>
<td>1,639.6</td>
<td>1,640.4</td>
<td>1,621.3</td>
<td>1,642.6</td>
</tr>
<tr>
<td>lc202DVRPPDCE</td>
<td>1487.8</td>
<td>1,569.7</td>
<td>1,566.6</td>
<td>1,594.3</td>
<td>1,532.5</td>
</tr>
<tr>
<td>lc203DVRPPDCE</td>
<td>851.3</td>
<td>1,280.1</td>
<td>1,237.2</td>
<td>1,254.3</td>
<td>1,344.1</td>
</tr>
<tr>
<td>lc204DVRPPDCE</td>
<td>1,138.1</td>
<td>980.2</td>
<td>1,170.5</td>
<td>1,076.0</td>
<td>1,479.8</td>
</tr>
<tr>
<td>lc205DVRPPDCE</td>
<td>1,419.1</td>
<td>1,588.9</td>
<td>1,584.7</td>
<td>1,559.0</td>
<td>1,632.8</td>
</tr>
<tr>
<td>lc206DVRPPDCE</td>
<td>1,275.5</td>
<td>1,383.6</td>
<td>1,406.4</td>
<td>1,649.1</td>
<td>1,627.3</td>
</tr>
<tr>
<td>lc207DVRPPDCE</td>
<td>1,332.1</td>
<td>1,544.2</td>
<td>1,585.4</td>
<td>1,503.6</td>
<td>1,691.2</td>
</tr>
<tr>
<td>lc208DVRPPDCE</td>
<td>1,131.5</td>
<td>1,516.9</td>
<td>1,501.6</td>
<td>1,571.2</td>
<td>1,586.6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>9,314.5</strong></td>
<td><strong>11,503.4</strong></td>
<td><strong>11,692.6</strong></td>
<td><strong>11,828.7</strong></td>
<td><strong>12,536.8</strong></td>
</tr>
</tbody>
</table>

For each test instance, the highest profits are marked as **bold** and lowest profits are marked in *italics*. With one exception (lc204DVRPPDCE), the lowest profits are obtained with the standard (no discount) strategy. In general, higher discounts are associated with higher profits, however, when looking at the overall results, a general trend cannot be noted.

Figure 5.10 shows that in most test cases (lc202, lc203, lc205, lc207, lc208), profits increase when discounts are applied – as expected. However, this is different for case 204 where the marginal cost policy (Case 1) results in higher profits than Case 2 and 4.

![Figure 5.10: Average profits of simulated random competition experiments with different discount rates](image-url)
5.8 Experiment 7: Request forecasting and price substitution

The previous section showed that discounting rates for early requests could help in maximising profits for the operation period. However, this assumes a rather uniform distribution of customer locations and an even occurrence of requests over time.

Next, a number of simple examples of forecasting based on historical demand are presented. To do so, historical demand is accumulated and presented in a grid that is laid over the service area. Each field of the grid is issued with a number of requests that are expected to arrive in the service area generating an expected demand matrix (Figure 5.11).

The number of expected requests in each grid subarea is determined as:

\[ \text{demExp}_{ij} = \sum s_{ij} \forall \text{grids } \in I,J \]  \hspace{1cm} (5.1)

Based on this matrix, incoming demands in certain grid areas can be discounted. Parameters such as grid size and discount settings have to be determined. Several experiments have been conducted to validate the forecasting functionality. For the experimental test instances, no historical data is available. Hence, expected demand matrices have been created by randomising the applied data sets. This has been done by creating values using a
Gaussian/normal distribution with a mean of the ‘real’ demand value and different standard deviation values.

Prices were calculated based on the notation presented in Chapter 3. Additionally, discounted prices were calculated if the market price was above the operator’s price. The discounted prices are calculated based on the expected demand and the discount parameter $\delta$ as per notation 3.29 as follows:

$$\text{discount} = \text{demExp}_{ij} \times \delta$$  \hspace{1cm} (5.2)

The experiments were conducted with three different grid sizes (3x3, 4x4, 5x5), four different standard deviations (0, 1, 2, 3) to account for diverse forecast quality, four different discount levels (5%, 15%, 35%, 50% for each request in sector) and four test cases (lc201DVRPPDCE, lc202DVRPPDCE, lc205DVRPPDCE, lc206DVRPPPDCE) being a total of 192 experiments. The forecast values are simulated with the altered standard deviations, in order to maintain comparability (The request forecast kept constant experimental combinations of grid sizes and discount levels). The experimental results are presented in Table 5.8.

### Table 5.8: Profit results from forecasting experiments

<table>
<thead>
<tr>
<th>No. of grids</th>
<th>File name</th>
<th>Reference profits</th>
<th>Std. Dev. = 0</th>
<th>Std. Dev. = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Discount values</td>
<td>Discount values</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5%</td>
<td>20%</td>
</tr>
<tr>
<td>3</td>
<td>lc201*</td>
<td>1,850.9</td>
<td>2,074.3</td>
<td>2,098.7</td>
</tr>
<tr>
<td></td>
<td>lc202*</td>
<td>1,918.2</td>
<td>1,939.7</td>
<td>2,057.9</td>
</tr>
<tr>
<td></td>
<td>lc205*</td>
<td>1,766.6</td>
<td>2,030.0</td>
<td>2,059.0</td>
</tr>
<tr>
<td></td>
<td>lc206*</td>
<td>1,637.3</td>
<td>2,126.1</td>
<td>2,135.7</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>7,173.0</td>
<td>8,170.0</td>
<td>8,351.2</td>
</tr>
<tr>
<td>4</td>
<td>lc201*</td>
<td>1,850.9</td>
<td>2,040.4</td>
<td>2,098.7</td>
</tr>
<tr>
<td></td>
<td>lc202*</td>
<td>1,918.2</td>
<td>1,961.1</td>
<td>1,992.1</td>
</tr>
<tr>
<td></td>
<td>lc205*</td>
<td>1,766.6</td>
<td>1,829.7</td>
<td>2,020.9</td>
</tr>
<tr>
<td></td>
<td>lc206*</td>
<td>1,637.3</td>
<td>2,045.6</td>
<td>2,123.5</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>7,173.0</td>
<td>7,876.8</td>
<td>8,235.2</td>
</tr>
<tr>
<td>5</td>
<td>lc201*</td>
<td>1,850.9</td>
<td>1,950.1</td>
<td>2,098.7</td>
</tr>
<tr>
<td></td>
<td>lc202*</td>
<td>1,918.2</td>
<td>1,881.1</td>
<td>2,038.3</td>
</tr>
<tr>
<td></td>
<td>lc205*</td>
<td>1,766.6</td>
<td>1,980.2</td>
<td>2,132.4</td>
</tr>
<tr>
<td></td>
<td>lc206*</td>
<td>1,637.3</td>
<td>1,911.4</td>
<td>2,127.9</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>7,173.0</td>
<td>7,722.7</td>
<td>8,397.2</td>
</tr>
<tr>
<td>No. of grids</td>
<td>File name</td>
<td>Reference profits</td>
<td>Std. Dev. = 2</td>
<td>Std. Dev. = 3</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>-------------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Discount values</td>
<td>Discount values</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5% 20% 35% 50%</td>
<td>5% 20% 35% 50%</td>
</tr>
<tr>
<td>3</td>
<td>lc201*</td>
<td>1,850.9</td>
<td>2,057.2 2,064.7 2,138.0 2,021.1</td>
<td>2,056.6 2,138.0 2,096.6 2,138.0</td>
</tr>
<tr>
<td></td>
<td>lc202*</td>
<td>1,918.2</td>
<td>2,026.8 2,020.9 2,056.1 2,060.6</td>
<td>1,911.8 2,016.6 2,045.8 2,057.9</td>
</tr>
<tr>
<td></td>
<td>lc205*</td>
<td>1,766.6</td>
<td>1,701.1 2,026.0 1,906.9 1,679.0</td>
<td>2,016.9 1,931.1 2,076.0 1,925.1</td>
</tr>
<tr>
<td></td>
<td>lc206*</td>
<td>1,637.3</td>
<td>2,022.3 2,123.8 2,085.4 2,132.3</td>
<td>2,030.8 2,079.0 2,133.7 2,133.9</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>7,173.0</td>
<td>7,807.4 8,235.5 8,186.4 7,893.0</td>
<td>8,016.0 8,164.7 8,352.1 8,254.9</td>
</tr>
<tr>
<td>4</td>
<td>lc201*</td>
<td>1,850.9</td>
<td>1,857.0 2,068.6 2,045.0 2,133.4</td>
<td>2,063.5 2,133.4 2,133.4 2,133.4</td>
</tr>
<tr>
<td></td>
<td>lc202*</td>
<td>1,918.2</td>
<td>1,974.6 2,066.2 2,009.7 2,001.8</td>
<td>1,959.8 2,060.6 2,027.2 2,060.6</td>
</tr>
<tr>
<td></td>
<td>lc205*</td>
<td>1,766.6</td>
<td>2,071.0 1,853.9 2,009.9 1,893.5</td>
<td>2,032.4 2,029.6 1,899.9 2,056.8</td>
</tr>
<tr>
<td></td>
<td>lc206*</td>
<td>1,637.3</td>
<td>2,000.9 2,126.3 2,132.9 2,140.3</td>
<td>1,865.6 2,133.3 2,139.3 2,133.4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>7,173.0</td>
<td>7,903.5 8,115.0 8,197.5 8,169.1</td>
<td>7,921.3 8,356.9 8,199.9 8,384.3</td>
</tr>
<tr>
<td>5</td>
<td>lc201*</td>
<td>1,850.9</td>
<td>1,993.5 2,094.5 2,138.0 2,138.0</td>
<td>1,982.1 2,082.3 2,082.3 2,082.3</td>
</tr>
<tr>
<td></td>
<td>lc202*</td>
<td>1,918.2</td>
<td>1,918.2 2,038.5 1,995.9 1,979.5</td>
<td>1,838.0 2,045.8 2,057.9 2,045.8</td>
</tr>
<tr>
<td></td>
<td>lc205*</td>
<td>1,766.6</td>
<td>2,023.9 2,037.9 2,130.1 2,061.0</td>
<td>1,995.6 2,025.7 2,068.8 2,121.8</td>
</tr>
<tr>
<td></td>
<td>lc206*</td>
<td>1,637.3</td>
<td>2,021.5 2,129.0 2,131.3 2,127.2</td>
<td>2,118.7 2,136.3 2,133.3 2,130.8</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>7,173.0</td>
<td>7,957.1 8,299.8 8,395.3 8,305.7</td>
<td>7,934.4 8,290.1 8,342.3 8,380.6</td>
</tr>
</tbody>
</table>

Note: The column ‘Reference profits’ describes an experiment for each test case without the use of forecasting. The column ‘discount cumulates discount applied per grid (i.e. if 3 more requests are expected in a sector and the discount value is 20%, then 3*20% means that up to 60% is discounted from an incoming request in that sector). The discount is capped at a maximum of 80%. The ‘Total’ values accumulate the resulting values for the different test sets for all experiment combinations. *The file names are abbreviated in order to save space. All end with “DVRPPDCE”.

The results in Table 5.8 demonstrate that the application of forecasting is superior to the reference in all test cases. The highest overall profits are created by using 20% as a discount value, followed by 50%. However, the values of 5% and 35% show lower profits. From the experiments conducted, there is a positive relation between discount and profits, but this depends on the case. To enable an easier comparison of the discount values, profits were further summarised in Table 5.9.
Table 5.9: Concluded results for discount values and number of grids

<table>
<thead>
<tr>
<th>No. of grids</th>
<th>Std. Dev.</th>
<th>5%</th>
<th>20%</th>
<th>35%</th>
<th>50%</th>
<th>Average profits /grid number</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>8,170.0</td>
<td>8,351.2</td>
<td>8,292.5</td>
<td>8,201.8</td>
<td><strong>8,176.8</strong></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>8,157.0</td>
<td>8,472.3</td>
<td>7,911.0</td>
<td>8,362.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7,807.4</td>
<td>8,235.5</td>
<td>8,186.4</td>
<td>7,893.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8,016.0</td>
<td>8,164.7</td>
<td>8,352.1</td>
<td>8,254.9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>7,876.8</td>
<td>8,235.2</td>
<td>8,100.3</td>
<td>8,153.6</td>
<td><strong>8,163.8</strong></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>8,030.5</td>
<td>8,295.0</td>
<td>8,353.3</td>
<td>8,328.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7,903.5</td>
<td>8,115.0</td>
<td>8,197.5</td>
<td>8,169.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7,921.3</td>
<td>8,356.9</td>
<td>8,199.9</td>
<td>8,384.3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>7,722.7</td>
<td>8,397.2</td>
<td>8,271.3</td>
<td>8,232.3</td>
<td><strong>8,173.1</strong></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>7,790.6</td>
<td>7,901.5</td>
<td>8,264.1</td>
<td>8,285.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7,957.1</td>
<td>8,299.8</td>
<td>8,395.3</td>
<td>8,305.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7,934.4</td>
<td>8,290.1</td>
<td>8,342.3</td>
<td>8,380.6</td>
<td></td>
</tr>
<tr>
<td><strong>Average profits/discount value</strong></td>
<td></td>
<td><strong>7,940.6</strong></td>
<td><strong>8,259.5</strong></td>
<td><strong>8,238.8</strong></td>
<td><strong>8,245.9</strong></td>
<td></td>
</tr>
</tbody>
</table>

Note: The ‘Average profits/discount value’ refer to the average profits per test instance resulting from the application of the corresponding discount value. The ‘Average profits/grid number refers to the average profits per test instance resulting from the application of the corresponding number of grids.

As stated in the price setup section (Section 3.2.3.1), discount is used when the market price is lower than the marginal costs. As a result, it is only used in some cases. Observations indicate that most discounts should be applied when the first requests arrive (Figure 5.12). This also explains why the discount value of 5% does not bring the necessary effect as is does not accumulate a total discount large enough to ‘win’ some requests that have RTD costs.

![Figure 5.12: Number of admitted discounts for numbers of requests included in vehicle routing](image-url)
Figure 5.12 shows the number of applied discounts against the number of requests included in routing. The relation is negative, suggesting that discounts are viewed as an incentive when only few transport assignments have been acquired, and they are less likely to be applied with increasing numbers of job assignments.

Overall, the results indicate that average profits for the number of grids = 3 are the highest (presented in the last column) and there is a weak positive relation with discounts especially when the number of grids is equal to 5. However, there are no clear patterns of association with the discount for number of grids = 3, therefore it is difficult to draw conclusive recommendations. Theoretically, a higher number of grids should lead to a higher degree of accuracy as it better resembles geographical proximity. However, this could not be shown in the experiments conducted here. A possible reason for this could be the experimental setting applied. Requests in sectors surrounding a sector of interest are not considered at all. Nevertheless, a bordering sector may accommodate future requests and thus offer the potential cost savings referred to earlier. Hence, future experiments could gradually incorporate surrounding sectors in order to take into account nearby customers and consequently save costs when requests are combined.

Figure 5.13 displays the discount that has been granted to customers in order to gain a transport assignment, based on the results from experiment 7.
Figure 5.13: Granted discount in relation to customer requests included in routing

The chart shows large numbers of entries scattered just below the x-axis. Entries of different discount values (5, 20, 35, and 50%) are also evenly distributed within these. The majority of entries seem to form ‘clouds’/clusters. However, no overall trend could be identified, nor the reason for the formation of these clusters. There are numerous outlier cases which all result from the three highest discount values.

Figure 5.14 displays the same data as in Figure 5.13 but as a percentage value of the marginal costs.
Figure 5.14: Granted discount in relation to customer requests included in routing (proportional)

Due to the limited number of applied test cases and runs, the experiments are not sufficient to give specific recommendations on the bidding process. However, this is not seen as a drawback, as the intention was to highlight the complexity of the problem, and highlight relevant factors which can be further explored in future research. Nevertheless, based on the numerous tests and experiments, the author believes that the results indicate advantages from the application of simple forecasting techniques and the positive role of discounts; specific calibration depends on the application.

This chapter presented experimental studies on the application of the proposed system using seven sets of experiments. Using cost indicators, it is shown that direct and roundtrip distance can lead to false cost estimations, thus the calculation of marginal costs is preferred and should be applied, if possible. Further, the effects of using different cost indicators over a complete planning horizon are presented. While the negative effects of incorrect cost estimations may sometimes be balanced out by including other requests, the use of marginal costs has been demonstrated to always be superior (Experiment 1). The relation between the cost/distance ratio and the number of requests accepted showed the importance for carriers to always evaluate incoming requests in combination with existing ones (Experiment 2). Two sets of experiments (Experiment 3 and 4) confirm the findings in a simulated market environment. Experiment 5 simulates strong regional competition and shows how the system avoids this and automatically detects ‘niche’ markets where operation generates profits. The variety of
experiments indicated how early requests (in a planning horizon) have comparably high costs as they do not yet benefit from the later economies of scale. Hence, Experiment 6 presents competition and different discount policies, with results indicating that giving discounts on ‘early’ requests is highly effective and increases profit in most cases. Finally, Experiment 7 includes the earlier described forecasting technique and demonstrates that knowledge about expected future demand can be very beneficial when generating bid prices under uncertainty.
6 Case study

This chapter presents an application using real-world data. For this thesis, transport request data has been collected from www.truckit.net, an online marketplace for freight transport requests in Australia. The data includes transport characteristics and price negotiation data for almost 9,000 requests over the Australian continent. The chapter demonstrates the capability of the system to improve the profit and efficiency of the operation of this freight market segment.

6.1 Data collection

Palletised transport requests were recorded using an online data mining approach. The website (www.truckit.net) is designed as a reverse auction: Carriers can place bids for the requests, then shippers make final decisions on whether one of the carriers is chosen for the transport assignment. The marketplace functions as a platform for a variety of types of transport requests, however for data collection, requests recorded between February 2015 and September 2017 have been pre-screened and only palletised requests have been selected. The decision was made based on two criteria: 1) the comparability of load size, implied weight restrictions and handling with light machinery such as forklifts for the pallets, which are regarded as the ‘foundation of small load unit design’ and are widely used across different industries (Yujie and Yaoqiu, 2008); 2) the prevalence of palletised transport requests on the website.

The data collection has been performed by a Java application that was implemented for this purpose. During the data collection, the website has been scanned for new palletised transport requests every 15 minutes. Every 8 hours, the program ran an updating sequence which checked already recorded requests with an ‘active’ auction for new information. All requests had a unique identifier (ID) which enables tracking. In parallel, shipper-carrier interaction in the related discussion forum has been recorded in separate text files for further analysis. Detailed information is not included in this thesis but is presented in a separate journal article which has been submitted for publication in October 201714.

The data set consists of listing ID, pick-up location, delivery location, driving distance, driving time, Customer ID, date listed, number of bids, loading location specification (commercial, residential), auction expiry date and time, lowest bid, unloading location specification, goods

---

14 Working title: “Understanding bid-price generation for road freight transport by analysing online market data”, submitted to Transportation Research A.
specification, number of pallets, pallet specification (standard, non-standard), pallet size, auction status, last bid, acceptance status.

The data was manually assessed and cleaned, with incomplete requests removed from the analysis and case study.

6.2 Data description

The recorded transport requests cover all Australian states, including Tasmania. The database includes requests with high variance in their characteristics, starting with urban consignments to be moved as little as 2 km and ending with requests for interstate movements up to 6,440 km (average 1,645.9 km). Average size of the requests was 4.3 pallets (maximum request 26) and 70% of them were completely flexible in terms of timing, whereas only 14.6% had fixed time windows. Unsurprisingly, there is higher activity on the Pacific coast of Australia and there are more unique destinations for transport requests in Western Australia (WA), Northern Territory (NT), Queensland (QLD), and Tasmania (TAS), as illustrated by the map in Figure 6.1 and Figure 6.2. The freight demand cumulated by state is given in Table 6.1.

Figure 6.1: Origins of the requests presented in www.truckit.net
Table 6.1 shows the accumulated origin and destination numbers for all transport requests by state.

Table 6.1: Freight flows within and between states for the collected data

<table>
<thead>
<tr>
<th>Origin state</th>
<th>ACT</th>
<th>NSW</th>
<th>NT</th>
<th>QLD</th>
<th>SA</th>
<th>TAS</th>
<th>VIC</th>
<th>WA</th>
<th>Grand Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT</td>
<td>1</td>
<td>28</td>
<td>1</td>
<td>32</td>
<td>7</td>
<td>5</td>
<td>17</td>
<td>7</td>
<td>98</td>
</tr>
<tr>
<td>NSW</td>
<td>71</td>
<td>821</td>
<td>48</td>
<td>747</td>
<td>180</td>
<td>64</td>
<td>537</td>
<td>327</td>
<td>2,795</td>
</tr>
<tr>
<td>NT</td>
<td>0</td>
<td>33</td>
<td>8</td>
<td>27</td>
<td>12</td>
<td>6</td>
<td>33</td>
<td>16</td>
<td>135</td>
</tr>
<tr>
<td>QLD</td>
<td>33</td>
<td>549</td>
<td>86</td>
<td>603</td>
<td>112</td>
<td>37</td>
<td>358</td>
<td>197</td>
<td>1,975</td>
</tr>
<tr>
<td>SA</td>
<td>6</td>
<td>129</td>
<td>30</td>
<td>124</td>
<td>46</td>
<td>13</td>
<td>106</td>
<td>92</td>
<td>546</td>
</tr>
<tr>
<td>TAS</td>
<td>1</td>
<td>23</td>
<td>5</td>
<td>24</td>
<td>9</td>
<td>5</td>
<td>24</td>
<td>12</td>
<td>103</td>
</tr>
<tr>
<td>VIC</td>
<td>67</td>
<td>706</td>
<td>77</td>
<td>619</td>
<td>194</td>
<td>101</td>
<td>346</td>
<td>278</td>
<td>2,388</td>
</tr>
<tr>
<td>WA</td>
<td>6</td>
<td>184</td>
<td>34</td>
<td>189</td>
<td>66</td>
<td>31</td>
<td>184</td>
<td>189</td>
<td>883</td>
</tr>
</tbody>
</table>

The highest numbers of requests are for intrastate transport within NSW (821) and interstate requests from VIC to NSW (706). There are no requests from NT to ACT and only one request for transport within ACT. The largest values are highlighted in bold. The distribution of freight flows is in line with the description of major freight corridors in governmental publications, which indicate that most freight movements are to or from or within major cities on the
Australian east and south coast, with only minor numbers in the centre or the north and northwestern regions (BITRE, 2014).

While the online collected data gives a good representation of freight flows across Australia, it only exemplifies a minor share of the overall market <1%. It is assumed that the majority of freight requests are traded via traditional means of communication (telephone, facsimile) or contract logistics.

### 6.3 Test case generation

For the case study experiments, test instances were generated from the data collected from [www.truckit.net](http://www.truckit.net). Each experimental test instance is based on a depot location in one of the Australian cities (Sydney, Melbourne, Brisbane) with the highest number of transport requests. After applying a number of location specific filters, transport requests were randomly chosen and added to the test instances. The filters ensured that only valid data, relevant to the depot location, was included in the test cases (e.g. a test instance based on a Melbourne depot would not contain an inner-city transport request in Brisbane).

Creating test instances from this large amount of real-world requests posed a new challenge of generating sets of requests that reflect the real market conditions. First experiments confirmed that the market is very competitive and certain sets of randomly selected requests (too specific) resulted in losses for small carriers with a limited number of vehicles. On the other hand, selection filters that are too general would ‘falsely’ generate test instances with numerous opportunities for the carriers (presenting an unrealistically optimistic situation). After revisiting the vehicle types required for transport, the experiments were run considering average outcome conditions. All case study instances were built with substantial randomness which may result in operational losses for some cases, but also guarantees that cases are not assembled in a biased manner to inflate profits.

A number of assumptions were made to deal with factors that were not included in the collected data or are necessary to reflect local operational conditions or governmental regulations. A six-day working week from Monday to Saturday was assumed and as the applied model does not take account of working and driving hour regulations, the average vehicle speed of 60 km/h was reduced by a factor of 3 (8h/24h per day = 1/3) to 20 km/h, to account for an eight-hour working day.
The pick-up and delivery time windows of the collected data only consider three cases (date specific, 3-day flexible, or completely flexible requests). For the experimental instances, the time windows have been converted simultaneously. As no data on the required time for pick-up and delivery was available, all service times have been assumed to equal 90 minutes.

Due to the large share of owner-operators in Australia (Bureau of Transport and Regional Economics, 2003), the case study instances have been constructed for one vehicle. Nevertheless, cases with larger fleet numbers can be calculated, as shown in the heuristic evaluation (Chapter 4) and the extended experiments (Chapter 5).

The heuristic starting solutions were generated by using Cartesian coordinates, Northing and Easting of the origin and destination locations, obtained from geocoding addresses on the website and converting them to kilometres\(^{15}\). To generate the case study cases from the request database, the following logical and case specific filters were applied:

- **Logical**: The auction had to be ended/closed, last bid accepted, requested number of pallets to be transported had to be no larger than the maximum vehicle capacity, request origin or destination must not be in the state of Tasmania (which includes sea transport).
- **Case specific**: Case files with different depot locations were created (Brisbane, Melbourne, Sydney). Depending on the depot location, requests along major routes or in the metropolitan region were filtered out before analysis.

Additionally, cases with less than 10% accepted requests were filtered out in a test run. This was due to the high randomness and ensured that the generated files have a minimum degree of realism/plausibility. The following table gives an overview of the routes and regions being considered.

---

\(^{15}\) For the conversion of geocodes into Northing and Easting, it was important to regard changes of Earth circumference according to the latitude. As the circumference for the most northern and most southern points are 109.3 km and 80.54 km, for the conversion, the medium value of 94.92 km has been chosen. The error occurring from using an average-value conversion has a maximum of 3.9 m per kilometre for the most northern and southern tips of Australia, deemed minor for the purpose of the research and therefore ignored.
Table 6.2: Overview of case instance setup

<table>
<thead>
<tr>
<th>Depot location</th>
<th>Distribution focus</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Brisbane</td>
</tr>
<tr>
<td>Brisbane</td>
<td>Within a metropolitan area of 300 km (M300)</td>
</tr>
<tr>
<td>Melbourne</td>
<td>Within a metropolitan area of 300 km (M300)</td>
</tr>
<tr>
<td>Sydney</td>
<td>Routes connecting major cities with locations within a 100 or 250 km radius of either city (R100, R250)</td>
</tr>
</tbody>
</table>

Table 6.2 should be read in the following way: Based on the depot location as listed in the left column, a number of cases with a ‘distribution focus’ (as described in the remaining part of the table) were created. The values in the cells (100, 250, 300) described the radius (in km) around the included cities in which the request locations were located. For example, for Sydney, cases were created that are either within 100 km radius around Sydney or Melbourne, 250 km radius around Sydney or Melbourne, or 300 km radius around Sydney only. This resembles the case of an operator who focuses on the major freight corridor between Sydney and Melbourne or exclusively on the Sydney metropolitan area. All cases included in one cell should be considered simultaneously. In total, 94 cases were created with a minimum of eight cases for each city and each operating option. A detailed overview of the cases can be found in the appendix.

6.3.1 Business operating cost

For the case study, a number of assumptions for the business operating cost have to be made. This includes vehicle and trailer finance, fuel, wages, service and maintenance, as well as a variety of other aspects. To keep the cost estimation generic, the author decided to use Freightmetrics (2017), which provides a comprehensive cost overview for truck operating companies in Australia. The effective date for the cost recordings was the 17th of October 2017. Cost fluctuations were ignored\(^\text{16}\) and costs were based on a six-day working week with 46 working weeks per year (see Table 6.3 and

\(^{16}\) Diesel prices vary over time and region. However, the crude oil price on the given date was above the average of the data collection period (U.S. Energy Information Administration, 2017).
Table 6.4). The truck uses a single slider trailer with a capacity of 26 Australian standard pallets. This is a type of truck widely used by the small operators, however smaller vehicles would be necessary if the demand is not high enough and the utilisation too low to justify this capacity. The average daily driving distance was set to 750 km.

Table 6.3: Detailed cost specification (1)

<table>
<thead>
<tr>
<th>Vehicle costs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Capital costs truck</td>
<td>$276,210.00</td>
</tr>
<tr>
<td>Vehicle stamp duty (based on a rate of 3%)</td>
<td>$8,286.00</td>
</tr>
<tr>
<td>Capital costs trailer</td>
<td>$81,406.00</td>
</tr>
<tr>
<td>Trailer stamp duty (based on a rate of 3%)</td>
<td>$2,442.00</td>
</tr>
<tr>
<td>Miscellaneous costs</td>
<td>$15,000.00</td>
</tr>
<tr>
<td>Principal (Load amount financed)</td>
<td>$383,344.00</td>
</tr>
<tr>
<td>Residual/Balloon (25%)</td>
<td>$95,836.00</td>
</tr>
<tr>
<td>Vehicle investment less residual</td>
<td>$287,508.00</td>
</tr>
<tr>
<td>Annual vehicle cost (considering a 5-year loan period)</td>
<td>$57,501.60</td>
</tr>
<tr>
<td>Annual vehicle interest (considering a 5-year loan period and a 9.5% interest rate)</td>
<td>$23,420.72</td>
</tr>
<tr>
<td><strong>Annual cost and interests (total)</strong></td>
<td><strong>$80,922.32</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General costs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Insurance</td>
<td>$13,042.00</td>
</tr>
<tr>
<td>Registration</td>
<td>$6,555.00</td>
</tr>
<tr>
<td>Accounting/Consultancy</td>
<td>$500.00</td>
</tr>
<tr>
<td>Depot/Rent for vehicle</td>
<td>$12,500.00</td>
</tr>
<tr>
<td>Depot insurance</td>
<td>$1,500.00</td>
</tr>
<tr>
<td>Road tolls</td>
<td>$5,520.00</td>
</tr>
<tr>
<td>Mobile cost</td>
<td>$1,440.00</td>
</tr>
<tr>
<td>Telephone cost</td>
<td>$3,540.00</td>
</tr>
<tr>
<td>Administration staff</td>
<td>$22,680.00</td>
</tr>
<tr>
<td>Office supplies</td>
<td>$2,880.00</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>$22,632.00</td>
</tr>
<tr>
<td>Driver wage</td>
<td>$86,736.00</td>
</tr>
<tr>
<td>Workers insurance</td>
<td>$4,076.59</td>
</tr>
<tr>
<td>Superannuation</td>
<td>$7,806.24</td>
</tr>
<tr>
<td><strong>Annual general costs (total)</strong></td>
<td><strong>$191,407.83</strong></td>
</tr>
</tbody>
</table>

Whereas Table 6.3 refers to the purchase and general costs (capital primarily) on an annual basis, Table 6.4 presents the running costs (maintenance and fuel) per km.
Table 6.4: Detailed cost specification (2)

<table>
<thead>
<tr>
<th>Service, maintenance and tyre wear</th>
<th>Cost/unit</th>
<th>Cost/km</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle service (per service interval every 18,000 km)</td>
<td>930.00</td>
<td>0.05</td>
</tr>
<tr>
<td>Vehicle maintenance (per maintenance interval every 20,000 km)</td>
<td>1670.00</td>
<td>0.08</td>
</tr>
<tr>
<td>Steer tyre cost and quantity * 2 (require replacement every 100,000 km)</td>
<td>774.00</td>
<td>0.02</td>
</tr>
<tr>
<td>Drive and trailer tyre cost and quantity * 20 (require replacement every 160,000 km)</td>
<td>700.00</td>
<td>0.09</td>
</tr>
<tr>
<td><strong>Total service and maintenance costs ($ per km)</strong></td>
<td><strong>0.24</strong></td>
<td></td>
</tr>
</tbody>
</table>

Fuel costs

<table>
<thead>
<tr>
<th></th>
<th>Cost/unit</th>
<th>Cost/km</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuel price (as of 10/2017)</td>
<td>$/l</td>
<td>1.4</td>
</tr>
<tr>
<td>Fuel rebate</td>
<td>$/l</td>
<td>0.12003</td>
</tr>
<tr>
<td>Fuel incl. delivery and rebate</td>
<td>$/l</td>
<td>1.27997</td>
</tr>
<tr>
<td>Fuel consumption</td>
<td>km/l</td>
<td>2.1</td>
</tr>
<tr>
<td><strong>Total fuel costs ($ per km)</strong></td>
<td><strong>$/km</strong></td>
<td><strong>0.6095</strong></td>
</tr>
</tbody>
</table>

Table 6.5 summarises the variable costs per kilometre and fixed costs per week. As shown by the average costs, any truck used in the optimisation will incur a minimum cost of $5,920, regardless of the number of requests completed during the week. Additionally, $0.85 occurs per kilometre driven, covering fuel, service, maintenance and tyre wear.

Table 6.5: Cost summary

<table>
<thead>
<tr>
<th>Fixed costs:</th>
<th>Annually</th>
<th>Weekly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle</td>
<td>$80,922.32</td>
<td>$1,759.18</td>
</tr>
<tr>
<td>General costs and wages</td>
<td>$191,407.83</td>
<td>$4,161.04</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$272,330.15</strong></td>
<td><strong>$5,920.22</strong></td>
</tr>
</tbody>
</table>

Variable costs

<table>
<thead>
<tr>
<th></th>
<th>Per km</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuel</td>
<td>$0.61</td>
</tr>
<tr>
<td>Service, maintenance and tyre wear</td>
<td>$0.24</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$0.85</strong></td>
</tr>
</tbody>
</table>

### 6.4 Experiments

The experiments in Chapter 5 demonstrated the proposed system’s capabilities and enabled calibration of system data. Given the real-world data availability, the following experiments
give the opportunity to investigate real-world performance and identify strengths and limitations of the proposed system.

6.4.1 System performance on real-world data

The focus of the first experiment is on the system’s performance in terms of generated profits and profit margins. In this experiment, for each case, 200 subsequently arriving requests were processed by the optimising system. The system automatically assesses incoming requests and sets a transport price according to the description in Chapter 3. The market price is known according to a second-price auction, which is equivalent to the online marketplace where the case study data was collected.

Three different cases are investigated: (1) R100 - Requests along freight routes, connecting two major cities pick-up and delivery location with a 100 km radius of one of those cities; (2) R250 - Requests along freight routes, connecting two major cities pick-up and delivery location within a 250 km radius of one of those cities; (3) M300 - Requests for a metropolitan area with a radius of 300 km. The experimental results are presented in the following three tables: Table 6.6, Table 6.7 and Table 6.8.

Table 6.6: Computational results for cases with focus around major cities (100 km radius)

<table>
<thead>
<tr>
<th>Filename</th>
<th>Route</th>
<th>Costs ($)</th>
<th>Revenues ($)</th>
<th>Profits ($)</th>
<th>Number of accepted requests</th>
<th>Profit margin (proportion)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AU_NSW_R100_101</td>
<td>Sydney - Melbourne</td>
<td>6,355.6</td>
<td>9,270.0</td>
<td>2,914.4</td>
<td>15</td>
<td>0.31</td>
</tr>
<tr>
<td>AU_NSW_R100_102</td>
<td>Sydney - Melbourne</td>
<td>8,386.4</td>
<td>18,643.0</td>
<td>10,256.6</td>
<td>28</td>
<td>0.55</td>
</tr>
<tr>
<td>AU_NSW_R100_103</td>
<td>Sydney - Melbourne</td>
<td>7,688.9</td>
<td>9,213.0</td>
<td>1,524.1</td>
<td>19</td>
<td>0.17</td>
</tr>
<tr>
<td>AU_NSW_R100_104</td>
<td>Sydney - Melbourne</td>
<td>6,239.5</td>
<td>2,717.0</td>
<td>-3,522.5</td>
<td>13</td>
<td>-1.30</td>
</tr>
<tr>
<td>AU_NSW_R100_105</td>
<td>Sydney - Melbourne</td>
<td>7,632.9</td>
<td>4,318.0</td>
<td>-3,314.9</td>
<td>12</td>
<td>-0.77</td>
</tr>
<tr>
<td>AU_NSW_R100_106</td>
<td>Sydney - Melbourne</td>
<td>6,313.8</td>
<td>16,334.0</td>
<td>10,020.2</td>
<td>22</td>
<td>0.61</td>
</tr>
<tr>
<td>AU_NSW_R100_107</td>
<td>Sydney - Melbourne</td>
<td>7,216.2</td>
<td>8,068.0</td>
<td>851.8</td>
<td>21</td>
<td>0.11</td>
</tr>
<tr>
<td>AU_NSW_R100_108</td>
<td>Sydney - Melbourne</td>
<td>7,763.7</td>
<td>8,345.0</td>
<td>581.3</td>
<td>22</td>
<td>0.07</td>
</tr>
<tr>
<td>AU_NSW_R100_109</td>
<td>Sydney - Melbourne</td>
<td>7,832.6</td>
<td>11,766.0</td>
<td>3,933.4</td>
<td>21</td>
<td>0.33</td>
</tr>
<tr>
<td>AU_NSW_R100_110</td>
<td>Sydney - Melbourne</td>
<td>7,120.6</td>
<td>7,986.0</td>
<td>865.4</td>
<td>15</td>
<td>0.11</td>
</tr>
<tr>
<td>AU_NSW_R100_111</td>
<td>Sydney - Melbourne</td>
<td>7,625.0</td>
<td>9,521.0</td>
<td>1,896.0</td>
<td>14</td>
<td>0.20</td>
</tr>
<tr>
<td>AU_NSW_R100_112</td>
<td>Sydney - Melbourne</td>
<td>7,744.8</td>
<td>6,264.0</td>
<td>-1,480.8</td>
<td>14</td>
<td>-0.24</td>
</tr>
<tr>
<td>AU_NSW_R100_113</td>
<td>Sydney - Melbourne</td>
<td>7,654.2</td>
<td>4,810.0</td>
<td>-2,844.2</td>
<td>13</td>
<td>-0.59</td>
</tr>
<tr>
<td>AU_NSW_R100_114</td>
<td>Sydney - Melbourne</td>
<td>7,105.1</td>
<td>8,005.0</td>
<td>899.9</td>
<td>21</td>
<td>0.11</td>
</tr>
<tr>
<td>AU_QLD_R100_101</td>
<td>Brisbane - Sydney</td>
<td>6,834.7</td>
<td>11,166.0</td>
<td>4,331.3</td>
<td>24</td>
<td>0.39</td>
</tr>
<tr>
<td>AU_QLD_R100_102</td>
<td>Brisbane - Sydney</td>
<td>7,932.5</td>
<td>8,330.0</td>
<td>397.5</td>
<td>19</td>
<td>0.05</td>
</tr>
<tr>
<td>AU_QLD_R100_103</td>
<td>Brisbane - Sydney</td>
<td>7,490.6</td>
<td>6,130.0</td>
<td>-1,360.6</td>
<td>14</td>
<td>-0.22</td>
</tr>
<tr>
<td>AU_QLD_R100_104</td>
<td>Brisbane - Sydney</td>
<td>8,567.8</td>
<td>9,389.0</td>
<td>821.2</td>
<td>22</td>
<td>0.09</td>
</tr>
<tr>
<td>AU_QLD_R100_105</td>
<td>Brisbane - Sydney</td>
<td>7,018.0</td>
<td>9,485.0</td>
<td>2,467.0</td>
<td>27</td>
<td>0.26</td>
</tr>
<tr>
<td>AU_QLD_R100_106</td>
<td>Brisbane - Sydney</td>
<td>7,318.2</td>
<td>8,767.0</td>
<td>1,448.8</td>
<td>18</td>
<td>0.17</td>
</tr>
<tr>
<td>AU_QLD_R100_107</td>
<td>Brisbane - Sydney</td>
<td>8,067.2</td>
<td>6,854.0</td>
<td>-1,213.2</td>
<td>24</td>
<td>-0.18</td>
</tr>
<tr>
<td>AU_QLD_R100_108</td>
<td>Brisbane - Sydney</td>
<td>7,613.3</td>
<td>15,042.0</td>
<td>7,428.7</td>
<td>20</td>
<td>0.49</td>
</tr>
<tr>
<td>Filename</td>
<td>Route</td>
<td>Costs ($)</td>
<td>Revenues ($)</td>
<td>Profits ($)</td>
<td>Number of accepted requests</td>
<td>Profit margin (proportion)</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------</td>
<td>------------</td>
<td>--------------</td>
<td>-------------</td>
<td>----------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>AU_QLD_R100_109</td>
<td>Brisbane - Sydney</td>
<td>7,162.3</td>
<td>8,227.0</td>
<td>1,064.7</td>
<td>22</td>
<td>0.13</td>
</tr>
<tr>
<td>AU_QLD_R100_110</td>
<td>Brisbane - Sydney</td>
<td>7,486.0</td>
<td>9,386.0</td>
<td>1,900.0</td>
<td>19</td>
<td>0.20</td>
</tr>
<tr>
<td>AU_QLD_R100_111</td>
<td>Brisbane - Sydney</td>
<td>7,471.6</td>
<td>9,674.0</td>
<td>2,202.4</td>
<td>17</td>
<td>0.23</td>
</tr>
<tr>
<td>AU_VIC_R100_101</td>
<td>Melbourne - Sydney</td>
<td>7,018.9</td>
<td>8,865.0</td>
<td>1,846.1</td>
<td>21</td>
<td>0.21</td>
</tr>
<tr>
<td>AU_VIC_R100_102</td>
<td>Melbourne - Sydney</td>
<td>6,343.2</td>
<td>4,092.0</td>
<td>-2,251.2</td>
<td>14</td>
<td>-0.55</td>
</tr>
<tr>
<td>AU_VIC_R100_103</td>
<td>Melbourne - Sydney</td>
<td>7,737.8</td>
<td>7,120.0</td>
<td>-617.8</td>
<td>12</td>
<td>-0.09</td>
</tr>
<tr>
<td>AU_VIC_R100_104</td>
<td>Melbourne - Sydney</td>
<td>7,662.7</td>
<td>5,969.0</td>
<td>-1,693.7</td>
<td>9</td>
<td>-0.28</td>
</tr>
<tr>
<td>AU_VIC_R100_105</td>
<td>Melbourne - Sydney</td>
<td>7,492.8</td>
<td>15,316.0</td>
<td>7,823.2</td>
<td>17</td>
<td>0.51</td>
</tr>
<tr>
<td>AU_VIC_R100_106</td>
<td>Melbourne - Sydney</td>
<td>7,141.9</td>
<td>8,973.0</td>
<td>1,831.1</td>
<td>24</td>
<td>0.20</td>
</tr>
<tr>
<td><strong>Mean</strong></td>
<td></td>
<td><strong>7,388.3</strong></td>
<td><strong>8,969.2</strong></td>
<td><strong>1,580.8</strong></td>
<td><strong>18.5</strong></td>
<td><strong>0.176</strong></td>
</tr>
</tbody>
</table>

Note: The column ‘Route’ shows included cities where the first named city also presents the depot location.

Table 6.6 presents the results for cases with customers close to urban centres (100 km radius) of two major Australian cities building a major freight corridor\textsuperscript{17}. The results show highly differing profit margins for the instances (e.g. -0.77 for AU_NSW_R100_105 and 0.61 for AU_NSW_R100_106). This is expected to be a result of the randomised case composition. However, it is also expected that profits fluctuate over planning periods and that this variability adequately represents real-world conditions. In total, a mean profit margin of 17.6\% could be achieved (0.176), as shown in the last row of Table 6.6, with profits obtained in 72\% of the cases. According to the Australian Bureau of Transport and Regional Economics (2003), the average industry profit margin is 9.3\% which is well below the experimental result. Acil Tasman (2004) presented an even lower value with about 7\% for the road freight sector and 4\% for the freight forwarding sector. Moreover, industry reports such as Magner (2017) state the highest profits for large operators, and not statistics for the whole sector. Despite using a single vehicle, the presented case still generates results above the industry averages. Assuming that the case study instances are representative for the industry in Australia, the results show that using a DSS such as the one presented here can help small operators to considerably improve their competitive situation. The system considers a set of transport requests that have a high similarity in their characteristics or fit well together. In this sense, the operator is assisted to find a niche market and to work more efficiently.

Next, the results for cases with customers within a greater radius (250 km) around city origins are considered (Table 6.7).

\textsuperscript{17} Detailed explanations can be found in Section 6.3.
The average marginal profits of cases within a 300 km radius are slightly lower within this group of case instances, as shown in Table 6.7.

### Table 6.7: Computational results for cases with focus around major cities (250 km radius)

<table>
<thead>
<tr>
<th>Filename</th>
<th>Route</th>
<th>Costs ($)</th>
<th>Revenues ($)</th>
<th>Profits ($)</th>
<th>Number of accepted requests</th>
<th>Profit margin (proportion)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AU_NSW_R250_101</td>
<td>Sydney - Melbourne</td>
<td>7,774.4</td>
<td>11,774.0</td>
<td>3,999.6</td>
<td>25</td>
<td>0.34</td>
</tr>
<tr>
<td>AU_NSW_R250_102</td>
<td>Sydney - Melbourne</td>
<td>7,796.1</td>
<td>5,571.0</td>
<td>-2,225.1</td>
<td>11</td>
<td>-0.40</td>
</tr>
<tr>
<td>AU_NSW_R250_103</td>
<td>Sydney - Melbourne</td>
<td>7,306.2</td>
<td>8,192.0</td>
<td>885.8</td>
<td>17</td>
<td>0.11</td>
</tr>
<tr>
<td>AU_NSW_R250_104</td>
<td>Sydney - Melbourne</td>
<td>7,211.8</td>
<td>6,255.0</td>
<td>-956.8</td>
<td>17</td>
<td>-0.15</td>
</tr>
<tr>
<td>AU_NSW_R250_105</td>
<td>Sydney - Melbourne</td>
<td>7,577.0</td>
<td>9,111.0</td>
<td>1,534.0</td>
<td>15</td>
<td>0.17</td>
</tr>
<tr>
<td>AU_NSW_R250_106</td>
<td>Sydney - Melbourne</td>
<td>7,638.4</td>
<td>8,427.0</td>
<td>788.6</td>
<td>13</td>
<td>0.09</td>
</tr>
<tr>
<td>AU_NSW_R250_107</td>
<td>Sydney - Melbourne</td>
<td>7,084.4</td>
<td>7,436.0</td>
<td>351.6</td>
<td>21</td>
<td>0.05</td>
</tr>
<tr>
<td>AU_NSW_R250_108</td>
<td>Sydney - Melbourne</td>
<td>7,598.3</td>
<td>5,502.0</td>
<td>-2,096.3</td>
<td>13</td>
<td>-0.38</td>
</tr>
<tr>
<td>AU_NSW_R250_109</td>
<td>Sydney - Melbourne</td>
<td>8,304.4</td>
<td>12,553.0</td>
<td>4,248.6</td>
<td>27</td>
<td>0.34</td>
</tr>
<tr>
<td>AU_QLD_R250_101</td>
<td>Brisbane - Sydney</td>
<td>7,979.0</td>
<td>15,807.0</td>
<td>7,828.0</td>
<td>20</td>
<td>0.50</td>
</tr>
<tr>
<td>AU_QLD_R250_102</td>
<td>Brisbane - Sydney</td>
<td>8,069.4</td>
<td>9,596.0</td>
<td>1,526.6</td>
<td>19</td>
<td>0.16</td>
</tr>
<tr>
<td>AU_QLD_R250_103</td>
<td>Brisbane - Sydney</td>
<td>8,531.7</td>
<td>11,440.0</td>
<td>2,908.3</td>
<td>22</td>
<td>0.25</td>
</tr>
<tr>
<td>AU_QLD_R250_104</td>
<td>Brisbane - Sydney</td>
<td>7,975.4</td>
<td>7,119.0</td>
<td>-856.4</td>
<td>22</td>
<td>-0.12</td>
</tr>
<tr>
<td>AU_QLD_R250_105</td>
<td>Brisbane - Sydney</td>
<td>6,822.4</td>
<td>8,323.0</td>
<td>1,500.6</td>
<td>25</td>
<td>0.18</td>
</tr>
<tr>
<td>AU_QLD_R250_106</td>
<td>Brisbane - Sydney</td>
<td>7,991.0</td>
<td>7,652.0</td>
<td>-339.0</td>
<td>22</td>
<td>-0.04</td>
</tr>
<tr>
<td>AU_QLD_R250_107</td>
<td>Brisbane - Sydney</td>
<td>7,391.0</td>
<td>10,829.0</td>
<td>3,438.0</td>
<td>25</td>
<td>0.32</td>
</tr>
<tr>
<td>AU_QLD_R250_108</td>
<td>Brisbane - Sydney</td>
<td>7,233.0</td>
<td>9,057.0</td>
<td>1,824.0</td>
<td>20</td>
<td>0.20</td>
</tr>
<tr>
<td>AU_QLD_R250_109</td>
<td>Brisbane - Sydney</td>
<td>8,186.1</td>
<td>9,145.0</td>
<td>958.9</td>
<td>23</td>
<td>0.10</td>
</tr>
<tr>
<td>AU_QLD_R250_110</td>
<td>Brisbane - Sydney</td>
<td>7,536.4</td>
<td>10,450.0</td>
<td>2,913.6</td>
<td>30</td>
<td>0.28</td>
</tr>
<tr>
<td>AU_QLD_R250_111</td>
<td>Brisbane - Sydney</td>
<td>8,335.8</td>
<td>9,356.0</td>
<td>1,020.2</td>
<td>13</td>
<td>0.11</td>
</tr>
<tr>
<td>AU_QLD_R250_112</td>
<td>Brisbane - Sydney</td>
<td>8,941.9</td>
<td>8,311.0</td>
<td>-630.9</td>
<td>23</td>
<td>-0.08</td>
</tr>
<tr>
<td>AU_QLD_R250_113</td>
<td>Brisbane - Sydney</td>
<td>7,581.9</td>
<td>9,516.0</td>
<td>1,934.1</td>
<td>19</td>
<td>0.20</td>
</tr>
<tr>
<td>AU_QLD_R250_114</td>
<td>Brisbane - Sydney</td>
<td>7,711.2</td>
<td>9,861.0</td>
<td>2,149.8</td>
<td>18</td>
<td>0.22</td>
</tr>
<tr>
<td>AU_VIC_R250_101</td>
<td>Melbourne - Sydney</td>
<td>7,692.9</td>
<td>10,140.0</td>
<td>2,447.1</td>
<td>12</td>
<td>0.24</td>
</tr>
<tr>
<td>AU_VIC_R250_102</td>
<td>Melbourne - Sydney</td>
<td>7,842.4</td>
<td>4,913.0</td>
<td>-2,929.4</td>
<td>11</td>
<td>-0.60</td>
</tr>
<tr>
<td>AU_VIC_R250_103</td>
<td>Melbourne - Sydney</td>
<td>7,410.5</td>
<td>3,845.0</td>
<td>-3,565.5</td>
<td>14</td>
<td>-0.93</td>
</tr>
<tr>
<td>AU_VIC_R250_104</td>
<td>Melbourne - Sydney</td>
<td>7,316.6</td>
<td>12,494.0</td>
<td>5,177.4</td>
<td>18</td>
<td>0.41</td>
</tr>
<tr>
<td>AU_VIC_R250_105</td>
<td>Melbourne - Sydney</td>
<td>7,989.1</td>
<td>10,635.0</td>
<td>2,645.9</td>
<td>23</td>
<td>0.25</td>
</tr>
<tr>
<td>AU_VIC_R250_106</td>
<td>Melbourne - Sydney</td>
<td>7,954.7</td>
<td>16,564.0</td>
<td>8,609.3</td>
<td>33</td>
<td>0.52</td>
</tr>
<tr>
<td>AU_VIC_R250_107</td>
<td>Melbourne - Sydney</td>
<td>7,440.9</td>
<td>5,028.0</td>
<td>-2,412.9</td>
<td>10</td>
<td>-0.48</td>
</tr>
<tr>
<td>AU_VIC_R250_108</td>
<td>Melbourne - Sydney</td>
<td>7,835.0</td>
<td>9,849.0</td>
<td>2,014.0</td>
<td>12</td>
<td>0.20</td>
</tr>
</tbody>
</table>

**Mean** | 7,743.8 | 9,185.5 | 1,441.7 | 19.1 | 0.157

Note: The column ‘Route’ shows the two included cities where the first named city is also the depot location.

The results of this group of case instances are slightly lower with average marginal profits of 15.7%, still above the industry average. As in the previous cases, there is a high variance in the final results, but more than 70% of the test cases display profits.

Table 6.8 describes cases which include requests with customer locations within a 300 km radius of major cities.
Similar to the two previous groups of case instances, there is a noticeably high variance of profit margins between cases. The average profit margin is at a level of 10.1%, which is only slightly above the market average. This is expected, considering the more homogeneous market and possibly higher competition. However, over 71% of the cases show a profit and if the catchment areas get separated as presented in Table 6.9, it can be observed that especially the Melbourne base cases generate low profit margins. This is consistent with the two previous
experiments (R100 and R250) and may be due to the highly competitive market in this region. Future research could further investigate this finding based on the case study data.

Table 6.9: Profit margin overview for markets and experimental setup

<table>
<thead>
<tr>
<th>Files</th>
<th>Depot</th>
<th>Profit margins (proportions)</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>AU_NSW_R100_101-114,</td>
<td>Sydney</td>
<td>R100: 0.1803</td>
<td>R250: 0.0873</td>
</tr>
<tr>
<td>AU_NSW_R250_101-109,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_M300_101-110</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R100_101-11,</td>
<td>Brisbane</td>
<td>R100: 0.1902</td>
<td>R250: 0.1918</td>
</tr>
<tr>
<td>AU_QLD_R250_101-114,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_M300_101-110</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R100_101-106,</td>
<td>Melbourne</td>
<td>R100: 0.1378</td>
<td>R250: 0.1631</td>
</tr>
<tr>
<td>AU_VIC_R250_101-108,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_VIC_M300_101-116</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td></td>
<td>R100: 0.1763</td>
<td>R250: 0.1570</td>
</tr>
</tbody>
</table>

When comparing all three groups of experiments, the results show highest profit margins for cases where requests are close to cities included in a major freight corridor (R100). While these are usually the most competitive regions, the algorithm has the advantage of being able to identify geographical clusters of requests. Hence, the relatively better access to customers and therefore lower running costs, enables higher profits. On average, the optimisation system offers an average benefit of $0.68/km or an average additional profit/request of $56.58. Nevertheless, these findings are based on the available data. On average, highest profit margins were achieved for depot locations in Sydney with 15.94%, closely followed by Brisbane (15.14%). The Melbourne average is at 11.2%, slightly above the industry average. As indicated in Table 6.5, a min cost of $5,920 is incurred by any truck, therefore given the relatively low demand values extracted from the website, it may be useful to test the system for other potential vehicles in the fleet.

6.4.2 Request forecasting from historical data

Based on the case study data, potential benefits from applying historical forecasting are investigated next. The forecasting method used in this research has been presented in Chapter 3 (3.2.3.3) and applied in the experimental cases for benchmarking in Section 5.8. Here, the forecasting information is based on requests with similar features in each file, not included in the training file. In other words, the files created for analysis were split into ‘training’ and ‘testing’ requests, to enable validation of the performance of the model. The case files include
forecasting data for 36 sub-regions, obtained for a 6x6 grid of horizontal and vertical lines. For the calculations, all experiments have been conducted using the same algorithm settings to allow for comparability.

The experiments were run on the same cases as used in experiment 1. The results including forecasting are compared to the previous results in Table 6.10.

Table 6.10: Experimental results for R100 cases when forecasting is included

<table>
<thead>
<tr>
<th>Filename</th>
<th>Forecasting excluded</th>
<th>Forecasting included</th>
<th>Profit margin change</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Profits ($)</td>
<td>Profit margin (prop.)</td>
<td>Costs ($)</td>
</tr>
<tr>
<td>AU_NSW_R100_101</td>
<td>2,914.4</td>
<td>0.314</td>
<td>6,395.3</td>
</tr>
<tr>
<td>AU_NSW_R100_102</td>
<td>10,256.6</td>
<td>0.550</td>
<td>7,614.9</td>
</tr>
<tr>
<td>AU_NSW_R100_103</td>
<td>1,524.1</td>
<td>0.165</td>
<td>7,629.8</td>
</tr>
<tr>
<td>AU_NSW_R100_104</td>
<td>-3,522.5</td>
<td>-1.296</td>
<td>6,256.7</td>
</tr>
<tr>
<td>AU_NSW_R100_105</td>
<td>-3,314.9</td>
<td>-0.768</td>
<td>8,454.9</td>
</tr>
<tr>
<td>AU_NSW_R100_106</td>
<td>10,020.2</td>
<td>0.613</td>
<td>7,774.8</td>
</tr>
<tr>
<td>AU_NSW_R100_107</td>
<td>851.8</td>
<td>0.106</td>
<td>6,960.7</td>
</tr>
<tr>
<td>AU_NSW_R100_108</td>
<td>581.3</td>
<td>0.070</td>
<td>7,669.2</td>
</tr>
<tr>
<td>AU_NSW_R100_109</td>
<td>3,933.4</td>
<td>0.334</td>
<td>7,055.3</td>
</tr>
<tr>
<td>AU_NSW_R100_110</td>
<td>865.4</td>
<td>0.108</td>
<td>7,574.6</td>
</tr>
<tr>
<td>AU_NSW_R100_111</td>
<td>1,896.0</td>
<td>0.199</td>
<td>7,258.9</td>
</tr>
<tr>
<td>AU_NSW_R100_112</td>
<td>-1,480.8</td>
<td>-0.236</td>
<td>8,209.6</td>
</tr>
<tr>
<td>AU_NSW_R100_113</td>
<td>-2,844.2</td>
<td>-0.591</td>
<td>7,104.4</td>
</tr>
<tr>
<td>AU_NSW_R100_114</td>
<td>899.9</td>
<td>0.112</td>
<td>7,485.9</td>
</tr>
<tr>
<td>AU_QLD_R100_101</td>
<td>4,331.3</td>
<td>0.388</td>
<td>7,071.8</td>
</tr>
<tr>
<td>AU_QLD_R100_102</td>
<td>397.5</td>
<td>0.048</td>
<td>7,332.4</td>
</tr>
<tr>
<td>AU_QLD_R100_103</td>
<td>-1,360.6</td>
<td>-0.222</td>
<td>8,062.0</td>
</tr>
<tr>
<td>AU_QLD_R100_104</td>
<td>821.2</td>
<td>0.087</td>
<td>8,626.4</td>
</tr>
<tr>
<td>AU_QLD_R100_105</td>
<td>2,467.0</td>
<td>0.260</td>
<td>7,515.1</td>
</tr>
<tr>
<td>AU_QLD_R100_106</td>
<td>1,448.8</td>
<td>0.165</td>
<td>7,218.2</td>
</tr>
<tr>
<td>AU_QLD_R100_107</td>
<td>-1,213.2</td>
<td>-0.177</td>
<td>8,018.5</td>
</tr>
<tr>
<td>AU_QLD_R100_108</td>
<td>7,428.7</td>
<td>0.494</td>
<td>7,511.6</td>
</tr>
<tr>
<td>AU_QLD_R100_109</td>
<td>1,064.7</td>
<td>0.129</td>
<td>7,030.8</td>
</tr>
<tr>
<td>AU_QLD_R100_110</td>
<td>1,900.0</td>
<td>0.202</td>
<td>7,507.3</td>
</tr>
<tr>
<td>AU_QLD_R100_111</td>
<td>2,202.4</td>
<td>0.228</td>
<td>7,446.1</td>
</tr>
<tr>
<td>AU_VIC_R100_101</td>
<td>1,846.1</td>
<td>0.208</td>
<td>7,044.6</td>
</tr>
<tr>
<td>AU_VIC_R100_102</td>
<td>-2,251.2</td>
<td>-0.550</td>
<td>6,358.2</td>
</tr>
<tr>
<td>AU_VIC_R100_103</td>
<td>-617.8</td>
<td>-0.087</td>
<td>6,912.2</td>
</tr>
<tr>
<td>AU_VIC_R100_104</td>
<td>-1,693.7</td>
<td>-0.284</td>
<td>7,567.5</td>
</tr>
<tr>
<td>AU_VIC_R100_105</td>
<td>7,823.2</td>
<td>0.511</td>
<td>7,485.1</td>
</tr>
<tr>
<td>AU_VIC_R100_106</td>
<td>1,831.1</td>
<td>0.204</td>
<td>7,095.8</td>
</tr>
<tr>
<td>Mean</td>
<td>1,580.8</td>
<td><strong>0.176</strong></td>
<td>7,395.1</td>
</tr>
<tr>
<td>Std. Dev.:</td>
<td><strong>0.403</strong></td>
<td></td>
<td><strong>0.254</strong></td>
</tr>
</tbody>
</table>

The results show that although forecasting improves the results, there are situations when it displays a completely different outcome between excluding and including forecasting (e.g. AU_NSW_R100_102). This is due to the evolution of the solution where slightly different
decisions at certain points may have significant influence on the objective outcome (i.e. Baita et al., 1998). Examples include an additional request or a slightly different routing that lead the algorithm in a completely different direction in terms of ‘fitting’ the requests, and thus changes the objective value.

The overall results show that including forecasting in the algorithm increases the average absolute profits from $1,580.8 to $1,862.5. The profit margin has been further increased from 17.6% to 20.1%. As already indicated, this is significantly higher than the market average of about 9.3% (Bureau of Transport and Regional Economics, 2003). Additionally, as anticipated, having information on future possible demand reduces uncertainty and the standard deviation is noticeably lower. This shows that the method helps to stabilise profit fluctuation and reduces risks for operators.

Acknowledging that the applied forecasting approach is rather simplistic, future research on this topic is expected to contribute to the improvement of operation in this sector. Moreover, considering that there is almost no application of forecasting techniques in the industry and only a little research for the application in road freight transport, models that fit real-world industry requirements could answer the lasting call for road freight productivity increases.

Next, results for cases with a larger catchment area (R250) around capital cities are presented in Table 6.11.

Table 6.11: Experimental results for R250 cases under application of forecasting

<table>
<thead>
<tr>
<th>Filename</th>
<th>Forecasting excluded</th>
<th>Forecasting included</th>
<th>Profit margin change</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Profits ($)</td>
<td>Profit margin (prop.)</td>
<td>Costs ($)</td>
</tr>
<tr>
<td>AU_NSW_R250_101</td>
<td>3,999.6</td>
<td>0.340</td>
<td>7,243.5</td>
</tr>
<tr>
<td>AU_NSW_R250_102</td>
<td>-2,225.1</td>
<td>-0.399</td>
<td>7,851.2</td>
</tr>
<tr>
<td>AU_NSW_R250_103</td>
<td>885.8</td>
<td>0.108</td>
<td>8,754.6</td>
</tr>
<tr>
<td>AU_NSW_R250_104</td>
<td>-956.8</td>
<td>-0.153</td>
<td>7,764.5</td>
</tr>
<tr>
<td>AU_NSW_R250_105</td>
<td>1,534.0</td>
<td>0.168</td>
<td>7,766.2</td>
</tr>
<tr>
<td>AU_NSW_R250_106</td>
<td>788.6</td>
<td>0.094</td>
<td>7,179.3</td>
</tr>
<tr>
<td>AU_NSW_R250_107</td>
<td>351.6</td>
<td>0.047</td>
<td>7,071.8</td>
</tr>
<tr>
<td>AU_NSW_R250_108</td>
<td>-2,096.3</td>
<td>-0.381</td>
<td>7,626.5</td>
</tr>
<tr>
<td>AU_NSW_R250_109</td>
<td>4,248.6</td>
<td>0.338</td>
<td>8,144.9</td>
</tr>
<tr>
<td>AU_QLD_R250_101</td>
<td>7,828.0</td>
<td>0.495</td>
<td>7,864.9</td>
</tr>
<tr>
<td>AU_QLD_R250_102</td>
<td>1,526.6</td>
<td>0.159</td>
<td>8,023.6</td>
</tr>
<tr>
<td>AU_QLD_R250_103</td>
<td>2,908.3</td>
<td>0.254</td>
<td>7,073.3</td>
</tr>
<tr>
<td>AU_QLD_R250_104</td>
<td>-856.4</td>
<td>-0.120</td>
<td>7,638.3</td>
</tr>
<tr>
<td>AU_QLD_R250_105</td>
<td>1,500.6</td>
<td>0.180</td>
<td>7,448.6</td>
</tr>
<tr>
<td>AU_QLD_R250_106</td>
<td>-339.0</td>
<td>-0.044</td>
<td>8,500.3</td>
</tr>
<tr>
<td>AU_QLD_R250_107</td>
<td>3,438.0</td>
<td>0.317</td>
<td>7,907.2</td>
</tr>
</tbody>
</table>
Finally, the results for cases representing the metropolitan areas of Brisbane, Melbourne and Sydney are shown in Table 6.12.

Table 6.12: Experimental results for M300 cases under application of forecasting

<table>
<thead>
<tr>
<th>Filename</th>
<th>Forecasting excluded</th>
<th></th>
<th>Forecasting included</th>
<th></th>
<th>Profit margin change</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Profits ($)</td>
<td>Profit margin (prop.)</td>
<td>Costs ($)</td>
<td>Revenues ($)</td>
<td>Profits ($)</td>
</tr>
<tr>
<td>AU_NSW_M300_101</td>
<td>367.7</td>
<td>0.048</td>
<td>7,579.3</td>
<td>8,800.0</td>
<td>1,220.7</td>
</tr>
<tr>
<td>AU_NSW_M300_102</td>
<td>3,853.5</td>
<td>0.351</td>
<td>7,582.3</td>
<td>13,193.0</td>
<td>5,610.7</td>
</tr>
<tr>
<td>AU_NSW_M300_103</td>
<td>2,193.4</td>
<td>0.233</td>
<td>7,546.3</td>
<td>9,505.0</td>
<td>1,958.7</td>
</tr>
<tr>
<td>AU_NSW_M300_104</td>
<td>2,857.1</td>
<td>0.265</td>
<td>7,633.5</td>
<td>9,987.0</td>
<td>2,353.5</td>
</tr>
<tr>
<td>AU_NSW_M300_105</td>
<td>2,155.9</td>
<td>0.225</td>
<td>7,531.9</td>
<td>10,284.0</td>
<td>2,752.1</td>
</tr>
<tr>
<td>AU_NSW_M300_106</td>
<td>5,707.6</td>
<td>0.415</td>
<td>7,882.0</td>
<td>11,412.0</td>
<td>3,530.0</td>
</tr>
<tr>
<td>AU_NSW_M300_107</td>
<td>774.6</td>
<td>0.095</td>
<td>7,640.4</td>
<td>8,941.0</td>
<td>1,300.6</td>
</tr>
<tr>
<td>AU_NSW_M300_108</td>
<td>-1,688.9</td>
<td>-0.278</td>
<td>7,917.2</td>
<td>10,994.0</td>
<td>3,076.8</td>
</tr>
<tr>
<td>AU_NSW_M300_109</td>
<td>708.8</td>
<td>0.089</td>
<td>7,634.5</td>
<td>11,828.0</td>
<td>4,193.5</td>
</tr>
<tr>
<td>AU_NSW_M300_110</td>
<td>3,064.1</td>
<td>0.288</td>
<td>7,345.1</td>
<td>7,262.0</td>
<td>-83.1</td>
</tr>
<tr>
<td>AU_QLD_M300_101</td>
<td>167.5</td>
<td>0.020</td>
<td>7,468.2</td>
<td>7,663.0</td>
<td>194.8</td>
</tr>
<tr>
<td>AU_QLD_M300_102</td>
<td>-726.2</td>
<td>-0.108</td>
<td>7,672.2</td>
<td>11,635.0</td>
<td>3,962.8</td>
</tr>
<tr>
<td>AU_QLD_M300_103</td>
<td>-787.6</td>
<td>-0.120</td>
<td>8,001.1</td>
<td>6,804.0</td>
<td>-1,197.1</td>
</tr>
</tbody>
</table>

Similar to the previous set of cases, the inclusion of forecasting results in a profit increase, with a profit margin of 16.6%, compared to 15.7% without forecasting. The overall change of profit margins are with 0.9% not statistically significant (p-value = 0.59). Again, the standard deviation of profit margins is much lower, 0.237 compared to 0.323 for the standard cases (statistically significant at 0.05 level).
The results for the metropolitan cases show the lowest profit margins across all the investigated cases. However, by applying forecasting, the profit margin could be increased by 4.2% percent from 10.1% to 14.3%. This is the highest increase over all three sets. The standard deviation of profits over the cases is 0.191, slightly lower than the value for the cases when forecasting is not included. As in the previous cases, in some instances, the profits are reduced, as a result of the distinct optimisation conditions.

Considering the three case sets, the variation of profits as shown by the standard deviation is much more stable when including forecasting (0.191 – 0.254). This value almost doubles for the non-forecasting approach with variance between 0.223 and 0.403. Table 6.13 summarises the results for all forecasting experiments.
Table 6.13: Profit margin overview for markets and experimental setup when forecasting is applied

<table>
<thead>
<tr>
<th>Files</th>
<th>Depot</th>
<th>Profit margins (proportion)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AU_NSW_R100_101-114,</td>
<td>Sydney</td>
<td>R100 0.2308</td>
<td>R250 0.1250</td>
<td>M300 0.2535</td>
<td>Mean 0.2031</td>
</tr>
<tr>
<td>AU_NSW_R250_101-109,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_M300_101-110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R100_101-11,</td>
<td>Brisbane</td>
<td>0.1529</td>
<td>0.1858</td>
<td>0.1299</td>
<td>Mean 0.1562</td>
</tr>
<tr>
<td>AU_QLD_R250_101-114,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_M300_101-110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R100_101-106,</td>
<td>Melbourne</td>
<td>0.2154</td>
<td>0.1725</td>
<td>0.0626</td>
<td>Mean 0.1502</td>
</tr>
<tr>
<td>AU_VIC_R250_101-108,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_VIC_M300_101-116</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td></td>
<td>0.2012</td>
<td>0.1658</td>
<td>0.1429</td>
<td></td>
</tr>
</tbody>
</table>

In general, the highest profit margins could be yielded for the cases with a small catchment area around urban centres (R100, with an average of 20.12%) and the lowest for the metropolitan cases (M300, with an average of 14.29%). However, as shown in Table 6.12, the highest profit increases by including forecasting were obtained for the metro cases (4.2%). A possible explanation for this could be in the fact that customer locations of the metropolitan cases (M300) present a higher degree of ‘spatial scattering’ and thus benefits from forecasting arise from additional knowledge of this spatial distribution. On the other hand, R100 and R250 cases benefit from cluster or agglomeration economies. A particular insight is that across all metro cases, Sydney presents the highest profit margins and Melbourne the lowest.

To draw conclusions from all cases, a multiple regression model was estimated, using Melbourne as the reference case. The results confirm the significant role of distance and number of requests, although the model overall has a limited explanatory power.

Everything else being equal, an additional request considered in the routing is associated with $79.5 and a km of driving with 68 cents. Compared to Melbourne, the lowest profits appear in Brisbane, as already highlighted by the descriptive statistics.
Table 6.14: Relation between profit and activity, expressed as distance, number of requests, and location

<table>
<thead>
<tr>
<th>Variable</th>
<th>Unstandardised Coefficients</th>
<th>t</th>
<th>Sig.</th>
<th>Collinearity Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td>Tolerance</td>
</tr>
<tr>
<td>(Constant)</td>
<td>-2.755.273</td>
<td>-2.751</td>
<td>0.006</td>
<td></td>
</tr>
<tr>
<td>Sydney</td>
<td>596.900</td>
<td>1.313</td>
<td>0.191</td>
<td>0.119</td>
</tr>
<tr>
<td>Brisbane</td>
<td>118.140</td>
<td>0.260</td>
<td>0.795</td>
<td>0.665</td>
</tr>
<tr>
<td>Forecasting</td>
<td>226.861</td>
<td>2.026</td>
<td>0.044</td>
<td>0.127</td>
</tr>
<tr>
<td>Metropolitan</td>
<td>259.698</td>
<td>1.989</td>
<td>0.048</td>
<td>0.859</td>
</tr>
<tr>
<td>Number of requests</td>
<td>79.506</td>
<td>2.872</td>
<td>0.005</td>
<td>0.873</td>
</tr>
<tr>
<td>Total distance (km)</td>
<td>0.680</td>
<td>2.338</td>
<td>0.018</td>
<td>0.794</td>
</tr>
</tbody>
</table>

Note: $R^2_{adj}=0.147$, $F$-test $=16.782$, $p<0.001$.

When calculating profits per request, it is obvious that there are only few cases with large profits and that most profits are around $50-100/request as shown in Figure 6.3.

![Figure 6.3: Distribution of profits per request ($)](image)

### 6.5 Additional case study observations

During the case study setup and while conducting the experiments, a number of interesting aspects became apparent. They are discussed below in terms of their influence on the case study experiments, but also as directions for future investigation.
6.5.1 Bid-price estimation

Initially, this research planned to test a second forecasting method based on transport market prices obtained from collected data from www.truckit.net. Knowledge about expected market prices could help transport carriers to become more competitive when ‘sealed auction’-type requests arrive. Examples are requests that arrive via phone or facsimile, where the shipper contacts multiple operators and selects the one offering the lowest price. In this case, the operator would not have information about competitors’ prices (sealed auction) compared to when inspecting i.e. www.truckit.net (non-sealed auction). A number of multivariate regression analyses were conducted to estimate bid-prices. The results showed that estimated functions depend on the request region and consignment type. However, a number of test experiments indicated that the estimated bid-prices were not reflecting the real market prices accurately. Reasons for this could lie in the high heterogeneity of the market or non-numerical factors that were not included (such as flexibility or special conditions for transport and delivery). A research paper addressing this problem is currently in the publication process.

6.5.2 Other observations

First observations of the collected data showed that a number of transport requests exceeded the vehicle capacity chosen for the experiments. While the number of requests of high volume is minor, they may represent a valuable profit source. Future research may extend the applied model by a load-splitting option as presented in Dror et al. (1994) and Ho and Haugland (2004). The split delivery option may be beneficial not only for requests that go over vehicle capacities, but also in cases where splitting smaller loads between vehicles would allow for better utilisation of the fleet and for the inclusion of requests that may have been rejected otherwise.

Further, the exact vehicle size, capacity and associated variable and fixed costs should be considered in the optimisation, whenever is possible. For this study, the choice of vehicle was based on a common type used in the industry, relevant for the relatively small number of pallets considered in the experiments. While operators’ vehicle fleets are not easily exchangeable, and this research is not including a fleet selection problem (e.g. List et al., 2003) here, future research should address what kind of requests suit certain fleet characteristics.

A final observation is related to the applied forecasting technique. While the forecasting method has been introduced as a simplified way of estimating simple requests, further research should aim for an optimisation of the applied parameters and also consider advanced models
that forecast load sizes and prospective revenues. Forecasting models have been widely studied in the field of revenue management as shown in Chapter 2 of this thesis.

Future research could compare cases from Australian online marketplaces to cases based on data from Europe or North America. This could give additional insights on market differences and how Australian operators can benefit if the share of online traded transport requests further increases.
7 Conclusions and further research

This chapter provides a summary of the topics examined in this thesis, its contributions, and suggests possible future research directions.

7.1 Focus of the work

Road freight transport operators deal with high competition, low-profit margins and uncertainty about future customer demand, which all represent substantial challenges in their activity. Despite the wide availability of highly developed IT-systems, GPS tracking and internet connectivity, RFT operation still involves a large amount of manual planning and lacks integration and analysis of the available data. This thesis proposed a profit optimising system for road freight carriers, featuring the examination of dynamically incoming transport requests. Marginal cost and prices were determined under consideration of current and future requests with the objective of maximising profits for the regarded time horizon.

The problem has drawn substantial scholarly attention, but most studies still focus on partial problems, consider ‘myopic’ cost structures, or are for applications outside RFT. In this thesis, the problem was separated into two sub-problems: (1) the cost determination problem; and (2) the price determination and request selection problem. The implemented model solves sub-problem (1) by calculating the marginal costs of single requests with a dynamic pick-up and delivery problem with time windows (DPDPTW). Based on the marginal cost, the profits are maximised by applying dynamic pricing (2) and a capacity control policy while considering the current market situation and future requests. The problem has been solved here using an adaptive Tabu Search heuristic. Based on the findings of this research, the research questions that guided this research and stated in Chapter 1 can be answered as follows:

• What is the current competitive market situation on the LTL road freight market in Australia and other industrialised countries and how does the literature address this topic?

The comprehensive market review provided in Chapter 2 shows that the RFT markets are highly competitive with low-profit margins for the carriers.

RFT has many specific features and many different business types of various sizes can operate. Characteristics of the transport refer to request types (pallets, parcels, bulk load, heavy loads, specialty loads, mixed loads), services (courier, line-haul, mixed), and types of transport
contracts (long-term, short-term, spot market). All these factors influence the optimisation problem and carriers’ operation decisions.

The rich literature combines both academic literature and practitioner reports. They indicate that in most developed countries, where deregulation is present (Bayliss, 1998), there are large numbers of small operators and only a few big operators with high influence on the prices. In these markets, the small operators are also more ‘vulnerable’ to the economic environment conditions. For example, as a result of the high price competition, savings from recently reduced oil prices were passed on to the customers.

- What are the determining factors necessary to decide transport prices in practice and are there decision support tools available today?

In Chapter 2 of this thesis, the overview of the RFT pricing process and the available decision support systems showed that today, markets are mostly liberalised and that prices are determined by a number of factors on both shipper and carrier sides. Transport distance, load weight and volume play an important role. However, there are other key factors such as good characteristics, service requirements or urgency. On the carrier’s side, capacity utilisation, cost and price structure, service quality as well as business risks also affect transport prices. The exact price determination varies between businesses, but it can be seen as a negotiation, where the above stated attributes are inputs along with other factors (e.g. competitive situation, carrier’s service area, corporate relationship). With the price determination as a ‘black box’, there are multiple models aiming to estimate price or decompose it.

Given the high uncertainty and need to quickly react to competition and demand, RFT uses less strategic planning. Consequently, a decision support system for demand management should be much more flexible and not rely on strategic decisions, but rather promptly react to changes.

- Based on the data that is available to the industry, which methods and techniques from the literature can bring benefits to the industry and what are the limitations?

As indicated, the two sub-problems of cost determination problem and pricing and request selection problem, were analysed separately in this research, before being joined for optimisation. ‘Vehicle routing’ was substantially discussed in the operations research literature and a variety of models with different extensions are available, some of them close enough to real-world size problems, hence with direct application in practice. Pricing and request selection has been partially studied for road freight transport. Nevertheless, the field of
‘revenue management’ presents a broad range of work undertaken on this problem. However, while most RM techniques have been proposed, there are many challenges in practice. Examples for these are a lack of load unitisation, information availability or manual planning. A broad overview is presented in Chapter 3. For this thesis, a profit optimisation is proposed that features a DPDPTW for the cost determination and capacity control, forecasting and dynamic pricing for the price determination and request selection problem. The main benefit of applying these methods is an automated request evaluation. Limitations lie in the need for standardisation of transport units and information availability and connectivity of IT-systems.

- What are the solving methods available for this type of problem and how can it be addressed efficiently?

Mathematical problems presented here can be solved in different ways. Possible options are simulation-based optimisation, exact solvers or heuristic methods. An outline is provided in Chapter 4. While there are often multiple choices, they highly depend on the operational constraints such as problem size or available time to solve. Given the size of the test cases and the NP-hard nature of these problem, they cannot be solved exactly in polynomial time. Hence, heuristics are needed. A Tabu Search metaheuristic and a number of improvement heuristics are proposed. Experimental results on standard benchmark cases show that good results can be found in reasonable computational time.

- What is the advantage to freight operators of having better knowledge of their costs and the competitive market situation?

Chapter 5 presents a variety of experiments on the proposed optimisation approach. The results show that calculating marginal transport costs, compared to myopic approaches, avoids over or underestimating transport costs. Even if at the operator-level these two sorts of ‘miscalculations’ balance each other, they reduce competitiveness and can generate losses for single requests. In sum, the use of MC led to higher profits for all experiments undertaken.

Chapter 5 provides experiments for all methods as proposed in the optimisation approach (Chapter 3). The results show positive effects from all components including dynamic pricing, capacity control and forecasting. For the forecasting method, different settings were chosen and results better than the benchmark values were obtained. However, within the experiments, no specific attribute combination has been found superior to the others. This suggests that
further experiments could focus on this aspect in future studies, as this testing was beyond the scope of the thesis.

- Are the findings transferable to the real-world, and what are the limitations and important factors to consider for optimising operation and pricing in road freight transport?

Chapter 6 presents a case study that applies the optimising system as presented in Chapter 3 to real-world data. In order to realise this, approximately 9,000 transport requests have been recorded from an online marketplace for road freight transport requests in Australia (www.truckit.net). The chapter describes in detail the characteristics of the requests and their spatial distribution. Although this segment is minor in terms of volume in relation to the freight operation in Australia, it mirrors the major freight corridors in the country, with most freight movements to, from, or within major cities on the Australian east and south coast, and only minor numbers in the centre, the north and north-western regions. The experiments considered the depot location in one of three major Australian cities (Sydney, Melbourne, Brisbane) and included freight routes with pick-up and delivery requests within 100km and 250km, and within metro areas (300km radius). In more than 70% of the experiments, the optimisation has led to profit margins above the average industry profit margin (9.3%). The improvements were even higher when a forecasting system was applied. Table 6.9 and Table 6.13 summarise the findings and highlight the importance of location factors (e.g., higher marginal profits in Sydney than the other capital cities), in addition of the level of activity (number of requests, distances, flexibility). The results also show the importance of including discounts at the beginning of the scheduling period, in anticipation of follow-up requests for the same time period. These findings have implications for practice, considering that operators currently rely on a cost per km approach to set their bids.

In conclusion, the thesis provides a decision support system (DSS), with a combined optimisation that determines the marginal costs of single requests within a DPDPTW problem and maximises profits by applying dynamic pricing, capacity control and forecasting based on secondary data. After benchmarking against published test cases, the optimisation was applied to the RFT palletised transport in Australia, which is an atomic market with numerous small owner-operator agents. The model shows the benefits of the system, offering profits above those recorded for the sector.
7.2 Limitations and possible future extensions

Future work may be carried out in four directions: a) fleet and operation; b) simultaneous demand request and outsourcing; c) methodology – different heuristics and mixed-methods applied to demand analysis; d) market-wide applications.

- The current model does not allow for splitting request loads between different vehicles. Models including this option have been presented in the literature before and could further increase operators’ profits by enabling the acceptance of requests that cannot be served under the current setting.

While the presented model presents a scenario of a carrier with a homogeneous vehicle fleet, future research could investigate fleets of differing vehicles. The outcomes could especially prove valuable for small operators as regarded in this thesis.

Driving and working hour regulations as well as congestion and delays are not specifically incorporated in this research. Both aspects are natural extensions that would increase the direct applicability of the proposed models as DSS to real-world operation. Models for these problems have been studied before (e.g. Goel, 2009; Goel et al., 2012; Kok et al., 2010), but are context specific and have to be adapted to the country or region of application.

- Currently, the simulation does not give special attention to parallel arriving transport requests. This means that simultaneously arriving requests are treated sequentially. Future research will compare different assessments under the objective of profit maximisation and meeting real-time computation times. This can include both complete re-optimisation and cheapest insertion options.

Besides the focus on small size parallel requests, future work may address the new challenges in the acquisition of a bundle of requests from single or different shippers. Consequently, parallel request assignment could result in searching for the best ‘cluster’ of requests from various shippers, similar to the concept of a combinatorial auction (Song and Regan, 2003).

With the availability of fast internet connections and online marketplaces, the outsourcing of transport requests, even on short notice, becomes more and more important in freight transport. This topic has gained attention in the literature, with
many studies using the term of collaborative transport planning (Moon et al., 2012; Zäpfel and Bögl, 2008). In addition to the use of fix transport alliances, a resale on the spot market may be considered.

The availability of semi-automated or automated outsourcing tools could improve not only single operator’s capacity utilisation, special request combination and profits, but also have a positive effect on the industry as a whole. There is a variety of interesting topics to focus research on: collaborative transport, niche markets for small operators, system-wide capacity utilisation, congestion reductions and carbon emission savings, reward systems, shipper benefits and risks, etc.

• Further developments are also envisaged from the methodological point of view. The Tabu Search heuristic used to solve the proposed model showed good results, under reasonable computation time. Yet future research could involve the implementation of parallel computing on different cores of the applied machines. Moreover, based on the existing improving heuristics, a heuristic comparison of different metaheuristics (i.e. adaptive large neighbourhood search, genetic algorithm or simulated annealing)\(^{18}\) could be undertaken.

Finally, in conjunction with this thesis, a mixed-method study on bid-price generation for road freight transport requests has been conducted. The research combines quantitative analysis of freight auction data and qualitative analysis of the shipper-carrier interactions. The results show that transport prices depend not only on the ‘hard’/objective determinants, but rather on a combination of ‘hard’ and ‘soft’ collaborative factors on both parties. The paper is currently under review at a major international journal.

• The thesis proposes a profit optimisation system and provides experiments for applications by single operators. Presuming a successful real-world application, the system would automatically draw attention of other market competitors which would result in other, similar systems. The author assumes that a certain degree of market heterogeneity will always remain (as a result of depot location, vehicle fleet configuration, customer loyalty, etc.) and hence, operators will be able to identify niche markets. However, future research should address this topic, apply multiple systems

\(^{18}\) For further information, please refer to Glover and Kochenberger (2006).
parallel and research limiting bounds, operational strategies and the effects on the transport market.
8 Bibliography


9 Appendices

- Flowchart of the optimisation system

Figure 9.1: Flowchart of the optimising system
Heuristic benchmark route sequences

Table 9.1 presents the route sequences for the results of the heuristic benchmark experiments as presented in Section 4.4. The column 'Route sequences' shows the sequence of visited locations after leaving and before returning to the depot. Each line represents one route unless routes are too long to fit into one line. In this case lines end with a '-' before the line break.

Table 9.1: Route sequences of the benchmark experiments

<table>
<thead>
<tr>
<th>Filename</th>
<th>Route sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>lc101HB</td>
<td>9-23-39-24-63-75-64-10-76-40</td>
</tr>
<tr>
<td></td>
<td>15-16-41-53-54-77-78-42</td>
</tr>
<tr>
<td></td>
<td>7-27-8-28-61-62-81-82-99-100-103-104-105-106</td>
</tr>
<tr>
<td></td>
<td>11-29-12-30-67-68-85-86-93-94</td>
</tr>
<tr>
<td></td>
<td>1-25-2-49-50-71-83-89-84-90-72-26</td>
</tr>
<tr>
<td></td>
<td>3-4-37-38-59-69-79-60-97-98-80-70</td>
</tr>
<tr>
<td></td>
<td>13-14-43-55-65-56-44-66</td>
</tr>
<tr>
<td></td>
<td>5-21-35-47-36-22-6-87-88-48-101-102</td>
</tr>
<tr>
<td></td>
<td>19-33-34-45-57-20-46-58-95-96</td>
</tr>
<tr>
<td>lc102HB</td>
<td>7-21-8-91-87-92-88-73-22-74</td>
</tr>
<tr>
<td></td>
<td>15-29-30-39-47-16-40-48-75-76</td>
</tr>
<tr>
<td></td>
<td>11-23-35-24-36-61-62-12</td>
</tr>
<tr>
<td></td>
<td>93-85-83-94-86-57-67-68-84-58</td>
</tr>
<tr>
<td></td>
<td>103-101-105-45-102-106-104-71-72-46-99-100</td>
</tr>
<tr>
<td></td>
<td>5-25-33-6-51-55-56-26-34-95-96-97-98-52</td>
</tr>
<tr>
<td></td>
<td>13-14-89-90-81-59-69-70-60-82</td>
</tr>
<tr>
<td></td>
<td>9-27-37-28-38-63-10-64</td>
</tr>
<tr>
<td></td>
<td>3-17-31-43-53-18-4-54-79-80-44-32</td>
</tr>
<tr>
<td>lc103HB</td>
<td>87-5-77-69-70-88-75-6-78-76</td>
</tr>
<tr>
<td></td>
<td>65-11-53-66-54-35-36-12</td>
</tr>
<tr>
<td></td>
<td>3-7-63-8-29-64-30-4</td>
</tr>
<tr>
<td></td>
<td>1-2-21-22-79-81-39-80-82-40</td>
</tr>
<tr>
<td></td>
<td>51-52-17-49-31-50-18-43-32-44</td>
</tr>
<tr>
<td>Filename</td>
<td>Route sequences</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Filename</td>
<td>Route sequences</td>
</tr>
<tr>
<td>----------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
| lc109HB  | 7-8-15-51-52-69-77-70-16-78-103-104  
35-36-43-59-44-60-87-88-99-100  
17-21-23-53-54-24-83-18-22-84  
29-33-34-57-30-75-58-93-94-76  
5-3-9-4-10-71-72-81-6-82  
43-44  
7-8  
<table>
<thead>
<tr>
<th>Filename</th>
<th>Route sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>lr102HB</td>
<td>107-108-43-44</td>
</tr>
<tr>
<td></td>
<td>105-3-4-59-57-58-85-86-106-60</td>
</tr>
<tr>
<td></td>
<td>103-25-37-26-38-104</td>
</tr>
<tr>
<td></td>
<td>95-23-96-24</td>
</tr>
<tr>
<td></td>
<td>97-7-98-8-99-81-82-100</td>
</tr>
<tr>
<td></td>
<td>109-9-110-10-63-83-84-64</td>
</tr>
<tr>
<td></td>
<td>1-2-89-90-51-87-52-88</td>
</tr>
<tr>
<td></td>
<td>93-19-29-41-42-20-94-30</td>
</tr>
<tr>
<td></td>
<td>21-22-53-65-66-73-74-54</td>
</tr>
<tr>
<td></td>
<td>11-12-33-34-67-68-75-91-92-76</td>
</tr>
<tr>
<td></td>
<td>13-14-55-56</td>
</tr>
<tr>
<td></td>
<td>39-77-78-40</td>
</tr>
<tr>
<td></td>
<td>45-46</td>
</tr>
<tr>
<td></td>
<td>49-61-62-50</td>
</tr>
<tr>
<td></td>
<td>5-31-32-69-6-70</td>
</tr>
<tr>
<td></td>
<td>15-16-47-71-48-72</td>
</tr>
<tr>
<td>lr103HB</td>
<td>87-41-42-47-88-48-75-76</td>
</tr>
<tr>
<td></td>
<td>7-8-23-37-38-71-72-24</td>
</tr>
<tr>
<td></td>
<td>29-35-36-97-30-98</td>
</tr>
<tr>
<td></td>
<td>91-9-92-31-10-32</td>
</tr>
<tr>
<td></td>
<td>101-13-19-14-49-50-20-102</td>
</tr>
<tr>
<td></td>
<td>11-12-45-59-60-63-46-64</td>
</tr>
<tr>
<td></td>
<td>85-5-79-80-6-86</td>
</tr>
<tr>
<td></td>
<td>77-61-62-33-34-78-69-70-93-94</td>
</tr>
<tr>
<td></td>
<td>103-17-25-26-18-104-81-82</td>
</tr>
<tr>
<td></td>
<td>99-95-100-89-96-90</td>
</tr>
<tr>
<td></td>
<td>3-57-58-73-43-51-4-74-44-52</td>
</tr>
<tr>
<td>lr104HB</td>
<td>79-51-47-48-52-33-34-13-14-80</td>
</tr>
<tr>
<td></td>
<td>101-102-3-7-11-12-4-8-87-88</td>
</tr>
<tr>
<td></td>
<td>83-84-97-98</td>
</tr>
<tr>
<td></td>
<td>95-96-31-32-9-55-63-10-53-54-56-64</td>
</tr>
<tr>
<td></td>
<td>75-76-73-35-36-74-29-30-43-44-81-82</td>
</tr>
<tr>
<td></td>
<td>99-77-100-5-17-18-37-78-38-6-103-104</td>
</tr>
<tr>
<td>lr105HB</td>
<td>47-63-48-77-79-64-80-78</td>
</tr>
<tr>
<td></td>
<td>11-43-44-61-75-76-62-129-10-39-40</td>
</tr>
<tr>
<td></td>
<td>35-37-36-38</td>
</tr>
<tr>
<td></td>
<td>7-21-51-65-22-66-52-8</td>
</tr>
<tr>
<td></td>
<td>25-15-16-26</td>
</tr>
<tr>
<td></td>
<td>17-18-41-42-81-82-95-96</td>
</tr>
<tr>
<td></td>
<td>1-3-2-4-69-70-101-102-105-106</td>
</tr>
<tr>
<td></td>
<td>5-19-53-6-20-54-93-94</td>
</tr>
<tr>
<td>Filename</td>
<td>Route sequences</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------</td>
</tr>
</tbody>
</table>
| lr106HB  | 97-98-25-45-26-47-71-48-72-46  
|          | 99-81-100-82-37-38-57-58  
|          | 95-87-96-27-28-41-88-79-80-42  
|          | 7-103-8-55-104-56-65-73-74-66  
|          | 93-85-11-86-94-12-69-70-75-76  
|          | 89-9-10-49-90-50-91-92  
|          | 5-6-15-16-43-44  
|          | 13-31-14-51-32-52  
|          | 77-17-78-35-59-61-62-36-60-18  
|          | 21-39-40-53-54-22-83-84  
|          | 19-33-29-20-63-34-64-30  
|          | 101-1-3-23-102-24-67-68-2-4  |
| lr107HB  | 23-31-24-97-95-32-98-96  
|          | 3-7-4-8-25-26-33-51-52-83-84-34  
|          | 103-81-82-11-15-12-39-59-60-104-40-16  
|          | 55-9-56-29-30-10-73-79-80-74  
|          | 5-71-72-17-6-41-18-61-42-62  
|          | 63-13-35-43-44-36-64-14  
|          | 1-53-54-37-2-69-45-46-38-70  
|          | 83-79-5-80-29-6-84-45-30-46  
|          | 99-3-87-85-88-23-100-4-24-33-34-86  
|          | 81-61-7-41-8-19-55-20-82-42-62-56  
|          | 95-91-96-15-92-75-25-76-53-26-54-16  
|          | 1-73-74-2-43-11-44-47-48-65-12-66  
|          | 93-71-49-51-17-94-72-50-52-18-77-78  |
| lr109HB  | 13-7-51-8-61-52-14-103-62-104  
|          | 11-53-31-29-32-30-12-54  
|          | 3-33-47-34-69-4-87-88-70-481-9-10-71-2-59-72-93-94-60  
|          | 21-35-36-89-90-22-105-106  
|          | 77-78  
|          | 23-24-79-67-68-99-80-100  
|          | 17-63-64-81-18-75-76-82  
<p>|          | 15-25-73-16-95-74-26-96  |</p>
<table>
<thead>
<tr>
<th>Filename</th>
<th>Route sequences</th>
</tr>
</thead>
</table>
| lr110HB  | 15-37-16-31-32-38  
|          | 9-10-55-101-23-24-56-102  
|          | 11-12-41-42-67-81-79-68-82-80  
|          | 1-91-92-57-58-63-103-104-2-64-87-88  
|          | 49-50-51-52-77-69-43-70-78-44  
|          | 3-93-27-94-28-4-99-59-60-100  
|          | 17-47-48-18-97-98  
|          | 53-5-85-33-34-89-90-6-54-86  
|          | 7-8-21-22-65-66  
|          | 7-5-39-25-76-26-45-46-40  
|          | 13-14-19-20-71-61-95-62-72-96  
|          | 83-29-35-36-30-84-73-74-80 |
| lr111HB  | 49-103-23-24-104-50-41-55-56-42-97-98  
|          | 83-84-27-7-11-63-8-28-64-12-93-94  
|          | 61-51-25-26-62-69-52-70-87-88  
|          | 19-33-37-34-20-67-68-38  
|          | 5-6-3-17-4-35-18-89-36-90  
|          | 105-106-75-15-16-53-76-77-54-78-107-108  
|          | 45-46-79-13-80-29-14-73-74-30 |
| lr112HB  | 9-10-83-84-69-70  
|          | 21-22-51-52-29-30-59-60  
|          | 55-89-56-95-37-96-38-90  
|          | 71-5-6-72-23-47-48-24-99-100  
|          | 35-87-36-53-13-43-44-73-14-54-74-88  
|          | 17-1-41-18-2-81-42-75-82-85-86-76  
|          | 45-46-11-15-16-67-68-12-103-104  
|          | 27-61-62-28-3-4-79-80  
|          | 49-77-31-78-32-93-50-94  
|          | 63-64-105-106-25-26 |
|          | 7-3-8-25-31-33-34-23-47-39-55-51-4-83-32-26-24-40-56-52-81-83-79-82-87-88-80-84  
| lr202HB  | 21-22  
<table>
<thead>
<tr>
<th>Filename</th>
<th>Route sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>Route sequences</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------</td>
</tr>
</tbody>
</table>
| lrc102HB | 105-23-45-106-24-46-79-80  
7-17-8-37-38-55-18-77-78-56  
31-32  
13-35-14-53-54-75-36-76  
103-25-26-51-52-104-97-98  
19-27-20-73-28-81-74-82-99-100  
15-16-41-61-47-62-48-42  
91-11-12-92-87-57-58-63-67-68-88-64  
89-90-43-44-59-60-85-86  
1-95-39-40-83-96-65-2-84-66 |
| lrc103HB | 97-67-98-17-21-18-31-32-87-22-88-68  
103-5-11-6-12-95-96-104  
105-106-61-41-62-42  
101-102-63-23-64-71-72-24-93-94  
99-73-100-74-25-26-53-43-54-44  
9-10-27-81-37-38-28-82-69-70  
83-7-65-66-84-8-33-34-47-48  
1-91-2-79-19-92-80-20-89-90  
3-4-77-78-35-36-39-40  
75-55-76-59-56-29-60-30-57-58-85-86 |
97-91-92-98-51-17-23-24-52-18  
1-2-37-35-36-33-38-25-31-32-26-34  
85-79-7-61-8-80-47-62-55-56-48-86  
3-99-4-81-100-49-29-30-50-82  
93-94-83-59-84-71-60-69-67-68-70-72  
105-103-106-89-5-9-95-96-104-6-90-10  
77-57-78-11-21-22-58-12 |
| lrc105HB | 3-69-47-70-59-63-4-48-105-106-64-60  
77-9-78-10-57-41-42-83-84-58-103-104  
91-29-95-92-30-96  
13-17-18-31-23-14-89-90-32-24  
7-8-75-35-36-76  
43-93-44-94-55-99-100-56  
37-11-27-12-38-28-97-98  
79-51-53-54-52-80  
1-71-21-33-22-2-72-34  
5-6-81-82  
61-62 |
33-34-25-26-57-58-103-97-98-104  
11-12-49-53-50-85-86-54-105-106  
<table>
<thead>
<tr>
<th>Filename</th>
<th>Route sequences</th>
</tr>
</thead>
</table>
| lrc107HB | 63-19-33-23-91-34-64-20-24-92  
51-7-55-56-8-39-52-101-40-102  
13-17-49-71-18-14-72-50  
15-43-44-16  
69-1-77-2-47-21-70-78-22-48  
57-5-11-6-99-100-58-12-75-76  
81-61-95-83-62-82-84-96-89-90  
3-4-59-60-35-73-36-74  
| lrc108HB | 39-11-12-13-31-32-14-40  
19-43-5-20-44-85-6-77-78-86  
49-89-90-50  
47-65-48-99-41-66-100-42-83-84  
69-53-54-17-18-75-91-70-37-76-92-38  
35-97-95-36-25-55-96-26-56-98  
1-3-4-101-102-2-103-104  
21-9-33-34-7-8-51-32-93-94-22-10  
73-71-72-74-61-63-67-64-68-62  
23-29-45-24-30-46-59-60-81-82 |
| lrc201HB | 1-7-23-11-55-59-41-49-53-24-60-56-2-54-73-74-81-82-8-12-42-50  
| lrc203HB | 7-8-73-19-74-15-20-16  
<table>
<thead>
<tr>
<th>Filename</th>
<th>Route sequences</th>
</tr>
</thead>
</table>
- Case study file description

The following tables (Table 9.2, Table 9.3, Table 9.4) present the setup of the case study instances. They include the depot location, the route or region served and the catchment area. An Appendix is represented by a radius around the participated cities. For example, the first case of Table 9.1 (AU_NSW_R100_101) features a depot located in Sydney, and includes customers who are located within a 100km radius of either of the cities Sydney or Melbourne.

Table 9.2: R100 case description

<table>
<thead>
<tr>
<th>Filename</th>
<th>Depot</th>
<th>Route/Region</th>
<th>Pickup and/or delivery in radius (km) of:</th>
<th>Sydney</th>
<th>Melbourne</th>
<th>Brisbane</th>
</tr>
</thead>
<tbody>
<tr>
<td>AU_NSW_R100_101</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R100_102</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R100_103</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R100_104</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R100_105</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R100_106</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R100_107</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R100_108</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R100_109</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R100_110</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R100_111</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R100_112</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R100_113</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R100_114</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R100_101</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R100_102</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R100_103</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R100_104</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R100_105</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R100_106</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R100_107</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R100_108</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R100_109</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R100_110</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R100_111</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R100_101</td>
<td>Melbourne</td>
<td>Melbourne - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R100_102</td>
<td>Melbourne</td>
<td>Melbourne - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R100_103</td>
<td>Melbourne</td>
<td>Melbourne - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R100_104</td>
<td>Melbourne</td>
<td>Melbourne - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R100_105</td>
<td>Melbourne</td>
<td>Melbourne - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R100_106</td>
<td>Melbourne</td>
<td>Melbourne - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 9.3: R250 case description

<table>
<thead>
<tr>
<th>Filename</th>
<th>Depot</th>
<th>Route/Region</th>
<th>Pickup and/or delivery in radius (km) of Sydney</th>
<th>Melbourne</th>
<th>Brisbane</th>
</tr>
</thead>
<tbody>
<tr>
<td>AU_NSW_R250_101</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R250_102</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R250_103</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R250_104</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R250_105</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R250_106</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R250_107</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R250_108</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_R250_109</td>
<td>Sydney</td>
<td>Sydney - Melbourne</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R250_101</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R250_102</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R250_103</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R250_104</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R250_105</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R250_106</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R250_107</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R250_108</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R250_109</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R250_110</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R250_111</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R250_112</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R250_113</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_QLD_R250_114</td>
<td>Brisbane</td>
<td>Brisbane - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R250_101</td>
<td>Melbourne</td>
<td>Melbourne - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R250_102</td>
<td>Melbourne</td>
<td>Melbourne - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R250_103</td>
<td>Melbourne</td>
<td>Melbourne - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R250_104</td>
<td>Melbourne</td>
<td>Melbourne - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R250_105</td>
<td>Melbourne</td>
<td>Melbourne - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R250_106</td>
<td>Melbourne</td>
<td>Melbourne - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R250_107</td>
<td>Melbourne</td>
<td>Melbourne - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_R250_108</td>
<td>Melbourne</td>
<td>Melbourne - Sydney</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
Table 9.4: M300 metro case description

<table>
<thead>
<tr>
<th>Filename</th>
<th>Depot and served region</th>
<th>Pickup and/or delivery in radius (km) of</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Sydney</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>AU_NSW_M300_101</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_M300_102</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_M300_103</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_M300_104</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_M300_105</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_M300_106</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_M300_107</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_M300_108</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_M300_109</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_NSW_M300_110</td>
<td>Sydney</td>
<td>x</td>
</tr>
<tr>
<td>AU_VIC_M300_101</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_M300_102</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_M300_103</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_M300_104</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_M300_105</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_M300_106</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_M300_107</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_M300_108</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_M300_109</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_M300_110</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_M300_111</td>
<td>Sydney</td>
<td></td>
</tr>
<tr>
<td>AU_VIC_M300_112</td>
<td>Sydney</td>
<td></td>
</tr>
</tbody>
</table>
- Case study route example

Figure 9.2 presents an example of a vehicle route from the case study (AU_NSW_R100_109). The map shows an excerpt of south-east Australia with some of its major cities and the road network. Green circles embody customer locations with diameters according to the request size (only acquired transport jobs are shown). The black line represents the vehicle route, starting from the depot in Sydney, serving all acquired transport requests and returning to the depot. It is to acknowledge that certain constraints such as visitation time windows and the dynamics of the problem have an influence on the solution.

Figure 9.2: Vehicle route of case study instance AU_NSW_R100_109
E.1. MyRMcontrol class

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

import jxl.Workbook;
import jxl.read.biff.BiffException;
import jxl.write.Label;
import jxl.write.NumberFormat;
import jxl.write.Number;
import jxl.write.WritableCell;
import jxl.write.WritableCellFormat;
import jxl.write.WritableFont;
import jxl.write.WritableSheet;
import jxl.write.WritableWorkbook;
import jxl.write.WriteException;

public class MyRMcontrol {
    static int rechnung;
    public static int requestNr;
    static FileWriter fw;
    static BufferedWriter bw;
    static DecimalFormat df = new DecimalFormat("#.00");
    static int kkk;
    static int VVV;
    static int vvv;
    static int nrOfCustomers;
    static List<Double[]> fileMatrix = new ArrayList<Double[]>();
    static List<Double> costs = new ArrayList<Double>();
    static String filename;
    static List<String> filesToCompute = new ArrayList<String>(Arrays.asList(
        /*VRPPDTWCE_AU_NSW_F_1","VRPPDTWCE_AU_NSW_F_2","VRPPDTWCE_AU_NSW_F_3","VRPPDTWCE_AU_NSW_F_4","VRPPDTWCE_AU_NSW_F_5","VRPPDTWCE_AU_NSW_F_6","VRPPDTWCE_AU_NSW_F_7","VRPPDTWCE_AU_NSW_F_8","VRPPDTWCE_AU_NSW_F_9","VRPPDTWCE_AU_NSW_F_10","VRPPDTWCE_AU_NSW_F_11","VRPPDTWCE_AU_NSW_F_12","VRPPDTWCE_AU_NSW_F_13","VRPPDTWCE_AU_NSW_F_14","VRPPDTWCE_AU_NSW_F_15","VRPPDTWCE_AU_NSW_F_16","VRPPDTWCE_AU_NSW_F_17","VRPPDTWCE_AU_NSW_F_18","VRPPDTWCE_AU_NSW_F_19","VRPPDTWCE_AU_NSW_F_20","VRPPDTWCE_AU_VIC_F_1","VRPPDTWCE_AU_VIC_F_2","VRPPDTWCE_AU_VIC_F_3","VRPPDTWCE_AU_VIC_F_4","VRPPDTWCE_AU_VIC_F_5","VRPPDTWCE_AU_VIC_F_6","VRPPDTWCE_AU_VIC_F_7","VRPPDTWCE_AU_VIC_F_8","VRPPDTWCE_AU_VIC_F_9","VRPPDTWCE_AU_VIC_F_10","VRPPDTWCE_AU_VIC_F_11","VRPPDTWCE_AU_VIC_F_12","VRPPDTWCE_AU_VIC_F_13","VRPPDTWCE_AU_VIC_F_14","VRPPDTWCE_AU_VIC_F_15","VRPPDTWCE_AU_VIC_F_16","VRPPDTWCE_AU_VIC_F_17","VRPPDTWCE_AU_VIC_F_18","VRPPDTWCE_AU_VIC_F_19","VRPPDTWCE_AU_VIC_F_20","VRPPDTWCE_AU_QLD_F_1","VRPPDTWCE_AU_QLD_F_2","VRPPDTWCE_AU_QLD_F_3","VRPPDTWCE_AU_QLD_F_4","VRPPDTWCE_AU_QLD_F_5","VRPPDTWCE_AU_QLD_F_6","VRPPDTWCE_AU_QLD_F_7","VRPPDTWCE_AU_QLD_F_8","VRPPDTWCE_AU_QLD_F_9","VRPPDTWCE_AU_QLD_F_10","VRPPDTWCE_AU_QLD_F_11","VRPPDTWCE_AU_QLD_F_12","VRPPDTWCE_AU_QLD_F_13","VRPPDTWCE_AU_QLD_F_14","VRPPDTWCE_AU_QLD_F_15","VRPPDTWCE_AU_QLD_F_16","VRPPDTWCE_AU_QLD_F_17","VRPPDTWCE_AU_QLD_F_18","VRPPDTWCE_AU_QLD_F_19","VRPPDTWCE_AU_QLD_F_20","*/
    ));
    public class MyRMcontrol {
        public MyRMcontrol {
            import java.io.BufferedReader;
            import java.io.BufferedWriter;
            import java.io.File;
            import java.io.FileNotFoundException;
            import java.io.FileReader;
            import java.io.FileWriter;
            import java.io.IOException;
            import java.text.DecimalFormat;
            import java.util.ArrayList;
            import java.util.Arrays;
            import java.util.List;
            import java.util.Scanner;
            import jxl.Workbook;
            import jxl.read.biff.BiffException;
            import jxl.write.Label;
            import jxl.write.NumberFormat;
            import jxl.write.Number;
            import jxl.write.WritableCell;
            import jxl.write.WritableCellFormat;
            import jxl.write.WritableFont;
            import jxl.write.WritableSheet;
            import jxl.write.WritableWorkbook;
            import jxl.write.WriteException;
            public class MyRMcontrol {
                static int rechnung;
                public static int requestNr;
                static FileWriter fw;
                static BufferedWriter bw;
                static DecimalFormat df = new DecimalFormat("#.00");
                static int kkk;
                static int VVV;
                static int vvv;
                static int nrOfCustomers;
                static List<Double[]> fileMatrix = new ArrayList<Double[]>();
                static List<Double> costs = new ArrayList<Double>();
                static String filename;
                static List<String> filesToCompute = new ArrayList<String>(Arrays.asList(
                    /*"VRPPDTWCE_AU_NSW_F_1","VRPPDTWCE_AU_NSW_F_2","VRPPDTWCE_AU_NSW_F_3","VRPPDTWCE_AU_NSW_F_4","VRPPDTWCE_AU_NSW_F_5","VRPPDTWCE_AU_NSW_F_6","VRPPDTWCE_AU_NSW_F_7","VRPPDTWCE_AU_NSW_F_8","VRPPDTWCE_AU_NSW_F_9","VRPPDTWCE_AU_NSW_F_10","VRPPDTWCE_AU_NSW_F_11","VRPPDTWCE_AU_NSW_F_12","VRPPDTWCE_AU_NSW_F_13","VRPPDTWCE_AU_NSW_F_14","VRPPDTWCE_AU_NSW_F_15","VRPPDTWCE_AU_NSW_F_16","VRPPDTWCE_AU_NSW_F_17","VRPPDTWCE_AU_NSW_F_18","VRPPDTWCE_AU_NSW_F_19","VRPPDTWCE_AU_NSW_F_20","VRPPDTWCE_AU_VIC_F_1","VRPPDTWCE_AU_VIC_F_2","VRPPDTWCE_AU_VIC_F_3","VRPPDTWCE_AU_VIC_F_4","VRPPDTWCE_AU_VIC_F_5","VRPPDTWCE_AU_VIC_F_6","VRPPDTWCE_AU_VIC_F_7","VRPPDTWCE_AU_VIC_F_8","VRPPDTWCE_AU_VIC_F_9","VRPPDTWCE_AU_VIC_F_10","VRPPDTWCE_AU_VIC_F_11","VRPPDTWCE_AU_VIC_F_12","VRPPDTWCE_AU_VIC_F_13","VRPPDTWCE_AU_VIC_F_14","VRPPDTWCE_AU_VIC_F_15","VRPPDTWCE_AU_VIC_F_16","VRPPDTWCE_AU_VIC_F_17","VRPPDTWCE_AU_VIC_F_18","VRPPDTWCE_AU_VIC_F_19","VRPPDTWCE_AU_VIC_F_20","VRPPDTWCE_AU_QLD_F_1","VRPPDTWCE_AU_QLD_F_2","VRPPDTWCE_AU_QLD_F_3","VRPPDTWCE_AU_QLD_F_4","VRPPDTWCE_AU_QLD_F_5","VRPPDTWCE_AU_QLD_F_6","VRPPDTWCE_AU_QLD_F_7","VRPPDTWCE_AU_QLD_F_8","VRPPDTWCE_AU_QLD_F_9","VRPPDTWCE_AU_QLD_F_10","VRPPDTWCE_AU_QLD_F_11","VRPPDTWCE_AU_QLD_F_12","VRPPDTWCE_AU_QLD_F_13","VRPPDTWCE_AU_QLD_F_14","VRPPDTWCE_AU_QLD_F_15","VRPPDTWCE_AU_QLD_F_16","VRPPDTWCE_AU_QLD_F_17","VRPPDTWCE_AU_QLD_F_18","VRPPDTWCE_AU_QLD_F_19","VRPPDTWCE_AU_QLD_F_20","*/
                ));
            }
        }
    }
```
public static void main (String args[]) throws IOException, WriteException, BiffException{
//for(int gridloop=3; gridloop<5; gridloop++){
    double stdDev = 0.0;
    System.out.println("GRIDLOOP: "+MyDynamicPricing.gridNumber+" ");
    for(int stdDevloop=1; stdDevloop<4; stdDevloop++)
        for(int set1=1; set1<4; set1++)
            System.out.println("StdDev Loop: "+stdDev);
    MyDynamicPricing.priceRebate = 20;
    System.out.println(filesToCompute.get(file));
    stdDev = filesToCompute.get(file);
    "directoryAndFilename = "/"C:\\Users\\Hendrik\\workspace\\TestFiles\\"+filename".txt;";*/
    //"C:\\Users\\21576959\\Workspace\\02 VRPPD test files\\04 MyRM test files\\"+filename".txt";"*/
0058 */
0059 */
0060 */
0061 });
0062 static String directoryAndFilename; = "C:\\Users\\21576959\\Workspace\\04 VRPPD test files\\"+filename".txt";"*/
0063 static String directory = "/"C:\\Users\\Hendrik\\workspace\\";"*/"C:\\Users\\21576959\\Workspace\\";"*/
0064 static List<Double[]> fileMatrixCurrent = new ArrayList<Double[]>();
0065 static List<Double[]> fileMatrixRoutes = new ArrayList<Double[]>();
0066 public static List<Double[]> fileMatrixCurrentNew = new ArrayList<Double[]>();
0067 public static List<Double[]> fileMatrixCurrentREM = new ArrayList<Double[]>();
0068 static double[][] timePosition; //time at each position
0069 static int[][][] previousW;
0070 public static double[] timePosition; //time at each position
0071 public static double[] timePositionRem;
0072 static int[] startingPosition;
0073 static double[] startingTime;
0074 static double[] startingPositionRem;
0075 static double[] startingTimeRem;
0076 static int[] startingPositionRem;
0077 static int[] startingTimeRem;
0078 public static List<Integer> wtourListRem = new ArrayList<Integer>();
0079 public static List<Integer> wtourListRemID = new ArrayList<Integer>();
0080 public static List<Double[]> fileMatrixCurrentREM = new ArrayList<Double[]>();
0081 static double alreadyDrivenDistance;
0082 static double alreadyDrivenDistanceIncremental;
0083 static double alreadyDrivenCapRm;
0084 static double alreadyDrivenCapRmIncremental;
0085 static int t;
0086 static boolean calculate = true;
0087 static boolean desktopPrintout = false;
0088 static List<Double[]> results = new ArrayList<Double[]>();
0089 static Double[] resultsRow;
0090 static double[] results2;
0091 static boolean ONLYvrp = false;
0092 static boolean caseStudy = true;
0093 static boolean fileForecast = true;
0094 static double stdDev = 0.0;
0095 
0096 public static void main (String args[]) throws IOException, WriteException, BiffException{
0097 //for(int gridloop=3; gridloop<5; gridloop++){
0098     double stdDev = 0.0;
0099     System.out.println("GRIDLOOP: "+MyDynamicPricing.gridNumber+" ");
0100     for(int stdDevloop=1; stdDevloop<4; stdDevloop++)
0101         for(int set1=1; set1<4; set1++)
0102             System.out.println("StdDev Loop: "+stdDev);
0103     MyDynamicPricing.priceRebate = 20;
0104     System.out.println(filesToCompute.get(file));
0105     filename = filesToCompute.get(file);
0106     directoryAndFilename = "C:\\Users\\Hendrik\\workspace\\TestFiles\\"+filename".txt;";"*/
0107     //"C:\\Users\\21576959\\Workspace\\02 VRPPD test files\\04 MyRM test files\\"+filename".txt";"*/
0108     directory = "C:\\Users\\21576959\\Workspace\\02 VRPPD test files\\04 MyRM test files\\"+filename".txt";"*/
0109     directory = "C:\\Users\\21576959\\Workspace\\02 VRPPD test files\\04 MyRM test files\\"+filename".txt";"*/
0110     directory = "C:\\Users\\21576959\\Workspace\\02 VRPPD test files\\04 MyRM test files\\"+filename".txt";"*/
//directory +"02 VRPPD test files\04 MyRM test files\DVRPPD CE Solomon\"+filename\".txt";

directory + "02 VRPPD test files\06 Case study instances\"+ filename\".txt";

//directory + "02 VRPPD test files\04 MyRM test files\"+filename\".txt";

//directory + "02 VRPPD test files\05 Heuristic benchmark\"+filename\".txt";

/** New initialization for file loop**/
readFile();
MyDynamicPricing.createDeviationMatrix(stdDev);
MyDynamicPricing.discountPercentage = .05;

//for(int set=1; set<=4; set++)
{//(This one calculates same things for every file)

List<Double> costs = new ArrayList<Double>();
fileMatrix.clear();
fileMatrixCurrent = new ArrayList<Double>();
fileMatrixCurrentNew = new ArrayList<Double>();
wtourListRem = new ArrayList<Integer>();
wtourListRemID = new ArrayList<Integer>();
fileMatrixCurrentREM = new ArrayList<Double>();
costs.clear();
/**/
costs.add(0.0);
costs.add(0.0);
// read test file:
readFile();
if(fileForecast ==false) MyDynamicPricing.createExpectedDemandMatrix(stdDev);

System.out.println("Discounted price:"+MyDynamicPricing.returnDiscountedPriceFromForecast(1000, 1, 2, MyDynamicPricing.expectedDemand));

alreadyDrivenDistance=0;
alreadyDrivenCapKm=0;
startingPosition = new int[kkk];
startTime = new double[kkk];
results.clear();
results2 = new double[35];
timePosition = new double[fileMatrix.size()+1][kkk];
fileMatrixCurrentREM = new ArrayList<Double>();
requestNr = 1;
if(desktopPrintout==true)MyCanvas.main3();
double starzeit = System.currentTimeMillis();
for(t= fileMatrix.get(0)[9].intValue(); t<=1000; t+=1){
T: if(checkIfRequestArrival(t) == true){
System.out.println("RM STARTED 1"+"GRIDLOOP: "+MyDynamicPricing.gridNumber+" StdDev : "+stdDev);
calculate = true;
resultsRow = new Double[18];
starzeit = System.currentTimeMillis();
startingPosition = new int[kkk];
System.out.println();
System.err.println("RM calculation started for time: "+ t);
System.out.println();
//remember current solution for later:
if(*(28/04t>10 &&/*t=fileMatrix.get(0)[9] && costs.get(costs.size()-1) < 100000){
startingPositionRem = Arrays.copyOf(startingPosition, startingPosition.length);
timePositionRem = Arrays.copyOf(timePosition, timePosition.length);
previousWRem = MySolution.deepCopyOf(previousW);
0169  //createWtourListREMiD();
0170  startingTimeRem = Arrays.copyOf(startingTime, startingTime.length);
0171  fileMatrixCurrentREM.clear();
0172  for(int i=0; i<fileMatrixCurrent.size(); i++){
0173    fileMatrixCurrentREM.add(Arrays.copyOf(fileMatrixCurrent.get(i), fileMatrixCurrent.get(i).length));
0174  }
0175  } // delete old files from fileMatrixCurrent && Update position of starting nodes:
0176  fileMatrixCurrentNew.clear();
0177  //create fileMatrixCurrent for t<0 //11/01/17:
0178  if((t==fileMatrix.get(0)[9].intValue())/*<=0*/ && ONLYvrp == false)){
0179    //createFileMatrixCurrent();
0180    insertIncomingRequests(t, requestNr);
0181    createResultsRowsForFirstRequests();
0182    System.out.println("rechnung 1.1.2");
0183  }
0184  else if(ONLYvrp == true) createFileMatrixCurrent();
0185  //*************** Remove old requests and calc current location: *********
0186  alreadyDrivenDistanceIncremental=0;
0187  alreadyDrivenCapKmIncremental = 0;
0188  if(t>0 && checkIfRequestArrival(t) == true && ONLYvrp == false) removeOldRequestsAndDetermineStartingPosMEW(t);
0189  resultsRow[0] = (double) requestNr;
0190  resultsRow[i] = (double) t;
0191  //**************************************************************************
0192  // first vehicle routing:**************************************************************************
0193  // Do this for initial solution
0194  * Always if a new request arrives (without that request)
0195  *******************************************************************************/
0196  System.out.println("First vehicle routing: t: "+t);
0197  rechnung = 1;
0198  createMyDataArrays();
0199  //timePosition = new double [VVV+1][kkk];
0200  //System.out.println("fileMatrixCurrent.size() "+fileMatrixCurrent.size());
0201  previousW = new int [fileMatrixCurrent.size()][fileMatrixCurrent.size()]+[fileMatrixCurrent.size()]+[kkk];
0202  previousW = new int [fileMatrixCurrent.size()][fileMatrixCurrent.size()][fileMatrixCurrent.size()]+[kkk];
0203  //**************************************************************************/
0204  //System.out.println("rechnung 1.1.2");
0205  if((calculate == true /*28/04&& t>=0*/) /*25/05/17*/|| ONLYvrp == true)
0206    System.out.println("First vehicle routing:");
0207    Main.main();
0208  if(desktopPrintout==true && caseStudy == true)MyCanvas.updateCanvasCS(MySolution.w, 0);
0209  else if(desktopPrintout==true)MyCanvas.updateCanvas(MySolution.wtour);
0210  moveArrayContentToList(wtourListRem, MySolution.wtour);
0211  System.out.println("Costs for full routing at time "+t+: "+resultsRow[7]);
0212  if((calculate == true && t<0) resultsRow[7] = costs.get(costs.size()-1);
0213  else if((t<0) && costs.get(costs.size()-1) < 100000 && ONLYvrp == false)
0214    moveArrayContentToList(wtourListRem, MySolution.wtour);
0215    System.out.println("1 wtour list size: "+wtourListRem.size());
0216  startingPositionRem = Arrays.copyOf(startingPosition, startingPosition.length);
0217  timePositionRem = Arrays.copyOf(timePosition, timePosition.length);
0218  previousWRem = MySolution.deepCopyOf(previousW);
0219  createWtourListREMiD();
0220  startingTimeRem = Arrays.copyOf(startingTime, startingTime.length);
fileMatrixCurrentREM.clear();
for (int i = 0; i < fileMatrixCurrent.size(); i++) {
    fileMatrixCurrentREM.add(Arrays.copyOf(fileMatrixCurrent.get(i), fileMatrixCurrent.get(i).length));
}

fileMatrixCurrentNew.clear();
for (int i = 0; i < fileMatrixCurrent.size(); i++) {
    fileMatrixCurrentREM.add(Arrays.copyOf(fileMatrixCurrent.get(i), fileMatrixCurrent.get(i).length));
}

fileMatrixCurrentNew.clear();
if (checkIfRequestArrival(t) == true) calculate = insertIncomingRequests(t, requestNr);
if (/*28/04*/t >= 0 && calculate == true) /*&& costs.get(costs.size() - 1) < 100000*/ {
    createMyDataArrays();
    //timePosition = new double [VVV+1][kkk];
    /*TEST*/ //previousWRem = MySolution.deepCopyOf(previousW);
    //MyObjectiveFunction.printSolution(previousWRem);
    previousW = new int [fileMatrixCurrent.size()+kkk][fileMatrixCurrent.size()+kkk][fileMatrixCurrent.size()+1][kkk];
    //System.out.println("PREVIOUS W" + previousW.length);
    Main.V = MyData.demand.length;
    if (calculate == true) {
        System.out.println("Second vehicle routing:!");
        Main.main();
        if (desktopPrintout == true && caseStudy == true) MyCanvas.updateCanvasCS(MySolution.w, 0);
        else if (desktopPrintout == true) MyCanvas.updateCanvas(MySolution.w, 670);
    }
    System.out.println("rechnung = 2;" + System.currentTimeMillis() - startzeit);
    //resultsRow[16] = stdDev;
    requestNr++;
    results.add(resultsRow);
    createMissingResultsRows();
    //Trickkiste.printRoute();
}
if (calculate == true && ONLYvrp == false) {
    capacityControl(requestNr);
}
else System.out.println("No incoming requests at t=\"+t +\" or previous w as infeasible");
//System.out.println("rechnung 2.3");
//--MyDynamicPricing
resultsRow[2] = System.currentTimeMillis() - startzeit;
//--resultsRow[16] = stdDev;
requestNr++;
results.add(resultsRow);
createMissingResultsRows();
//Trickkiste.printRoute();
if (caseStudy == true) writeResultsGIS();
    //printResults();
    System.gc();
    MyDynamicPricing.discountPercentage+=.15;

    //set 2 loop end (This one calculates same things for every file)
    } /** File loop end**/
    //stdDev ++;
    MyDynamicPricing.priceRebate+=5;
    } //set1 loop end
    //MyDynamicPricing.gridNumber++;
    //MyDynamicPricing.gridSize = 100/MyDynamicPricing.gridNumber;
    } ///gridLoop end

    private static void createResultsRowsForFirstRequests() {
        for (int i=1; i<fileMatrix.size(); i+=2) {
            if ((fileMatrix.get(i)[9]==0 && fileMatrix.get(i+2)[9]==0) ^
                fileMatrix.get(i)[9]==fileMatrix.get(0)[9]){
                resultsRow[0] = (double) requestNr;
                resultsRow[1] = fileMatrix.get(i)[9];
                resultsRow[2] = 0.0;
                resultsRow[3] = fileMatrix.get(i+1)[0];
                resultsRow[4] = fileMatrix.get(i+1)[0];
                resultsRow[5] = norm(fileMatrix.get(i)[1], fileMatrix.get(i+1)[1], fileMatrix.get(i)[2], fileMatrix.get(i+1)[2]);
                resultsRow[6] = resultsRow[5] + norm(fileMatrix.get(0)[1], fileMatrix.get(0)[1], fileMatrix.get(0)[2], fileMatrix.get(0)[2]);
                for (int j=7; j<resultsRow.length; j++){
                    resultsRow[j] = 0.0;
                }
            }
            for (int j=0; j<resultsRow.length; j++){
                System.out.print(resultsRow[j]+"\t");
            }
            System.out.println();
            requestNr++;
            results.add(resultsRow);
        }

        resultsRow = new Double[17];
    }

    private static void createFileMatrixCurrent() {
        System.out.println("Create fileMatrixCurrent");
        for (int i=0; i<fileMatrix.size(); i++) {
            if (fileMatrix.get(i)[9]==t) fileMatrixCurrent.add(fileMatrix.get(i));
        }
    }

    private static void capacityControl(int requestNr) {
        boolean acceptAllFeasible = false;
        resultsRow[9] = resultsRow[8] - resultsRow[7]; //costs for inserting
        //MyDynamicPricing.simulateCompetitionRandom(resultsRow[3].intValue(), resultsRow[4].intValue());
        MyDynamicPricing.setCompetitionFromFile(resultsRow[3].intValue(), resultsRow[4].intValue());
        //MyDynamicPricing.simulateCompetitionRegional(resultsRow[3].intValue(), resultsRow[4].intValue());
        MyDynamicPricing.applyDP(resultsRow[9], resultsRow[3].intValue(), resultsRow[4].intValue());
    }
Row[4].intValue());
0341  // CONSOLE PRINTOUT:
0342  System.out.println("Request from " + resultsRow[3] + " to " + resultsRow[4] + " a t time: " + t);
0343  System.out.println("Costs: " + df.format(resultsRow[9]) + " Market price: " + df.format(resultsRow[14]) + " Proposed price: " + df.format(resultsRow[10]));
0344  if (resultsRow[9] < 0) resultsRow[9] = 0.0;
0346  resultsRow[11] = 0.0;
0347  resultsRow[12] = 0.0;
0348  resultsRow[13] = 0.0;
0349  restorePreviousSolution();
0350  System.out.println("Solution invalid, request rejected, previous solution restored");
0351  } else if (acceptAllFeasible == true || MyDynamicPricing.myPrice <= MyDynamicPricing.marketPrice) {
0352  resultsRow[11] = 1.0;
0353  resultsRow[12] = 1.0;
0354  resultsRow[13] = 1.0;
0355  System.out.println("Request accepted");
0356  alreadyDrivenDistance += alreadyDrivenDistanceIncremental;
0357  alreadyDrivenCapKm += alreadyDrivenCapKmIncremental;
0358  results2[8] = resultsRow[8];
0359  moveArayContentToList(wtourListRem, MySolution.wtour);
0360  previousWRem = MySolution.deepCopyOf(previousW);
0361  createWtourListREMID();
0362  startingPositionRem = Arrays.copyOf(startingPosition, startingPosition.length);
0363  }
0364  else {
0365  resultsRow[11] = 1.0;
0366  resultsRow[12] = 0.0;
0367  resultsRow[13] = 0.0;
0368  System.out.println("Request rejected, competitor wins request");
0369  restorePreviousSolution();
0370  }
0371  }
0372  }
0373  private static void printResults() {
0374  System.out.println();
0375  System.out.println("Overall results:");
0376  for (int i = 0; i < results.size(); i++) {
0377  if (results.get(i)[0] != 0 || results.get(i)[1] == -10) {
0378  for (int j = 0; j < results.get(i).length; j++) {
0379  if (results.get(i)[j] != null) System.out.print(df.format(results.get(i)[j]) + "	");
0380  }
0381  System.out.println();
0382  }
0383  }
0384  private static void restorePreviousSolution() {
0385  for (int i = 1; i < fileMatrixCurrent.size(); i++) {
0386  if (fileMatrixCurrent.get(i)[9] == t && fileMatrixCurrent.get(i)[0] <
1000) {
0402     //    fileMatrixCurrent.remove(i);
0403     // i--;
0404     }
0405     //
0406     fileMatrixCurrent.clear();
0407     for(int i=0; i<fileMatrixCurrentREM.size(); i++){
0408         fileMatrixCurrent.add(Arrays.copyOf(fileMatrixCurrentREM.get(i), fileMa-
0409         trixCurrentREM.get(i).length));
0410     }
0411     previousW = MySolution.deepCopyOf(previousWRem);
0412     timePosition = Arrays.copyOf(timePositionRem, timePositionRem.length);
0413     startingPosition = Arrays.copyOf(startingPositionRem, startingPositionRem
0414     .length);
0415     //System.
0416     //out.println("starting
0417     //positionsREM
0418     //3:")+startingPositionRem[0]+"+
0419     //t"+startingPositionRem[1]+"
0420     //startingPositionRem[2]);
0421     //checkSolution();
0422     //is
0423     //just
0424     //a
0425     //printout
0426     //System.out.println("File
0427     //set
0428     //for
time
0429     //t
0430     //"+t+" after restoring old
0431     //set");
0432     //
0433     //print out current set:
0434     //for(int i=0; i<
0435     //fileMatrixCurrent.size(); i++){
0436         System.out.print(df.format(fileMatrixCurrent.get(i)[j])+
0437         "t"};
0438     //System.out.println();
0439     // System.out.println();
0440     //
0441     //private static void updateStartingLocation(double x1, double y1, double x2,
0442     double y2, int row, int id, int t, int q, int k, int iToRemove) {
0443         double tan;
0444         double xNew;
0445         double yNew;
0446         double r;
0447         if (q==1 && t<10)
0448             r = (t/Main.v);
0449         else /*if (q==1) */
0450             r = (t - startingTimeRem[k]/Main.v);/*
0451         tan = (Math.sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1)) - r);
0452         xNew = ((r*x2)+(tan*x1))/(r+tan);   
0453         yNew = ((r*y2)+(tan*y1))/(r+tan);   
0454         if (x1==x2)
0455             xNew = (x1+x2)/2; 
0456         if (y1==y2)
0457             yNew = (y1+y2)/2; 
0458         if (x2>x1 && (xNew>x2 || xNew<x1))xNew = x2; 
0459         else if (x2<x1 && (xNew<x2 || xNew>x1))xNew = x2; 
0460         if (y2>y1 && (yNew>y2 || yNew<y1))yNew = y2;   
0461         else if (y2<y1 && (yNew<y2 || yNew>y1))yNew = y2;   
0462         // add already driven distance to total costs:
0463         double distanceToAdd = norm(x1, y1, xNew, yNew);
0464         System.out.println("Add already driven distance to total costs: "+distance
0465         eToAdd);
0466         //Main.calculateVehicleLoad();
0467         alreadyDrivenDistanceIncremental += distanceToAdd;
0468         alreadyDrivenCapKmIncremental += Main.load[iToRemove] * distanceToAdd;
0469         System.out.println("t: "+t "startingTimeRem/v: "+startingTimeRem[k] + 
0470         "timePos/v: "+timePositionRem[q][k]);
0471         System.out.println("x1: "+x1", y1: "+y1);
0472         System.out.println("x2: "+x2", y2: "+y2);
System.out.println("r: "+r", t: "+t);
System.out.println("xNew: "+xNew +" in row "+row+" of Matrix new with si-

ze "+fileMatrixCurrentNew.size());
System.out.println("yNew: "+yNew);
for(int i=0; i<fileMatrixCurrentNew.size(); i++){
    if(fileMatrixCurrentNew.get(i)[0]==id){
        fileMatrixCurrentNew.get(i)[1] = xNew;
        fileMatrixCurrentNew.get(i)[2] = yNew;
        System.out.println("Koordinaten angepasst in row"+i);
    }
}
if(x1<x2 && (xNew<x1 || xNew>x2)) System.err.println("ERROR IN SP g-
eration x");
else if (x1>x2 && (xNew>x1 || xNew<x2)) System.err.println("ERROR IN S-
P generation x");
if (y1<y2 && (yNew<y1 || yNew>y2)) System.err.println("ERROR IN SP g-
eration y");
else if (y1>y2 && (yNew>y1 || yNew<y2)) System.err.println("ERROR IN S-
P generation y");
}
private static void checkSolution() {
    System.out.println("Previous sol:");
    for(int k=0; k<kkk; k++){  
        for(int q=1; q<previousW[0].length; q++){  
            for(int i=0; i<previousW.length; i++){  
                for(int j=0; j<previousW[0].length; j++){  
                    if (previousW[i][j][q][k]==1) System.out.println("w["+i+""]["+j+""]["+q+"]["+k+"] \t");
                }
            }
        }
    }
    private static boolean checkIfRequestArrival(int t){
        boolean request = false;
        for(int i=0; i<fileMatrix.size(); i++){  
            if(fileMatrix.get(i)[9]==t){
                request = true;
                break I;
            }
        }
        return request;
    }
    static boolean insertIncomingRequests(int t, int requestNr) {
        boolean newRequest = false;
        //fuege Depot hinzu:
        if(fileMatrixCurrent.size()==0 && ONLYvrp ==false){
            for(int i=0; i/<5*/1; i++){  
                fileMatrixCurrent.add(fileMatrix.get(i));
            }
        } else if(fileMatrixCurrent.size()==0 && ONLYvrp ==true){
            createFileMatrixCurrent();
        } else{
            // check other lines:
            L: for(int i=1; i<fileMatrix.size(); i++){
if (fileMatrix.get(i)[9] == t && doesMatrixContainCustomer(fileMatrix.get(i)[0].intValue()) == false)
    
    fileMatrixCurrent.add(fileMatrix.get(i));

    // find partner ID if not 0:
    if (fileMatrix.get(i)[8] > 0) {
        fileMatrixCurrent.add(fileMatrix.get(i + 1));
    }

    newRequest = true;

    if (fileMatrix.get(i)[3] > 0 && t >= 0) {
        resultsRow[3] = fileMatrix.get(i)[0];
    }

    else if (fileMatrix.get(i)[3] < 0 && t >= 0) {
        resultsRow[4] = fileMatrix.get(i)[0];
    }

    fileMatrix.remove(i);
    fileMatrix.remove(i);

    // i:
    resultsRow[3] = fileMatrixCurrent.get(fileMatrixCurrent.size() - 2)[0];
    // j:
    resultsRow[4] = fileMatrixCurrent.get(fileMatrixCurrent.size() - 1)[1],
    fileMatrixCurrent.get(fileMatrixCurrent.size() - 1)[2]);

    resultsRow[5] = norm(fileMatrixCurrent.get(fileMatrixCurrent.size() - 2)[1],
    fileMatrixCurrent.get(fileMatrixCurrent.size() - 2)[2],
    fileMatrixCurrent.get(fileMatrixCurrent.size() - 1)[1],
    fileMatrixCurrent.get(fileMatrixCurrent.size() - 1)[2]);

    fileMatrixCurrent.get(fileMatrixCurrent.size() - 2)[2],
    fileMatrixCurrent.get(0)[1], fileMatrixCurrent.get(0)[2])
    + norm(fileMatrixCurrent.get(0)[1], fileMatrixCurrent.get(0)[2],
    fileMatrixCurrent.get(fileMatrixCurrent.size() - 1)[1],
    fileMatrixCurrent.get(fileMatrixCurrent.size() - 1)[2]);


    break L;

    System.out.println("or request nr: "+requestNr+" time t "+t+" after removing and inserting requests");

    for(int i=0; i<fileMatrixCurrent.size(); i++){
        for(int j=0; j<fileMatrix.get(0).length; j++){
            System.out.print(df.format(fileMatrix.get(i)[j])+	"t");
        }
        System.out.println();
    }

    System.out.println();

    System.out.println("FileMatrix for time t "+t+" after removing and inserting requests");

    for(int i=0; i<fileMatrix.size()/2; i++){
        for(int j=0; j<fileMatrix.get(0).length; j++){
            System.out.print(df.format(fileMatrix.get(i)[j])+	"t");
        }
        System.out.println();
    }

    return newRequest;
private static boolean doesMatrixContainCustomer(int c) {
    for(int i=0; i<fileMatrixCurrent.size(); i++) {
        if (fileMatrixCurrent.get(i)[0]==c) return true;
    }
    return false;
}

static void insertIncomingAll(int t) {
    for(int i=0; i<fileMatrix.size(); i++) {
        if (fileMatrix.get(i)[9]==t) {
            Double[] row = new Double[fileMatrix.get(i).length];
            for(int j=0; i<fileMatrix.get(i).length; j++) {
                row[j] = (double) fileMatrix.get(i)[j];
            }
            fileMatrixCurrent.add(row);
        }
    }
}

private static void removeOldRequestsAndDetermineStartingPosMEW(int t) {
    System.out.println("removeOldRequestsAndDetermineStartingPosMEW");
    alreadyDrivenDistanceIncremental = 0;
    System.out.println("start: alreadyDrivenDistanceIncremental = 0");
    double alreadyDrivenDistanceIncremental = 0;
    System.out.println("alreadyDrivenDistanceIncremental = 0");
    //return array pos to starting ID:
    for(int k = 0; k<previousW.length; k++) {
        startingPosition[k] = intValue(fileMatrixCurrent.get(startingPositionRem)[k][0]);
    }
    System.out.println("alreadyDrivenDistanceIncremental = 0");
    System.out.println("alreadyDrivenDistanceIncremental = 0");
    // Determine starting positions:
    for(int k=0; k<previousW[0].length; k++) {
        previousQhasBeenDeleted = true;
        for(int q=1; q<previousW[0].length; q++) {
            for(int i=0; i<Math.min(previousW.length, fileMatrixCurrent.size()); i++) {
                if (previousW[i][j][q][k] == 1 && t == timePosition[q][k] && timePosition[q][k] != 0 && i != 0 && i != (1000+k)) {
                    System.out.println("Entferne "+i"(ID "+intValue(fileMatrixCurrent.get(j)[0])+") als neue starting pos");
                }
                else {
                    startingPositionID[k] = 0;
                    System.out.println("Entferne "+i"(ID "+intValue(fileMatrixCurrent.get(j)[0])+") als neu starting pos");
                }
            }
        }
    }
    alreadyDrivenDistanceIncremental += X * MyData.distance[i][j];
    alreadyDrivenCapKmIncremental += MyData.distance[i][j];
    System.out.println("Nod: "+ i ", ID "+intValue(fileMatrixCurrentNew.get(i)[0]);
}
get(m)[0] + " wird aus matrix entfernt";
if (fileMatrixCurrentNew.get(m)[0] < 1000) {
    fileMatrixRoutes.add(fileMatrixCurrentNew.get(m));
    fileMatrixRoutes.get(fileMatrixRoutes.size()-1)[9] = (double) k;
}
fileMatrixRoutes.get(fileMatrixRoutes.size()-1)[4] = 0.0;
}
fileMatrixCurrentNew.remove(m);
System.out.println("remove "+m);
}
}
System.out.println("MyData.service.length: "+MyData.Service.length);
/**determine starting time (trifft nur bei diesen kunden zu!**/
if (previousW[i][j][q][k]==1 && timePosition[q][k]==t && timePosition[q][k]!=0){
    // position exakt bei ankunft
    startingTime[k] = t + MyData.Service[j];
}
else if (previousW[i][j][q][k]==1 && timePosition[q][k]<t && (timePosition[q][k]+/*MyData.Service[j]*/fileMatrixCurrent.get(j)[6])>t && timePosition[q][k]!=0){
    // position waehrend service
    startingTime[k] = timePosition[q][k]+/*MyData.Service[j]*/fileMatrixCurrent.get(j)[6];
}
else if (previousW[i][j][q][k]==1 && t<timePosition[q][k] && timePosition[q][k]!=0 && fileMatrixCurrent.get(j)[0]!=(1000+k) || fileMatrixCurrent.get(i)[0]==(1000+k) || previousQhasBeenDeleted == true )){
    // change starting 0 and 1000+k:
    int ID1=intValue(fileMatrixCurrent.get(i)[0]);
    int ID2=intValue(fileMatrixCurrent.get(j)[0]);
    boolean contains = false;
    for(int line=0; line<fileMatrixCurrentNew.size(); line++){
        if(fileMatrixCurrentNew.get(line)[0]==(1000+k)){
            contains = true;
            fileMatrixCurrentNew.remove(line); //hier wird 1000+k geloescht, falls es bereits existiert
        }
    }
    System.out.println("Add dummy location "+(1000+k));
    Double dummy[]= new Double [fileMatrixCurrent.get(i).length];
    dummy[0]=(double) 1000+k;
    for(int l=1; l<9; l++){
        dummy[l] = fileMatrixCurrent.get(i)[l];
    }
    /**
    * Change vehicle load to counted load at old position i:
    **/
    for(int n=0; n<Main.load.length; n++) {
        System.out.println("load["+n+"]": "+ Main.load[n]);
    }
    System.out.println("Set dummy load as: load["+i+"/"+fileMatrixCurrent.get(i)[0]+": "+ Main.load[i]);
    dummy[3] = (double) Main.load[j];
    */
}
if(dummy[7] >= 1000) dummy[7] = 0.0;
dummy[9] = (double) t;
fileMatrixCurrentNew.add(dummy);
fileMatrixCurrent.add(dummy);

// Hier wird der alte sender geloescht und der empfaenger bekomm
t 1000+k als sender zugeteilt
for(int line=0; line<fileMatrixCurrentNew.size(); line++){
    if(fileMatrixCurrentNew.get(line)[0] == ID1 && /*ID1 != (1000 + k)*/ ID1 < 1000 /*teste id1<1000 um zu vermeiden, dass andere 1000s geloescht werden*/ && ID1 != 0)
    {
        System.out.println("ID1 = " +ID1);
        //finde receiptent falls vorhande:
        for(int r = 0; r < fileMatrixCurrentNew.size(); r++){
            if(fileMatrixCurrentNew.get(r)[7] == ID1 && fileMatrixCurrentNew.get(r)[0] < 1000)
            {
                fileMatrixCurrentNew.get(r)[7] = (double)(1000+k);
            }
        }
        System.out.println("Remove old customer"+ID1);
    }
    if(fileMatrixCurrentNew.get(line)[0] < 1000) {
        fileMatrixCurrentNew.get(line)[7] = (double)(1000+k);
    }
    System.out.println("Update DUMMY starting Location: i: " + ID1 + " j: " +ID2);
    System.out.println("ID 1 test: "+ID1+ " == " +fileMatrixCurrent.get(i)[0]);
    System.out.println("ID 2 test: "+ID2+ " == " +fileMatrixCurrent.get(j)[0]);
    updateStartingLocation(fileMatrixCurrent.get(i)[1], fileMatrixCurrent.get(i)[2], fileMatrixCurrent.get(j)[1], fileMatrixCurrent.get(j)[2], fileMatrixCurrent.size()-1, (1000+k), t, q, k, i);
    startingPositionID[k]=1000+k;
    System.out.println("starting pos of k "+startingPositionID[k]);
    System.out.println();
    startingTime[k]=t;
    previousQhasBeenDeleted = false;
}
System.out.println("filematrixcurrent.size() before: "+fileMatrixCurrent.size());
fileMatrixCurrent.clear();
fileMatrixCurrent.addAll(fileMatrixCurrentNew);
System.out.println("filematrixcurrent.size() after: "+fileMatrixCurrent.size());

// reset 1000+k starting loads:
for(int i= 0; i<fileMatrixCurrent.size(); i++){
    if(fileMatrixCurrent.get(i)[0]>=1000) {
        System.out.println("filematrixcurrent.size() after: "+fileMatrixCurrent.size());
        fileMatrixCurrent.clear();
        fileMatrixCurrent.addAll(fileMatrixCurrentNew);
        System.out.println("filematrixcurrent.size() after: "+fileMatrixCurrent.size());
    }
}
```
fileMatrixCurrent.get(i)[3] = 0.0;
for(int j= 0; j<fileMatrixCurrent.size(); j++){
    if(fileMatrixCurrent.get(j)[7].intValue() == fileMatrixCurrent.get(i)[0].intValue()) {
        fileMatrixCurrent.get(i)[3] += fileMatrixCurrent.get(j)[3] * -1;
    }
}

//set staringpos as array position:
for(int k = 0; k<kkk; k++){
    k: for(int i= 0; i<fileMatrixCurrentNew.size(); i++){
        MyData.distance[i][j] = MyData.distance[j][i] = norm(fileMatrixCurrent.get(i)[1], fileMatrixCurrent.get(i)[2], fileMatrixCurrent.get(j)[1], fileMatrixCurrent.get(j)[2]);
    }
}
System.out.println("File set for time t "+t+" after removing old requests ");
//print out current set:
for(int i= 0; i<fileMatrixCurrent.size(); i++){
    for(int j= 0; j< fileMatrixCurrent.get(0).length; j++){
        System.out.print(df.format(fileMatrixCurrent.get(i)[j]) + "	");
    }
    System.out.println();
}
System.out.println("NEW STARTING POSITIONS: ");
for(int i=0; i<startingPosition.length; i++){
    System.out.print(startingPosition[i]+", ");
}
System.out.println();
static void createMyDataArrays() {
    MyData.distance = new double[fileMatrixCurrent.size()][fileMatrixCurrent.size()];
    for(int i = 0; i < fileMatrixCurrent.size(); i++ ){
        System.out.println(fileMatrixCurrent.get(i)[0]);
        //for( int j = i+1; j < fileMatrixCurrent.size(); j++ ){
        if(i==j) MyData.distance[i][j]=0.0;
        else{
            MyData.distance[i][j] = MyData.distance[j][i] = norm(fileMatrixCurrent.get(i)[1], fileMatrixCurrent.get(i)[2], fileMatrixCurrent.get(j)[1], fileMatrixCurrent.get(j)[2]);
        }
    }
    System.out.println();
}
MyData.demand = new double[fileMatrixCurrent.size()][fileMatrixCurrent.size()];
for( int i = 0; i < fileMatrixCurrent.size(); i++ ){
    //System.out.println(fileMatrixCurrent.get(i)[0]);
    System.out.println(
        System.out.println(MyData.distance[i][j]+"\t");
    }
    System.out.println();
    }
    System.out.println();
    }
```
```java
0812     }
0813     else{
0814         MyData.demand[j][i] = 0.0;
0815     }  
0816     }
0817     else{
0818         MyData.demand[j][i] = 0.0;
0819     }
0820 }
0821     //System.out.print("TWstart= {\t");
0822     MyData.TWstart= new double[fileMatrixCurrent.size()];
0823     for( int i = 0; i < fileMatrixCurrent.size(); i++){
0824         //System.out.print (fileMatrixCurrent.get(i)[4]+", ");
0825         MyData.TWstart[i]= fileMatrixCurrent.get(i)[4];
0826     }
0827     //System.out.println();
0828     //System.out.print("TWend= {\t");
0829     MyData.TWend= new double[fileMatrixCurrent.size()];
0830     for( int i = 0; i < fileMatrixCurrent.size(); i++){
0831         //System.out.print (fileMatrixCurrent.get(i)[5]+", ");
0832         MyData.TWend[i]= fileMatrixCurrent.get(i)[5];
0833     }
0834     //System.out.println();
0835     //System.out.print("Service= {\t");
0836     MyData.Service= new double[fileMatrixCurrent.size()];
0837     for( int i = 0; i < fileMatrixCurrent.size(); i++){
0838         //System.out.print (fileMatrixCurrent.get(i)[6]+", ");
0839         MyData.Service[i]= fileMatrixCurrent.get(i)[6];
0840     }
0841     System.out.println();
0842     
0843     public static void readFile() throws FileNotFoundException, IOException{
0844         BufferedReader br = new BufferedReader(new FileReader(directoryAndFilename));
0845         try {
0846             MyDynamicPricing.marketPriceList.clear();
0847             @SuppressWarnings("resource")
0848             Scanner scanner = new Scanner(new File(directoryAndFilename));
0849             System.out.println();
0850             int line=0;
0851             System.out.println();
0852             kkk = scanner.nextInt();
0853             VVV = scanner.nextInt();
0854             vvv = scanner.nextInt();
0855             //if(scanner.hasNextInt()) {
0856                 //MyData.reduceX = scanner.nextDouble();
0857                 //MyData.reduceY = scanner.nextDouble();
0858                 //}
0859                 //kkk=kkk;
0860                 System.out.println();
0861                 int line=0;
0862                 System.out.println();
0863                 System.out.println();
0864                 while(scanner.hasNextLine()){
0865                 Double[] row = new Double[10];
0866                 //read first element
0867                 scanner.nextLine();
0868                 //for(int i=0; i<VVV; i++){
0869                 //while(scanner.hasNextLine()){
0870                 for(int j=0; j<10; j++){
```
```
for (int l = 0; l <= 98; l++) {
    Double[] row = new Double[10];
    for (int j = 0; j < 10; j++) {
        row[j] = (double) scanner.nextInt();
        // fileMatrix[line][j] = scanner.nextInt();
        System.out.print(row[j] + " ");
    }
    System.out.println();
    fileMatrix.add(row);
    // Revenues extra column:
    if (scanner.hasNextInt()) {
        MyDynamicPricing.marketPriceList.add((double) scanner.nextInt());
        // System.out.print(MyDynamicPricing.marketPriceList.get(line) + " ");
    }
    // Revenues end
    line++;
}
if (scanner.hasNextLine()) scanner.nextLine();
}
```
public static void moveArrayContentToList(List<Integer> list, int[] array) {
    list.clear();
    for (int i = 0; i < array.length; i++) {
        list.add(array[i]);
    }
}

static double norm(double x1, double y1, double x2, double y2) {
    double xDiff = x2 - x1;
    double yDiff = y2 - y1;
    return Math.sqrt(xDiff*xDiff + yDiff*yDiff);
}

private static int intValue(Double double1) {
    return double1.intValue();
}

static void createMissingResultsRows() {
    if (t == fileMatrix.get(0)[9]) {
        for (int i = 1; i < fileMatrixCurrent.size() - 1; i++) {
            if (fileMatrixCurrent.get(i)[9] == t) {
                Double[] resultsRowX = new Double[17];
                resultsRowX[0] = (double) requestNr;
                resultsRowX[1] = (double) t;
                resultsRowX[2] = (double) 0;
                resultsRowX[3] = (double) fileMatrixCurrent.get(i)[0];
                resultsRowX[4] = (double) fileMatrixCurrent.get(i)[0];
                resultsRowX[5] = norm(fileMatrixCurrent.get(i)[0], fileMatrixCurrent.get(i)[1], fileMatrixCurrent.get(i)[2], fileMatrixCurrent.get(i)[3], fileMatrixCurrent.get(i)[4], fileMatrixCurrent.get(i)[5]);
                resultsRowX[6] = resultsRowX[5] + norm(fileMatrixCurrent.get(0)[1], fileMatrixCurrent.get(0)[2], fileMatrixCurrent.get(i)[1], fileMatrixCurrent.get(i)[2]);
                for (int j = 7; j < resultsRowX.length; j++) {
                    resultsRowX[j] = 0.0;
                }
                results.add(resultsRowX);
                requestNr++;
            }
        }
    }
}

public static void writeExcel() throws IOException, WriteException, BiffException {
    // file manipulation:
    for (int i = 0; i < results.size(); i++) {
        if (results.get(i)[9] < 0) results.get(i)[14] = 0.0;
        else results.get(i)[14] = results.get(i)[9];
    }
    // results[i][11] = (results[i][6] - results[i][10]) * results[i][9];
    // create rows for time = 0

    Workbook existingWorkbook = Workbook.getWorkbook(new File(directory+"02 V RPPD test files\V04 MYRM test files\ExperimentsResultsRM.xls"));
    WritableWorkbook workbookCopy = Workbook.createWorkbook(new File(directory+"02 VRPPD test files\V04 MYRM test files\ExperimentsResultsRM.xls"), existingWorkbook);
    WritableSheet sheetToEdit = workbookCopy.getSheet("Sheet1");
WritableCell cell;
String file = null;
int nrOfRows = sheetToEdit.getRows();
int startColumn;
WritableFont numberFont = new WritableFont(WritableFont.TIMES);
WritableCellFormat numberFormat = new WritableCellFormat(numberFont, new NumberFormat("#0.0");
WritableCellFormat stringFormat = new WritableCellFormat(numberFont);
Label l1 = new Label(0, nrOfRows, filename + "FC\"/*+MyDynamicPricing.gridNumber+"_"+stdDev+"_"+MyDynamicPricing.discountPercentage*/", stringFormat);
cell = l1;
sheetToEdit.addCell(cell);

NumberCell numberCell;
for (int line=0; line<results.size(); line++){  //System.out.println("results line: "+results.get(line)[0]+ " "+results .get(line)[1]);
  for(int c=0; c<results.get(line).length; c++){
    if(results.get(line)[0] != 0 || results.get(line)[1] == -10) //funktioniert nicht
      //System.out.print(results.get(line)[c]+", ");
    if(results.get(line)[c] == null) numberCell= new Number(c, nrOfRows +1+line, 0.0, numberFormat);
    else numberCell = new Number(c, nrOfRows+1+line, results.get(line)[c], numberFormat);
    cell = numberCell;
    sheetToEdit.addCell(cell);
  }
  //System.out.println("I AM WRITING EXCEL 1");
  workbookCopy.write();
  workbookCopy.close();
  existingWorkbook.close();
}

public static void writeExcel2() throws IOException, WriteException, BiffException {
  Workbook existingWorkbook = Workbook.getWorkbook(new File(directory+"02 VRPPD test files\ExperimentsResultsRM.xls"));
  WritableWorkbook workbookCopy = Workbook.createWorkbook(new File(directory+"02 VRPPD test files\ExperimentsResultsRM.xls"), existingWorkbook);  
  WritableSheet sheet2 = workbookCopy.getSheet("Sheet2");
  WritableCell cell2;
  int nrOfRows = sheet2.getRows();
  //startColumn;
  WritableFont numberFont = new WritableFont(WritableFont.TIMES);
  WritableCellFormat numberFormat = new WritableCellFormat(numberFont, new NumberFormat("#0.0");
  WritableCellFormat stringFormat = new WritableCellFormat(numberFont);
  Label l2 = new Label(0, nrOfRows, filename + "FC\"/*+MyDynamicPricing.gridNumber+"_"+stdDev+"_"+MyDynamicPricing.discountPercentage*/", stringFormat);
  cell2 = l2;
  sheet2.addCell(cell2);

  double[] results2 = new double[11];
  if(ONLYvrp==false){
    for(int i=0; i<results.size(); i++){
      if(results.get(i)[7] >0) results2[0]++;
    }
  }
  for(int i=0; i<results.size(); i++){
    if(results.get(i)[7] >0) results2[0]++;
  }
  / number of requests
  results2[2]= results.get(i)[2];  // to tal runtime
if(results.get(i)[13] !=null) results2[3] += (results.get(i)[5] * results.get(i)[13]); // total DD
if(results.get(i)[13] !=null) results2[4] += (results.get(i)[6] * results.get(i)[13]); // total RTD
if(results.get(i)[11] !=null) results2[5] += results.get(i)[11]; // valid solutions
if(results.get(i)[12] !=null) results2[6] += results.get(i)[12]; // total won auctions
if(results.get(i)[12] !=null && results.get(i)[13] >0) results2[7] +=1; // costs for inserted requests
if(results.get(i)[13] !=null) results2[8] += alreadyDrivenDistance/*results.get(i)[9] * results.get(i)[13]*/; // costs for inserted requests
if(results.get(i)[15] !=null) results2[10] += (results.get(i)[10] * results.get(i)[15]); // Profits
}
}

results2[8] += Main.fixedCostsWeek + alreadyDrivenDistance*Main.variablesNrOfRows;
System.out.println("Already driven CapKm: "+alreadyDrivenCapKm);
results2[12] += alreadyDrivenDistance; // total distance
results2[11] = ((Main.calculateCapKmFromCurrPosition() + alreadyDrivenKm)/results2[12])/Main.C; // average of accepted requests

results2[0] = Main.iterations; // number of requests
results2[1] = fileMatrix.size();
results2[2] = results.get(0)[2]; // total runtime
results2[3] = 0; // total DD
results2[4] = 0; // total RTD
/results2[5] = 0; // total valid solutions
/results2[6] = 0; // total won auctions
/results2[7] = 0; // number of accepted requests
/results2[8] += alreadyDrivenDistance; // costs for inserted requests
results2[9] = 0; // Revenues
results2[10] = 0; // Profits
for(int h=0; h<MoveManager.length(); h++) {
  results2[h+11] = MoveManager[0][h];
}

Label 13 = new Label(1+results2.length, nrOfRows, wToString(previousW), stringFormat);
cell2 = 13;
newCell.addCell(cell2);
}
// opdistribution:
for(int h=0; h<MoveManager.size(); h++) {
  results2[h+13] = MoveManager[0][h];
}

Number numberCell;
for(int c=0; c<results2.length; c++){
  System.out.print(results2[c]+"\t");
  numberCell = new Number(l+c, nrOfRows, results2[c], numberFormat);
cell2 = numberCell;
 sheet2.addCell(cell2);
}
}
workbookCopy.write();
public static void writeResultsGIS() {
try {
    fw = new FileWriter("C:\\Users\\21576959\\Workspace\\02 VRPPD test file s\\07 GIS results\\"+filename+\\_results.txt");
    bw = new BufferedWriter(fw);
    bw.write("Latitude " + "\t");
    bw.write("Longitude " + "\t");
    bw.write("ID" + "\t");
    bw.write("Loadsize" + "\t");
    bw.newLine();
    System.out.println();
    fileMatrixRoutes.add(0, fileMatrixCurrent.get(0));
    for(int k=0; k<kkk; k++){
        for(int j=0; j<previousW[0].length; j++){
            if(previousW[i][j][q][k]==1) {
                System.out.println("w[" + i + "]","j","q","k] = 1");
                fileMatrixRoutes.add(fileMatrixCurrent.get(j));
                fileMatrixRoutes.get(fileMatrixRoutes.size()-1)[9] = (double) k;
            }
        }
    }
    // remove duplicates
    for(int i=0; i<fileMatrixRoutes.size()-1; i++) {
        if(fileMatrixRoutes.get(i)[0] == fileMatrixRoutes.get(i+1)[0] 
            && fileMatrixRoutes.get(i)[1] == fileMatrixRoutes.get(i+1)[1] 
            && fileMatrixRoutes.get(i)[2] == fileMatrixRoutes.get(i+1)[2]) {
            fileMatrixRoutes.remove(i+1);
            i--;
        }
    }
    // sort by vehicle and print
    for(int k=0; k<kkk; k++){
        double x = (fileMatrixRoutes.get(0)[1]/111) + MyData.reduceX;
        double y = ((fileMatrixRoutes.get(0)[2]/94.92) - MyData.reduceY)*-1;
        bw.write(x + "\t");
        bw.newLine();
    }
    bw.newLine();
    for(int i=1; i<fileMatrixRoutes.size(); i++) {
        if(fileMatrixRoutes.get(i)[9] == (double)k) {
            x = (fileMatrixRoutes.get(i)[1]/111) + MyData.reduceX;
            y = ((fileMatrixRoutes.get(i)[2]/94.92) - MyData.reduceY)*-1;
            bw.newLine();
            bw.newLine();
    bw.newLine();
    bw.newLine();
}
    System.out.println("Results GIS printer");
    bw.close();
}

} catch (java.io.IOException ex) {
    System.out.println("IO Error: " + ex);
}

static boolean doesListcolumnContain(List<Double[]> list, int column, int item){
    for(int i=0; i<list.size(); i++){
        //System.out.println(list.get(i)[column] + " == ? " + item);
        if(list.get(i)[column] == item) return true;
    }
    return false;
}

static String wToString(int[][][][] w){
    String solutionString;
    StringBuilder builder = new StringBuilder();
    for(int k=0; k<Main.k; k++){
        for(int q=0; q<w[0][0].length; q++){
            for(int j=0; j<w.length; j++){
                if(w[i][j][q][k]==1){
                    builder.append(j);
                    builder.append(",");
                }
            }
        }
    }
    solutionString = builder.toString();
    return solutionString;
}

static void createWtourListREMiD(){
    wtourListRemID.clear();
    for(int i=0; i<wtourListRem.size(); i++){
        if(doesListcolumnContain(fileMatrixCurrent, 0, fileMatrixCurrent.get(wtourListRem.get(i))[0].intValue())){
            wtourListRemID.add(fileMatrixCurrent.get(wtourListRem.get(i))[0].intValue());
        }
    }
}

Java2html
E.2. Main class

```java
import java.text.DecimalFormat;
import java.util.Arrays;
import java.io.*;

import jxl.Workbook;
import jxl.read.biff.BiffException;
import jxl.write.Label;
import jxl.write.WritableCell;
import jxl.write.WritableSheet;
import jxl.write.WritableWorkbook;
import jxl.write.WriteException;

public class Main {
    public static boolean RMsystem = true;
    public static int k = MyRMcontrol.kkk;
    public static int C = /*MyRMcontrol.VVV*/26;
    public static double v = .333;
    public static double variableCostsKM = /*1.11527917*/.85;
    public static double fixedCostsWeek = /*6786.687609*/5920.22;
    static int [] load;

    public static double [] TWStart /*;*/ = MyData.TWstart ;
    public static double [] TWEnd /*;*/ = MyData.TWend;
    public static double [] service /*;*/ = MyData.Service;
    public static int [] startingPositionK /*= new int[3];*/ = Arrays.copyOf(MyRMcontrol[startingPosition, MyRMcontrol[startingPosition.length]);
    public static int iterations =150;
    public boolean request = false;
    public static String[] excel= new String[8];
    public static int V;
    public static int fromFileNr =1;
    public static int toFileNr =1;
    // measure computation time:
    public static double measureConstraints;
    public static double measureOperators;
    public static int[] operatorPerformance = new int[MyMoveManager.opDistribution.length];
    public static int[] operatorUtilisation = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    public static long startzeit;
    public static String fileDescription ="";

    //public static void main (String args[]) throws /*WriteException,*/ IOException, NullPointerException {
    public static void main() throws WriteException, IOException, NullPointerException {
        if(MyRMcontrol.ONLYvrp==false && MyRMcontrol.fileMatrixCurrent.size()<30 )iterations = /*250 20*//*100;*/**150; //changed 7/11/17
        else if(MyRMcontrol.ONLYvrp==false && MyRMcontrol.fileMatrixCurrent.size ()<50)iterations = /*100 12*//*50;*/**100;
        else if(MyRMcontrol.ONLYvrp==false && MyRMcontrol.fileMatrixCurrent.size ()>=50)iterations = /*70 10*//*30;*/**50;

        String filePath;
        String letter="c";
        String plus="";
        String capLetter="N";
    }
```
DecimalFormat df = new DecimalFormat("#.00");
startzeit = System.currentTimeMillis();
// Initialize our objects

//double[][] customers = new double[V][V]; //[20][2];

//Read in data set:
try {
    String filename = "C:\Users\21576959\Workspace\02 VRPPD test files\01 Created files\Instance"+fileNr+".txt";//02 VRPPD test files\02 SINTEF_pdp_100\lc101_N1.txt"; // 01 Created files\Instance"+fileNr+".txt"; //Datei(en) auswählen.
    String filename = "C:\\Users\\21576959\\Workspace\\02 VRPPD test files\\04 MyRM test files\\lc101 RM2format.txt"; //02 SINTEF_pdp_100"+filepath+".txt";
    MyData data = new MyData(filename);
    /**startingPositionK[0]=12;
    startingPositionK[1]=0;
    startingPositionK[2]=2;**/
    if(RMsystem == true) startingPositionK=Arrays.copyOf(MyRMcontrol.startingPosition, MyRMcontrol.startingPosition.length);
    V=MyData.distance[0].length;//*/MyRMcontrol.fileMatrixCurrent.size()/*;
    TWStart = MyData.TWstart;
    TWEnd = MyData.T Wend;
    service = MyData.Service;
    double[][] customers = new double[V][V];
    //double[][] customers = new double[V][V]; //[20][2];

    for( int i = 0; i < V; i++ ){
        for( int j = 0; j < V; j++ ){
            customers[i][j] = MyData.distance[i][j];
        }
    }
    //Trickkiste.printRoute();

    ObjectiveFunction objFunc = new MyObjectiveFunction( customers );
    Solution initialSolution = new MySolution( customers );
    MoveManager moveManager = new MyMoveManager();
    Tabulist tabulist = new /*ComplexTabulist/**/SimpleTabulist( 7 ); // In OpenTS package
    // MyCanvas.main2();
    // Create Tabu Search object
    TabuSearch tabuSearch = new SingleThreadedTabuSearch( /**/MultiThreadedTabuSearch( initialSolution, moveManager, objFunc, tabulist, new BestEverAspirationCriteria(), // In OpenTS package false ); // maximizing = yes/no; false means minimizing
tabuSearch.addTabuSearchListener( new MyTabuList() );
    // Start solving
tabuSearch.setIterationsToGo(new MyTabuList());
    // Start solving
tabuSearch.setIterationsToGo( iterations );
    tabuSearch.startSolving();
}
113
114      // Show solution
115      long runtime = System.currentTimeMillis() - startzeit;
116      System.out.println();
117      System.out.println("File: " + MyRMcontrol.filename + " running time: " + run
118      time + " ms");
119      System.out.println("Time consumption in % constraints = " + df.format(m
119      easureConstraints/runtime*100));
120      System.out.println("Time consumption in % operators = " + df.format(needle
120      measuresOperators/runtime*100));
121      System.out.println("Operator Distribution: ");
122      for (int i = 0; i < MyMoveManager.opDistribution.length; i++) {
123          System.out.print(" + MyMoveManager.opDistribution[i];
124      }
125      System.out.println();
126      System.out.println("Operator Performance:");
127      for (int i = 0; i < operatorPerformance.length; i++) {
128          System.out.println(" + operatorPerformance[i]);
129      }
130      System.out.println();
131      System.out.println("Operator Utilisation:");
132      for (int i = 0; i < operatorUtilisation.length; i++) {
133          System.out.println(" + operatorUtilisation[i]);
134      }
135      System.out.println();
136      System.out.println("Operator Perf / Distr:");
137      for (int i = 0; i < operatorPerformance.length; i++) {
138          if (operatorUtilisation[i] > 0 && operatorPerformance[i] > 0) System.out.pr
139              int(" + df.format(operatorPerformance[i]/operatorUtilisation[i]));
140          else System.out.println(" 0");
141      }
142      System.out.println();
143      System.out.println("Time: " + MyRMcontrol.t);
144      MySolution best = (MySolution) tabuSearch.getBestSolution();
145      System.out.println("Objective Value: " + best.getObjectiveValue()[0]);
146      System.out.println("Delay: " + best.getObjectiveValue()[1]);
147      System.out.println("Waiting time: " + best.getObjectiveValue()[2]);
148      System.out.println("- fixed costs: " + fixedCostsWeek);
149      MyRMcontrol.results2[12] = best.getObjectiveValue()[3]; // total dr
150      ive distance
151      System.out.println("Starting positions: " + MyRMcontrol.startingPosi
152      tion[0] + ",\" + MyRMcontrol.startingPosition[1] + ",\" + MyRMcontrol.startingPosi
153      tion[2] ");
154      int w[][][][] = /*MySolution.w*/ best.w;
155      int wtour[] = /*MySolution.wtour*/ best.wtour;
156      //print out decision variable
157      System.out.println("Sequence: [ ");
158      for (int b = 0; b < wtour.length - 1; b++) System.out.println(" + wtour[b]+ ");
159      System.out.println("0 ]");
160      int qq = 1;
161      int kk = 0;
162      double dist = 0;
163      double time = 0;
164      double startTime = -0;*/ MyRMcontrol.startTime[kk]; //MyData.Service[Main.s
165      tartingPositionK[kk]]")
166      double waitingTime = 0;
167      double SumServiceTime = 0;
168      //printout
169      System.out.println();
170      System.out.println("w[] seq.: [ 0 ");
171      for (int k = 0; k < Main.k; k++) {
172          for (int q = 0; q < /*Main.V*/ w[0][0].length; q++) {
173              for (int i = 0; i < /*Main.V*/ w.length; i++) {
for (int j = 0; j < /*Main.V*/w.length; j++) {
    if (w[i][j][q][k] == 1 && j != 0) {
        System.out.print(j + " ");
    } else if (w[i][j][q][k] == 1 && MyObjectiveFunction.doesArrayContain
        (MyRMcontrol.startingPosition, j)) {
        System.out.print(j + " ");
    }
    //
    //System.out.println("w[" +wtour[b]+ "]["+wtour[b+1]+"]["+qq+"]["+kk+"]
        \t");
    if (RMsystem == true) {
        MyRMcontrol.previousW[wtour[b]][wtour[b+1]][qq][kk] = 1;
    }
    dist += MyData.distance[wtour[b]][wtour[b+1]];
    time += (double) (MyData.distance[wtour[b]][wtour[b+1]])/v;
    if (time < MyData.TWstart[wtour[b+1]]){
        waitingTime += (MyData.TWstart[wtour[b+1]] - time);
        time = MyData.TWstart[wtour[b+1]];
    }
    //System.out.println("q: "+qq+" k: "+kk);
    if (RMsystem == true) MyRMcontrol.timePosition[qq][kk] = time;
    System.out.print("\t"+df.format(MyData.distance[wtour[b]][wtour[b+1]])); //no waiting times respected
    System.out.print("\t"+MyData.TWstart[wtour[b+1]]);
    System.out.print("\t"+df.format(time));
    System.out.print("\t" + MyData.TWend[wtour[b+1]]);
    System.out.print("\t" + MyData.Service[wtour[b+1]]);
    time += MyData.Service[wtour[b+1]];
    SumServiceTime += MyData.Service[wtour[b+1]];
    
    for (int i2 = 0; i2 < MyData.distance.length; i2++){
        if (MyData.demand[i2][wtour[b+1]] != 0) System.out.print("\t"+MyData.
            demand[i2][wtour[b+1]]+" units from p"+ i2);
    }
    
    else /*if(/*MyRMcontrol.startingPosition[k] == wtour[b+1] && (wtour[b]
        != wtour[b+1]) == true)="/"
        System.out.print("w[" +wtour[b]+ "]["+qq+"]["+kk+"] \t");
    //System.out.println("w[" +MyRMcontrol.fileMatrixCurrent.get(wtour[b])
    */
}
if (RMsystem = true) MyRMcontrol.previousW[wtour[b]][0][qq][kk] = 1;

dist+=MyData.distance[wtour[b]][0];
time+=(double)(MyData.distance[wtour[b]][0]) / v;

if (time<MyData.TWstart[0]){ 
waitingTime+= (MyData.TWstart[0]-time);
time=MyData.TWstart[0];
}
MyRMcontrol.timePosition[qq][kk]=time;

System.out.println(" \t"+df.format(MyData.distance[wtour[b]][0]));
//no waiting times respected
System.out.println(" \t"+MyData.TWstart[0]);
System.out.println(" \t"+df.format(time));
System.out.println(" \t" + + MyData.TWend[0]);
System.out.println(" \t" + + MyData.Service[0]);

time+= MyData.Service[0];
SumServiceTime+=MyData.Service[0];

for (int i2=0; i2<MyData.distance.length; i2++){
if (MyData.demand[i2][0]!=0)System.out.print(" \t"+MyData.demand[i2]+
[0]+" units from p"+ i2);
}

//qq++; // 0 0 check
if (((wtour[b+1]===O && MyObjectiveFunction.doesArrayContain(MyRMcontrol .startingPosition,wtour[b+1])) && kk<Main.k-1) || MyRMcontrol.previousW[wtour[b]][0][qq][kk]==1) {/** MySolution.startingPositions.contains(wtour[b+1]) */}

kk++;
qk=1;
if (kk<Main.k) time=MyRMcontrol.startTime[kk];
SumServiceTime=0;
System.out.println();

else { // 0 0 check
if (kk == MyRMcontrol.kkk) break G;
System.out.println("Distance = " + dist + " delay " + waitingTime);
calculateVehicleLoad();

//Trickkiste.addDrawComp();
// Trickkiste.printRoute();
// MyCanvas.updateCanvas(MyRMcontrol.previousW);
// MyCanvas.addDetails();
}

}

// end main

}0+ "]0["+qq"]","+kk"] \t");

if (RMsystem = true) MyRMcontrol.previousW[wtour[b]]0][qq][kk] = 1;

dist+=MyData.distance[wtour[b]][0];
time+=(double)(MyData.distance[wtour[b]][0]) / v;

if (time<MyData.TWstart[0]){ 
waitingTime+= (MyData.TWstart[0]-time);
time=MyData.TWstart[0];
}
MyRMcontrol.timePosition[qq][kk]=time;

System.out.println(" \t"+df.format(MyData.distance[wtour[b]][0]));
//no waiting times respected
System.out.println(" \t"+MyData.TWstart[0]);
System.out.println(" \t"+df.format(time));
System.out.println(" \t" + + MyData.TWend[0]);
System.out.println(" \t" + + MyData.Service[0]);

time+= MyData.Service[0];
SumServiceTime+=MyData.Service[0];

for (int i2=0; i2<MyData.distance.length; i2++){
if (MyData.demand[i2][0]!=0)System.out.print(" \t"+MyData.demand[i2]+
[0]+" units from p"+ i2);
}

//qq++; // 0 0 check
if (((wtour[b+1]===O && MyObjectiveFunction.doesArrayContain(MyRMcontrol .startingPosition,wtour[b+1])) && kk<Main.k-1) || MyRMcontrol.previousW[wtour[b]][0][qq][kk]==1) {/** MySolution.startingPositions.contains(wtour[b+1]) */}

kk++;
qk=1;
if (kk<Main.k) time=MyRMcontrol.startTime[kk];
SumServiceTime=0;
System.out.println();

else { // 0 0 check
if (kk == MyRMcontrol.kkk) break G;
System.out.println("Distance = " + dist + " delay " + waitingTime);
calculateVehicleLoad();

//Trickkiste.addDrawComp();
// Trickkiste.printRoute();
// MyCanvas.updateCanvas(MyRMcontrol.previousW);
// MyCanvas.addDetails();
}

}

// end main
public static void calculateVehicleLoad() {
    double load = new int[MyData.demand.length];

    for (int k = 0; k < MYRMcontrol.previousW[0][0].length; k++) {
        for (int q = 0; q < MYRMcontrol.previousW[0][0].length; q++) {
            for (int i = 0; i < MYRMcontrol.previousW.length; i++) {
                if (MYRMcontrol.previousW[i][j][k][q] == 1) {
                    load[i] = MYRMcontrol.fileMatrixCurrent.get(i)[3].intValue();
                    System.out.println(i + "+ j + " : sVehicle load at id: " + MYRMcontrol.fileMatrixCurrent.get(i)[0].intValue() + "= +load[j] ;}
                } else if (MYRMcontrol.previousW[i][j][q][k] == 1) {
                    load[j] = (load[i] + MYRMcontrol.fileMatrixCurrent.get(i)[3].intValue());
                    System.out.println(i + "+ j + " : sVehicle load at id: " + MYRMcontrol.fileMatrixCurrent.get(i)[0].intValue() + "= +load[j] ;
                }
            }
        }
    }
    return capKm;
}

public static double calculateCapKmFromCurrPosition() {
    double capKm = 0;

    for (int k = 0; k < MYRMcontrol.previousW[0][0].length; k++) {
        for (int q = 0; q < MYRMcontrol.previousW[0].length; q++) {
            for (int i = 0; i < MYRMcontrol.previousW.length; i++) {
                for (int j = 0; j < MYRMcontrol.previousW.length; j++) {
                    if (MYRMcontrol.previousW[i][j][k][q] == 1) {
                        capKm += load[i] * MyData.distance[i][j];
                        System.out.println("Acc. capKM at q " + q + " : l: " + load[i] + " dist: " + MyData.distance[i][j]);
                    }
                }
            }
        }
    }
    return capKm;
}

public static void writeExcel() throws IOException, WriteException, BiffException {
    Workbook existingWorkbook = Workbook.getWorkbook(new File("C:\\Users\\21576959\\Workspace\\02 VRPPD test files\\01 Created files\\ExperimentsEvaluation.xls"));
    WritableWorkbook workbookCopy = Workbook.createWorkbook(new File("C:\\Users\\21576959\\Workspace\\02 VRPPD test files\\01 Created files\\ExperimentsEvaluation.xls"), existingWorkbook);
    WritableSheet sheetToEdit = workbookCopy.getSheet("Sheet1");
    WritableCell cell;

    String file = null;
    int nrOfRows = sheetToEdit.getRows();
    int startColumn;
    if (request == true) {
        file = fileDescription; //"Request" + Main.excel[7];
        startColumn = 8;
        Label 12 = new Label(16, nrOfRows, Double.toString(MyData.distance[MyData.distance.length-2][MyData.distance.length-1]));
        cell = 12;
        sheetToEdit.addCell(cell);

Label l1 = new Label(17, nrOfRows, Double.toString(roundtrip));
cell = l1;
sheetToEdit.addCell(cell);

} else {
    file= fileDescription;"+Main.excel[0];
    startColumn= 0;
}

Label l1 = new Label(0+startColumn, nrOfRows, file);
cell = l1;
sheetToEdit.addCell(cell);

for(int c=0; c<Main.excel.length-1; c++){
    Label l = new Label((1+c+startColumn), nrOfRows, Main.excel[c]);
cell = l;
sheetToEdit.addCell(cell);
}
workbookCopy.write();
workbookCopy.close();
existingWorkbook.close();

} // end class Main
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.text.DecimalFormat;

public class DemandGenerator {
  static FileWriter fw;
  static BufferedWriter bw;
  static DecimalFormat df = new DecimalFormat("#.00");
  public static double[][] distance;
  public static double[][] demand;
  public static double[] TWstart;
  public static double[] TWend;
  public static double[] Service;
  public static double[][] coordinates;

  DemandGenerator(String filename) throws IOException, InputDataReader.InputDataReaderException {
    InputDataReader reader = new InputDataReader(filename);
    distance = reader.readDoubleArrayArray();
    demand = reader.readDoubleArrayArray();
    TWstart = reader.readDoubleArray();
    TWend = reader.readDoubleArray();
    Service = reader.readDoubleArray();
    coordinates = reader.readDoubleArrayArray();
  }

  public static void main(String args[]) {
    String filename = "C:\\Users\\21576959\\Workspace\\02 VRPPD test files\\Created files\\Instance30.txt"; // Datei(en) auswählen.
    DemandGenerator data = new DemandGenerator(filename);
    int nrOfRequestSets = 300;
    for (int n = 291; n <= nrOfRequestSets; n++) {
      fw = new FileWriter("C:\\Users\\21576959\\Workspace\\02 VRPPD test files\\Created files\\Request\\Request" + n + ".txt");
      bw = new BufferedWriter(fw);
      System.out.println(coordinates[0][1]);
      // Create new arrays:
      double[] newdistance = new double[distance.length + 2][distance.length + 2];
      double[] newdemand = new double[distance.length + 2][distance.length + 2];
      double[] newTWstart = new double[distance.length + 2];
      double[] newTWend = new double[distance.length + 2];
      double[] newService = new double[distance.length + 2];
      double[][] newcoordinates = new double[distance.length + 2][2];
      java.util.Random r = new java.util.Random(); // 12345
      java.util.Random r = new java.util.Random(); // 12345
      // Copy old and create new coordinates:
      for (int i = 0; i < newcoordinates.length; i++) {
        for (int j = 0; j < 2; j++) {
          if (i < newcoordinates.length - 2)
            newcoordinates[i][j] = coordinates[i][j];
          else
            newcoordinates[i][j] = r.nextDouble() * 100;
        }
      }
    }
  }
}
// Create new distance matrix:
for( int i = 0; i < newdistance.length; i++ )
  for( int j = i+1; j < newdistance.length; j++ )
    newdistance[i][j] = newdistance[j][i] = norm(
      newcoordinates[i][0], newcoordinates[i][1],
      newcoordinates[j][0], newcoordinates[j][1] );

//Start
System.out.print("distance="); bw.write("");
for( int i = 0; i < newdistance.length; i++ ){
  if(i==0){
    System.out.print("[");
    bw.write("[");
  } else {
    System.out.print("\t [");
    bw.write("[");
  }
  for( int j = 0; j < newdistance.length; j++ ){
    if(i==j){
      System.out.print(newdistance[i][j]);//"0.0"
      bw.write("0.0");
    } else{
      System.out.print( df.format(newdistance[i][j]));//df.format(
        r.nextDouble()*100));
      bw.write( df.format(newdistance[i][j]));//r.nextDouble()*100
        )
    } if(j<newdistance.length-1){
      System.out.print(" , ");
      bw.write(" , ");
    }
  if(i<newdistance.length-1){
    System.out.println("\n");
    bw.write("],");
  } else{
    System.out.println("]");
    bw.newLine();
  } if(i<newdistance.length-1){
    System.out.println("\n");
    bw.newLine();
  }
}
System.out.println("\n");bw.newLine();
boolean pair=true;
int demand;
System.out.print("demand = {"); bw.write("{");
for( int i = 0; i < newdistance.length; i++ ){
  if(i==0){
    System.out.print("\t {");
    bw.write("[");
  } else System.out.print("\t {");
  for( int j = 0; j < newdistance.length; j++ ){
    if(i==j){
      System.out.print("0");
      bw.write("0");
    } else if(i!=0 && j==i+1 && pair==true){
      demand= r.nextInt(3)+i;
      System.out.print(demand);
      bw.write("2");
    }
```java
pair=false;
}
else if(i!=0 && j==i+1 && pair==false){
    System.out.print(0);
    bw.write("0");
pair=true;
}
else{
    System.out.print(0);
    bw.write("0");
    if(j<newdistance.length-1){
        System.out.print(" ");bw.write(" ");
    }
    if(i<newdistance.length-1){
        System.out.print(" ");bw.write(" ");
    }
}
```
```
System.out.print("\n"); bw.newLine(); bw.newLine();

// print out x-y coordinates:
System.out.println("Coordinates = "); bw.write("[");
for (int i=0; i<newcoordinates.length; i++){
    System.out.println("["+df.format(newcoordinates[i][0])+", \	"+df.format(newcoordinates[i][1])+"],");
    bw.write("["+df.format(newcoordinates[i][0])+", \t"+df.format(newcoordinates[i][1])+"],");
    if (i<newcoordinates.length-1){
        bw.newLine();
    }
    else bw.write("]]");
    bw.newLine();
} System.out.println("]);
bw.close();

} catch (InputDataReader.InputDataReaderException ex) {
    System.out.println("Data Error: " + ex);
}
catch (java.io.IOException ex) {
    System.out.println("IO Error: " + ex);
}

/** Calculate distance between two points. */
private static double norm( double x1, double y1, double x2, double y2 )
{
    double xDiff = x2 - x1;
    double yDiff = y2 - y1;
    return Math.sqrt( xDiff*xDiff + yDiff*yDiff );
} // end norm

}
E.4. DrawingComponent class

```java
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import javax.imageio.ImageIO;
import javax.swing.JComponent;
import java.awt.Rectangle;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.text.DecimalFormat;

public class DrawingComponent extends JComponent{

    static int sFactor = 7;
    static int shift = 350;
    static DecimalFormat df = new DecimalFormat("#.00");
    static Graphics2D g2;

    @Override
    public void paintComponent(Graphics g) {
        g2 = (Graphics2D) g;
        Color darkGreen = new Color(10, 170, 10);
        g2.setColor(Color.white);
        g2.fill(rect1);
        g2.setColor(Color.black);
        g2.fill(depot);
        //Nodes:
        Ellipse2D.Double ellipse2;
        for (int i=1; i<MyRMcontrol.fileMatrix.size(); i++){
            if (MyRMcontrol.fileMatrix.get(i)[3]>0) drawNod(MyRMcontrol.fileMatrix.get(i)[1]*sFactor-shift, (MyRMcontrol.fileMatrix.get(i)[2]*sFactor)-5, 10, 10);
            else drawNod(MyRMcontrol.fileMatrix.get(i)[1]*sFactor-shift, MyRMcontrol.fileMatrix.get(i)[2]*sFactor, g2, false, MyRMcontrol.fileMatrix.get(i)[0]);
        }

        //Lines:
        if (Trickkiste.drawList == true) drawFromList();
        else{
            Line2D.Double connector;
            int pos = 80;
            double timex = 0;
            for (int k=0; k<=2; k++){
                double centroidX = 0, centroidY = 0;
                int nNoFKnots = 0;
                timex=0;
                if (k==0) g2.setColor(Color.blue);
                if (k==1) g2.setColor(darkGreen);
                if (k==2) g2.setColor(Color.red);
                for (int q=1; q<MyRMcontrol.previousW.length-3; q++){
                    // Lines...
                }
            }
        }
    }
}
```
for(int i=0; i<MyRMcontrol.previousW.length-3; i++){
    for(int j=0; j<MyRMcontrol.previousW.length-3; j++){
        if(MyRMcontrol.previousW[i][j][q][k] == 1){
            centroidX += MyRMcontrol.fileMatrix.get(i)[1];
            centroidY += MyRMcontrol.fileMatrix.get(i)[2];
            nrOfKnots++;
        }
    }
}

for(int i=0; i<MyRMcontrol.previousW.length-3; i++){  //for(int j=0; j<MyRMcontrol.previousW.length-3; j++){
    connector = new Line2D.Double(MyRMcontrol.fileMatrix.get(i)[1]*sFactor+shift, MyRMcontrol.fileMatrix.get(i)[2]*sFactor, MyRMcontrol.fileMatrix.get(j)[1]*sFactor+shift, MyRMcontrol.fileMatrix.get(j)[2]*sFactor);
    g2.draw(connector);
    g2.drawString("["+i+"]["+j+"]", 10, pos);
    timex+=(MyData.distance[i][j]/Main.v);
    if(timex<MyData.TWstart[j]){timex=MyData.TWstart[j];}
    g2.drawString("t"+MyData.TWstart[j], 40, pos);
    g2.drawString("t"+df.format(timex)+MyData.TWend[j]);
    g2.drawString("t"+df.format(timex)+MyData.Service[j]);
    timex+=MyData.Service[j];
    pos+=15;
    centroidX += MyRMcontrol.fileMatrix.get(i)[1];
    centroidY += MyRMcontrol.fileMatrix.get(i)[2];
    nrOfKnots++;
}

    return new Point(centroidX / knots.size(), centroidY / knots.size());
    Rectangle2D.Double center = new Rectangle2D.Double((centroidX /nrOfKnots)*sFactor-shift-10, (centroidY /nrOfKnots)*sFactor-10, 20, 20);
    g2.setFillColor(Color.orange);
    g2.fill(center);
}

if(Trickkiste.drawfList == false){
    g2.setColor(Color.black);
    g2.drawString("Filename: "+MyRMcontrol.filename, 10, 20);
    g2.drawString("Number of iterations: "+Main.iterations, 10, 40);
    g2.drawString("Obj. Value: "+MyRMcontrol.costs.get(MyRMcontrol.costs.size()-1), 10, 60);
}
else{
    g2.setColor(Color.black);
    g2.drawString("Filename: "+Trickkiste.filename, 10, 20);
}

public static void drawNod(double d, double e, Graphics2D g2, boolean fill, Double nodNr){
    g2.setStroke(new BasicStroke(1));
    g2.setColor(color);
    g2.draw(new Ellipse2D.Double(d-5, e-5, 10, 10));
}

public static void drawFromList(){
    Line2D.Double connector;
    g2.setColor(Color.blue);
    int pos=80;
double timex = 0;
for(int i=0; i<Trickkiste.r1.length-1; i++){
    connector = new Line2D.Double(MyRMcontrol.fileMatrix.get(Trickkiste.r1[i])[1]*sFactor+shift, MyRMcontrol.fileMatrix.get(Trickkiste.r1[i])[2]*sFactor, MyRMcontrol.fileMatrix.get(Trickkiste.r1[i+1])[1]*sFactor+shift, MyRMcontrol.fileMatrix.get(Trickkiste.r1[i+1])[2]*sFactor);
g2.draw(connector);
g2.drawString("["+Trickkiste.r1[i]+"]"+
/*Trickkiste.r1[i+1]+" ", 10, pos)
    timex+=MyData.distance[Trickkiste.r1[i]][Trickkiste.r1[i+1]];
    if(timex<MyData.TWstart[Trickkiste.r1[i+1]])
        timex=MyData.TWstart[Trickkiste.r1[i+1]];
    
    g2.drawString(" ", 55, pos);
    g2.drawString("t"+MyData.TWstart[Trickkiste.r1[i+1]], 55, pos);
    g2.drawString("t"+df.format(timex), 105, pos);
    System.out.print("t"+df.format(MyData.distance[i][j]));
    //no waiting times respected
    g2.draw(connector);
g2.setColor(darkGreen);
for(int i=0; i<Trickkiste.r2.length-1; i++){
    connector = new Line2D.Double(MyRMcontrol.fileMatrix.get(Trickkiste.r2[i])[1]*sFactor+shift, MyRMcontrol.fileMatrix.get(Trickkiste.r2[i])[2]*sFactor, MyRMcontrol.fileMatrix.get(Trickkiste.r2[i+1])[1]*sFactor+shift, MyRMcontrol.fileMatrix.get(Trickkiste.r2[i+1])[2]*sFactor);
g2.draw(connector);
g2.setColor(Color.red);
for(int i=0; i<Trickkiste.r3.length-1; i++){
    connector = new Line2D.Double(MyRMcontrol.fileMatrix.get(Trickkiste.r3[i])[1]*sFactor+shift, MyRMcontrol.fileMatrix.get(Trickkiste.r3[i])[2]*sFactor, MyRMcontrol.fileMatrix.get(Trickkiste.r3[i+1])[1]*sFactor+shift, MyRMcontrol.fileMatrix.get(Trickkiste.r3[i+1])[2]*sFactor);
g2.draw(connector);
}
// random line:
addLine(g2);
}
public static void updatePlot(){
    g2.dispose();
g2.clearRect(0, 0, 1280, 720);
g2.drawRect(0, 0, 1280, 720);
    g2.draw(connector);
    
    // random line:
    addLine(g2);
}
public static void addLine(Graphics2D g2){
int x1 = (int) (Math.random()*320);
int x2 = (int) (Math.random()*320);
int y1 = (int) (Math.random()*200);
int y2 = (int) (Math.random()*200);
, \((\text{float})\text{Math.random()}\));
179       g2.drawLine(x1, y1, x2, y2);
180
181   }
182   public Point centroid() {
183       double centroidX = 0, centroidY = 0;
184
185       for(Point knot : knots) {
186           centroidX += knot.getX();
187           centroidY += knot.getY();
188       }
189       return new Point(centroidX / knots.size(), centroidY / knots.size());
190   }
E.5. ExcelFiller class

```java
public class ExcelFiller {
    // public static void main(String[] args) throws IOException, WriteException, BiffException {
    Workbook existingWorkbook = Workbook.getWorkbook(new File("C:\\Users\\21576959\\Workspace\\02 VRPPD test files\\01 Created files\\ExperimentsEvaluationFinal.xls"));
    WritableWorkbook workbookCopy = Workbook.createWorkbook(new File("C:\\Users\\21576959\\Workspace\\02 VRPPD test files\\01 Created files\\ExperimentsEvaluationFinal.xls"), existingWorkbook);
    WritableSheet sheetToEdit = workbookCopy.getSheet("Sheet1");
    WritableCell cell;
    for(int i=1; i<=100; i++){
        DecimalFormat df = new DecimalFormat("#.00");
        long startzeit = System.currentTimeMillis();
        // Initialize our objects
        double[][] customers = new double[21][21]; // [20][2];
        // Read in data set:
        String filename = "C:\Users\21576959\Workspace\02 VRPPD test files\01 Created files\Request"+i+".txt"; // Datei(en) auswählen.
        MyData data = new MyData(filename);
        int nrOfRows =44;
        double roundtrip = data.distance[0][21]+data.distance[21][22]+data.distance[22][0];
        Label l1 = new Label(17, (nrOfRows+i), Double.toString(roundtrip));
        cell = (WritableCell) l1;
        sheetToEdit.addCell(cell);
        // Read in data set:
    }
    workbookCopy.write();
    workbookCopy.close();
    existingWorkbook.close();
}
```

**E.6. InputDataReader class**

```java
import java.io.*;

public class InputDataReader {
    public static class InputDataReaderException extends Exception {
        private static final long serialVersionUID = 1021L;
        public InputDataReaderException(String file) {
            super(""" + file + "' contains bad data format");
        }
    }

    StreamTokenizer _tokenizer;
    Reader _reader;
    String _fileName;

    public InputDataReader(String fileName) throws IOException {
        _reader = new FileReader(fileName);
        _fileName = fileName;
        _tokenizer = new StreamTokenizer(_reader);
    }

    @Override
    protected void finalize() throws Throwable {
        _reader.close();
    }

    public double readDouble() throws InputDataReaderException, IOException {
        int ntType = _tokenizer.nextToken();
        if (ntType != StreamTokenizer.TT_NUMBER )
            throw new InputDataReaderException(_fileName);
        return _tokenizer.nval;
    }

    public int readInt() throws InputDataReaderException,
```
IOException {
    int ntType = _tokenizer.nextToken();
    if (ntType != StreamTokenizer.TT_NUMBER)
        throw new InputDataReaderException(_fileName);
    return (new Double(_tokenizer.nval)).intValue();
}

double[] readDoubleArray() throws InputDataReaderException, IOException {
    int ntType = _tokenizer.nextToken();  // Read the '['
    if (ntType != '[')
        throw new InputDataReaderException(_fileName);
    DoubleArray values = new DoubleArray();
    ntType = _tokenizer.nextToken();
    while (ntType == StreamTokenizer.TT_NUMBER) {
        values.add(_tokenizer.nval);
        ntType = _tokenizer.nextToken();
        if (ntType == ',') {
            ntType = _tokenizer.nextToken();
        } else if (ntType != ']') {
            throw new InputDataReaderException(_fileName);
        }
    }
    if (ntType != ']')
        throw new InputDataReaderException(_fileName);
    // Allocate and fill the array.
    double[] res = new double[values.getSize()];
    for (int i = 0; i < values.getSize(); i++) {
        res[i] = values.getElement(i);
    }
    return res;
}

double[][] readDoubleArrayArray() throws InputDataReaderException, IOException {
    int ntType = _tokenizer.nextToken();  // Read the '['
    if (ntType != '[')
        throw new InputDataReaderException(_fileName);
    DoubleArrayArray values = new DoubleArrayArray();
    ntType = _tokenizer.nextToken();
    while (ntType == '[') {
        _tokenizer.pushBack();
        values.add(readDoubleArray());
        ntType = _tokenizer.nextToken();
        if (ntType == ',') {
            ntType = _tokenizer.nextToken();
        } else if (ntType != ']') {
            throw new InputDataReaderException(_fileName);
        }
    }
    if (ntType != ']')
        throw new InputDataReaderException(_fileName);
// Allocate and fill the array.
  double[][] res = new double[values.getSize()][];
  for (int i = 0; i < values.getSize(); i++) {
    res[i] = new double[values.getSize(i)];
    for (int j = 0; j < values.getSize(i); j++) {
      res[i][j] = values.getElement(i, j);
    }
  }
  return res;
}

int[] readIntArray() throws InputDataReaderException, IOException {
  int nType = _tokenizer.nextToken(); // Read the ']
  if (nType != ']')
    throw new InputDataReaderException(_fileName);

  IntArray values = new IntArray();
  nType = _tokenizer.nextToken();
  while (nType == StreamTokenizer.TT_NUMBER) {
    values.add(_tokenizer.nval);
    nType = _tokenizer.nextToken();
    if (nType == ',') {
      nType = _tokenizer.nextToken();
    } else if (nType != ']') {
      throw new InputDataReaderException(_fileName);
    }
  }

  if (nType != ']')
    throw new InputDataReaderException(_fileName);

  // Allocate and fill the array.
  int[] res = new int[values.getSize()];
  for (int i = 0; i < values.getSize(); i++) {
    res[i] = values.getElement(i);
  }
  return res;
}

int[][] readIntArrayArray() throws InputDataReaderException, IOException {
  int nType = _tokenizer.nextToken(); // Read the ']
  if (nType != ']')
    throw new InputDataReaderException(_fileName);

  IntArrayArray values = new IntArrayArray();
  nType = _tokenizer.nextToken();
  while (nType == '[') {
    _tokenizer.pushBack();
    values.add(readIntArray());
    nType = _tokenizer.nextToken();
    if (nType == ',') {
      nType = _tokenizer.nextToken();
    } else if (nType != ']') {
      throw new InputDataReaderException(_fileName);
    }
  }
  return values;
}
if ( ntType != '\' )
    throw new InputDataReaderException(_fileName);

    // Allocate and fill the array.
    int[][] res = new int[values.getSize()[]];
    for (int i = 0; i < values.getSize(); i++) {
        res[i] = new int[values.getSize(i)];
        for (int j = 0; j < values.getSize(i); j++) {
            res[i][j] = values.getElement(i, j);
        }
    }
    return res;
}

private class DoubleArray {
    int _num = 0;
    double[] _array = new double[32];

    final void add(double dval) {
        if (_num >= _array.length ) {
            double[][] array = new double[2 * _array.length][];
            System.arraycopy(_array, 0, array, 0, _num);
            _array = array;
            _array[_num++] = dval;
        }
    }

    final double getElement(int i) { return _array[i]; }
    final int getSize() { return _num; }
}

private class DoubleArrayArray {
    int _num = 0;
    double[][] _array = new double[32][];

    final void add(double[] dray) {
        if (_num >= _array.length ) {
            double[][] array = new double[2 * _array.length][];
            for (int i = 0; i < _num; i++) {
                array[i] = _array[i];
            }
            _array = array;
            _array[_num] = new double[dray.length];
            System.arraycopy(dray, 0, _array[_num], 0, dray.length);
            _num++;
        }
    }

    final double getElement(int i, int j) { return _array[i][j]; }
    final int getSize() { return _num; }
    final int getSize(int i) { return _array[i].length; }
}

private class IntArray {
    int _num = 0;
    int[] _array = new int[32];

    final void add(int ival) {
        if (_num >= _array.length ) {
            int[] array = new int[2 * _array.length];
            System.arraycopy(_array, 0, array, 0, _num);
            _array = array;
            _array[_num++] = (int)Math.round(ival);
        }
    }
}
```java
269     final int getElement(int i) { return _array[i]; }
270     final int getSize() { return _num; }
271 }
272
private class IntArrayArray {
    int _num = 0;
    int[][] _array = new int[32][];

    final void add(int[] iray) {
        if (_num >= _array.length) {
            int[][] array = new int[2 * _array.length][];
            for (int i = 0; i < _num; i++) {
                array[i] = _array[i];
            }
            _array = array;
        }
        _array[_num] = new int[iray.length];
        System.arraycopy(iray, 0, _array[_num], 0, iray.length);
        _num++;
    }

    final int getElement(int i, int j) { return _array[i][j]; }
    final int getSize() { return _num; }
    final int getSize(int i) { return _array[i].length; }
}
```
E.7. MyCanvas class

```java
import java.awt.image.BufferedImage;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;
import java.awt.geom.Rectangle2D;
import javax.imageio.ImageIO;
import javax.imageio.ImageWriteParam;
import javax.imageio.ImageWriter;
import javax.imageio.stream.FileImageOutputStream;
import javax.imageio.stream.ImageOutputStream;
import javax.swing.*;
import java.util.ArrayList;
import java.util.Random;
import java.io.*;

public class MyCanvas {
    static JLabel view;
    static BufferedImage surface;
    static Random random = new Random();
    static double sFactor = .4;
    static int xShift = /*350*/50;
    static DecimalFormat df = new DecimalFormat("#.00");
    static Color darkGreen = new Color(10, 170, 10);
    static ArrayList<Double[]> previousStartingPositionBlue = new ArrayList<Double[]>();
    static ArrayList<Double[]> previousStartingPositionGreen = new ArrayList<Double[]>();
    static ArrayList<Double[]> previousStartingPositionRed = new ArrayList<Double[]>();
    static ActionListener listener;

    public MyCanvas(int width) {
        surface = new BufferedImage(width, 640, BufferedImage.TYPE_INT_RGB);
        view = new JLabel(new ImageIcon(surface));
        Graphics g = surface.getGraphics();
        g.setColor(Color.white);
        g.fillRect(0, 0, width, 640);
        g.setColor(Color.BLACK);
        g.setColor(Color.white);
        g.fillRect(0, 0, width, 640);
        g.setColor(Color.BLACK);
        g.dispose();

        ActionListener listener = new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                addNewElement();
            }
        };
        Timer timer = new Timer(20, listener);
        timer.start();
    }
}
```
public static void addNewElement() {
    boolean drawArc = random.nextBoolean();
    int x = random.nextInt(60);
    int y = random.nextInt(40);
    Graphics g = surface.getGraphics();
    if (drawArc) {
        g.setColor(Color.BLUE);
        int xx = random.nextInt(60);
        int yy = random.nextInt(40);
        drawArc(x, y, xx, yy, g);
    } else {
        g.setColor(Color.BLACK);
        int xx = random.nextInt(60);
        int yy = random.nextInt(40);
        drawNode(x, y, xx, yy, g);
    }
    g.dispose();
    view.repaint();
}

public static void main2(/*String[] args*/) {
    previousStartingPositionBlue.clear();
    previousStartingPositionGreen.clear();
    previousStartingPositionRed.clear();
    MyCanvas canvas = new MyCanvas(640);
    JFrame frame = new JFrame();
    frame.getContentPane().setBackground(Color.white);
    int vertexes = 0;
    // Change this next part later to be dynamic.
    vertexes = 10;
    int canvasSize = vertexes * vertexes;
    frame.setSize(canvasSize, canvasSize);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setContentPane(canvas.view);
    frame.pack();
    frame.setLocationByPlatform(true);
    frame.setVisible(true);
    //MyCanvas.updateCanvas(MySolution.w);
}

public static void main3(/*String[] args*/) // hauptprogramm {
    previousStartingPositionBlue.clear();
    previousStartingPositionGreen.clear();
    previousStartingPositionRed.clear();
    MyCanvas canvas = new MyCanvas(1280);
    JFrame frame = new JFrame();
    frame.getContentPane().setBackground(Color.white);
    int vertexes = 0;
    // Change this next part later to be dynamic.
    vertexes = 10;
    int canvasSize = vertexes * vertexes;
    frame.setSize(canvasSize, canvasSize);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setContentPane(canvas.view);
    frame.pack();
    frame.setLocationByPlatform(true);
    frame.setVisible(true);
    //MyCanvas.updateCanvas(MySolution.w);
}
public static void drawPath(Graphics g) {
    g.setColor(Color.LIGHT_GRAY);
    int xLoc;
    int yLoc;
    int xxLoc;
    int yyLoc;

    for (int i = 0; i < previousStartingPositionBlue.size() - 1; i++) {
        xLoc = (previousStartingPositionBlue.get(i)[0].intValue()) * sFactor + xShift;
        yLoc = (previousStartingPositionBlue.get(i)[1].intValue()) * sFactor;
        xxLoc = (previousStartingPositionBlue.get(i + 1)[0].intValue()) * sFactor + xShift;
        yyLoc = (previousStartingPositionBlue.get(i + 1)[1].intValue()) * sFactor;
        g.drawLine(xLoc, yLoc, xxLoc, yyLoc);
    }

    for (int i = 0; i < previousStartingPositionGreen.size() - 1; i++) {
        xLoc = (previousStartingPositionGreen.get(i)[0].intValue()) * sFactor + xShift;
        yLoc = (previousStartingPositionGreen.get(i)[1].intValue()) * sFactor;
        xxLoc = (previousStartingPositionGreen.get(i + 1)[0].intValue()) * sFactor + xShift;
        yyLoc = (previousStartingPositionGreen.get(i + 1)[1].intValue()) * sFactor;
        g.drawLine(xLoc, yLoc, xxLoc, yyLoc);
    }

    for (int i = 0; i < previousStartingPositionRed.size() - 1; i++) {
        xLoc = (previousStartingPositionRed.get(i)[0].intValue()) * sFactor + xShift;
        yLoc = (previousStartingPositionRed.get(i)[1].intValue()) * sFactor;
        xxLoc = (previousStartingPositionRed.get(i + 1)[0].intValue()) * sFactor + xShift;
        yyLoc = (previousStartingPositionRed.get(i + 1)[1].intValue()) * sFactor;
        g.drawLine(xLoc, yLoc, xxLoc, yyLoc);
    }
}

public static void drawNode(double x, double y, Graphics g, boolean fill, Double ID, int nodeNr) {
    int size = (int) (8 + Math.abs(MyRMcontrol.fileMatrixCurrent.get(nodeNr)[3]) / 10);

    int xLoc = ((int)x) * sFactor + xShift - size / 2;
    int yLoc = ((int)y) * sFactor - size / 2;
    g.setColor(Color.black);
    if (fill == true) g.fillOval(xLoc, yLoc, size, size);
    else g.drawOval(xLoc, yLoc, size, size);

    int xLoc = ((int)x) * sFactor + xShift - 5;
    int yLoc = ((int)y) * sFactor - 5;
    g.setColor(Color.black);
    if (fill == true) g.fillOval(xLoc, yLoc, 10, 10);
    else g.drawOval(xLoc, yLoc, 10, 10);
    g.drawString("" + nodeNr + "+ID.intValue(), xLoc, yLoc - 5);
}

public static void drawNode2(double x, double y, Graphics g, boolean fill, Double ID, int nodeNr) {
    int size = 5;
    int xLoc = (int) (x * sFactor);
    int yLoc = (int) (y * sFactor);
}
System.out.println("x: "+xLoc+" y: "+yLoc);
g.setColor(Color.black);
if(fill==true)g.fillOval(xLoc, yLoc, size, size);
else g.drawOval(xLoc, yLoc, size, size);
g.drawString(" +nodeNr+"|"+ID.intValue(), xLoc, yLoc-5);
}
public static void drawSuburbs(double x, double y, Graphics g, boolean fill, Double ID, String name)
{
    int size = 50;
    int xLoc = (int)(x*sFactor);
    int yLoc = (int)(y*sFactor);
    g.setColor(Color.blue);
    if(fill==true)g.fillOval(xLoc, yLoc, size, size);
    else g.drawOval(xLoc, yLoc, size, size);
    g.drawString(name+"|"+ID.intValue(), xLoc, yLoc-5);
}
public static void drawArc(int x, int y, int xx, int yy, Graphics g)
{
    int xLoc = (int)(x*sFactor+xShift);
    int yLoc = (int)(y*sFactor);
    int xxLoc = (int)(xx*sFactor+xShift);
    int yyLoc = (int)(yy*sFactor);
    g.drawLine(xLoc, yLoc, xxLoc, yyLoc);
}
public static void updateCanvas(int w[][],[], int thisxShift){
    Graphics g = surface.getGraphics();
    xShift = thisxShift;
    // dispose previous drawing:
    g.setColor(Color.white);
    if(xShift==50) g.fillRect(0,0,/*1280*/640, 640);
    else g.fillRect(0,0,1280, 640);
    g.setColor(Color.black);
    g.drawString(" +nodeNr+"|"+ID.intValue(), xLoc, yLoc-5);
}
for(int i=1; i<MyRMcontrol.fileMatrixCurrent.size(); i++){
    if(MyRMcontrol.fileMatrixCurrent.get(i)[3]>0)
        g.fillRect(MyRMcontrol.fileMatrixCurrent.get(i)[1], MyRMcontrol.fileMatrixCurrent.get(i)[2], g, false, MyRMcontrol.fileMatrixCurrent.get(i)[0], i);
    else drawNode(MyRMcontrol.fileMatrixCurrent.get(i)[1], MyRMcontrol.fileMatrixCurrent.get(i)[2], g, true, MyRMcontrol.fileMatrixCurrent.get(i)[0], i);
}
for(int k=0; k<Main.k; k++)
    if(k==0) g.setColor(Color.blue);
    else if(k==1) g.setColor(darkGreen);
    else if(k==2) g.setColor(Color.red);
else{
Color randomColor = new Color(random.nextFloat(), random.nextFloat(), random.nextFloat());
g.setColor(randomColor);
}
for (int q=1; q<MyRMcontrol.fileMatrixCurrent.size()/*-3*/; q++){
for (int i=0; i<Math.min(MyRMcontrol.fileMatrixCurrent.size(), w.length)/*-3*/; i++){
for (int j=0; j<Math.min(MyRMcontrol.fileMatrixCurrent.size(), w.length)/*-3*/; j++){
if (w[i][j][q][k] == 1){
drawArc(MyRMcontrol.fileMatrixCurrent.get(i)[1].intValue(), MyRMcontrol.fileMatrixCurrent.get(i)[2].intValue(), MyRMcontrol.fileMatrixCurrent.get(j)[1].intValue(), MyRMcontrol.fileMatrixCurrent.get(j)[2].intValue(), g);
}
}
}
}
g.dispose();
view.repaint();
}
public static void updateCanvasCS(int w[][],[], int thisxShift){
Graphics g = surface.getGraphics();
sFactor = '/.4/* ((double)630/(double)(MyDynamicPricing.gridNumber*MyDynamicPricing.gridSizeY));
System.out.println("//////////////////////////////////////MyDynamicPricing.gridSizeY" + MyDynamicPricing.gridNumber);
System.out.println("/////////////////////////////////////MyDynamicPricing.gridNumber:");
System.out.println("sFactor = " + sFactor + " + ((double)630/(double)(MyDynamicPricing.gridNumber*MyDynamicPricing.gridSizeY));
System.out.println("//////////////////////////////////////");
xShift = /*thisxShift*/0;
// dispose previous drawing:
g.setColor(Color.white);
if(xShift >= 50)  g.fillRect(0,0/*1280*//640, 640);
else  g.fillRect(0,0,1280, 640);
g.setColor(Color.black);
drawNode2(0, 0, g, true, 0.0, 0.0);
drawNode2(0, 1000, g, true, 0.0, 1000);
drawNode2(1000, 0, g, true, 1000.0, 0.0);
drawNode2(1000, 1000, g, true, 1000.0, 1000);
//draw all nodes:
g.setColor(Color.CYAN);
for (int i=1; i<MyRMcontrol.fileMatrix.size(); i++){
if(MyRMcontrol.fileMatrix.get(i)[3]>=0) drawNode2(MyRMcontrol.fileMatrix.get(i)[1], MyRMcontrol.fileMatrix.get(i)[2], g, false, MyRMcontrol.fileMatrix.get(i)[0], i);
else drawNode2(MyRMcontrol.fileMatrix.get(i)[1], MyRMcontrol.fileMatrix.get(i)[2], g, true, MyRMcontrol.fileMatrix.get(i)[0], i);
}
g.setColor(Color.black);
drawSuburbs(115.75*111, -32.055*-94.92, g, true, 1000.0, "Fremantle");
drawSuburbs(500, 500, g, true, 1000.0, "Fremantle");
drawGrid(g);
//
// drawPreviousStartingPositionsBlue(g);
// drawPreviousStartingPositionsGreen(g);
// drawPreviousStartingPositionsRed(g);
// addDetails();
// drawPath(g);

// draw nodes:
for (int i=1; i<MyRMcontrol.fileMatrixCurrent.size(); i++){
  if (MyRMcontrol.fileMatrixCurrent.get(i)[3]>0)
drawNode2(MyRMcontrol.fileMatrixCurrent.get(i)[1], MyRMcontrol.fileMatrixCurrent.get(i)[2], g, false, MyRMcontrol.fileMatrixCurrent.get(i)[0], i);
  else drawNode2(MyRMcontrol.fileMatrixCurrent.get(i)[1], MyRMcontrol.fileMatrixCurrent.get(i)[2], g, true, MyRMcontrol.fileMatrixCurrent.get(i)[0], i);
}

// draw arcs:
for (int k=0; k<Main.k; k++){
  if (k==0) g.setColor(Color.blue);
  else if (k==1) g.setColor(darkGreen);
  else if (k==2) g.setColor(Color.red);
  else{
    Color randomColor = new Color(random.nextFloat(), random.nextFloat(), random.nextFloat());
g.setColor(randomColor);
  }
  for (int q=1; q<MyRMcontrol.fileMatrixCurrent.size()-3; q++)
    for (int i=0; i<Math.min(MyRMcontrol.fileMatrixCurrent.size(), w.length)-3; i++)
      for (int j=0; j<Math.min(MyRMcontrol.fileMatrixCurrent.size(), w.length)-3; j++)
        if (w[i][j][q][k] == 1){
drawArc(MyRMcontrol.fileMatrixCurrent.get(i)[1].intValue(), MyRMcontrol.fileMatrixCurrent.get(i)[2].intValue(), MyRMcontrol.fileMatrixCurrent.get(j)[1].intValue(), MyRMcontrol.fileMatrixCurrent.get(j)[2].intValue(), g);
  }
}
g.dispose();
view.repaint();

public static void addDetails(){
  Graphics g = surface.getGraphics();
g.setColor(Color.black);
g.drawString("Filename: "+MyRMcontrol.filename, 10, 20);
g.drawString("Number of iterations: "+Main.iterations, 10, 40);
g.drawString("Obj. Value: "+MyRMcontrol.costs.get(MyRMcontrol.costs.size()-1), 10, 60);
g.drawString("Time: "+MyRMcontrol.t, 10, 80);
g.dispose();
  view.repaint();
}

try {
  BufferedImage image = new BufferedImage(/*getWidth()*/1280, /*getHeight()*/640, BufferedImage.TYPE_INT_RGB);
g.drawImage(image, Color.black);
g.drawString("Filename: "+MyRMcontrol.filename, 10, 20);
g.drawString("Number of iterations: "+Main.iterations, 10, 40);
g.drawString("Obj. Value: "+MyRMcontrol.costs.get(MyRMcontrol.costs.size()-1), 10, 60);
g.drawImage(image, Color.black);
}
public static void drawPreviousStartingPositionsBlue(Graphics g)
{
    Double[] loc = new Double[2];
    //Add current starting locations:
    loc[0] = MyRMcontrol.fileMatrixCurrent.get(MyRMcontrol.startingPosition[0][1]);
    loc[1] = MyRMcontrol.fileMatrixCurrent.get(MyRMcontrol.startingPosition[0][2]);
    previousStartingPositionBlue.add(loc);
    for (int i = 0; i < previousStartingPositionBlue.size(); i++)
    {
        int xLoc = (previousStartingPositionBlue.get(i)[0].intValue()) * sFactor + xShift - 5;
        int yLoc = (previousStartingPositionBlue.get(i)[1].intValue()) * sFactor - 5;
        g.setColor(Color.blue);
        g.fillOval(xLoc, yLoc, 10, 10);
    }
}

public static void drawPreviousStartingPositionsGreen(Graphics g)
{
    Double[] loc = new Double[2];
    //Add current starting locations:
    loc[0] = MyRMcontrol.fileMatrixCurrent.get(MyRMcontrol.startingPosition[1][1]);
    loc[1] = MyRMcontrol.fileMatrixCurrent.get(MyRMcontrol.startingPosition[1][2]);
    previousStartingPositionGreen.add(loc);
    for (int i = 0; i < previousStartingPositionGreen.size(); i++)
    {
        int xLoc = (previousStartingPositionGreen.get(i)[0].intValue()) * sFactor + xShift - 5;
        int yLoc = (previousStartingPositionGreen.get(i)[1].intValue()) * sFactor - 5;
        g.setColor(Color.green);
        g.fillOval(xLoc, yLoc, 10, 10);
    }
}

public static void drawPreviousStartingPositionsRed(Graphics g)
{
    Double[] loc = new Double[2];
    //Add current starting locations:
    loc[0] = MyRMcontrol.fileMatrixCurrent.get(MyRMcontrol.startingPosition[2][1]);
    loc[1] = MyRMcontrol.fileMatrixCurrent.get(MyRMcontrol.startingPosition[2][2]);
    previousStartingPositionRed.add(loc);
    for (int i = 0; i < previousStartingPositionRed.size(); i++)
    {
        int xLoc = (previousStartingPositionRed.get(i)[0].intValue()) * sFactor + xShift - 5;
        int yLoc = (previousStartingPositionRed.get(i)[1].intValue()) * sFactor - 5;
        g.setColor(Color.red);
  g.fillOval(xLoc, yLoc, 10, 10);
}

public static void drawDepot(Graphics g){
    // Depot:
    int xLoc = (40*sFactor)+xShift -5;
    int yLoc = (50*sFactor) -5;
    g.setColor(Color.black);
    //g.drawRect(xLoc, yLoc, 10, 10);
    g.fillRect((MyRMcontrol.fileMatrix.get(0)[1].intValue()*sFactor)+xShift -5, (MyRMcontrol.fileMatrix.get(0)[2].intValue()*sFactor) -5, 10, 10);
}

public static void drawGrid(Graphics g) {
    double x1 = 0;
    double y1 = 0;
    double x2 = 0;
    double y2 = 100;
    g.setColor(Color.gray);
    for(int i = 0; i<=MyDynamicPricing.gridNumber; i++) {
        drawArc((int)x1, 0, (int)x2, (MyDynamicPricing.gridNumber*MyDynamicPricing.gridSizeY), g);
        g.drawString(i+"", (int)((x1+MyDynamicPricing.gridSizeX/2)*sFactor+xShift), 15);
        x1 += MyDynamicPricing.gridSizeX;
        x2 += MyDynamicPricing.gridSizeX;
    }
    y2 = 0;
    for(int i = 0; i<=MyDynamicPricing.gridNumber; i++) {
        drawArc(0, (int)y1, (MyDynamicPricing.gridNumber*MyDynamicPricing.gridSizeX), (int)y2, g);
        g.drawString(i+"", 10+xShift, (int)((y1+MyDynamicPricing.gridSizeY/2)*sFactor));
        y1 += MyDynamicPricing.gridSizeY;
        y2 += MyDynamicPricing.gridSizeY;
    }
}
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.text.DecimalFormat;

public class MyData {
    static FileWriter fw;
    static BufferedWriter bw;
    static DecimalFormat df = new DecimalFormat("#.00");

    public static double[][] distance;
    public static double[][] demand;
    public static double[] TWstart;
    public static double[] TWend;
    public static double[] Service;
    public static double cap;
    public static double reduceX;
    public static double reduceY;

    MyData(String filename) throws /**IloException,**/ java.io.IOException, InputDataReader.InputDataReaderException {
        if(Main.RMsystem == false){
            InputDataReader reader = new InputDataReader(filename);
            distance = reader.readDoubleArrayArray();
            demand = reader.readDoubleArrayArray();
            TWstart = reader.readDoubleArray();
            TWend = reader.readDoubleArray();
            Service = reader.readDoubleArray();
            //cap = reader.readDouble();
        }
    }

    /** Calculate distance between two points. */
    // private static double norm( double x1, double y1, double x2, double y2 )
    // { double xDiff = x2 - x1;
    //    double yDiff = y2 - y1;
    //    return Math.sqrt( xDiff*xDiff + yDiff*yDiff );
    // } // end norm

}
E.9. MyDynamicPricing class

```java
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class MyDynamicPricing {
    public static double distance = 1500;
    public static double directDistance;
    public static double roundtripDistance;
    public static int nrOfUnits = 1;
    public static double marginalCosts;
    public static double profitContribution;
    public static double myPrice;
    public static double traditionalPrice;
    public static boolean competition = false;
    public static int competitionLevel;
    public static int competitionSector = 0;
    public static double marketPrice = 0;
    public static ArrayList<Double> marketPriceList = new ArrayList<Double>();
    public static double[] allTimeProfits = new double[2];
    static int gridNumber = 6;
    static int gridSizeX = 100/gridNumber;
    static int gridSizeY = 100/gridNumber;
    public static double[][] deviationMatrix = new double[gridNumber][gridNumber];
    public static double[][] expectedDemand = new double[gridNumber][gridNumber];
    public static double discountPercentage = .025;
    public static double priceRebate = 10;
    static Random randomGenerator = new Random();

    public static void applyDP(double costs, int customerID1, int customerID2){
        myPrice = /*roundtripDistance // costs * (MyRMcontrol.resultsRow[17]/(Main.C*.15)) + costs *.1*/ costs * (.25*MyRMcontrol.resultsRow[17]/Main.C) + costs *.75 ;
        // Forecast pricing
        if(MyRMcontrol.fileForecast == true) myPrice = returnDiscountedPriceFromForecast(myPrice, customerID1, customerID2, expectedDemand);
        // Pricing with fixed rebate:
        if(marketPrice > costs) myPrice=marketPrice;
        else if(marketPrice >= costs*((100-priceRebate)/100))myPrice=marketPrice;
        // Pricing up to direct distance costs:
        if(marketPrice >= directDistance)myPrice=marketPrice;
    }
}
```
// Set Price as no lower than (expected) market price:
if(marketPrice> myPrice) myPrice=marketPrice;

//double estimatedMarketPrice = directDistance * 0.7862;
if(estimatedMarketPrice> myPrice) myPrice=estimatedMarketPrice;

MyRMcontrol.resultsRow[10]=myPrice;
System.out.println("Market price = "+marketPrice+ " My costs = "+costs + " My price = "+myPrice);

public static boolean checkProfitability(){
    boolean accept = false;
    if(marginalCosts <= marketPrice(distance, competitionLevel)) accept=true;
    else System.out.println("Reject customer request");
    return accept;
}

public static double marketPrice(int NrOfUnits, double distance, int competitionLevel){
    double KMprice = 2;
    marketPrice = distance*KMprice - 20*competitionLevel + 90;
    System.out.println("Market price = "+marketPrice);
    return marketPrice;
}

determinePrice(double costs, double marketprice, double timeToExecution, double distance){
    double priceLb = costs;
    double priceUb = marketprice;
    double KMprice = 2;
    myPrice = /*distance*KMprice + costs;*/distance * KMprice + 50;
    if(myPrice > marketprice) myPrice = marketprice - 1;
    return myPrice;
}

determinePriceTraditionally(){
    double KMprice = 2;
    traditionalPrice = distance * KMprice + 50;
    System.out.println("Traditional price: "+traditionalPrice);
}

simulateCompetitionRandom(int customerID1, int customerID2){
    directDistance = MyData.distance[searchbyID(MyRMcontrol.fileMatrixCurrent, customerID1)][searchbyID(MyRMcontrol.fileMatrixCurrent, customerID2)];
    roundtripDistance = MyData.distance[0][searchbyID(MyRMcontrol.fileMatrixCurrent, customerID1)]
        + MyData.distance[searchbyID(MyRMcontrol.fileMatrixCurrent, customerID2)][0]
        + directDistance;
    System.out.println("Request DD = "+directDistance+" RTD = " + roundtripDistance);
    //marketPrice = directDistance + randomGenerator.nextDouble() * (roundtripDistance-directDistance);/** roundtripDistance+1;*/
    //randomno.nextGaussian()*(((rtd-dd)/4)+(dd+((rtd-dd)/2));
    marketPrice = randomGenerator.nextGaussian()*(roundtripDistance-directDistance)/2+directDistance+((roundtripDistance-directDistance)/2);
    MyRMcontrol.resultsRow[14]=marketPrice;
}

simulateCompetitionRegional(int customerID1, int customerID2){
    directDistance = MyData.distance[searchbyID(MyRMcontrol.fileMatrixCurrent, customerID1)][searchbyID(MyRMcontrol.fileMatrixCurrent, customerID2)];
    roundtripDistance = MyData.distance[0][searchbyID(MyRMcontrol.fileMatrixCurrent, customerID1)]
112             + MyData.distance[searchbyID(MyRMcontrol.fileMatrixCurrent, customerID2)][0]
113             + directDistance;
114             
115             marketPrice = /*roundtripDistance*//*/marketPriceList.get(customerID1);
116             
117             if(centerIsInSector(competitionSector, customerID1, customerID2)){
118                 marketPrice = marketPrice*.6; //directDistance*.8;
119                 System.out.println("At least one customer is in sector, market price = " +marketPrice);
120             }
121             else{
122                 //marketPrice = randomGenerator.nextGaussian()/*((roundtripDistance-
123                     directDistance)/2)+((roundtripDistance-directDistance)/2);
124                 System.out.println("No customer in sector, market price = "+marketPrice)
125             }
126             }  
127             MyRMcontrol.resultsRow[14]=marketPrice;
128             
129             public static void setCompetitionFromFile(int customerID1, int customerID2){
130                 directDistance = MyData.distance[searchbyID(MyRMcontrol.fileMatrixCurrent, customerID1)][0][0]+MyData.distance[searchbyID(MyRMcontrol.fileMatrixCurrent, customerID2)][0][0];
131                 roundtripDistance = MyData.distance[0][searchbyID(MyRMcontrol.fileMatrixCurrent, customerID1)][0]+MyData.distance[searchbyID(MyRMcontrol.fileMatrixCurrent, customerID2)][0]
132                     + directDistance;
133                 System.out.println("Request DD = "+directDistance+" RTD = "+roundtripDistance);
134                 marketPrice = marketPriceList.get(customerID1);
135                 MyRMcontrol.resultsRow[14]=marketPrice;
136             }
137             
138             static boolean isInSector(int competitiveSector, int customerID){
139                 boolean isinSector = false;
140                 System.out.println("x:"+MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[1]+"y:"+MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[2]);
141                 if(competitiveSector == 1 && MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[1] >= MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[2] && MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[2] >= MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[0][1] && MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[0][0] && MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[0][1] >= MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[2][1] && MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[2][0] >= MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[0][0])isinSector = true;
144                 if(competitiveSector == 3 && MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[1] >= MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[2] && MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[2] >= MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[0][1] && MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[0][0] && MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[0][1] >= MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[2][1] && MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[2][0] >= MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID))[0][0])isinSector = true;
146                 return isinSector;
147             }
148             
149             static boolean centerIsInSector(int competitiveSector, int customerID1, int customerID2){
150                 boolean isinSector = false;
151                 // determine centre:
152                 double xCust = (MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID1))[1] + MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID2))[1]) / 2;
153                 double yCust = (MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID1))[2] + MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID2))[2]) / 2;
154 }
System.out.println("cID1:"+customerID1 + " pos: "+ searchbyID(MyRMcontrol.fileMatrixCurrent, customerID1) + " = customer: "+ MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID1))[0]);
System.out.println("cID1:"+customerID2 + " pos: "+ searchbyID(MyRMcontrol.fileMatrixCurrent, customerID2) + " = customer: "+ MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID2))[0]);
System.out.println("sector:"+competitiveSector + " xCust: "+xCust + " yCust: "+yCust + " xDepot: "+MyRMcontrol.fileMatrixCurrent.get(0)[1]+ " yDepot: "+MyRMcontrol.fileMatrixCurrent.get(0)[2]);

if (xCust >= MyRMcontrol.fileMatrix.get(0)[1] && yCust <= MyRMcontrol.fileMatrix.get(0)[2]) {
  MyRMcontrol.resultsRow[16] = 1.0;
  System.out.println("A Is in sector "+ 1);
}
else if (xCust < MyRMcontrol.fileMatrix.get(0)[1] && yCust <= MyRMcontrol.fileMatrix.get(0)[2]) {
  MyRMcontrol.resultsRow[16] = 2.0;
  System.out.println("A Is in sector "+ 2);
}
else if (xCust < MyRMcontrol.fileMatrix.get(0)[1] && yCust > MyRMcontrol.fileMatrix.get(0)[2]) {
  MyRMcontrol.resultsRow[16] = 3.0;
  System.out.println("A Is in sector "+ 3);
}
else if (xCust >= MyRMcontrol.fileMatrix.get(0)[1] && yCust > MyRMcontrol.fileMatrix.get(0)[2]) {
  MyRMcontrol.resultsRow[16] = 4.0;
  System.out.println("A Is in sector "+ 4);
}
System.out.println("Z Is in sector "+ MyRMcontrol.resultsRow[16]);
if (competitiveSector == MyRMcontrol.resultsRow[16]) isInSector = true;
return isInSector;
}

static int searchbyID(List<Double[]> List, int ID){
  int item = 9999;
  for(int i = 0; i< List.size(); i++){
    if(List.get(i)[0] == ID) item = i;
  }
  return item;
}

//Forecasting:

static void createExpectedDemandMatrix(double stdDev) {
  System.out.println("Expected demand matrix:");
  for(int i=0; i<expectedDemand.length; i++) {
    for(int j=0; j<expectedDemand.length; j++) {
      expectedDemand[i][j] = 0;
    }
  }
  System.out.println("Exp dem matrix size: "+expectedDemand.length);
  System.out.println("gridSizeX: "+gridSizeX+ "gridSizeY: "+gridSizeY);
}


double sectorX = 0;
double sectorY = 0;
for(int i=1; i<MyRMcontrol.fileMatrixCurrent.size()-1; i++) {
  sectorX = numDiv(MyRMcontrol.fileMatrixCurrent.get(i)[1].intValue()+MyRMcontrol.fileMatrix.get(i+1)[1].intValue())/2, gridSizeX);
  sectorY = numDiv(MyRMcontrol.fileMatrixCurrent.get(i)[2].intValue()+MyRMcontrol.fileMatrix.get(i+1)[2].intValue())/2, gridSizeY);
  if(sectorX==1 && sectorY==2) System.out.println("secX "+sectorX+"secY: "+ sectorY);
  expectedDemand[(int)sectorX][(int)sectorY]++;
System.out.println("\n"+MyRMcontrol.fileMatrix.get(i)[0].intValue()+" "+MyRMcontrol.fileMatrix.get(i+1)[0].intValue()+" gridPos: "+ sectorX+" +sectorY;
        }
        System.out.println("Expected demand matrix:");
        for(int i=0; i<expectedDemand.length; i++) {
            for(int j=0; j<expectedDemand.length; j++) {
                System.out.print(expectedDemand[i][j]+"\t");
            }
            System.out.println();
        }
        System.out.println("Deviation matrix:");
        for(int i=0; i<deviationMatrix.length; i++) {
            for(int j=0; j<deviationMatrix.length; j++) {
                System.out.print(deviationMatrix[i][j]+"\t");
            }
            System.out.println();
        }
        System.out.println("Randomised expected demand matrix with stdDev: "+stdDev+"\n");
        for(int i=0; i<expectedDemand.length; i++) {
            for(int j=0; j<expectedDemand.length; j++) {
                expectedDemand[i][j] = /*randomGenerator.nextGaussian()*/ * stdDev*/deviationMatrix[i][j] + expectedDemand[i][j];
            }
            System.out.println();
        }
        
        public static void determineGridSize() {
            double xMin = Integer.MAX_VALUE;
            double xMax = Integer.MIN_VALUE;
            double yMin = Integer.MAX_VALUE;
            double yMax = Integer.MIN_VALUE;

            for(int i=0; i<MyRMcontrol.fileMatrix.size(); i++) {
                if ((MyRMcontrol.fileMatrix.get(i)[1] < xMin) 
                    xMin = MyRMcontrol.fileMatrix.get(i)[1];
                else if ((MyRMcontrol.fileMatrix.get(i)[1] > xMax) 
                    xMax = MyRMcontrol.fileMatrix.get(i)[1];
                if ((MyRMcontrol.fileMatrix.get(i)[2] < yMin) 
                    yMin = MyRMcontrol.fileMatrix.get(i)[2];
                else if ((MyRMcontrol.fileMatrix.get(i)[2] > yMax) 
                    yMax = MyRMcontrol.fileMatrix.get(i)[2];
            }

            gridSizeX = (int) ((xMax/gridNumber);
            gridSizeY = (int) ((yMax/gridNumber);
        }

        static double returnDiscountedPriceFromForecast(double costs2, int customerID1, int customerID2, double[][] expectedDemand) {
            double price = costs2;
            int sectorX = numDiv((MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID1))[1].intValue()
                +MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID2))[1].intValue()/*
                /2, gridSizeX);
            int sectorY = numDiv((MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID1))[2].intValue()
                +MyRMcontrol.fileMatrixCurrent.get(searchbyID(MyRMcontrol.fileMatrixCurrent, customerID2))[2].intValue()/*
                /2, gridSizeY);
            double returnDiscountedPriceFromForecast = 2625.75 * gridPos; /*
                +MyRMcontrol.fileMatrix.get(customerID2)[2].intValue() 
                /2, gridSize);"+ sectorX}
System.out.println("Y Centroid of 2 customer locations is at : "+ sectorY);

double discount = 0;

System.out.println("Adapted expected demand matrix before:");
for(int i=0; i<expectedDemand.length; i++) {
    for(int j=0; j<expectedDemand.length; j++) {
        System.out.print(expectedDemand[i][j]+" \t");
    }
    System.out.println();
}

System.out.println("customers in grid sec: "+expectedDemand[sectorX][sectorY]*discountPercentage);
if(expectedDemand[sectorX][sectorY]>0) discount = (expectedDemand[sectorX][sectorY]*discountPercentage);
System.out.println("Discount: "+discount+" with discount percentage: "+discountPercentage);

//MyRMcontrol.resultsRow[16] = (discount*100);
//MyRMcontrol.resultsRow = discount;
//remove request from expected demand matrix:
expectedDemand[sectorX][sectorY]=--;
System.out.println("Adapted expected demand matrix:");
for(int i=0; i<expectedDemand.length; i++) {
    for(int j=0; j<expectedDemand.length; j++) {
        System.out.print(expectedDemand[i][j]+" \t");
    }
    System.out.println();
}

// mx discount
if(discount > .2) discount = .2;
return price*(1-discount);

static void createDeviationMatrix(double stdDev) {
    //discountPercentage = .5;
    for(int i=0; i<deviationMatrix.length; i++) {
        for(int j=0; j<deviationMatrix.length; j++) {
            deviationMatrix[i][j] = 0;
        }
    }
    for(int i=0; i<deviationMatrix.length; i++) {
        for(int j=0; j<deviationMatrix.length; j++) {
            deviationMatrix[i][j] = randomGenerator.nextGaussian() * stdDev;
        }
    }
    System.out.println();
}

public static void simulateRequest(){
    distance = randomGenerator.nextDouble()*100;
    marginalCosts = distance;
}

public static int numDiv(int a, int b) {
    int result = 0;
    System.out.println("a "+a+" b: "+b);
    while (a>b) {
        a-=b;
        result++;
    }
    return result;
}
public class MyMoveManager implements MoveManager

    { public static int operator;
    public static double[] opDistributionStandard = /*{0, 0, 70, 23, 4, 0, 2, 0, 1}; */ /*{0, 100, 68, 23, 0, 0, 2, 0, 0, 0}; */ /*{0, 2, 60, 3, 2, 2, 2, 2, 0, 2}
, 20, 7}; */
    public static double[] opDistributionDiversificate = {0, 50, 0, 0, /*5* /0, 0, 0, 0, 0, 0, 0, 0, 50, 10};
    public static double[] opDistributionLocalImp = /*for lc202*/ /*{0, 2, 70, 3, 2, 2, 0, 2, 10, 7}; */
    public static double[] opDistributionDiversificate = {0, 2, 60, 3, 2, 2, 2, 0, 2, 20, 7};
    public static double[] opDistributionDiversificate = {0, 2, 60, 3, 2, 2, 2, 0, 2, 20, 7};
    public static double[] opDistributionDiversificate = {0, 2, 60, 3, 2, 2, 2, 0, 2, 20, 7};
    public static double[] opDistributionDiversificate = {0, 2, 60, 3, 2, 2, 2, 0, 2, 20, 7};
    public static double[] opDistributionDiversificate = {0, 2, 60, 3, 2, 2, 2, 0, 2, 20, 7};
    public static double[] opDistributionDiversificate = {0, 2, 60, 3, 2, 2, 2, 0, 2, 20, 7};
    public static double[] opDistributionDiversificate = {0, 2, 60, 3, 2, 2, 2, 0, 2, 20, 7};
public Move[] getAllMoves( Solution solution ) { // for W
    if(MyRMcontrol.ONLYvrp==false)Arrays.copyOf(opDistributionLocalImp, opDistributionLocalImp.length);
    int[][] w = MySolution.w;
    Move[] buffer = new Move[ w.length*w.length ];
    List<Integer> singleCtoMove = new ArrayList<Integer>();
    int nextBufferPos = 0;
    Random randomGenerator = new Random();
    int exit=0;
    int i = 1;
    int j = 1;
    operator=0;
    double opSelection= randomGenerator.nextInt(100) + .01;
    if(MyRMcontrol.fileMatrixCurrent.size()<=4) {
        Move[] moves = new Move[ nextBufferPos ];
        System.arraycopy( buffer, 0, moves, 0, nextBufferPos );
        return moves;
    }
    else if (opSelection <= opDistribution[0]) {
        operator=2; // Swap Pair
        return moves;
    } else if (opSelection > opDistribution[0]) {
        opSelection = accumulateArrayVal(opDistribution, 0); // Swap Pair with TWs > TWe of followers
        if(opSelection > accumulateArrayVal(opDistribution, 1)) opSelection = accumulateArrayVal(opDistribution, 2); // Swap C in Route (distance)
        if(opSelection > accumulateArrayVal(opDistribution, 2)) operator= 3; // Swap C in Route (TW)
        if(opSelection > accumulateArrayVal(opDistribution, 3)) operator= 4; // Swap Pair to closest route center (GEO)
        if(opSelection > accumulateArrayVal(opDistribution, 4)) operator= 5; // Postpone single C?
        if(opSelection > accumulateArrayVal(opDistribution, 5)) operator= 6; // Postpone C?...
        if(opSelection > accumulateArrayVal(opDistribution, 6)) operator= 7; // Steep angle
        if(opSelection > accumulateArrayVal(opDistribution, 7)) operator= 8; // PD Exchange
        if(opSelection > accumulateArrayVal(opDistribution, 8)) operator= 9; // XExchange
        if(opSelection > accumulateArrayVal(opDistribution, 9)) operator= 10; // XExchange
        if(opSelection > accumulateArrayVal(opDistribution, 10)) operator= 11; // XExchange
        if(opSelection > accumulateArrayVal(opDistribution, 11)) operator= 12; // Postpone large TW end customer
        if(opSelection > accumulateArrayVal(opDistribution, 12)) operator= 13; // SwapPairRandIncr
        if(opSelection > accumulateArrayVal(opDistribution, 13)) {
            operator= 14; // Shaw removal
        }
    }
// int q = */5;}//4 /* randomGenerator.nextInt(6)-1*/;
0094 //
0095 i = removedCustomers.get(0).get(0).intValue();
0096 j = removedCustomers.get(0).get(1).intValue();
0097 buffer[nextBufferPos++] = new MySwapMove( i, j, operator );
0098 */Worst removal */ else if (opSelection > accumulateArrayVal(opDistribution, 13) && opSelection <= accumulateArrayVal(opDistribution, 14)) {
0099 operator= 15; // Shaw removal
0100 //
0101 int q = */5;}//4 /* randomGenerator.nextInt(6)-1*/;
0102 shawRemoval(q, 4);
0103 i = removedCustomers.get(0).get(0).intValue();
0104 j = removedCustomers.get(0).get(1).intValue();
0105 buffer[nextBufferPos++] = new MySwapMove( i, j, operator );
0106//
0107 else if (opSelection > accumulateArrayVal(opDistribution, 14) && opSelection <= accumulateArrayVal(opDistribution, 15)) {
0108 operator= 16; // Shaw removal
0109 //
0110 int q = */5;}//4 /* randomGenerator.nextInt(6)-1*/;
0111 worstRemovalLocal(q, 1, true);
0112 i = removedCustomers.get(0).get(0).intValue();
0113 j = removedCustomers.get(0).get(1).intValue();
0114 buffer[nextBufferPos++] = new MySwapMove( i, j, operator );
0115//
0116 else if (opSelection > accumulateArrayVal(opDistribution, 15) && opSelection <= accumulateArrayVal(opDistribution, 16)) {
0117 operator= 17; // Shaw removal
0118 //
0119 int q = */5;}//4 /* randomGenerator.nextInt(6)-1*/;
0120 worstRemoval(q, 1);
0121 if(removedCustomers.size()>0) {
0122 i = removedCustomers.get(0).get(0).intValue();
0123 j = removedCustomers.get(0).get(1).intValue();
0124 buffer[nextBufferPos++] = new MySwapMove( i, j, operator );
0125 }
0126 //
0127 else if (opSelection > accumulateArrayVal(opDistribution, 17) && opSelection <= accumulateArrayVal(opDistribution, 18)) {
0128 operator= 19; // Shaw removal
0129 //
0130 int q = */5;}//4 /* randomGenerator.nextInt(6)-1*/;
0131 worstRemoval(q, 1);
0132 worstRemovalByRoute(q, 1, false);
0133 i = removedCustomers.get(0).get(0).intValue();
0134 j = removedCustomers.get(0).get(1).intValue();
0135 buffer[nextBufferPos++] = new MySwapMove( i, j, operator );
0136 //
0137 else if (opSelection > accumulateArrayVal(opDistribution, 18) && opSelection <= accumulateArrayVal(opDistribution, 19)) {
0138 operator= 20; // Shaw removal
0139 //
0140 randomRemoval("Math.max(1,(int)(MyRMcontrol.fileMatrixCurrent.
0141 size()/20))*/ q);
0142 // eliminateRoute();
0143 System.out.println("c: "+i+ " m: "+j+ " o: "+operator);
0144 i = removedCustomers.get(0).get(0).intValue();
0145 j = removedCustomers.get(0).get(1).intValue();
0146 buffer[nextBufferPos++] = new MySwapMove( i, j, operator );
0147 */
operator= 21; // Shaw removal
randomRemoval(q);
if(removedCustomers.size()>0) {
    i = removedCustomers.get(0).get(0).intValue();
    j = removedCustomers.get(0).get(1).intValue();
    buffer[nextBufferPos++] = new MySwapMove( i, j, operator );
}
}
else{
    operator= 22; // Shaw removal
    randomRemoval(/*Math.max(1,(int)(MyRMcontrol.fileMatrixCurrent.size()/20))*/ q);
    eliminateRoute();
    i = removedCustomers.get(0).get(0).intValue();
    j = removedCustomers.get(0).get(1).intValue();
    buffer[nextBufferPos++] = new MySwapMove( i, j, operator );
}
}
}
}
}
Operator.fourDto2DListe(MySolution.w); //I do not need this all the time...check
//System.out.println("OPERATOR: "+operator);
if(operator == 1){ // New 04/04/17:
    findFurthestList1(w);
    if(highDistCustomers.size()>0){
        i = highDistCustomers.get(randomGenerator.nextInt(highDistCustomers.size()));
        if(i!=j /**/ & MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, i)==false /**/){
            buffer[nextBufferPos++] = new MySwapMove( i, j, operator );
        }
    }
    //Random customer operator// SwapPair operator:
    if(operator == 8){
        i = (randomGenerator.nextInt(MyRMcontrol.fileMatrixCurrent.size()-1)+1);
        if(MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, i )==false){
            //System.out.println("Operator 8, move customer "+i);
            buffer[nextBufferPos++] = new MySwapMove( i, 1, operator );
        }
    }
    else if(operator==2){
        System.out.println("Operator 2 ");
    }
}
for(int k=0; k< Main.k; k++){
    double timex = 0;
    Q: for(int q=1; q<w.length; q++){
        for(int ii=0; ii<w.length; ii++){
            for(int jj = 0; jj<Main.V; jj++){
                if(w[ii][jj][q][k]==1){
                    timex+=w[ii][jj][q][k] * (MyData.distance[ii][jj]/Main.v);
                    if(timex<MyData.TWstart[jj]) timex=MyData.TWstart[jj];
                }
            }
        }
    }
    if(timex > MyData.TWend[jj]){  
        singleCtoMove.add(ii);
        singleCtoMove.add(jj);
        //System.out.println("customer added to move list: "+ii + " "+jj);
0204     break Q;
0205   }
0206   else{
0207     timex+= MyData.Service[jj];
0208   }
0209  
0210  }
0211  }
0212  }
0213  }
0214  if(singleCtoMove.size()>0)i=singleCtoMove.get(randomGenerator.nextInt(singleCtoMove.size()));
0215  //System.out.println("i: "+i + "length= "+singleCtoMove.size());
0216  /**\*if(MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, i)==false)\*/buffer[nextBufferPos++]= new MySwapMove( i, 2, 2);
0217  }
0218  
0219  else if(operator==3 || operator ==4|| operator ==10|| operator ==11){
0220  // System.out.println("Operator 3/4/10/11 ");
0221  /*new 24/04/17:*/
0222  if(MyObjectiveFunction.customersToMove.size() < 5){
0223     /*29/04/17*/findFurthestInList();
0224     MyObjectiveFunction.customersToMove.addAll(highDistCustomers);
0225  }
0226  }
0227  System.out.println("ctmList: ");
0228  for(int l=0; l<MyObjectiveFunction.customersToMove.size();l++){
0229     System.out.println(MyObjectiveFunction.customersToMove.get(l)+" ");
0230  }
0231  System.out.println();
0232  
0233  if(MyObjectiveFunction.customersToMove.size() != 0 && MyObjectiveFunction.customersToMove.size()<=31 /*//*4050*/){
0234      int choice2 = randomGenerator.nextInt(100);
0235      if(choice2 <=80){
0236        System.out.println("c2m size: "+MyObjectiveFunction.customersToMove.size());
0237        if(operator == 3 || operator == 10)i = MyObjectiveFunction.customersToMove.get(randomGenerator.nextInt(MyObjectiveFunction.customersToMove.size()));
0238      }else{
0239        operator=3;  // changed for lrc
0240      }
0241    }
0242    }else if(choice2 > 80 && choice2 <=95){
0243      boolean i=false; //new 24/04/17
0244      findFurthestList1(w);
0245      findFurthestInList();
0246      findHighDistTWrelation(w); //test 160815
0247      if(highDistCustomers.size()>0){
0248        i = highDistCustomers.get(randomGenerator.nextInt(highDistCustomers.size()));
0249      }else{
0250        i = (randomGenerator.nextInt(w.length-1)+1);
0251        while(MySolution.startingPositions.contains(i)==true && exit< 50)
0252          {
i = (randomGenerator.nextInt(w.length-1)+1);
exit++;
}
}
else{
operator=3; // changed for lrc //back
int choice2 = randomGenerator.nextInt(100);
if(choice2 <=90){
  boolean i\_j = randomGenerator.nextBoolean();
  //findFurthestList1(w);
  findFurthestInList();
  findHighDistTWrelation(w); //test 160815
} if(highDistCustomers.size()>0){
i = highDistCustomers.get(randomGenerator.nextInt(highDistCustomers.size()));
}
exit++;
}
else{
operator=3; // changed for lrc
}
}
else{
i = (randomGenerator.nextInt(w.length-1)+1);
while(MySolution.startingPositions.contains(i)==true && exit<50)
{
i = (randomGenerator.nextInt(w.length-1)+1);
exit++;
}
}
else{
operator=3; // changed for lrc
}
else if(operator==5){ //Swap Pair GEO
  System.out.print("Operator 5 ");
  i=0;
  //introduced to avoid operator if no "i" was found
  Operator.fourDto2DListe(w);
  double[] centroidX = new double[Operator.wLists.size()], centroidY = new double[Operator.wLists.size()];
  for(int k=0; k<Operator.wLists.size(); k++){
    int nrOfKnots =0;
    for(int pos=0; pos<Operator.wLists.get(k).size(); pos++){
      centroidX[k] += MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos))[1];
      centroidY[k] += MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos))[2];
    }
    nrOfKnots++;
  }
  List<Integer[]> customerToMoveList = new ArrayList<Integer[]>();
double x1;
double y1;
double currentHigh = Integer.MIN_VALUE;
double distToCentroid = 0;
double distToOther = 0;
int futureK = 1;
int c2;
// Integer[] newSet = (0,0);
for(int k=0; k<Operator.wLists.size(); k++){
    for(int pos=/*0*/1; pos<Operator.wLists.get(k).size(); pos++){
        //check 04/04/17: before was tried to be moved
        if(MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos))[8]>0){
            //System.out.print("check sender "+ Operator.wLists.get(k).get(pos));
            c2 = MySolution.searchibyID(MyRMcontrol.fileMatrixCurrent, MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos))[8].intValue());
            if(c2==9999) break;
            x1 = ( MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos))[1] + MyRMcontrol.fileMatrixCurrent.get(c2)[1] )/2;
            y1 = ( MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos))[2] + MyRMcontrol.fileMatrixCurrent.get(c2)[2] )/2;
            distToCentroid = MyRMcontrol.norm(x1, y1, centroidX[k], centroidY[k]);
            double minDistToOther = 1000;
distToOther= 1000;
            Integer[] newSet = /*new Integer [2]*/{0,0};
            for(int kk=0; kk<Main.k; kk++){
                distToOther = MyRMcontrol.norm(x1, y1, centroidX[kk], centroidY[kk]);
                if(distToCentroid > (distToOther*.95) && distToOther < minDistToOther && k!=kk){
                    minDistToOther = distToOther;
                    //System.out.print(" new min to other: "+ distToOther);
                    newSet[0] = Operator.wLists.get(k).get(pos);
                    newSet[1] = kk;
                }
            }
            //System.out.println();
            //add to list
            if(newSet[0]!=0) customerToMoveList.add(newSet);
        }
    }
}
if(customerToMoveList.size()>0){
    int rand = randomGenerator.nextInt(customerToMoveList.size());
    i= customerToMoveList.get(rand)[0];
    j= customerToMoveList.get(rand)[1];
}
//System.out.println("GEO: "+i+" to k: "+ j);
if(i != 0 /*&& MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, i)==false*/ && MyRMcontrol.fileMatrixCurrent.get(i)[8]>0) buffer[nextBufferPos++]= new MySwapMove( i, j, operator );
else if(operator==6){
    System.out.print("Operator 6 ");
}
double oldDist;

double newDist;

int zaehler = 0;

K: while(i==1 && zaehler < 4){
   int k = /*randomGenerator.nextInt(Main.k)*/ selectK();

   double oldDist = MyData.distance[Operator.wLists.get(k).get(pos-1)][Operator.wLists.get(k).get(pos)]
   + MyData.distance[Operator.wLists.get(k).get(pos)][Operator.wLists.get(k).get(pos+1)]
   + MyData.distance[Operator.wLists.get(k).get(pos)][Operator.wLists.get(k).get(pos+2)];

   if(newDist < oldDist && Operator.wLists.get(k).get(pos+1) != MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos)) [8].intValue() ){
      i = Operator.wLists.get(k).get(pos);
      j = pos+1;
   } else if(MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, i)==false) buffer[nextBufferPos++] = new MySwapMove( i, j, operator);

   if(MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, i)==false) buffer[nextBufferPos++] = new MySwapMove( i, j, operator);

   for(int pos=1; pos<Operator.wLists.size() ; pos++){
      line1 = new Line2D.Float((MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos)) [1]).floatValue(),
                              (MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos)) [2]).floatValue(),
                           (MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos+1)) [1]).floatValue(),
                           (MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos+1)) [2]).floatValue());
      line2 = new Line2D.Float((MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos+1)) [0]).floatValue(),
                              (MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos+2)) [1]).floatValue(),
                           (MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos+2)) [2]).floatValue(),
                           (MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos+3)) [1]).floatValue());
241
0423 // if(currentSteepest>angleBetween2Lines(line1, line2)){
0424 // currentSteepest = angleBetween2Lines(line1, line2);
0425 // i = Operator.wLists.get(k).get(pos+1);
0426 // j = pos+2;
0427 // }
0428 if((currentSteepest>angleBetween2Lines(line1, line2) ^ angleBetween2Lines(line1, line2) < minAngle){
0429 singleCtoMove.add(0, Operator.wLists.get(k).get(pos+1));
0430 if(singleCtoMove.size()>3)singleCtoMove.remove(singleCtoMove.size()-1);
0431 }
0432 }
0433 if((currentSteepest==0) buffer[nextBufferPos++] = new MySwapMove(1, 3, 3);
0435 // else if(!i==0) buffer[nextBufferPos++] = new MySwapMove(i, j, operator);
0436 // else System.out.println("SingleCtoMove.size(): "+singleCtoMove.size());
0437 if((singleCtoMove.size()>0 && !MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, i)==false)buffer[nextBufferPos++] = new MySwapMove( singleCtoMove.get(randomGenerator.nextInt(singleCtoMove.size())), 3, operator); //3
0439 else if(operator == 9){ // CROSS-EXCHANGE
0440 System.out.println("Operator 9");
0441 int i2 = 1;
0442 boolean found = false;
0443 int intersectionLength;
0444 int intersectionMin;
0445 Operator.fourDto2DListe(w);
0446 K: for(int kk=0; kk<Operator.wLists.size();kk++){
0447 int k = randomGenerator.nextInt(Main.k); //um nicht am gleichen fehler haengen zu bleiben
0448 //System.out.println("start searching route "+k);
0449 intersectionMin = Integer.MAX_VALUE;
0450 for(int pos=0; pos<Operator.wLists.get(k).size()-3; pos++)
0451 Line2D linel = new Line2D.Float((MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos)))[1]).floatValue(),
0452 (MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos+1))).floatValue(),
0453 (MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos+2))).floatValue(),
0454 (MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos+3))).floatValue();
0457 //for(int pos2=pos2; pos2<Operator.wLists.get(k).size()-1; pos2++)
0459 for(int pos2=Operator.wLists.get(k).size()-1; pos2>pos2; pos2--)
0460 Line2D line2 = new Line2D.Float((MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos2)))[1]).floatValue(),
0461 (MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos2+1))).floatValue(),
0462 (MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos2+2))).floatValue(),
0463 (MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos2+3))).floatValue();
0464 if(line2.intersectsLine(linel) == true && MyData.distance[Operator.wLists.get(k).get(pos+1)]<operator.wLists.get(k).get(pos2)=

241
if (intersectionLength < intersectionMin /*&& MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos2))[3]>0 && MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos+1))[8] != MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos2-1))[0]*/){
    intersectionMin = intersectionLength;
    System.out.println("intersection length: "+intersectionLength);
}

i = (MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos2))[0]).intValue(); //check if pos2-1 works better (there was a case where this did not work!

intersectionMin = intersectionLength;

//System.out.println("Intersection length: "+intersectionLength);

i = Operator.wLists.get(k).get(pos);

i2 = (MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos2-1))[0]).intValue();

j = pos+1; //all abgeschnitten for Xex2 test

i = Operator.wLists.get(k).get(pos);

j = Operator.wLists.get(k).get(pos2);

found = true;

System.out.println(pos+" - " + (pos+1)+" intersects with "+pos2+" - " + (pos2-1)+" in k: "+k);
System.out.println("pos = "+i+" pos 2 = "+j);
break K;

}//for k

if (intersectionMin < 50) break K;

System.out.println("Move 9 sent");

if (i != j /**& MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, i)==false**/ buffer[nextBufferPos++] = new MySwapMove( i, j, operator );

//Test ob mit op 3/4 besser:
buffer[nextBufferPos++] = new MySwapMove( 89, 40, operator );
buffer[nextBufferPos++] = new MySwapMove( 89, 40, operator );

//buffer[nextBufferPos++] = new MySwapMove( i2, 9, 3);

double minDistance = Integer.MAX_VALUE;

for (int pos2=Operator.wLists.get(k).size()-1; pos2>Math.max(pos+1, Operator.wLists.get(k).size()-5); pos3--){
    double thisDist = MyData.distance[Operator.wLists.get(k.get(pos3)][

if (thisDist < minDistance)
    i = Operator.wLists.get(k).get(pos2);
    minDistance = thisDist;
    futurePos = pos3-1; //respects taking c from prev pos
    System.out.println("Op12 barrier: "+Operator.wLists.get(k).get(pos) +" move i: "+i+" with min dist: "+minDistance+" to pos "+futurePos);
else 
    
//System.out.println("Op12 barrier: "+Operator.wLists.get(k).get(pos) +" move i: "+i+" with min dist: "+minDistance+" to pos "+futurePos);
    
if (futurePos!= 0 /*MyObjectiveFunction.customersToMove.size()<7*/)
    buffer[nextBufferPos++] = new MySwapMove(i, futurePos, operator);
    /* else buffer[nextBufferPos++] = new MySwapMove(i, futurePos, 3 );*/
    else if(operator==13) //Swap Pair randCL1 to best incr.
        i=0; //introduced to avid operator if no "i" was found
        
if (MyObjectiveFunction.customersToMove.size() != 0){
    i=MyObjectiveFunction.customersToMove.get(randomGenerator.nextInt(MyObjectiveFunction.customersToMove.size()));
    
    while(MySolution.customersToStayInK.contains(i)==true && exit<50){
        i=MyObjectiveFunction.customersToMove.get(randomGenerator.nextInt(MyObjectiveFunction.customersToMove.size()));
        exit++;
    }
    for(int jj=1; jj<Main.V; jj++){
        if(MyData.demand[i][jj]>0 && MySolution.startingPositions.contains(i)==false && MySolution.startingPositions.contains(jj)==false)
            buffer[nextBufferPos++] = new MySwapMove(i, jj, operator);
    }
    for(int jj=1; jj<Main.V; jj++){
        if(MyData.demand[i][jj]>0 && MySolution.startingPositions.contains(i)==false && MySolution.startingPositions.contains(jj)==false)
            buffer[nextBufferPos++] = new MySwapMove(jj, i, operator);
    }
}

// Trim buffer
Move[] moves = new Move[ nextBufferPos ];
System.arraycopy( buffer, 0, moves, 0, nextBufferPos );
return moves;
} // end getAllMoves

// new removal heuristics:
public void shawRemoval(int q, int p){
    System.out.println("Shaw removal heuristic with q = "+q);
    removedCustomers.clear();
    int r = 0;
    int ri;
    int rj;
while(r==0 || MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, r)==true || MySolution.customersToStayInK.contains(r)==true){
    r = (randomGenerator.nextInt(MyRMcontrol.fileMatrixCurrent.size()-1)+1);
    // r = MyObjectiveFunction.customersToMove.get(randomGenerator.nextInt(MyObjectiveFunction.customersToMove.size()));
}
if(MyRMcontrol.fileMatrixCurrent.get(r)[0].intValue() > 0 && MyRMcontrol.fileMatrixCurrent.get(r)[3].intValue() > 0 && MyRMcontrol.fileMatrixCurrent.get(r)[8].intValue() > 0){
    double rVal;
    double phi = 9/6/*6*/;
    double chi = 3/*6*/;
    double psi = 2;
    // double omega = 5;
    double y/* = randomGenerator.nextDouble()*/;
    double yExpP /*= (int) Math.pow(y, p)*/;
    MyObjectiveFunction.evaluateQuick(false);
    for(int ii=0; ii<MyRMcontrol.fileMatrixCurrent.size()-1; ii++){
        if(MyRMcontrol.fileMatrixCurrent.get(ii)[0].intValue() > 0 && MyRMcontrol.fileMatrixCurrent.get(ii)[3].intValue() > 0 && MyRMcontrol.fileMatrixCurrent.get(ii)[8].intValue() > 0){
            i=ii;
            j=ii+1;
            System.out.println(MyData.distance.length + " i "+i" j "+j + " ri "+ri + " rj "+rj);
        } else{
           京津冀
            System.out.println("rVal: "+ rVal);
            System.out.println("rVal+= rVal*Math.pow(randomGenerator.nextDouble(), p); p = randomGenerator.nextInt(6)+1; y = randomGenerator.nextDouble(); yExpP = (int) Math.pow(y, p); rVal **= (1-yExpP);
        } System.out.println("rVal+= rVal*Math.pow(randomGenerator.nextDouble(), p); p = randomGenerator.nextInt(6)+1; y = randomGenerator.nextDouble(); yExpP = (int) Math.pow(y, p); rVal **= (1-yExpP);
        System.out.println("rVal+= rVal*Math.pow(randomGenerator.nextDouble(), p); p = randomGenerator.nextInt(6)+1; y = randomGenerator.nextDouble(); yExpP = (int) Math.pow(y, p); rVal **= (1-yExpP);
        System.out.println("rVal+= rVal*Math.pow(randomGenerator.nextDouble(), p); p = randomGenerator.nextInt(6)+1; y = randomGenerator.nextDouble(); yExpP = (int) Math.pow(y, p); rVal **= (1-yExpP);
        System.out.println("removedCustomers.add(new ArrayList<Double>(Arrays.asList((double) i, (double) j, rVal, 1.0, 2.0, 0.0))));
    }
Note: r - random does not have to be added as it scores the lowest R(ij) anyway
removedCustomers.add(new ArrayList<Double>(Arrays.asList((double) rj, 0.0, 0.0, 0.0, 0.0)));

Collections.sort(removedCustomers, new Comparator<ArrayList<Double>>() {
  private static final int INDEX = 2;
  public int compare(ArrayList<Double> o1, ArrayList<Double> o2) {
    return Double.compare(o1.get(INDEX), o2.get(INDEX));
  }
});
removedCustomers.subList(Math.min(q, removedCustomers.size()), removedCustomers.size()).clear();

for (int is = 0; is<removedCustomers.size(); is++) {
  System.out.println(removedCustomers.get(is).toString());
}
}

public void worstRemoval(int q, int p){
  //System.out.println("Worst removal heuristic with q = "+q);
  removedCustomers.clear();
  double savings;
  double y/* = randomGenerator.nextDouble()*/;
  double yExpP /*= (int) Math.pow(y, p)*/;
  Operator.fourDto2DListePlusReturn(MySolution.w);
  double currentCosts = MyObjectiveFunction.evaluateQuick(true);
  double newCosts;
  for(int r=1; r<MyRMcontrol.fileMatrixCurrent.size(); r++){
    if(MyRMcontrol.fileMatrixCurrent.get(r)[3]>0){
      i=MyRMcontrol.fileMatrixCurrent.get(r)[0].intValue();
      j=MyRMcontrol.fileMatrixCurrent.get(r)[8].intValue();
      //test for dynamic:
      i=r;
      j=r+1;
      //create wLists
      Operator.fourDto2DListePlusReturn(MySolution.w);
      //method: remove customer pair from wlists
      Operator.RemoveSingleFromList(i);
      Operator.RemoveSingleFromList(j);
      //randomizer:
      p = randomGenerator.nextInt(3)+1;
      y = randomGenerator.nextDouble();
      yExpP = (int) Math.pow(y, p);
      // evaluate new wlists
      newCosts = MyObjectiveFunction.evaluateQuick(true);
      // calculate c- c_ -ij
      savings = (currentCosts - newCosts) * (1-yExpP);
      removedCustomers.add(new ArrayList<Double>(Arrays.asList((double) i, (double) j, savings, 0.0, 0.0, 0.0)));
    }
  }
  // Select only a fraction of customers by random:
  selectNrOfCustomerPairs2(25, currentCosts);
Collections.sort(removedCustomers, new Comparator<ArrayList<Double>>()
{
    private static final int INDEX = 2;
    public int compare(ArrayList<Double> o1, ArrayList<Double> o2) {
        return Double.compare(o1.get(INDEX), o2.get(INDEX))*(-1);
    }
});
for (int is = 0; is<removedCustomers.size(); is++) {
    System.out.println(removedCustomers.get(is).toString());
}
removedCustomers.subList(q, removedCustomers.size()).clear();

worstRemovalByRoute(int q, int p, boolean penalty)
{
    removedCustomers.clear();
    double savings;
    double y/* = randomGenerator.nextDouble()*/;
    double yExpP /* = (int) Math.pow(y, p)*/;
    Operator.fourDto2DListePlusReturn(MySolution.w);
    //replacement:
    int r;
    double currentCosts;
    double newCosts;
    int[] ijPos;
    for(int k=0; k<Operator.wLists.size(); k++) {
        currentCosts = MyObjectiveFunction.evaluateSingleRouteQuick(k, penalty);
        for(int l=1; l< Operator.wLists.get(k).size()-1; l++) {
            System.out.print(Operator.wLists.get(k).get(l)+" ");
            r = Operator.wLists.get(k).get(l);
            if(MyRMcontrol.fileMatrixCurrent.get(r)[3]>0){
                i=MyRMcontrol.fileMatrixCurrent.get(r)[0].intValue();
                j=MyRMcontrol.fileMatrixCurrent.get(r)[8].intValue();
                //test for dynamic:
                i=r;
                j=r+1;
            }
            // method: remove customer pair from wlists
            ijPos = Operator.removePairFromListnReturnPos(i, j, k);
            //randomizer:
            p = randomGenerator.nextInt(3)+1;
            y = randomGenerator.nextDouble();
            yExpP = (int) Math.pow(y, p);
            // evaluate new wlists
            newCosts = MyObjectiveFunction.evaluateSingleRouteQuick(k, penalty);
            //rebuild previous wLists:
            Operator.wLists.get(k).add(ijPos[0], i);
            Operator.wLists.get(k).add(ijPos[1], j);
            // calculate c- c_ij
            savings = (currentCosts - newCosts) * (1-yExpP);
removedCustomers.add(new ArrayList<Double>(Arrays.asList((double) i, (double) j, savings, 1.0, 2.0, 0.0)));

Collections.sort(removedCustomers, new Comparator<ArrayList<Double>>() {
    private static final int INDEX = 2;
    public int compare(ArrayList<Double> o1, ArrayList<Double> o2) {
        return Double.compare(o1.get(INDEX), o2.get(INDEX)) * (-1);
    }
});
removedCustomers.subList(Math.min(q, removedCustomers.size()), removedCustomers.size()).clear();

public void worstRemovalLocal(int q, int p, boolean penalty) {
    //System.out.println("Worst removal single route q = "+q);
    removedCustomers.clear();
    double savings;
    double y;
    double yExpP;
    Operator.fourDto2DListePlusReturn(MySolution.w);
    int r;
    double currentCosts;
    double newCosts;
    int[] ijPos;
    int k = selectK();
    currentCosts = MyObjectiveFunction.evaluateSingleRouteQuick(k, penalty);
    for(int l=1; l< Operator.wLists.get(k).size()-1; l++) {
        r = Operator.wLists.get(k).get(l);
        if(MyRMcontrol.fileMatrixCurrent.get(r)[3]>0){
            i = MyRMcontrol.fileMatrixCurrent.get(r)[0].intValue();
            j = MyRMcontrol.fileMatrixCurrent.get(r)[8].intValue();
        }
        //test for dynamic:
        i=r;
        j=r+1;
        // method: remove customer pair from wlists
        ijPos = Operator.removePairFromListnReturnPos(i, j, k);
        //randomizer:
        p = randomGenerator.nextInt(3)+1;
        y = randomGenerator.nextDouble();
        yExpP = (int) Math.pow(y, p);
        // evaluate new wlists
        newCosts = MyObjectiveFunction.evaluateSingleRouteQuick(k, penalty);
        //rebuild previous wLists:
        Operator.wLists.get(k).add(ijPos[0], i);
Operator.wLists.get(k).add(ijPos[1], j);

// calculate c= c_ij
savings = (currentCosts - newCosts) * (1-yExpP);

removedCustomers.add(new ArrayList<Double>(Arrays.asList((double) i, (double) j, savings, 1.0, 2.0, 0.0)));
}

Collections.sort(removedCustomers, new Comparator<ArrayList<Double>>()
{
    private static final INDEX = 2;
    public int compare(ArrayList<Double> o1, ArrayList<Double> o2) {
        return Double.compare(o1.get(INDEX), o2.get(INDEX))*(-1);
    }
});

for (int is = 0; is<removedCustomers.size(); is++) {
    System.out.println(removedCustomers.get(is).toString());
}

if(q<removedCustomers.size()-2)removedCustomers.subList(q, removedCustomers.size()).clear();

System.out.println("Customers to be removed C1:");
for (int is = 0; is<removedCustomers.size(); is++) {
    System.out.println(MyRMcontrol.fileMatrixCurrent.get(removedCustomers.get(is).get(0).intValue()));
}

public static void randomRemoval(int q){
    System.out.println("Random removal heuristic with q: "+q);
    removedCustomers.clear();
    int r;
    int counter = 0;
    while(removedCustomers.size() < q ^ counter< MyRMcontrol.fileMatrixCurrent.size()){
        r= (int) (Math.random()*(MyRMcontrol.fileMatrixCurrent.size()-1)+1);
        if(checkIfAlreadyIncluded(r) ==false
            && MyRMcontrol.fileMatrixCurrent.get(r)[8]>0
            && MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, r)==false
            && MySolution.customersToStayInK.contains(r)==false ){
            i=r;
            j=r+1;
            System.out.println("RR i: "+i+ " j: "+j);
        removedCustomers.add(new ArrayList<Double>(Arrays.asList((double) i, (double) j, 0.0, 1.0, 2.0, 0.0)));
    }
    counter ++;
}

int shortestRoute = 0;
int shortestRouteSize = Integer.MAX_VALUE;
for(int k=0; k<Operator.wLists.size(); k++) {
if (Operator.wLists.get(k).size() >= 4 && Operator.wLists.get(k).size() < shortestRouteSize)
    shortestRoute = k;
    shortestRouteSize = Operator.wLists.get(k).size();
    */
    System.out.println("New shortest route size : "+shortestRouteSize);
    */
    shortestRoute = k;
    shortestRouteSize = Operator.wLists.get(k).size();
    shortes

double newCosts;

id = randomGenerator.nextInt(MyRMcontrol.fileMatrixCurrent.size());

while(removedCustomers.size()<amountOfPairs)

if(MyRMcontrol.fileMatrixCurrent.get(id)[3]>0{
  i=MyRMcontrol.fileMatrixCurrent.get(id)[0].intValue();
  j=MyRMcontrol.fileMatrixCurrent.get(id)[8].intValue();

  // create wlists
  Operator.fourDto2DListePlusReturn(MySolution.w);

  // method: remove customer pair from wlists
  Operator.RemoveSingleFromList(i);
  Operator.RemoveSingleFromList(j);

  // evaluate new wlists
  newCosts = MyObjectiveFunction.evaluateQuick(true);

  savings = currentCosts - newCosts;

  removedCustomers.add(new ArrayList<Double>(Arrays.asList((double)i, (double)j, savings, 0.0, 0.0, 0.0)));
}

if(id<MyRMcontrol.fileMatrixCurrent.size()-1) id++;
else id = 1;

// System.out.println(id);

}

public double accumulateArrayVal(double[] array, int pos){
  double value=0;
  for(int p=0; p<=pos; p++){
    value+= array[p];
  }
  return value;
}

public void findFurthestPair(int i, int j, boolean ij, int[][][] w){
  /**ALTERNATIVE hat sich als ungünstig erwiesen.
   * Update 08/01/16: Finde hiermit verbesserte loesungen**/
  double currenthighest=Integer.MIN_VALUE;
  double currentDist = 0;

  for(int q=1; q<w.length; q++){
    for(int ii=1; ii<w.length; ii++){
      for(int jj = 1; jj<Main.V; jj++){
        if(w[ii][jj][q][k]==1){
          currentDist = MyData.distance[ii][jj];
          // for(int j2=1; j2<Main.V; j2++) if(w[jj][j2][q+1][k]==1) {
            currentDist +=MyData.distance[jj][j2];
          // break;
          // }
        }
      }
    }
  }

  if(currentDist>currenthighest){
    currenthighest=currentDist;
    // i=jj;
    //boolean ij = randomGenerator.nextBoolean();
    if(ij==true){
      i = jj;
    }
    else i = ii;
  }

  // System.out.println("currentHighest = " +currenthighest + "; i ="+i);
}
```java
public void findFurthestList1(int[][][][] w){
    highDistCustomers.clear();
    double totalDist = 0;
    for(int k=0; k<Main.k; k++){
        for(int q=1; q<=w.length; q++){
            for(int ii=1; ii<w.length; ii++){
                for(int jj=1; jj<Main.V; jj++){
                    if(w[ii][jj][q][k]==1){
                        totalDist+=MyData.distance[ii][jj];
                    }
                }
            }
        }
        double distanceToBeBelow = (totalDist/MyData.distance.length)*1.6;
        // was 1.3 for lc201+202
        for(int k=0; k<Main.k; k++){
            for(int q=1; q<=w.length; q++){
                for(int ii=1; ii<w.length; ii++){
                    for(int jj=1; jj<Main.V; jj++){
                        if(w[ii][jj][q][k]==1 && MySolution.startingPositions.contains(ii) && MyData.distance[ii][jj] > distanceToBeBelow){
                            highDistCustomers.add(ii);
                            highDistCustomers.add(jj);
                        }
                    }
                }
            }
            while(highDistCustomers.size()>MyRMcontrol.fileMatrixCurrent.size()/5){
                highDistCustomers.remove(randomGenerator.nextInt(highDistCustomers.size()));
            }
        }
    }
}
```

if(highDistCustomers.size()>5){
    highDistCustomers.remove(highDistCustomers.size()-1);
    highDistCustomerValues.remove(highDistCustomerValues.size()-1);
}
else{
    if(highDistCustomers.size()<1){
        highDistCustomers.add(Operator.wLists.get(k).get(pos));
        highDistCustomerValues.add(thisDistance/withoutDistance);
    }
    else{
        L: for(int listPos=0; listPos<highDistCustomers.size(); listPos++){
            if(thisDistance/withoutDistance > highDistCustomerValues.get(listPos)){
                highDistCustomers.add(listPos, Operator.wLists.get(k).get(pos))
                highDistCustomerValues.add(listPos, thisDistance/withoutDistance);
                break L;
            }
        }
        if(highDistCustomers.size()>5){
            highDistCustomers.remove(highDistCustomers.size()-1);
            highDistCustomerValues.remove(highDistCustomerValues.size()-1);
        }
    }
}
// System.out.println("High distance customers by incremental distance:");
// for(int p=0; p<highDistCustomers.size(); p++) System.out.println(highDistCustomerValues.get(p)+", ");
System.out.println();
}
public void findFurthestList(int[][][][] w){
    highDistCustomers.clear();
    double totalDist = 0;
    int k = selectK();
    //for(int k=0; k<Main.k; k++){
    for(int q=1; q<=w.length; q++){
        for(int ii=1; ii<w.length; ii++){
            for(int jj =1; jj<Main.V; jj++){
                if(w[ii][jj][q][k]==1){
                    totalDist+=MyData.distance[ii][jj];
                }
            }
        }
    }
    //}
    double distanceToBeBelow = (totalDist/MyData.distance.length)*1.6;
    // was 1.3 for lc201+202
    //for(int k=0; k<Main.k; k++){
    for(int q=1; q<=w.length; q++){
        for(int ii=1; ii<w.length; ii++){
            for(int jj =1; jj<Main.V; jj++){
                if(w[ii][jj][q][k]==1 && MySolution.startingPositions.contains(ii) == false && MyData.distance[ii][jj] > distanceToBeBelow){
                    highDistCustomers.add(ii);
                } else{
                    highDistCustomers.add(jj);
                }
            }
        }
    }
    //}
```java
public void findHighDistTWrelation(int[][][] w) {
    double twDiff;
    double highDiff = Integer.MIN_VALUE;
    double diffCurrent;
    double timex;
    int customer = 0;
    for (int k = 0; k < Main.k; k++) {
        timex = 0;
        highDiff = Integer.MIN_VALUE;
        for (int q = 1; q < w.length; q++) {
            for (int ii = 0; ii < w.length; ii++) {
                for (int jj = 0; jj < Main.V; jj++) {
                    if (w[ii][jj][q][k] == 1 && MySolution.startingPositions.contains(ii) == false) {
                        timex += MyData.Service[ii];
                        timex += MyData.distance[ii][jj] / Main.v;
                        if (MyData.TWstart[jj] - timex > 300) {
                            highDistCustomers.add(jj);
                            break k;
                        }
                        else if (timex < MyData.TWstart[jj]) timex = MyData.TWstart[jj];
                    }
                    twDiff = Math.abs(MyData.TWend[jj] - MyData.TWend[ii]);
                    diffCurrent = MyData.distance[ii][jj] / twDiff;
                    if (MyData.TWstart[ii] > MyData.TWend[jj]) {
                        highDistCustomers.add(jj);
                        highDistCustomers.add(ii);
                        break k;
                    }
                    if (Math.abs(MyData.TWstart[jj] - MyData.TWstart[ii]) > 500) {
                        highDistCustomers.add(jj);
                        System.out.println("c: "+ii);
                        break k;
                    }
                    if (customer != 0) highDistCustomers.add(customer);
                }
            }
        }
        if (customer != 0) highDistCustomers.add(customer);
    }
    System.out.println("High TW relation candidate list:");
    for (int p = 0; p < highDistCustomers.size(); p++) System.out.print(highDistCustomers.get(p) +", ");
    System.out.println();
}
```
errors += i;

if(errors>0) {
    k = randomGenerator.nextInt(errors);
    int valueToMeet = 0;
    L: for(int i=0; i<MyObjectiveFunction.nrOfViolationsInRoutes.length; i++)
    {
        valueToMeet+= MyObjectiveFunction.nrOfViolationsInRoutes[i];
        if( k <= valueToMeet)
            k=i;
        break L;
    }
    return k;
}

public int checkifpostponingimproves(int i, int j){
    double oldDist;
    double newDist;
    for(int k=0; k<Operator.wLists.size(); k++)
    {
        if(Operator.wLists.get(k).size()>=3){
            for(int pos=1; pos<Operator.wLists.get(k).size()-2; pos++) {
                oldDist = MyData.distance[Operator.wLists.get(k).get(pos)][Operator.wLists.get(k).get(pos+1)][Operator.wLists.get(k).get(pos+1)][Operator.wLists.get(k).get(pos+2)];
                newDist = MyData.distance[Operator.wLists.get(k).get(pos)][Operator.wLists.get(k).get(pos+1)][Operator.wLists.get(k).get(pos+1)][Operator.wLists.get(k).get(pos+2)];
                if(newDist < oldDist && Operator.wLists.get(k).get(pos+1) != MyRM control.fileMatrixCurrent.get(Operator.wLists.get(k).get(pos))[8].intValue()){
                    i = Operator.wLists.get(k).get(pos);
                    j = pos+1;
                }
            }
        }
    }
    System.out.println(" postpone customer "+ i);
    return i;
}

public static double arraySum(double[] array){
    double sum = 0;
    for (int i =0; i< array.length; i++)
        sum += array[i];
    return sum;
}

private static boolean checkIfAlreadyIncluded(int id){
    boolean result = false;
    I: for(int i=0; i<removedCustomers.size(); i++)
    {
        if(removedCustomers.get(i).get(0).intValue() == id) {
            result = true;
            break I;
        }
        else {
            result = false;
        }
    }
    return result;
}
public static double angleBetween2Lines(Line2D line1, Line2D line2) {
    double angle1 = Math.atan2(line1.getY1() - line1.getY2(), line1.getX1() - line1.getX2());
    double angle2 = Math.atan2(line2.getY1() - line2.getY2(), line2.getX1() - line2.getX2());
    return Math.abs(Math.PI - Math.abs(angle1 - angle2));
} // end class MyMoveManager
import java.text DecimalFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import org.coinor.opents.*;

class MyObjectiveFunction implements ObjectiveFunction {
    public static double time[] = new double[Main.V];
    public static double penalty;
    public static double penalty2;
    public static double[] t; = new double[Main.V];
    static double[] tkreturn= new double[Main.k];
    static double[] tArrivalCheck;
    public static double[] TArrival = new double[MyRMcontrol.fileMatrixCurrent.size()+Main.k];;
    public static double TArrival1;
    double[] tWait = new double[MyData.distance.length];
    public static List<List<Integer>> cK = new ArrayList<List<Integer>>();
    static List<Integer> customersToMove = new ArrayList<Integer>();
    static List<Integer> customersToMove2 = new ArrayList<Integer>();
    static List<Integer> customersToMoveTime = new ArrayList<Integer>();
    static int[] nrOfViolationsInRoutes;
    public static boolean print = true;
    public static boolean printSOL = false;
    public static int[] pos1w;
    public static int[] pos2w;
    static double waitingTime;
    static double delay;
    public static double correctionfactor = 1;

    @SuppressWarnings("unlikely-arg-type")
    public static boolean contains(final int[] customersk, final int i) {
        return Arrays.asList(customersk).contains(i);
    }

    static DecimalFormat df = new DecimalFormat("#.0");
    public MyObjectiveFunction( double[][] customers ) {
        //matrix = createMatrix(customers);
    }

    // Standard VRP constraints:
    // 1.0 Jeder Knoten \0 darf genau einmal angefahren werden
    for( int j = 1; j < w.length; j++ ){
        int arrivals=0;
        for(int i=0; i<w.length; i++){
            for(int q=0; q<w.length; q++){
                for(int k=0; k<Main.k; k++){
                    arrivals+=w[i][j][q][k];
                }
            }
        }
        if(arrivals!=1 && doesArrayContain(MyRMcontrol.startingPosition, j )==false){
            penalty2+=(100000); //2nd part for dynamic
        }
    }

    public static void evaluateStandardConstraints(int w[][][][]){
        //
    }
}
Every Vehicle that leaves the depot (starting Position) and has to return to the depot

for (int k = 0; k < Main.k; k++) {
    int sumW0j = 0;
    int sumWi0 = 0;
    for (int j = 0; j < w.length; j++) {
        // changed for dynamic, vorher start bei j = 1
        sumW0j += w[Main.startingPositionK[k]][j][1][k];
    }
    for (int i = 0; i < w.length; i++) {
        for (int q = 0; q < w.length; q++) {
            sumWi0 += w[i][0][q][k];
        }
    }
    if (sumW0j != sumWi0) {
        if (print == true) System.err.println("LeaveReturn for starting position: "+MyRMcontrol.startingPosition[k]+": "+sumW0j+" "+sumWi0);
        penalty2+=100000;
    }
}

// 3.0 Jeder Knoten darf genau einmal verlassen werden
for (int i = 1; i < w.length; i++) {
    int departures = 0;
    for (int j = 0; j < w.length; j++) {
        for (int q = 0; q < w.length; q++) {
            // hier war zwischenzeitig q<w.length
            departures += w[i][j][q][k];
        }
    }
    if (departures != 1 && contains(Main.startingPositionK, i) == false) {
        penalty2+=(100000);
        if (print == true) System.err.println("Error depart 3.0 at "+ i +" departures: "+ departures);
    }
}

// 4.0 A vehicle must not drive from i to i
for (int i = 0; i < w.length; i++) {
    for (int q = 0; q < w.length; q++) {
        for (int k = 0; k < Main.k; k++) {
            // System.out.println("w"+i+"["+i+"][q]"+q+"["+k+"]");
            if (w[i][i][q][k] == 1 && i != 0 && contains(Main.startingPositionK, i) == false) {
                penalty2+=(100000);
                if (print == true) System.err.println(" Error i=i 4.0 at customer: "+i);
            }
        }
    }
}

public static double evaluateConstraints (int w[][][][]) {
    t = new double[Main.V];
    double now = System.currentTimeMillis();
    // determine variable size:
    int[][]Q = new int[Main.V][Main.k];
    // double l[][][] = new double [MyData.distance.length][Main.k]; // koennen evtl auf int geandert werden
    double L[][][] = new double [MyData.distance.length][Main.k];
    waitingTime=0;
    delay=0;
}
```java
double thisWaitingTime=0;

cK.clear(); // after heap space error
customersToMove.clear();
customersToMove2.clear();
customersToMoveTime.clear();
nrOfViolationsInRoutes = new int [Main.k];
penalty2 = 0;

// 9.8 by Lists
Operator.fourDto2DListePlusReturn(w);

double TArrival[][] = new double[Main.V][Main.k]; //erstelle T

tArrivalCheck =new double [MyRMcontrol.fileMatrixCurrent.size()][4];
for (int k=0; k<Operator.wLists.size(); k++){
    double timex = MyRMcontrol.startingTime[k];
    //double dist = 0;
    for (int i=0; i< Operator.wLists.get(k).size()-1; i++){
        timex+= MyData.distance[Operator.wLists.get(k).get(i)][Operator.wLists.get(k).get(i+1)]/Main.v;
        timex+= MyData.Service[Operator.wLists.get(k).get(i)];
        thisWaitingTime = MyData.TWstart[Operator.wLists.get(k).get(i+1)] - timex;
        if(thisWaitingTime>0){
            if(thisWaitingTime >50){
                customersToMove.add(Operator.wLists.get(k).get(i+1));
                customersToMoveTime.add(Operator.wLists.get(k).get(i+1));
                //System.out.println("Customer "+ Operator.wLists.get(k).get(i+1)+ " added to move list (waiting time)");
                waitingTime+= thisWaitingTime;
                timex=MyData.TWstart[Operator.wLists.get(k).get(i+1)];
            }
        }
        TArrival[Operator.wLists.get(k).get(i+1)][k]=timex;
        tArrivalCheck[Operator.wLists.get(k).get(i+1)][0] = timex;
    }
    if(print ==true)System.out.println("Customer "+ Operator.wLists.get(k).get(i+1)+ " added to move list (waiting time)");
}
}

// check TW end LISTS:
for (int k=0; k<Operator.wLists.size(); k++){
    for (int i=0; i< Operator.wLists.get(k).size()-1; i++){
        if(MyData.TWend[Operator.wLists.get(k).get(i)]<TArrival[Operator.wLists.get(k).get(i)][k]){ // T muss <= TW end sein
tArrivalCheck[Operator.wLists.get(k).get(i)][1]++;            //System.out.println("Delay at customer: "+Operator.wLists.get(k).get(i)+": " + (TArrival[Operator.wLists.get(k).get(i)][k] - MyData.TWend[Operator.wLists.get(k).get(i)]));
        if(i!=0 && doesArrayContain(MyRMcontrol.startingPosition, Operator.wLists.get(k).get(i))==false && customersToMove.contains(Operator.wLists.get(k).get(i))==false){ //2nd term for dynamic //check if already in for lc204,5,6 etc
customersToMove.add(Operator.wLists.get(k).get(i));
customersToMove2.add(Operator.wLists.get(k).get(i));
nrOfViolationsInRoutes[k]++;
```

---

258
if(i>1 && doesArrayContain(MyRMcontrol.startingPosition, Operator.wLists.get(k).get(i-1))){
    customersToMove.add(Operator.wLists.get(k).get(i-1));
}
//System.out.println(customerToMove);
if(print == true)System.err.println("Violation in 9.8 at customer " + Operator.wLists.get(k).get(i));  //hier wird nur tarrival gechecked
}
//System.out.println(customerToMove);
if(print == true)System.err.println("Fehler in 9.7 between customer " + Operator.wLists.get(k).get(i) + " and " + Operator.wLists.get(k).get(i+1) + " T[i][k] " + TArrival[Operator.wLists.get(k).get(i)][k] + " s[i] ...");
}
}
for(int k=0; k<Operator.wLists.size(); k++){
    for(int i=1; i< Operator.wLists.get(k).size()-2; i++){
        if({(TArrival[Operator.wLists.get(k).get(i)][k] + MyData.Service[Operator.wLists.get(k).get(i)]* (MyData.distance[Operator.wLists.get(k).get(i)][Operator.wLists.get(k).get(i+1)][Main.v]) - (TArrival[Operator.wLists.get(k).get(i+1)][k+1]) > 0} + 1 wegen rundungsfehlern!

        penalty2 += 100000;
        if(print == true)System.err.println("Fehler in 9.7 between customer " + Operator.wLists.get(k).get(i) + " and " + Operator.wLists.get(k).get(i+1) + " T[i][k] " + TArrival[Operator.wLists.get(k).get(i)][k] + " s[i] ...");
    }
}
foreach(int k=0; k<Operator.wLists.size(); k++)
    pickupTrue = new boolean[MyRMcontrol.fileMatrixCurrent.size()] ; //i if here-> alsoC9
    for(int i=0; i< Operator.wLists.get(k).size()-2; i++){
        Q[Operator.wLists.get(k).get(i)][k]=q;
        tArrivalCheck[Operator.wLists.get(k).get(i)][2] = q;
        q++;
        pickupTrue[Operator.wLists.get(k).get(i)] = true;  //check each position that has been visited
        if(MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(i))[7]>0)
            & pickupTrue[MySolution.searchbyID(MyRMcontrol.fileMatrixCurrent, MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(i))[7].intValue())] = false{
                penalty2+=100000;
                //System.out.println(MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(i))[7] +" should have been before " + Operator.wLists.get(k).get(i));
                if(print == true)System.err.println("Error in 8.0 - Delivery before pickup location" ); // or in different route);
                }
            
for(int k=0; k<Main.k; k++)
    } //k2 to enable all k=k2 pairs
boolean ii=false;
boolean jj=false;
for(int i=0; i<w.length; i++){
    for(int j=1; j<w.length; j++){
        if(MyData.demand[i][j]!=0 && MyRMcontrol.startingPosition[k]!=i && MyRMcontrol.startingPosition[k]!=j) {
            //System.out.println("Demand zwischen " + i + " und " + j);
            if(Q[i][k]!=0){
                //System.out.println("i ist in tour " + Q[i][k]);
                ii = true;
            }
            if(Q[j][k]!=0){
                //System.out.println("j ist in tour " + Q[j][k]);
                jj = true;
            }
        }
    }
}
if(ii!=jj || jj!=ii){
    penalty2+=100000;
    if(print ==true)System.err.println("Error in 9.0 - Pickup and delivery customers are not in the same route");
}
/**/

//10 Capacity constraints LISTS:
int[] vehicleCap = new int[Main.k];
for(int k=0; k<Operator.wLists.size(); k++){
    for(int i=0; i<Operator.wLists.get(k).size(); i++){
        vehicleCap[k]+=MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(i))[3].intValue();
    }
    if(vehicleCap[k]>Main.C){
        penalty2+=100000;
        if(print ==true)System.err.println("Error in 13.0 - Capacity outreach");
    }
}
Main.measureConstraints +=System.currentTimeMillis()-now;
return (penalty2);
}

public static double evaluateConstraintsNORMAL (int w[][][][]){
t=new double[Main.V];
double now = System.currentTimeMillis();
// determine variable size:
int[][]Q = new int[Main.V][Main.k];
double L[][] = new double [MyData.distance.length][Main.k]; // koennen evtl auf int geandert werden
double L[][] = new double [MyData.distance.length][Main.k];
waitingTime=0;
cK.clear(); // after heap space error
customersToMove.clear();
customersToMove2.clear();

    // Setze alle elemente q[][][]=0:
    for(int k=0; k<Main.k; k++){
        for(int j=0; j<Main.V; j++){
            Q[j][k]=0;
        }
    }

    penalty=0;

    // Standard VRP constraints:
    // 1.0 Jeder Knoten \0 darf genau einmal angefahren werden
    for( int j = 1; j < w.length; j++ ){
        arrivals=0;
        for(int i=0; i<w.length; i++){
            for(int q=0; q<w.length; q++){
for(int k=0; k<Main.k; k++)
    arrivals+=w[i][j][q][k];
}

if(arrivals!=1 && doesArrayContain(MyRMcontrol[startingPosition], j) ==false) {
    penalty+=(100000);   //2nd part for dynamic
    if(print == true)System.err.println("#Arrivals: "+ arrivals);
}

// 2.0 Every Vehicle that leaves the depot(starting Position) and has to r
return to the depot
for(int k=0; k<Main.k; k++)
    int sumW0j =0;
    int sumW10 =0;
for( int j = 0; j < w.length; j++) {   //changed for dynamic, vorher st
art bei j=1
    sumW0j += w[Main.startingPositionK[k]][j][1][k];
    for(int i=0; i<w.length; i++)
        for(int q=0; q<w.length; q++){
            sumWi0 += w[i][0][q][k];
        }
    if(sumW0j != sumWi0)
        if(print != true)System.err.println("LeaveReturn for starting position:
+MyRMcontrol[startingPosition]k]+": "+sumW0j +" "+" +sumW10);
        penalty+=100000;
}

// 3.0 Jeder Knoten \0 darf genau einmal verlassen werden
for( int i = 1; i < w.length; i++){
    int departures=0;
    for(int j=0; j<w.length; j++){
        for(int q=0; q<w.length; q++){
            // hier war zwischenzeitig
            q<w.length
for( int k=0; k<Main.k; k++){
    if(w[i][j][q][k]==1 ){
        departures+=w[i][j][q][k];
    }
    if(departures!=1 && contains(Main.startingPositionK, i) ==false) {
        penalty+=100000;
        if(print == true)System.err.println(" Error depart 3.0 at "+ i +
departures: "+ departures);
    }
    if(w[i][i][q][k]==1 && i!=0 && contains(Main.startingPositionK, i) ==
false){
        penalty+=(100000);
        if(print == true)System.err.println(" Error i=i 4.0 at customer: "+i);
    }
}
    }
}
// Time window and capacity constraints
/*new 9.8:
double TArrival[][] = new double[Main.V][Main.k];  // erstelle T
for (int ka=0; ka<Main.k; ka++) {
    double timex = /*0;*/ MyRMcontrol.startingTime[ka];//MyData.Service
    [Main.startingPosition[ka]];
    double dist = 0;
    for (int qu=1; qu<MyData.distance.length; qu++){
        for (int i=0; i<MyData.distance.length; i++){
            if (w[i][j][qu][ka] == 1) {
                System.out.println();
                System.out.print("w[" +i+ "]" +"["+(j)+"]" +"["+qu+"]["+ka+"]+=="
                + w[i][j][qu][ka]);
                dist+=w[i][j][qu][ka] * MyData.distance[i][j];
                timex+=w[i][j][qu][ka] * (MyData.distance[i][j]/Main.v);
                timex+= MyData.Service[i]; //inserted for dynamic
                //
                if (timex<MyData.TWstart[j]){
                    timex=MyData.TWstart[j];
                }
                //
                TArrival[j][ka]=timex;
                // tArrivalCheck[j][1] = timex;
                // tArrivalCheck[j][0] = timex;
                ///
                //**timex+= MyData.Service[j];**/ /*Removed for dynamic
            }
        }
}
// check TW end:
for(int k=0; k<Main.k; k++){
    for(int i=0; i< Main.V; i++ ){
        if(MyData.TWend[i]<TArrival[i][k]){
            waitingTime+= (TArrival[i][k] - MyData.TWend[i]);
            // penalty+= (TArrival[i][k] - MyData.TWend[i])/100;
            penalty+= 100000 ;  // T muss <= TW end sein
            if(i!=0 & & doesArrayContain(MyRMcontrol.startingPosition, i)==false
                & & customersToMove.contains(i)==false)  //2nd term for dynamic //check if alrea
                in for lc204,5,6 etc
                customersToMove.add(i);
                customersToMove2.add(i);
                for(int q=1; q<w.length; q++){
                    for(int h=1; h<w.length; h++){
                        if(w[h][i][q][ka]==1 & & h!=3){
                            customersToMove.add(h);  // Customer to move f
                            or MyMoveManager
                    }
                }
                //System.out.println(customerToMove);
                if(print == true)System.err.println("Violation in 9.8 at customer " +
                i);  //hier wird nur tarrival gechockered
                }
                //System.out.println("T["+i+"]["+k+"]="+TArrival[i][k]);
            }
        }
        if(print == true)System.err.println("CustomersToMove List length: "+customersToMove.size( ));
        // New 9.7:
for (int k=0; k<Main.k; k++) {
    for (int q=1; q<w.length; q++) {
        for (int i=1; i<w.length; i++) {
            for (int j=1; j<w.length; j++) {
                if ((w[i][j][q][k] == 1) &&
                }
            }
        }
    }
} // Beaufschlagung von waiting time:
    //penalty+=waitingTime;
} // 7.0 Position in route (for P-D check):
for (int k=0; k<Main.k; k++) {
    for (int q=1; q<w.length; q++) {
        for (int i=0; i<w.length; i++) {
            for (int j=1; j<w.length; j++) {
                if (w[i][j][q][k] == 1) {
                    System.out.println("w[" + i + "] [" + j + "] [" + q + "] [" + k + "] +" = + w[i][j][q][k]);
                    Q[i][j][k]=q;
                    System.out.println("\" Q[" + j + "] [" + k + "] +" = + Q[j][k]);
                }
            }
        }
    }
} // 8.0 Check positions in route (P-D):
boolean c80 = false;
for (int k=0; k<Main.k; k++) {
    for (int q=1; q<w.length; q++) {
        for (int l=0; l<w.length; l++) {
            if (MyData.demand[i][j] != 0) {
                if (Q[i][k] > Q[j][k] && Q[i][k] != 0) {
                    penalty+=100000;
                    c80 = true;
                }
            }
        }
    }
    System.out.println("i: "+i+" j: "+j);
} if (c80==true) {
    if (print == true) System.err.println("Error in 8.0 - Delivery before pickup location");
} // or in different route};
} // 9.0 wenn i-j P - D, dann muss j teil von K sein:
for (int k=0; k<Main.k; k++) {
    //k2 to enable all k-k2 pairs
    boolean ii=false;
boolean jj=false;

for(int i=0; i<w.length; i++){
    for(int j=1; j<w.length; j++) {
        if(MyData.demand[i][j]!=0 && MyRMcontrol.startingPosition[k]!=i && MyRMcontrol.startingPosition[k]!=j) {
            //System.out.println("Demand zwischen " + i + " und " + j);
        }
    }
}

if(Q[i][k]!=0) {
    //System.out.println("i ist in tour " + Q[i][k]);
    ii=true;
}

if(Q[j][k]!=0) {
    //System.out.println("j ist in tour " + Q[j][k]);
    jj=true;
}

if(ii!=jj || jj!=ii){
    penalty+=100000;
}

if(print==true)System.err.println("Error in 9.0 - Pickup and delivery customers are not in the same route");

/
/
// 10.0 Capacity constraints:
// Determine customers served by each vehicle k:
for(int k=0; k<Main.k; k++){
    List<Integer> row = new ArrayList<Integer>();
    for(int i=0; i<w.length; i++){
        for(int j=1; j<w.length; j++){
            if(w[i][j][q][k]==1)
                row.add(i);
        }
    }
    cK.add(row);
    //System.out.println(cK.get(k));
}

// 11 Determination of l_i
for(int k=0; k<Main.k; k++){
    for(int i=0; i<w.length; i++){
        for(int j=0; j<w.length; j++) {
            //System.out.println(contains(customers[k], i));
            if(MyData.demand[i][j]!=0 && cK.get(k).contains(i) && cK.get(k).contains(j)){
                //if(Arrays.asList(customers[k]).contains(j)){
                    l[i][k]+=MyData.demand[i][j];
                    l[j][k]=MyData.demand[i][j];
                    //System.out.println("l["+i+"]["+k+"]: " + l[i][k]);
            }
        }
    }
}

// 12 L_ijkq
boolean cl2=false;

for(int k=0; k<Main.k; k++){
    L[Main.startingPositionK[k]][k]=l[Main.startingPositionK[k]][k];
    // Change dynamic: was L[0][k]=l[0][k]
    for(int q=1; q<w.length; q++){
        for(int i=0; i<w.length; i++){
            for(int j=1; j<w.length; j++) {
                if(w[i][j][q][k]==1)l[j][k]=L[i][k]+1[l[j][k] - L[i][k]]!=0{
                    System.out.println("w[" + i + "][" + j + "](" + q + ")" + "L["+i+"]["+k+"]+L["+j+"]["+k+"]+l["+j+"]["+k+"]");
                }
            }
        }
    }
}

// System.out.println("L["+i+"]["+k+"]= " + L[i][k]);
System.out.println("1["+i+"]["+k+"] = " + l[i][k]);

if (c12==true) {
    penalty+=100000;
    if (print == true) System.err.println("Error in 12.0 - load-unload inequality");
}
// 13.0 Vehicle capacity
boolean c13=false;
for (int k=0; k<Main.k; k++) {
    for (int i=0; i<w.length; i++) {
        if (l[i][k]>L[i][k]) {
            c13=true;
            //System.out.println("1. " +l[i][k] +" > " + L[i][k]);
        }
        if (L[i][k]>Main.C) {
            c13=true;
            //System.out.println("2. " +L[i][k] +" > " + Main.C);
        }
    }
    if (c13==true) {
        penalty+=100000;
        if (print == true) System.err.println("Error in 13.0 - Capacity outreach");
    }
}
if (Main.measureConstraints +=System.currentTimeMillis()-now;
return (penalty);

// 14.0 Objective Value:
double dist;
costs = 0;
if (move == null) {
    dist = 0;
    for (int i = 0; i < wx.length; i++) {
        for (int j=0; j<wx.length; j++) {
            for (int q=0; q<wx.length; q++) {
                for (int k=0; k<Main.k; k++) {
                    if (wx[i][j][q][k] == 1) dist+= MyData.distance[i][j];
                }
            }
        }
    }
    double[][] tArrivalCheck =new double [MyRMcontrol.fileMatrixCurrent.size()][2];
evaluateConstraints(wx);
evaluateStandardConstraints(wx); //These errors do not occur usually
evaluateConstraintsNORMAL(wx);
// Objective Value:
costs = dist*Main.variableCostsKM + delay +penalty2; // + waitingTime;
System.out.println("Dist: "+dist);
System.out.println("Delay: "+delay);
System.out.println("Waiting: "+waitingTime);
System.out.println("Penalty2: "+penalty2);

return new double[]{costs, delay, waitingTime, dist};

// Else calculate incrementally

MySwapMove mv = (MySwapMove)move;
int[] pos1w = new int[8]; // Entfernt fuer methoden!
int[] pos2w = new int[8];
boolean cmpair = false;
boolean mcpair = false;

double now = System.currentTimeMillis();

System.out.println("Operator: "+mv.operator+"customer: "+mv.customer+"movement: "+mv.movement);

if(mv.operator==1){
Operator.SwapPair(solution, pos1w, pos2w, cmpair, mcpair, true, mv.customer, mv.movement, 0);
System.out.println("Operator 1");
}
else if (mv.operator==2) { 
Operator.SwapCustomerInRouteListeEco(solution, mv.customer, mv.movement, 7); 
System.out.println("OPERATOR 2"); 
} else if (mv.operator == 3 || mv.operator==1 || mv.operator == 8) { 
    // **TEST** */Operator.SwapCustomerInRoute(wx, pos1w, true, mv.customer, mv.movement, 0); 
    Operator.SwapCustomerInRouteListeEco(solution, mv.customer, mv.movement, 0); 
} else if (mv.operator == 4) { 
    // Operator.SwapCustomerInRoute(wx, pos1w, true, mv.customer, mv.movement, 1); 
    Operator.SwapCustomerInRouteListeEco(solution, mv.customer, mv.movement, 1); 
} else if (mv.operator == 5) { 
    // Operator.SwapCustomerInRoute(wx, pos1w, true, mv.customer, mv.movement, 2); 
    Operator.SwapPairGEO(solution, mv.customer, mv.movement); 
} else if ( mv.operator == 6) { 
    // Operator.SwapPair(solution, /* wx, */ pos1w, pos2w, cmpair, mcpair, true, mv.customer, mv.movement, 2); 
    Operator.SwapCustomerInRouteListeEco(solution, mv.customer, mv.movement, 4); 
    System.err.println("OPERATOR 6"); 
} else if ( mv.operator == 7) { 
    Operator.SwapCustomerInRouteListeEco(solution, mv.customer, mv.movement, 0); 
} else if (mv.operator == 8) { 

else if (mv.operator == 9) Operator.XExchange2(solution, mv.customer, mv.movement);
else if (mv.operator == 10) Operator.SwapCustomerInRouteListEco(solution, mv.customer, mv.movement, /*0*/ / 5);
else if (mv.operator == 11) Operator.SwapCustomerInRouteListEco(solution, mv.customer, mv.movement, /*1*/ / 7);
else if (mv.operator == 12) Operator.SwapCustomerInRouteToPos(solution, mv.customer, mv.movement);
else if (mv.operator == 13) Operator.SwapPairRandIncr(solution, mv.customer, mv.movement);
else if (mv.operator == 14) Operator.RemoveFromList(solution, mv.customer, mv.movement, 1, false);
else if (mv.operator == 15) Operator.RemoveFromList(solution, mv.customer, mv.movement, 2, false);
else if (mv.operator == 16) Operator.RemoveFromList(solution, mv.customer, mv.movement, Main.k, false);
else if (mv.operator == 17) Operator.RemoveFromList(solution, mv.customer, mv.movement, 1, true);
else if (mv.operator == 18) Operator.RemoveFromList(solution, mv.customer, mv.movement, 2, false);
else if (mv.operator == 19) Operator.RemoveFromList(solution, mv.customer, mv.movement, Main.k, false);
else if (mv.operator == 20) Operator.RemoveFromList(solution, mv.customer, mv.movement, 1, false);
else if (mv.operator == 21) Operator.RemoveFromList(solution, mv.customer, mv.movement, Main.k, false);
else if (mv.operator == 22) Operator.RemoveFromList(solution, mv.customer, mv.movement, Main.k, false); //eliminate Route

else if (mv.operator == 15) Operator.RemoveFromList/*RegretDist*/(solution, mv.customer, mv.movement);

Main.measureOperators += System.currentTimeMillis() - now;

if (printSOL == true) System.out.println("-----------------------------");

// Calculate driven distance
dist = 0;
double timex = 0;
for (int ka=0; ka<Main.k; ka++){
timex = 0;
for (int qu=1; qu<MyData.distance.length; qu++){
for (int i=0; i<MyData.distance.length; i++){
for (int j=0; j<MyData.distance.length; j++){
if (wx[i][j][qu][ka] == 1)
if (printSOL == true) System.out.println();
if (printSOL == true) System.out.print("w[" + i + "]" + (j) + "]
" + wx[i][j][qu][ka] + "");
System.out.println();
System.out.print("w[" + i + "]" + (j) + "]" + " + "");
System.out.println();
dist= wx[i][j][qu][ka] * MyData.distance[i][j];
timex= wx[i][j][qu][ka] * (MyData.distance[i][j]/Main.v);

}
}
if (timex<MyData.TWstart[j])
timex=MyData.TWstart[j];
}

System.out.println(" \t" + df.format(MyData.distance[i][j])
); //no waiting times respected
System.out.print(" \"t"+MyData.TWstart[j]);
System.out.print("\t"+df.format(timex));
System.out.print("\t" + MyData.TWend[j]);
System.out.print("\t" + MyData.Service[j]);

timex+= MyData.Service[j];

for(int i2=0; i2<MyData.distance.length; i2++){
  if(MyData.demand[i2][j]!=0)System.out.print("\t"+MyData.demand[i2][j]+" units from p"+i2);
}

if(printSOL == true)System.out.println();
System.out.println();
System.out.println();

double[][] tArrivalCheck =new double [MyRMcontrol.fileMatrixCurrent.size()][2];
evaluateConstraints(wx);
evaluateStandardConstraints(wx);  //These errors do not occur usually

evaluateConstraintsNORMAL(wx);

Operator.rememberW2 = new int[wx.length][wx.length][wx.length][Main.k];
Operator.rememberW2 = MySolution.deepCopyOf(MySolution.w);

if( mv.operator == 6 || mv.operator == 7){
  Operator.SwapPairReverse(solution, true);
}
else if( mv.operator == 8 ^ mv.operator == 9){
  Operator.operatorReverse(solution);
}
else Operator.SwapCustomerInRouteReverse(true);

costs = dist*Main.variableCostsKM + delay + penalty2; // + waitingTime;

for(int i=0; i<MyRMcontrol.fileMatrixCurrent.size();i++){
  System.out.println("i:"+i+" "+tArrivalCheck[i][2] + " +tArrivalCheck[i][3])
}
System.out.println("Constrains: "+df.format(penalty)+ " - "+df.format(penalty2));
System.out.println("Dist: "+ dist);
System.out.println("Delay: "+ delay);
System.out.println("Waiting: "+ waitingTime);
System.out.println("Penalty2: "+ penalty2);

return new double[]{costs, delay, waitingTime, dist};
}

public static double evaluateQuick(boolean completeEval){
double costs = 0;
double delay = 0;
int c1;
int c2;
// 9.8 by Lists
//System.out.println("Main.V: "+Main.V);
double TArrivalK[][] = new double[Main.V][Main.k];       //erstelle T
        double[] TArrival = new double[MyRMcontrol.fileMatrixCurrent.size()];
        for(int k=0; k<Operator.wLists.size(); k++){
            double timex = MyRMcontrol.startingTime[k];
            for(int i=0; i< Operator.wLists.get(k).size()-1; i++){
                c1 = Operator.wLists.get(k).get(i);
                c2 = Operator.wLists.get(k).get(i+1);
                costs += MyData.distance[c1][c2];
                timex+= MyData.Service[c1];
                timex+= MyData.distance[c1][c2]/Main.v;
                if(timex<MyData.TWstart[c2]){   //costs += (MyData.TWstart[c2]-timex)*.3;
                    timex = MyData.TWstart[c2];
                }
            } //System.out.println("Tarrival size:",TArrival.length + " custome r: "+Operator.wLists.get(k).get(i+1)+ "time: "+timex);
            TArrivalK[Operator.wLists.get(k).get(i+1)][k]=timex;
            TArrival[Operator.wLists.get(k).get(i+1)]=timex;
        }
        // check TW end LISTS:
        for(int k=0; k<Operator.wLists.size(); k++){
            for(int i=0; i< Operator.wLists.get(k).size()-1; i++){
                c1 = Operator.wLists.get(k).get(i);
                c2 = Operator.wLists.get(k).get(i+1);
                if(MyData.TWend[c1]<TArrivalK[c1][k]){   //costs += (MyData.TWstart[c2]-timex)*.3;
                    if(completeEval == true){
                        costs+= 100000;
                        delay+= (TArrivalK[c1][k] - MyData.TWend[c1]);
                    }else delay+= (TArrivalK[c1][k] - MyData.TWend[c1]);//**.001*/;
                }
            }
        }
        // System.out.println("distance: " +costs+ " delay: "+delay);
        return costs+delay;
}

public static double evaluateSingleRouteQuick(int k, boolean penalty){
    double costs = 0;
    double delay = 0;
    int c1;
    int c2;
    double timex = MyRMcontrol.startingTime[k];
    for(int i=0; i< Operator.wLists.get(k).size()-1; i++){
        c1 = Operator.wLists.get(k).get(i);
        c2 = Operator.wLists.get(k).get(i+1);
        System.out.println("c1: "+c1+ " c2: "+c2);
        costs += MyData.distance[c1][c2];
        timex+= MyData.Service[c1];
        timex+= MyData.distance[c1][c2]/Main.v;
        if(timex<MyData.TWstart[c2]){   //costs += (MyData.TWstart[Operator.wLists.get(k).get(i+1)]-timex)*.3;
            timex = MyData.TWstart[c2];
        }
    }
    MyMoveManager.TArrival[c2]=timex;
    for(int i=0; i< Operator.wLists.get(k).size()-1; i++){
        c1 = Operator.wLists.get(k).get(i);
        if(MyData.TWend[c1]<MyMoveManager.TArrival[c1])

delay+= (MyMoveManager.TArrival[c1] - MyData.TWend[c1]);
if(penalty == true) delay+= 100000;
if(penalty == true) delay+= (MyMoveManager.TArrival[c1] - MyData.TWend[c1]) * 1000;
return (costs + delay);
}

public static void printSolution(int[][][] wx) {

double timex = 0;
for (int ka = 0; ka < Main.k; ka++) {
timex = 0;
for (int qu = 1; qu <= wx.length; qu++) {
for (int i = 0; i < wx.length; i++) {
if (wx[i][j][qu][ka] == 1) {
    System.out.println();
    System.out.print(wx[i][j][qu][ka] + 1);
    System.out.println();
    timex+=wx[i][j][qu][ka] * (MyData.distance[i][j]/Main.v);
    ///
    if(timex<MyData.TWstart[j]){
      timex = MyData.TWstart[j];
    }
    ///
    System.out.print(" \	"+df.format(MyData.distance[i][j]));
    //no waiting times respected
    System.out.println(" \	"+MyData.TWstart[j]);
    System.out.println(" \	"+df.format(timex));
    System.out.println(" \	" + MyData.TWend[j]);
    System.out.println(" \	" + MyData.Service[j]);
    timex+= MyData.Service[j];
    }
    if(MyData.demand[i2][j]!=0)System.out.print(" \	"+MyData.demand[i2][j]+" units from p"+ i2);
}
}
public void adaptMoveManager(double objectiveValue){
if(objectiveValue>1000000){
  if(MyMoveManager.opDistribution[0]<8 && MyMoveManager.opDistribution[1]>0){
    MyMoveManager.opDistribution[0]+=8;
    MyMoveManager.opDistribution[1]+=8;
  }
  if(MyMoveManager.opDistribution[1]<84 && MyMoveManager.opDistribution[2]>0){
    MyMoveManager.opDistribution[3]+=8;
    MyMoveManager.opDistribution[2]+=8;
  }
}
else if(objectiveValue<1000000){
  if(MyMoveManager.opDistribution[0]>0 && MyMoveManager.opDistribution[1]<8){
    MyMoveManager.opDistribution[0]-=8;
    MyMoveManager.opDistribution[1]-=8;
  }
    MyMoveManager.opDistribution[3]-=8;
  }
}
}
n[2]<84}\{ 
894     MyMoveManager.opDistribution[3] -= 8; 
895     MyMoveManager.opDistribution[2] += 8; 
896 } 
897 } 
898 // printout: 
899 // for(int i=0; i<MyMoveManager.opDistribution.length; i++) {
900 //     System.out.print(MyMoveManager.opDistribution[i] + ", "); 
901 // } 
902 // System.out.println(); 
903 } 
904 static boolean doesArrayContain(int[] array, int c) { 
905 for(int i=0; i<array.length; i++) {
906     if (array[i] == c) return true; 
907 } 
908 return false; 
909 } 
910 // end class MyObjectiveFunction
E.12. MySolution class

```java
import java.io.FileNotFoundException;
import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.ConcurrentModificationException;
import java.util.List;
import org.coinor.opents.*;

public class MySolution extends SolutionAdapter {
    public static int [][][] w;
    public int findNearest;
    public int belongingJ;
    public int deleteInt;
    public static List<List<Integer>> routes = new ArrayList<List<Integer>>();
    public static List<List<Integer>> delRoutes = new ArrayList<List<Integer>>();
    public static List<Double> solutionsCloned = new ArrayList<Double>();
    public static List<Integer> customersToStayInK = new ArrayList<Integer>();
    public static List<Integer> startingPositions = new ArrayList<Integer>();
    public static List<Integer> wtourList = new ArrayList<Integer>();
    public static int[] wtour;

    public MySolution() {
        // Appease clone()
    }

    public MySolution(double[][] customers) throws FileNotFoundException {
        // Starting solution
        /**w = new int [Main.V][Main.V][Main.V+1][Main.k];**/
        // This method creates a starting solution from the normal data files
        createStartingSolutionMain();
        // This method creates a starting solution from a file that describes one particular solution (i.e. best previous solution)
        createStartingSolutionFromPreviousResults("current");
        // As previous but adding an extra set of new customers
        createStartSolFromPrevAndAdd("current");
        System.out.println("Solution check");
        MyObjectiveFunction.printSolution(w);
        MyObjectiveFunction.evaluateConstraints(w);
    }

    private void createStartSolFromPrevAndAdd(String file) throws FileNotFoundException {
        customersToStayInK.clear();
        // find all pickup locations:
        List<Integer> pickupsInRoute = new ArrayList<Integer>();
        List<Integer> deliveriesInRoute = new ArrayList<Integer>();
        for (int i=1; i<Main.V; i++){
            for (int j=0; j<Main.V; j++){
                if((MyData.demand[i][j]>0 || MyData.demand[j][i]>0) && startingPositions.contains(i)==true){
                    // System.out.println("STARTING POSITION FOUND "+ i + " "+j);
                    customersToStayInK.add(i);
                    customersToStayInK.add(j);
                } else if(MyData.demand[i][j]>0 && startingPositions.contains(j)==false)
                pickupsInRoute.add(i);
                // System.out.print(i+" ");
            }
        }
    }
```
```java
else if(MyData.demand[i][j]<0 && startingPositions.contains(i)==false)
    deliveriesInRoute.add(i);
    //System.out.print(i+", ");
}

// If there is no prior solution existing:
if(wtour == null|| MyRMcontrol.wtourListRemID.size() == 0 /*|| MyRMcontrol.
    fileMatrixCurrent.size()(<20*/){
    w = new int [Main.V][Main.V][Main.V+1][Main.k];
    System.out.println("Create initial solution");
    createStartingSolutionMain();
    System.out.println("w[ ][ ][ ] sizeXXX: "+w.length +" x " + w[0].length +" x "
        + w[0][0].length +" x " + w[0][0][0].length);

    // if there is a solution existing:
    else{
        System.out.println("wtourListRem solution:");
        for (int i=0; i<MyRMcontrol.wtourListRem.size(); i++){
            System.out.print(MyRMcontrol.wtourListRem.get(i)+", ");
        }
        System.out.println();
        wtourList.clear();
        System.out.println("wtour list size "+wtourList.size());
        wtourListRemID.clear();
        if(c!=9999){
            wtourList.add(c);
            System.out.print(c+", ");
        }else if(MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPositionRem, c)){
            wtourList.add(c);
            System.out.print(c+", ");
        }else
        {
            System.out.println("wListsize before "+wtourList.size());
            for(int i=0; i<wtourList.size(); i++){
                System.out.println("remove "+wtourList.get(i));
                wtourList.remove(i);
                i--;
            }
        }
        //insert new customers:
        Operator.insertPairInList(wtourListRem, 0);
        //Operator.insertPairInList(MyRMcontrol.fileMatrixCurrent.size()-
            2, MyRMcontrol.fileMatrixCurrent.size()-1, MyRMcontrol.wtourListRem);
        for(int i=0; i<MyRMcontrol.fileMatrixCurrent.size(); i++)
        {
            if(MyRMcontrol.fileMatrixCurrent.get(i)[9] == MyRMcontrol.t && MyRMcontrol.
                wtourListRem.contains(i)==false && MyRMcontrol.fileMatrixCurrent.get(i)[0] <
                10000)
                Operator.insertPairInList(i, i+1, wtourList);  //keine IDs
            break 0;
```
System.out.println("fileMatrix current REM sizes: "+MyRMcontrol.fileMatrixCurrent.size()+"+MyRMcontrol.fileMatrixCurrentREM.size());

for(int i=0; i<3; i++){
    System.out.println(MyRMcontrol.startingPositionRem[i] + " - ");
    System.out.println(MyRMcontrol.startingPosition[i] + " ");
}

//insert starting positions:
System.out.println("New sol with starting positions:");
int k=0;
int ii=0;
while(k<Main.k){
    if(ii>=wtourList.size())wtourList.add(0);
    //System.out.println("ii: "+ii);
    if(wtourList.get(ii) == 0 || MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, wtourList.get(ii))){
        wtourList.remove(ii);
        wtourList.add(ii, MyRMcontrol.startingPosition[k]);
        //System.out.println(" a: "+wtourList.get(ii)+", ");
        k++;
        }else{
        //System.out.println(" b: "+wtourList.get(ii)+", ");
        }
        ii++;
    }else{
    //System.out.println(" ");
}
//System.out.println("Array:");
//convert wtour list to array:
int[] wtour2 =new int[wtourList.size()];
wtour = wtour2;
for (int i=0; i<wtourList.size(); i++){
    wtour[i] = wtourList.get(i);
    //System.out.println(wtour[i]+", ");
}
//System.out.println();
//convert array[] to array[][][][]
int[][][][] wNew = new int[Main.V][Main.V][Main.V+1][Main.k];
w = wNew;
int q=1;
int numberOfStarts = 0;
for(int i=0; i<wtour.length-1; i++){
    if(k==3) break;
    if(wtour[i+1] != 0 && MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, wtour[i+1])==false){
        w[wtour[i]][wtour[i+1]][q][k] = 1;
        System.out.println("1 w["+wtour[i]+","+wtour[i+1]+"]");
        q++;
    }else if(MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, wtour[i+1])){
        w[wtour[i]][0][q][k] = 1;
    }
System.out.println("2 w["+wtour[i]+"][0]");
k++;
q=1;
System.out.println("2 w["+wtour[i]+"][0]");
k++;
q=1;
i++;
}
if(wtour[i] == 0 || MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, wtour[i])){
    numberOfStarts++;
}
}
operator.wLists.clear();
List<Integer> row1 = new ArrayList<Integer>();
List<Integer> row2 = new ArrayList<Integer>();
List<Integer> row3 = new ArrayList<Integer>();
System.out.println();
for(int i=0; i<Trickkiste.r1.length; i++){
    row1.add(Trickkiste.r1[i]);
}    
System.out.println("New start solution:");
for (int i = 0; i<wtour3.length; i++){
    wtour3[i] = wtourList.get(i);
}    
System.out.print(wtour3[i]+", ");
/**
* Wtour copy below has been deleted 29/04/17
*/
/*wtour = wtour3;*/
//MyObjectiveFunction.evaluateConstraints(w);  
private void createStartingSolutionFromPreviousResults(String file) throws FileNotFoundException {
    String filename = file;
    w = new int [Main.V][Main.V][Main.V+1][Main.k];
    Trickkiste.readRouteSequence(filename);
    Operator.wLists.clear();
    List<Integer> row1 = new ArrayList<Integer>();
    List<Integer> row2 = new ArrayList<Integer>();
    List<Integer> row3 = new ArrayList<Integer>();
    System.out.println();
    for(int i=0; i<Trickkiste.r1.length; i++){
        row1.add(Trickkiste.r1[i]);
    }    
    System.out.println("Trickkiste.r1[i]+" ");
    System.out.println();
    Operator.wLists.add(row1);
    //row.clear();
    for(int i=0; i<Trickkiste.r2.length; i++){
        row2.add(Trickkiste.r2[i]);
    }    
    System.out.println("Trickkiste.r2[i]+" ");
}
Operator.wLists.add(row2);
//row.clear();

for (int i=0; i<Trickkiste.r3.length; i++){
    row3.add(Trickkiste.r3[i]);
    System.out.print(Trickkiste.r3[i]+" ");
}

Operator.wLists.add(row3);
//row.clear();

for (int i=0; i<Trickkiste.r3.length; i++){
    row3.add(Trickkiste.r3[i]);
    System.out.print(Trickkiste.r3[i]+" ");
}

for (int k=0; k<Operator.wLists.size(); k++){
    for (int q=0; q<Operator.wLists.get(k).size(); q++){
        if (w[i][j][q][k] == 1){
            System.out.println("w["+i+"]["+j+"]["+q+"]["+k+"]=1");
        }
    }
}

private void createStartingSolutionMain() {
    startingPositions.clear();
    customersToStayInK.clear();
    //CREATE GREEDY START SOLUTION
    for (int k=0; k<Main.startingPositionK.length; k++){
        if (Main.startingPositionK[k]>0){
            startingPositions.add(Main.startingPositionK[k]);
            System.out.println("Starting position "+k+" +\n"+Main.startingPositionK[k]+"\n");
        }
    }
    //wtour= new int[Main.V+Main.k-startingPositions.size()];
    MyObjectiveFunction.printSolution(w);
    for (int ka=0; ka<Main.k; ka++){
        for (int qu=0; qu<MyData.distance.length; qu++){
            for (int i=1; i<Main.V; i++){
                if ((MyData.demand[i][j]>0 || MyData.demand[j][i]>0) && startingPositions.contains(i)==true){
                    //System.out.println("STARTING POSITION FOUND "+i+" "+j);
                    customersToStayInK.add(i);
                    customersToStayInK.add(j);
                    wTourLength--;
                    if (MyData.demand[j][i]>0 && startingPositions.contains(i)==true)wTourLength--;
                }
            }
        }
    }
    wTourLength = MyRMcontrol.fileMatrixCurrent.size()+Main.k;
    //find all pickup locations:
    List<Integer> pickupsInRoute = new ArrayList<Integer>();
    List<Integer> deliveriesInRoute = new ArrayList<Integer>();
    //System.out.println("PickupsInROute:");
    for (int i=1; i<Main.V; i++){
        for (int j=0; j<Main.V; j++){
            if (MyData.demand[i][j]>0 || MyData.demand[j][i]>0) && startingPositions.contains(i)==true{
                //System.out.println("STARTING POSITION FOUND "+i+" "+j);
                customersToStayInK.add(i);
                customersToStayInK.add(j);
                wTourLength--;
                if (MyData.demand[j][i]>0 && startingPositions.contains(i)==true)wTourLength--;
else if (MyData.demand[i][j] > 0 && startingPositions.contains(j) == false) {
    pickupsInRoute.add(i);
    //System.out.print(i +", ");
} else if (MyData.demand[i][j] < 0 && startingPositions.contains(i) == false) {
    deliveriesInRoute.add(i);
    //System.out.print(i +", ");
}
}

wtour = new int [wtourLength];
System.out.println("wtour length :" + wtour.length);
for (Integer element : customersToStayInK) {
    //System.out.print(element +", ");
}

// erstelle Routen listen:
//hier ansetzen um dynamics zu etablieren
for (int k=0; k<Main.k; k++) {
    List<Integer> singleRoute = new ArrayList<Integer>();
    //System.out.println("add starting position " + Main.startingPositionK[k] + " ID: " + MyRMcontrol.fileMatrixCurrent.get(Main.startingPositionK[k])[0]);
    singleRoute.add(MyRMcontrol.startingPosition[k]);
    for (int j=0; j<Main.V; j++) {
        if (MyData.demand[/*Main.startingPositionK[k]*/ MyRMcontrol.startingPosition[k]][j] > 0) {
            //System.out.println("Add delivery location " + j + " ID: " + MyRMcontrol.fileMatrixCurrent.get(j)[0]);
            singleRoute.add(j);
        }
    }
    routes.add(singleRoute);
}

// Erstelle Routen (AUSWAHL HIER)
//createInitialSolutionSimple(pickupsInRoute);
if (MyRMcontrol.ONLYvrp==false) createInitialSolutionMIXED(pickupsInRoute);
else createInitialSolutionPolar(pickupsInRoute);
//createInitialSolutionByCountingTime(pickupsInRoute);
//createInitialSolutionPickupsFirstThenDeliveries(pickupsInRoute);
System.out.println("Main.k: "+Main.k);
System.out.println("w[][][][] size: " + w.length + " x " + w[0].length + " x " + w[0][0].length + " x "
                    + w[0][0][0].length);
System.out.println(MyRMcontrol.ONLYvrp);

// verbinde den letzten customer mit dem depot
for (int kk=0; kk<Main.k; kk++) {
    System.out.println("k: "+kk + " +routes.get(kk).size()");
    if (routes.get(kk).size() > 0) w[routes.get(kk).get(routes.get(kk).size()-1)][0][routes.get(kk).size()][kk]=1;
}

// Drucke routen aus:
for (int ka=0; ka<Main.k; ka++){
//System.out.print("Route " + ka + ": ");
for (int qu=0; qu<routes.get(ka).size(); qu++){
  //System.out.print(routes.get(ka).get(qu)+", ");
}
//System.out.println();

for (int qu=0; qu<routes.get(ka).size(); qu++){
  //System.out.print(routes.get(ka).get(qu)+", ");
}
//System.out.println();

// Arrays leeren:
pickupsInRoute.clear();
routes.clear();
delRoutes.clear();

//print out starting solution:
int zaehler=0;
for (int ka=0; ka<Main.k; ka++){
  for (int i=0; i<MyData.distance.length; i++){
    for (int j=0; j<MyData.distance.length; j++){
      if (w[i][j][qu][ka] == 1 && zaehler<wtour.length){
        System.out.println("w[" + i + "]["+j+"]["+qu+"]["+ka+"]" +
                        "= " + w[i][j][qu][ka] + ");
        wtour[zaehler]=i;
        zaehler++;
      }
      //
    }
    //
  }
  //
}
wtour = wtour= new int[zaehler+1];

private void createInitialSolutionPolar(List<Integer> pickupsInRoute) throws ConcurrentModificationException{
  double[][] centers = new double[Main.k][2];
  for (int k=0; k<Main.k; k++){
    centers[k][0] = MyRMcontrol.fileMatrix.get(0)[1] + 20*Math.cos((2*Math.PI)/Main.k)*(k+1))/2;
    centers[k][1] = MyRMcontrol.fileMatrix.get(0)[2] + 20*Math.sin((2*Math.PI)/Main.k)*(k+1))/2;
    //System.out.println("Center "+k+: x: "+ centers[k][0]+ 	 y:"	+centers[k][1]);
  }
  double x1 = 0;
  double y1 = 0;
  double distToCenter;
  int futureK = 0;
  int belongingJ=0;
  for (int p=0; p<pickupsInRoute.size(); p++){
    for (int jj=1; jj<Main.V; jj++){
      if (MyData.demand[pickupsInRoute.get(p)][jj]>0/* && MySolution.startingPositions.contains(i)==false*/){
        belongingJ= jj;
        x1 = ( MyRMcontrol.fileMatrix.get(pickupsInRoute.get(p))[1] +MyRMcontrol.fileMatrix.get(jj)[1] )/2;
        y1 = ( MyRMcontrol.fileMatrix.get( pickupsInRoute.get(p)))[2] +MyRMcontrol.fileMatrix.get(jj)[2] )/2;
      }
    }
    double minDist = Integer.MAX_VALUE;
    for (int kk=0; kk<Main.k; kk++){
      distToCenter = MyRMcontrol.norm(x1, y1, centers[kk][0], centers[kk][1]);
      if (distToCenter < minDist){
        minDist = distToCenter;
        futureK=kk;
      }
    }
  }
private void createInitialSolutionMIXED(List<Integer> pickupsInRoute) throws ConcurrentModificationException{
    System.out.println("CreateInitialsolutionMixed " + MyRMcontrol.fileMatrix.size());
    int currentCustomer=0;
    int k=0;
    int[] pickups = new int[pickupsInRoute.size()];
    for (int f=0; f<pickups.length; f++){
        pickups[f] = pickupsInRoute.get(f);
        //System.out.println(pickups[f]);
    }
    while (pickupsInRoute.isEmpty()==false){
        double time = 100000;
        int i=0; i<pickups.length; i++;
        if(pickups[i]!=0 && currentCustomer==0 && MyData.TWend[pickups[i]]<time && startingPositions.contains(pickups[i])==false){
            time=MyData.TWend[pickups[i]];
            findNearest=pickups[i];
            deleteInt=i;
            for(int j=0; j<Main.V; j++){
                if(MyData.demand[pickups[i]][j]>0) belongingJ=j;
            }
        }
        else if(pickups[i]!=0 && MyData.TWend[pickups[i]]<time && MyData.TWend[pickups[i]]>= (MyData.TWend[currentCustomer]+MyData.distance[currentCustomer][pickups[i]]) && startingPositions.contains(pickups[i])==false){
            time=MyData.TWend[pickups[i]];
            findNearest=pickups[i];
            deleteInt=i;
            for(int j=0; j<Main.V; j++){
                if(MyData.demand[pickups[i]][j]>0) belongingJ=j;
            }
        }
        else if(findNearest == 0 && MyData.TWend[pickups[i]]<time && pickups[i]!=0 && startingPositions.contains(pickups[i])==false){
            time=MyData.TWend[pickups[i]];
            findNearest=pickups[i];
            deleteInt=i;
            for(int j=0; j<Main.V; j++){
                if(MyData.demand[pickups[i]][j]>0) belongingJ=j;
            }
        }
        findPosAndInsertCustomer(routes.get(k), findNearest, belongingJ, k);
    }
    //System.out.println("index to remove: " + deleteInt + " object:" + findNearest);
pickupsInRoute.remove(0);
pickups[deleteInt]=0;
k++;
if(k==Main.k){
k=0;
}
currentCustomer=routes.get(k).get(routes.get(k).size()-1);
}
System.out.println("I am here");
buildRoutesFromLists(routes);
}

private void findPosAndInsertCustomer(List<Integer> route, int customer, int recipient, int k){
    int newPos = route.size();
    List<Integer> changedRoute = new ArrayList<Integer>();
    changedRoute.addAll(route);
    //System.out.println("route size:"+route.size());
    // find Sender position
    for(int i=route.size()-2; i>=0; i--){
        //System.out.println(i);
        if((MyData.TWend[route.get(i)]+MyData.distance[route.get(i)][customer]/*+MyData.Service[customer]*/) <= MyData.TWend[customer] & &
            (MyData.TWend[customer]+MyData.distance[customer][route.get(i+1)]/*+MyData.Service[route.get(i+1)]*/) <= MyData.TWend[route.get(i+1)]){
            //System.out.println("findPosAndInsertCustomer - Customer inserted at alternative Pos successful inserted at:"+route.get(i));
            newPos = i;
        }
    }
    else{
        newPos = route.size();
        System.out.println("findPosAndInsertCustomer - Customer inserted at the end of the route");
    }
    //System.out.println("Inserting customer "+customer +" at position "+ newPos);
    changedRoute.add(newPos, customer);
    routes.set(k, changedRoute);
    //return route;
}

private int[][][][] buildRoutesFromLists(List<List<Integer>> routes2){
    for(int k=0; k<routes2.size(); k++){
        for(int q=1; q<routes2.get(k).size(); q++){
            w[routes2.get(k).get(q-1)][routes2.get(k).get(q)][q][k]=1;
            System.out.println(routes2.get(k).get(q)++ " ");
        }
        System.out.println();
    }
    return w;
}

private int[][][] buildRoutesFromListsAndSortNew(List<List<Integer>> routes2){
    List<List<Integer>> routesSorted = new ArrayList<List<Integer>>();
    List<Integer> thisRoute = new ArrayList<Integer>();
    int c1=0;
    int c2=0;
    for(int k=0; k<routes2.size(); k++){
        thisRoute = new ArrayList<Integer>();
        thisRoute.add(0, Main.startingPositionK[k]);
        routes2.get(k).remove(0);
        }
while (routes2.get(k).size() > 1) {
    double currentFurthest = Integer.MIN_VALUE;
    double currentDist;
    double[] currentRouteCentre = new double[2];
    // determine current route centre:
    for (int i = 0; i < thisRoute.size(); i++) {
        currentRouteCentre[0] += MyRMcontrol.fileMatrixCurrent.get(thisRoute.get(i))[1];
        currentRouteCentre[1] += MyRMcontrol.fileMatrixCurrent.get(thisRoute.get(i))[2];
    }
    currentRouteCentre[0] /= thisRoute.size();
    currentRouteCentre[1] /= thisRoute.size();
    // find furthest customer from route centre:
    for (int r = 0; r < routes2.get(k).size(); r++) {
        currentDist = MyRMcontrol.norm(currentRouteCentre[0], currentRouteCentre[1], MyRMcontrol.fileMatrixCurrent.get(routes2.get(k).get(r))[1], MyRMcontrol.fileMatrixCurrent.get(routes2.get(k).get(r))[2]);
        if (currentDist > currentFurthest) {
            currentFurthest = currentDist;
            c1 = routes2.get(k).get(r);
            c2 = MyRMcontrol.fileMatrixCurrent.get(routes2.get(k).get(r))[8].intValue();
        } else {
            c1 = MyRMcontrol.fileMatrixCurrent.get(routes2.get(k).get(r))[7].intValue();
        }
    }
    // remove c1 & c2 from route list:
    for (int i = 0; i < routes2.get(k).size(); i++) {
        if (routes.get(k).get(i) == c1 || routes.get(k).get(i) == c2) {
            routes.get(k).remove(i);
        }
    }
    // System.err.println("c1: "+c1+" c2: "+c2);
    // System.out.println("k size: "+routes2.get(k).size());
    if (thisRoute.size() == 1) {
        thisRoute.add(c1);
        thisRoute.add(c2);
    } else {
        // find minimum distance positions for c1 & c2:
        double posDist = Integer.MAX_VALUE;
        int posC1 = 0;
        int posC2 = 0;
        // reuse: double currentDist;
        for (int i = 0; i < thisRoute.size() - 1; i++) {
            currentDist = (MyData.distance[thisRoute.get(i)][c1] + MyData.distance[c1][thisRoute.get(i + 1)]) - MyData.distance[thisRoute.get(i)][thisRoute.get(i + 1)];
            if (currentDist < posDist) {
                posDist = currentDist;
                posC1 = i + 1;
                posC2 = c1;
            }
        }
    }
}
thisRoute.add(posC1, c1);

posDist = Integer.MAX_VALUE;
for(int i=posC1; i<thisRoute.size()-1; i++){
    currentDist = (MyData.distance[thisRoute.get(i)][c2] + MyData.distance[c2][thisRoute.get(i+1)] - MyData.distance[thisRoute.get(i)][thisRoute.get(i+1)]);
    //System.out.println("current distance: "+ currentDist);
    if(currentDist < posDist)
    {
        posDist = currentDist;
        posC2 = i+1;
        //System.out.println("New min distance c2 : "+ currentDist);
    }
}
thisRoute.add(posC2, c2);

//for(int i=0; i<thisRoute.size();i++){
//    System.err.print(thisRoute.get(i)+" ");
//} //System.out.println();
thisRoute.remove(thisRoute.size()-1);
routesSorted.add(thisRoute);

routes.clear();

for(int kk=0; kk<Main.k; kk++){
    //for(int pos=0; pos<routesSorted.get(kk).size(); pos++){
    //    System.out.print(routesSorted.get(kk).get(pos)+", ");
    //} //System.out.println();
    //System.out.println(routesSorted.get(kk).get(0));
    routes.add(routesSorted.get(kk));
    //routesSorted.get(0).add(0, Main.startingPositionK[0]);
    buildRoutesFromLists(routesSorted);
}

@Override
public Object clone()
{
    //Vorher:
    MySolution copy = (MySolution)super.clone();
    MySolution.w = deepCopyOf(w);
    MySolution.wtour = MySolution.wtour.clone();
    //solutionsCloned.add(getObjectiveValue()[0]);
    //Nachher:
    MySolution copy = (MySolution)super.clone();
    copy.wtour = wtour.clone();
    copy.w = deepCopyOf(w);

    return copy;
} // end clone
/** TRIAL **/
public static void returnToBestSolution(MySolution best){
    wtour = best.wtour.clone();
    w = deepCopyOf(best.w);
System.out.println("Solution restored");
MyObjectiveFunction.evaluateConstraints(w);

public static int[][][] deepCopy2(int[][][] original) {
    if (original == null) {
        return null;
    }

    final int[][][] result = new int[original.length][original[0].length + 1][original[0][0].length];
    for (int i = 0; i < original.length; i++) {
        for (int j = 0; j < original.length; j++) {
            for (int q = 0; q <= original.length; q++) {
                result[i][j][q] = Arrays.copyOf(original[i][j][q], original[i][j][q].length);
            }
        }
    }

    return result;
}

public static int[][][] deepCopy3(int[][][] original) {
    if (original == null) {
        return null;
    }

    final int[][][] result = new int[original.length][original[0].length + 1][original[0][0].length];
    for (int i = 0; i < original.length; i++) {
        for (int j = 0; j < original.length; j++) {
            for (int q = 0; q <= original.length; q++) {
                result[i][j][q] = w[i][j][q].clone();
            }
        }
    }

    return result;
}

@SuppressWarnings("unchecked")
public static <T> T[] deepCopy2Of(T[] array) {
    if (0 >= array.length) return array;

    return (T[]) deepCopyOf(array,
    Array.newInstance(array[0].getClass(), array.length), 0);
}

@SuppressWarnings("unchecked")
public static <T> T[] deepCopyOf(T[] array) {
    if (0 >= array.length) return array;

    return (T[]) deepCopyOf(array,
    Array.newInstance(array[0].getClass(), array.length), 0);

/** End Trial ***/
```java
private static Object deepCopyOf(Object array, Object copiedArray, int index) {
    if (index >= Array.getLength(array)) return copiedArray;
    Object element = Array.get(array, index);
    if (element.getClass().isArray()) {
        Array.set(copiedArray, index, deepCopyOf(
            element,
            Array.newInstance(element.getClass().getComponentType(),
            Array.getLength(element)),
            0));
    } else {
        Array.set(copiedArray, index, element);
        return deepCopyOf(array, copiedArray, ++index);
    }
}

@Override
public String toString() //For decision variable W
{
    StringBuffer s = new StringBuffer();
    s.append("Solution value: " +getObjectiveValue()[0] +"\n");
    s.append("Sequence: [ 0, ");
    // System.out.println("solution clone: ");
    for( int k = 0; k < Main.k; k++) {
        for( int q = 0; q < /*Main.V*/w[0][0].length; q++) {
            for( int j = 0; j < /*Main.V*/w.length; j++) {
                if(w[i][j][q][k]==1 && j!=0)
                s.append( j);
                //System.out.print(j+" ");
                s.append( ", " );
            } else if(w[i][j][q][k]==1 && MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, j)){
                s.append( j);
                //System.out.print(j+" ");
                s.append( ", " );
            }
        }
    }
    s.append("] ");
    return s.toString();
}
```

```java
static int searchibyID(List<Double[]> List, int ID){
    int item = 9999;
    for(int i = 0; i<List.size(); i++){
        if(List.get(i)[0] == ID) item = i;
    }
    return item;
}
```

```java
// end class MySolution
```
E.13. MySwapMove class

```java
import java.util.Objects;
import org.coinor.opents.*;

public class MySwapMove implements /*Complex*/Move {
    public int customer; //both were not static before
    public int movement;
    public int operator;
    //public static int[][][] w; //wenn dies aktiviert wird, wird die loesung nicht veraendert!
    public static int[] pos1w; //= new int[8];
    public static int[] pos2w; //= new int[8];
    public static boolean cmpair; //= false;
    public static boolean mcpair; //= false;

    public MySwapMove(int customer, int movement, int operator) {
        System.out.println("c:" + customer + " m:" + movement+ " O:" +operator);
        this.customer = customer; //both changes from this.customer=cus
tomer;
        this.movement = movement;
        this.operator = operator;
    }

    public void operateOn(Solution soln) { // for W
        int[][][][] w = MySolution.w; //Before: ((MySolution)soln).w;
        int[] tour = MySolution.wtour;
        //int w[[][]][] = MySolution.w;

        //New Operators:
        double now = System.currentTimeMillis();
        //MySolution.w = Arrays.copyOf(Operator.rememberW2, Operator.rememberW
2.length);
        MySolution.w = MySolution.deepCopyOf(Operator.rememberW2);
        if(operator==1) Operator.SwapPair(soln,/*w,*/pos1w, pos2w, cmpair, mcpair, false, customer, movement, 0);
        else if(operator==2) Operator.SwapCustomerInRouteListeEco(soln, customer, movement, 3);
        else if(operator==3) { //Operator.SwapCustomerInRoute(w, pos1w,** pos2w, cmpair, mcpair, */false, customer, movement, 0);
            Operator.SwapCustomerInRouteListeEco(soln, customer, movement, 0);
            } else if(operator==4) { //Operator.SwapCustomerInRoute(w, pos1w,** pos2w, cmpair, mcpair, */false, customer, movement, 1);
            Operator.SwapCustomerInRouteListeEco(soln, customer, movement, 1);
            } else if(operator==5) { //Operator.SwapCustomerInRoute(w, pos1w,** pos2w, cmpair, mcpair, */false, customer, movement, 2);
            Operator.SwapCustomerInRouteListeEco(soln, customer, movement, 2);
            } else if(operator==6) Operator.SwapCustomerInRouteListeEco(soln, cus
tomer, movement, 4);
056  // else if(operator==7) Operator.MoveAngle(soln, customer, movement);
057  // else if(operator==8) Operator.PDExchange(soln, /*w,*/ pos1w, pos2w, cmpair, mcpair, false, customer, movement, 4);
058  // else if(operator==9) Operator.XExchange2(soln, customer, movement);
059  // else if(operator==10) Operator.SwapCustomerInRouteListeEco(soln, customer, movement, 5);
060  // else if(operator==11) Operator.SwapCustomerInRouteListeEco(soln, customer, movement, 7);
061  
062 }  /**
063  * Identify a move for SimpleTabuList
064  */
065  @Override
066  public int hashCode() {
067  return customer;
068  }
069  }
070  }
071  //System.out.println(String.format("Operator: \"+ operator+ \" \"+dist);
072  //System.out.println();
073  }  // end operateOn
074  
075  /** Identify a move for SimpleTabuList */
076  /** test 28/03/17: test of setting up a move for the tabu list**/
077  @Override
078  public int hashCode() {
079  return customer;
080  }
081  }
082  
083  //System.out.println(String.format("Operator: \"+ operator+ \" \"+dist);
084  //System.out.println();
085  }  // end operateOn
086  
087  /*
088  * @Override
089  */
090  
091  
092  
093  
094  
095  
096  
097  
098  
099  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  

E.14. MyTabuList class

```java
public class MyTabuList extends SimpleTabuList
{
    public /*final*/ static int MAX_TENURE = 10; // 30;
    private static final Main TabuSearch = null;
    public static int iterationsWithoutNewBest = 0;
    public static int iterationCounter = 0;
    public static TabuSearch theTS;
    boolean checkSwitch = true;
    boolean checkSwitch2 = false;
    boolean checkSwitch3 = false;
    List<Double> visitedSolutions = new ArrayList<Double>();

    public void newBestSolutionFound(TabuSearchEvent evt)
    {
        // Decrease tenure
        // System.out.println("current tenure: " + getTenure()); /**
        setTenure( Math.max( 1, (int) ( 0.75 * getTenure() ) ) );
        //System.out.println("MYTABUSEARCHLISTENER newBestSolutionFound - tenure decreased: " + getTenure());
        iterationsWithoutNewBest = 0;
        if(MyRMcontrol.ONLYvrp==true){
            MyRMcontrol.results2[5] = evt.getTabuSearch().getIterationsCompleted();
            MyRMcontrol.results2[6] = System.currentTimeMillis() - Main.startzeit;
        }
        // Print best solution:
        System.out.println(evt.getTabuSearch().getIterationsCompleted() + ": " +
                          evt.getTabuSearch().getBestSolution().getObjectiveValue()[0]);
        MyObjectiveFunction.print = true;
        // MyObjectiveFunction.evaluateConstraints(MySolution.w);
        if(MyRMcontrol.desktopPrintout==true && MyRMcontrol.caseStudy == true)
            MyCanvas.updateCanvasCS(MySolution.w, 0);
        else if(MyRMcontrol.desktopPrintout==true) MyCanvas.updateCanvas(MySolution.w, 670);
        // Statistics: Has a solution been visited before:
        Main.operatorPerformance[MyMoveManager.operator-1] += 20;
        //System.out.println("New best solution move made");
        changeOperators(MyMoveManager.opPostOptimization);
        checkSwitch2 = true;
        if(evt.getTabuSearch().getBestSolution().getObjectiveValue()[0]<588.4)evt.getTabuSearch().stopSolving();
        if(MyRMcontrol.ONLYvrp==true && evt.getTabuSearch().getIterationsCompleted() > 30){
            changeOperators(MyMoveManager.opPostOptimization);
            checkSwitch3 = true;
        }
    }
}
```
public void unimprovingMoveMade(TabuSearchEvent evt) {
    // Increase tenure
    System.out.println("current tenure: " + getTenure());
    setTenure(Math.min(MAX_TENURE, getTenure() + 2));
    System.out.println("MYTABUSEARCHLISTENER unimprovingMoveMade - tenure increased " + getTenure());

    //Statistics: Has a solution been visited before:
    if (visitedSolutions.contains(evt.getTabuSearch().getCurrentSolution().getObjectiveValue()[0]) == false) {
        visitedSolutions.add(evt.getTabuSearch().getCurrentSolution().getObjectiveValue()[0]);
        Main.operatorPerformance[MyMoveManager.operator-1] += 9;
        System.out.println("Unimproving move made");
    }
}

public void newCurrentSolutionFound(TabuSearchEvent evt) {
    /*
     * MAX_TENURE = Math.min(MyRMcontrol.fileMatrixCurrent.size()/3, 30);
     */
    if (MyRMcontrol.desktopPrintout==true && MyRMcontrol.caseStudy == true) MyCanvas.updateCanvasCS(MySolution.w, 0);
    else if (MyRMcontrol.desktopPrintout==true) MyCanvas.updateCanvas(MySolution.w, 50);
    if (evt.getTabuSearch().getIterationsCompleted()%25==0) System.out.println("ITERATION "+ evt.getTabuSearch().getIterationsCompleted());

    // q:
    if (evt.getTabuSearch().getIterationsCompleted()%10==0) {
        MyMoveManager.q = 10;
        System.out.println("q reset to 10");
    } else if (evt.getTabuSearch().getIterationsCompleted()%2==0) {
        MyMoveManager.q = Math.max(1, MyMoveManager.q-1);
        System.out.println("q reduced to "+MyMoveManager.q);
    }

    System.out.println("q: "+MyMoveManager.q+" fMC size: " + MyRMcontrol.fileMatrixCurrent.size()+" fMC size/2: " + MyRMcontrol.fileMatrixCurrent.size()/2);
    if (MyMoveManager.q >= MyRMcontrol.fileMatrixCurrent.size()-Main.k/2) {
        MyMoveManager.q = Math.max(1, (MyRMcontrol.fileMatrixCurrent.size()-Main.k)/2);
        System.out.println("Adjusted q fMC size: "+MyMoveManager.q+" fMC size/2: " + MyRMcontrol.fileMatrixCurrent.size()/2);
    }

    // System.out.println("q in MyTabuList: "+ MyMoveManager.q);

    if(iterationsWithoutNewBest==50){
        System.out.println("Best solution restored");
        MySolution.returnToBestSolution((MySolution)evt.getTabuSearch().getBestSolution());
    }

    iterationsWithoutNewBest++;

    // Adaption of operator selection:
    if (MyRMcontrol.t==0) {
        MyMoveManager.opDistribution[21] = 0;
        Main.operatorPerformance[21] = 0;
        Main.operatorUtilisation[21] = 0;
    }
}
if(evt.getTabuSearch().getIterationsCompleted()%5 == 0) {
    //change operator selection, clear performance & util., print new selection
    System.out.println("Adjustment of move operators: ");
    for(int i=0; i<MyMoveManager.opDistribution.length; i++) {
        if(Main.operatorPerformance[i] > 0 && Main.operatorUtilisation[i] > 0)
            MyMoveManager.opDistribution[i] = MyMoveManager.opDistribution[i] * (1 - .2) + (.2 * (Main.operatorPerformance[i]/Main.operatorUtilisation[i]));
    }
    System.out.println("\n\n| Sum opDistribution: "+MyMoveManager.arraySum(MyMoveManager.opDistribution));
}

if(evt.getTabuSearch().getIterationsCompleted() > 10 && checkSwitch == true && /*Main.iterations >= 300*/MyRMcontrol.ONLYvrp==true)
    changeOperators(MyMoveManager.opDistributionLocalImp);

if(checkSwitch3 == true && iterationsWithoutNewBest>=2)
    changeOperators(MyMoveManager.opDistributionLocalImp);

if(checkSwitch == true) iterationCounter++;

public void tabuSearchStarted(TabuSearchEvent evt) {}
etObjectiveValue()[0] == false) {
    Main.operatorPerformance[MyMoveManager.operator-1] += 13;
    //System.out.println("Improving move made");
}

public void noChangeInValueMoveMade(TabuSearchEvent arg0) {
    //System.out.println("no Change In Value Move Made");
}

public static void stopTS(){
    theTS.stopSolving();
}

static int[][][][] wtourtoW (int[] wtour){
    int[][][][] w = new int[Main.V][Main.V][Main.V+1][Main.k];
    int q=1;
    int k=0;
    for(int i=0; i<wtour.length-1; i++){
        if(wtour[i+1] != 0){
            w[wtour[i]][wtour[i+1]][q][k] = 1;
            //System.out.println("w[" +wtour[i]+ "][" +wtour[i+1]+ "]["+q+"]["+k+" ] \t");
            q++;
        } else{
            w[wtour[i]][0][q][k] = 1;
            //System.out.println("w[" +wtour[i]+ "][0]["+q+"]["+k+" ] \t");
            k++;
            q=1;
        }
    }
    return w;
}

private void changeOperators(double[] distribution){
    MyMoveManager.opDistribution=Arrays.copyOf(distribution, distribution.length);
    //MyMoveManager.opDistributionStandart
    printOperators();
}

private void printOperators(){
    for(int i=0; i<MyMoveManager.opDistribution.length; i++) System.out.print(MyMoveManager.opDistribution[i]+", ");
    System.out.println();
}
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.Random;
import org.coinor.opents.Solution;

public class Operator {
    public static int[] position = new int[4];
    public static int[][][] rememberW;
    public static int[][][] rememberW2;
    public static int[] p3w;
    public boolean cm;
    public boolean mc;
    static Random randomGenerator = new Random();
    private static boolean cmpair;
    private static boolean mcpair;
    static int[][][] wNew; /// changed to static to check solution problem
    static List<Integer> wList = new ArrayList<Integer>();
    public static List<List<Integer>> wLists = new ArrayList<List<Integer>>();
    static boolean error = false;
    static int previousPosition = 0;

    public static void findPositions2(int[][][] w, int customer, int movement, int[] p1w, int[][] p2w){
        for (int i = 0; i < w.length; i++){
            for(int j=0; j<w.length; j++){
                for(int q=0; q<w.length; q++){
                    for(int k=0; k<Main.k; k++){
                        if (w[i][j][q][k]==1 && i==movement && j==customer ){
                            mcpair = true;
                            //System.out.println("mc"+mcpair+"w[" +i+ "]"+"["+j+"]"+"["+
                            +k+"]"+ w[customer][movement][q][k]);
                        }
                        if (w[i][j][q][k]==1 && i==customer && j==movement ){
                            cmpair=true;
                            //System.out.println("cm"+cmpair +"w[" +i+ "]"+"["+j+"]"+"["+
                            +k+"]"+ w[i][j][q][k]);
                        }
                        if( w[i][customer][q][k] == 1 ) {
                            plw[0] = i;
                            plw[1] = customer;
                            plw[2] = q;
                            plw[3] = k;
                        }
                        if( w[customer][j][q][k] == 1 ) {
                            plw[4] = customer;
                            plw[5] = j;
                            plw[6] = q;
                            plw[7] = k;
                        }
                    }
                }
            }
            //pos2 = pos1 + movement;
            //int customer2 = pos1 + movement; //Aenderung: movement ist jetzt c
            for( int i = 0; i < w.length; i++){
for (int j = 0; j < w.length; j++) {
  for (int q = 0; q <= w.length; q++) {
    for (int k = 0; k < Main.k; k++) {
      if (w[i][customer][q][k] == 1 && i == movement) mcpair = true;
    }
  }
}
for (int j = 0; j < w.length; j++) {
  for (int q = 0; q <= w.length; q++) {
    for (int k = 0; k < Main.k; k++) {
      if (w[customer][j][q][k] == 1 && j == movement) cmpair = true;
    }
  }
}
if (w[i][movement][q][k] == 1) {
p2w[0] = i;
p2w[1] = movement;
p2w[2] = q;
p2w[3] = k;
}
if (w[movement][j][q][k] == 1) {
p2w[4] = movement;
p2w[5] = j;
p2w[6] = q;
p2w[7] = k;
}
MySwapMove.pos1w = Arrays.copyOf(p1w, p1w.length);
MySwapMove.pos2w = Arrays.copyOf(p2w, p2w.length);
MySwapMove.cmpair = cmpair;
MySwapMove.mcpair = mcpair;
}
static void SwapCustomerInRouteListeEco(Solution soln, int customer, int movement, int selectionCriteria){
  wLists.clear();
  int[][][][] w = MySolution.w;
  int[][][][] wNew = new int[Main.V][Main.V][Main.V+3][Main.k];
  rememberW = MySolution.deepCopyOf(w);
  error = false;
  fourDto2DListe(w);
  RemoveSingleFromList(customer);
  if (wLists.get(position[2]).size() < 3) error = true;
  findSinglePositionList(customer, movement, selectionCriteria);
  Lists2DToWnew(wNew);
  if (error == false) MySolution.w = MySolution.deepCopyOf(wNew); // Arrays.copyOf(wNew, wNew.length); // try deepcopy // try lists to wnew(w)
}
static void SwapCustomerInRouteToPos(Solution soln, int customer, int movement){
  wLists.clear();
  int[][][][] w = MySolution.w;
  int[][][][] wNew = new int[Main.V][Main.V][Main.V+3][Main.k];
  rememberW = MySolution.deepCopyOf(w);
  error = false;
  fourDto2DListe(w);
  RemoveSingleFromList(customer);
  wLists.get(position[2]).add(movement, customer);
  error = false;
  localRouteOpt(wLists.get(position[2]).size() - 6, wLists.get(position[2]).size() - 1, position[2], wLists);
  Lists2DToWnew(wNew);
  if (error == false) MySolution.w = MySolution.deepCopyOf(wNew); // Arrays.copyOf(wNew, wNew.length); // try deepcopy // try lists to wnew(w)
}
public static void SwapCustomerInRouteListe(int[][][][] w, int[] plw, boolean OFSC, int customer, int movement, int selectionCriteria){

int[][][] wNew = new int[Main.V][Main.V][Main.V+3][Main.k];
//Remember W = MySolution.deepCopyOf(w);
findSinglePosition(w, customer, plw);
findGreedyInRoute(w, plw[1]/**, position**/, selectionCriteria, movement);
fourDtoListe(w);
int listPosBeforeCurrentK=0;
int route=0;
//Zaehle positionen vor current K:
while(route < position[2]){
    listPosBeforeCurrentK++;
    if(wList.get(listPosBeforeCurrentK)==0) route++;
}
//finde currentQ:
int currentQ=0;
pos: for(int i=listPosBeforeCurrentK; i<wList.size(); i++){
    if(wList.get(i)==customer){
        currentQ=i;
        break pos;
    }
}
//Insert customer at new position:
//System.out.println("C: "+customer+" in K: "+position[0]+" to position "+position[2]);
//System.out.println("currentQ: "+ currentQ);
//System.out.println("position[0]: "+ (listPosBeforeCurrentK+position[0]));
if(position[0]!=0){
    if (currentQ < (listPosBeforeCurrentK+position[0])){
        // Insert:
        wList.add((listPosBeforeCurrentK+position[0]), customer);
        // Remove:
        if(route==0) wList.remove(currentQ); //geandert
        else wList.remove(currentQ);
    } else if (currentQ > (listPosBeforeCurrentK+position[0])){
        // Insert:
        wList.add((listPosBeforeCurrentK+position[0]), customer);
        // Remove:
        wList.remove(currentQ+1);
    }
}
//System.out.println("Route "+route);
//System.out.println("List pos before current K "+listPosBeforeCurrentK);
//System.out.println("current K "+position[2]);
ListeToWnew(wNew);
MySolution.w=Arrays.copyOf(wNew, wNew.length);
private static void ListeToWnew(int wNew[][][][]){
    int k=0;
    int q=1;
    for(int j=1; j<wList.size(); j++){
        wNew[wList.get(j-1)][wList.get(j)][q][k]=1;
        //System.out.println("WNew["+wList.get(j-1)+"]["+wList.get(j)+"]["+q+"]["+k+"]= 1");
        q++;
        if(wList.get(j)==0 && j!= wList.size()){

private static void fourDtoListe(int w[][][][]){
    wList.clear();
    for(int k=0; k<Main.k; k++){
        for(int q=1; q<Main.V; q++){
            for(int i=0; i<Main.V; i++){
                for(int j=0; j<Main.V; j++){
                    if(w[i][j][q][k]==1){
                        wList.add(i);
                        //System.out.print(i + " ,");
                    }
                }
            }
        }
        wList.add(0);
        //System.out.print("0 ,");
    }
    wList.clear();
    for(int k=0; k<Main.k; k++){
        List<Integer> row = new ArrayList<Integer>();
        for(int q=1; q<Main.V; q++){
            for(int i=0; i<Main.V; i++){
                for(int j=0; j<Main.V; j++){
                    if(w[i][j][q][k]==1){
                        row.add(i);
                        //System.out.print(row.get(row.size()-1) + " ,");
                    }
                }
            }
            row.add(0);
            wLists.add(row);
            //System.out.println();
        }
        //System.out.print("0 ,");
    }
    wLists.clear();
    for(int k=0; k<Main.k; k++){
        List<Integer> row = new ArrayList<Integer>();
        for(int q=1; q<Main.V; q++){
            for(int i=0; i<Main.V; i++){
                for(int j=0; j<Main.V; j++){
                    if(w[i][j][q][k]==1){
                        //System.out.print(i);
                        row.add(i);
                        //System.out.print(row.get(row.size()-1) + " ,");
                    }
                }
            }
            row.add(0);
        }
    }
    wLists.clear();
    for(int k=0; k<Main.k; k++){
        List<Integer> row = new ArrayList<Integer>();
        for(int q=1; q<Main.V; q++){
            for(int i=0; i<Main.V; i++){
                for(int j=0; j<Main.V; j++){
                    if(w[i][j][q][k]==1){
                        //System.out.print(i);
                        row.add(i);
                        //else System.out.println("ERROR");
                    }
                }
            }
            row.add(0);
        }
    }
}

static void fourDto2DListe(int w[][][][]){
    wLists.clear();
    for(int k=0; k<Main.k; k++){
        List<Integer> row = new ArrayList<Integer>();
        for(int q=1; q<Main.V; q++){
            for(int i=0; i<Main.V; i++){
                for(int j=0; j<Main.V; j++){
                    if(w[i][j][q][k]==1){
                        //System.out.print(i);
                        row.add(i);
                        //else System.out.println("ERROR");
                    }
                }
            }
            row.add(0);
            wLists.add(row);
            //System.out.println();
        }
        //System.out.print("0 ,");
    }
    wLists.clear();
    for(int k=0; k<Main.k; k++){
        List<Integer> row = new ArrayList<Integer>();
        for(int q=1; q<Main.V; q++){
            for(int i=0; i<Main.V; i++){
                for(int j=0; j<Main.V; j++){
                    if(w[i][j][q][k]==1){
                        //System.out.print(i);
                        row.add(i);
                        //else System.out.println("ERROR");
                    }
                }
            }
            row.add(0);
        }
    }
}

static void fourDto2DListePlusReturn(int w[][][][]){
    wLists.clear();
    for(int k=0; k<Main.k; k++){
        List<Integer> row = new ArrayList<Integer>();
        for(int q=1; q<Main.V; q++){
            for(int i=0; i<Main.V; i++){
                for(int j=0; j<Main.V; j++){
                    if(w[i][j][q][k]==1){
                        //System.out.print(i);
                        row.add(i);
                        //else System.out.println("ERROR");
                    }
                }
            }
            row.add(0);
        }
    }
}
private static void Lists2DToWnew(int wNew[][][][]) {
    // printRouteFromList();
    for (int k = 0; k < wLists.size(); k++) {
        int q = 1;
        for (int i = 0; i < wLists.get(k).size() - 1; i++) {
            wNew[wLists.get(k).get(i)][wLists.get(k).get(i + 1)][q][k] = 1;
            q++;
        }
        if (wLists.get(k).size() > 0) wNew[wLists.get(k).get(wLists.get(k).size() - 1)][0][q][k] = 1;
    }
}

private static void ListsPlusReturnToWnew(int wNew[][][][]) {
    // System.out.println("wLists to wNew");
    for (int k = 0; k < wLists.size(); k++) {
        int q = 1;
        for (int i = 0; i < wLists.get(k).size() - 2; i++) {
            // System.out.println(wLists.get(k).get(i) + " - " + wLists.get(k).get(i + 1));
            wNew[wLists.get(k).get(i)][wLists.get(k).get(i + 1)][q][k] = 1;
            q++;
        }
        // System.out.println();
        if (wLists.get(k).size() > 1) wNew[wLists.get(k).get(wLists.get(k).size() - 2)][0][q][k] = 1;
    }
}

private static void removeFromLists(int customer, int k) {
    // for (int k = 0; k < wLists.size(); k++) {
        if (wLists.get(k).get(i) == customer) wLists.get(k).remove(i);
    // }
    // System.out.println("Remove customer " + customer + " from vehicle "+ k);
    // }
    // printRouteFromList();
}

private static void insertInLists(int customer, int position, int vehicle) {
    // System.out.println("position: " + position);
    // printRouteFromList();
    if (position <= wLists.get(vehicle).size()) wLists.get(vehicle).add(position, customer);
    else if (wLists.get(vehicle).size() == 0) {
        wLists.get(vehicle).add(0);
        wLists.get(vehicle).add(customer);
    } else wLists.get(vehicle).add(customer);
}

static void printRouteFromList() {
    for (int k = 0; k < wLists.size(); k++) {
        for (int i = 0; i < wLists.get(k).size(); i++) {
            System.out.print(wLists.get(k).get(i) + ",");
        }
        System.out.println();
    }
    System.out.println();
}
//private static void printW(int[][][][] ww){
    // for (int ka=0; ka<Main.k; ka++) {
        // for (int qu=1; qu<MyData.distance.length; qu++) {
            // for (int i=0; i<MyData.distance.length; i++) {
                // System.out.println();
                // System.out.print("w["
                    + i
                    + "][" + j
                    + "][" + qu
                    + "][" + ka
                    + "]" =
                    ww
                    [i
                    ][j
                    ][qu
                    ][ka
                    ]);  
            }
        // }
    // }
    System.out.println();
    //}
}

public static void SwapCustomerInRouteReverse (boolean OFSCRR){
    MySolution.w=Arrays.copyOf(rememberW,
            rememberW.length);
    //MyObjectiveFunction.wx = Arrays.copyOf(rememberW,
        rememberW.length);
}

public static int[][][][] SwapPairListe (Solution soln,
    int[][][] p1w, int[][][] p2w, boolean cm, boolean mc, boolean OFSM,
    int customer, int movement,
    int[] selectionCriteria){
    int[][][][] w = MySolution.w;
    int[][][][] wNew = new int[Main.V][Main.V][Main.V+3][Main.k];
    rememberW = MySolution.deepCopyOf(w);
    findPositions(w,
            customer,
            movement,
            p1w,
            p2w);
    findBestGreedyPosition(w,
            p1w[1],
            p2w[1],
            position,
            selectionCriteria);
    findEmptyPositions(w);
    fourDtoListe(w);
    if(position[0]!=0){
        find pos
        int listPosBeforeCurrentK=0;
        int route=0;
        // Zähle positionen vor current K:
        while(route < position[3]){
            listPosBeforeCurrentK++;
        }
        int listPosBeforeFutureK=0;
        int route2=0;
        // Zähle positionen vor future K:
        while(route2 < position[2]){
            listPosBeforeFutureK++;
        }
        //finde currentQ:
        int currentQc=0;
        int currentQm=0;
        for(int i=listPosBeforeCurrentK; i<list.size(); i++){
            if(list.get(i)==customer){
                currentQc=i;
            }
            if(list.get(i)==movement){
                currentQm=i;
            }
        }
    }
}
```
if (currentQc < (listPosBeforeFutureK+position[0])){
    wList.add({listPosBeforeFutureK+position[0]}, customer);
    wList.add({listPosBeforeFutureK+position[1]}, movement);
    // insert
    wList.remove(currentQm);
    wList.remove(currentQc);
}
else if (currentQc > (listPosBeforeFutureK+position[0])){
    wList.remove(currentQm);
    wList.remove(currentQc);
    // remove
    wList.add((listPosBeforeFutureK+position[1]), movement);
    wList.add((listPosBeforeFutureK+position[0]), customer);
}

for(int i=0; i<wList.size(); i++){
    System.out.print(wList.get(i)+", ");
}
System.out.println();

MySolution.w=Arrays.copyOf(wNew, wNew.length);
return w;
}

public static void SwapPairReverse (Solution solution, boolean OFSM){
    int[][][] w = ((MySolution)solution).w;
    MyObjectiveFunction.wx = Arrays.copyOf(rememberW, rememberW.length);
    MySolution.w=Arrays.copyOf(rememberW, rememberW.length);
    //Arrays.fill(rememberW, 0);
}

public static void PDExchange(Solution soln,
    int[] p1w, int[] p2w, boolean cm, boolean mc,
    boolean OFSM, int customer, int movement, int selectionCriteria){
    wLists.clear();
    int[][][] w = MySolution.w;
    int[][][] wNew = new int[Main.V][Main.V][Main.V+3][Main.k];
    rememberW = MySolution.deepCopyOf(w);
    int[] secondPair = new int[2];
    findPositions(w, customer, movement, p1w, p2w); // sucht 2D-Liste der Werte
    findBestGreedyPosition(w, p1w[1], p2w[1]/**, position/**, selectionCriteria ); // sucht greedy locations zum einordnen von p1, p2
    findEmptyPositions(w);
    System.out.println("SwapPair: c= "+ customer + " m = "+ movement + "from k : "+position[3]++ to "+position[2]++ q1:"+ position[0]++ q2:"+position[1]);
    printW(w);
    fourDto2DListe(w);
    find2ndPair(w, position[2], secondPair);
    if(wLists.get(0).size()+wLists.get(1).size()+wLists.get(2).size()>33){
        System.out.println("ZU LANG");
        MyTabuList.stopTS();
    }
    printRouteFromList();
    removeFromLists(customer, position[3]);
    removeFromLists(movement, position[3]);
    if(secondPair[0]!= customer && secondPair[0] != 0){
```

removeFromLists(secondPair[0], position[2]);
removeFromLists(secondPair[1], position[2]);

findPositionsArrayList(customer, movement, position[2]);
insertInLists(customer, position[0], position[2]);
insertInLists(movement, position[1], position[2]);

if(secondPair[0]!= customer && secondPair[0] != 0){
    findPositionsArrayList(secondPair[0], secondPair[1], position[3]);
    insertInLists(secondPair[0], position[0], position[3]);
    insertInLists(secondPair[1], position[1], position[3]);
}


Lists2DToWnew(wNew);
MySolution.w=MySolution.deepCopyOf(wNew);//Arrays.copyOf(wNew, wNew.length);
//try deepcopy // try lists to wnew(w)
// System.out.println("w AFTER: ");
// printW(((MySolution)soln).w);
/*return w;*/

public static void XExchange2(Solution soln, int customer, int movement){
    wLists.clear();
    int[][][] w = MySolution.w;
    int[][][] wNew = new int[Main.V][Main.V][Main.V+3][Main.k];
    rememberW = MySolution.deepCopyOf(w);
    fourDto2DListe(w);

    int cpos = 0;
    int mpos = 0;
    int kk = 0;
    K: for(int k=0; k<wLists.size(); k++){
        for(int i=0; i<wLists.get(k).size(); i++){
            kk = k;
            if(wLists.get(k).get(i) == customer{
                cpos = i;
            }
            if(wLists.get(k).get(i) == movement{
                mpos = i;
                break K;
            }
        }
    } //zur proble:
    reSortSelection2(cpos, mpos, kk, wLists);
    checkForPDerror(cpos, mpos, kk, wLists);
    Lists2DToWnew(wNew);
    MySolution.w=MySolution.deepCopyOf(wNew);//Arrays.copyOf(wNew, wNew.length);
    //try deepcopy // try lists to wnew(w)
public static void MoveAngle(Solution soln, int customer, int movement){
    wLists.clear();
    int[][][][] w = MySolution.w;
    int[][][][] wNew = new int[Main.V][Main.V][Main.V+3][Main.k];
    rememberW = MySolution.deepCopyOf(w);
    fourDto2DListe(w);
    RemoveSingleFromList(customer);
    insertInLists(customer, movement, position[2]);
    Lists2DToWnew(wNew);
    MySolution.w=MySolution.deepCopyOf(wNew);
    //Arrays.copyOf(wNew, wNew.length);
}  

public static void SwapPairGEO(Solution soln, int customer, int movement){
    wLists.clear();
    int[][][][] w = MySolution.w;
    int[][][][] wNew = new int[Main.V][Main.V][Main.V+3][Main.k];
    rememberW = MySolution.deepCopyOf(w);
    fourDto2DListe(w);
    position[2]=movement;
    int KtoGo = movement;
    // System.out.println("j: "+position[2]);
    movement = MySolution.searchbyID(MyRMcontrol.fileMatrixCurrent,MyRMcontrol.fileMatrixCurrent.get(customer)[8].intValue());
    RemoveSingleFromList(customer);
    RemoveSingleFromList(movement);
    findPositionsIncrementalAndInsert(customer, movement, KtoGo);
    //System.out.println("SwapPairGEO: move customer= "+ customer + " and "+ movement + "to vehicle "+KtoGo +" with "+ wLists.get(KtoGo).size()+ " positions ");
    Lists2DToWnew(wNew);
    MySolution.w=MySolution.deepCopyOf(wNew);
    //Arrays.copyOf(wNew, wNew.length);
    //try deepcopy // try lists to wnew(w)
}

public static void SwapPairRandIncr(Solution soln, int customer, int movement){
    wLists.clear();
    int[][][][] w = MySolution.w;
    int[][][][] wNew = new int[Main.V][Main.V][Main.V+3][Main.k];
    rememberW = MySolution.deepCopyOf(w);
    fourDto2DListe(w);
    int position[2]=movement;
    int[] KtoGo = movement;
    // System.out.println("j: "+position[2]);
    movement = MySolution.searchbyID(MyRMcontrol.fileMatrixCurrent,MyRMcontrol.fileMatrixCurrent.get(customer)[8].intValue());
    RemoveSingleFromList(customer);
    RemoveSingleFromList(movement);
    findPositionsIncrementalAndInsertAnyK(customer, movement);
    //System.out.println("SwapPairGEO: move customer= "+ customer + " and "+ movement + "to vehicle "+KtoGo +" with "+ wLists.get(KtoGo).size()+ " positions ");
    Lists2DToWnew(wNew);
    MySolution.w=MySolution.deepCopyOf(wNew);
    //Arrays.copyOf(wNew, wNew.length);
    //try deepcopy // try lists to wnew(w)
}

public static void SwapPairRandIncr4(Solution soln, int customer, int movement){
    wLists.clear();
    int[][][][] w = MySolution.w;
    int[][][][] wNew = new int[Main.V][Main.V][Main.V+3][Main.k];
    rememberW = MySolution.deepCopyOf(w);
    fourDto2DListe(w);
    int[] position[2]=movement;
    int[] KtoGo = movement;
    // System.out.println("j: "+position[2]);
    movement = MySolution.searchbyID(MyRMcontrol.fileMatrixCurrent,MyRMcontrol.fileMatrixCurrent.get(customer)[8].intValue());
    RemoveSingleFromList(customer);
    RemoveSingleFromList(movement);
    findPositionsIncrementalAndInsertAnyK(customer, movement);
    //System.out.println("SwapPairGEO: move customer= "+ customer + " and "+ movement + "to vehicle "+KtoGo +" with "+ wLists.get(KtoGo).size()+ " positions ");
    Lists2DToWnew(wNew);
    MySolution.w=MySolution.deepCopyOf(wNew);
    //Arrays.copyOf(wNew, wNew.length);
    //try deepcopy // try lists to wnew(w)
}

public static void RemoveFromListRegretDist(Solution soln, int customer, int movement){
    wLists.clear();
    int[][][][] w = MySolution.w;
    int[][][][] wNew = new int[Main.V][Main.V][Main.V+3][Main.k];
    rememberW = MySolution.deepCopyOf(w);
    fourDto2DListe(w);
    RemoveSingleFromList(customer);
    RemoveSingleFromList(movement);
    findPositionsIncrementalAndInsertAnyK(customer, movement);
    //System.out.println("SwapPairGEO: move customer= "+ customer + " and "+ movement + "to vehicle "+KtoGo +" with "+ wLists.get(KtoGo).size()+ " positions ");
    Lists2DToWnew(wNew);
    MySolution.w=MySolution.deepCopyOf(wNew);
    //Arrays.copyOf(wNew, wNew.length);
    //try deepcopy // try lists to wnew(w)
}
wLists.clear();
int[][][] w = MySolution.w;
int[][][] wNew = new int[Main.V][Main.V][Main.V+3][Main.k];
rememberW = MySolution.deepCopyOf(w);

for(int i=0; i<MyMoveManager.removedCustomers.size(); i++){
    RemoveSingleFromList(MyMoveManager.removedCustomers.get(i).get(0).intValue());
    RemoveSingleFromList(MyMoveManager.removedCustomers.get(i).get(1).intValue());
}

//Greedy insertion
for(int i=0; i<MyMoveManager.removedCustomers.size(); i++){
    FindPositionsIncrementalAndInsertAnyK(MyMoveManager.removedCustomers.get(i).get(0).intValue(),
            MyMoveManager.removedCustomers.get(i).get(1).intValue());
    regretInsertation();
}

if(m==Main.k || m>2) regretMInsertion(m, penalty);
else if(m==2) regretTWOInsertion(penalty);
else if(m==1) regretONEInsertation(); //regretOne looks for regret between first and secondBest choice

// RegretInsertion only looks for simple distance between 2 points (not used anymore)
if(m==Main.k || m>2) regretMInsertion(m, penalty);
else if(m==2) regretTWOInsertion(penalty);
else if(m==1) regretONEInsertation(); //regretOne looks for regret between first and secondBest choice

//System.out.println("SwapPairGEO: move customer= " + customer + " and movement = " + movement + " to vehicle " +KtoGo + " with "+ wLists.get(KtoGo).size()+ " positions ");
Lists2DToWnew(wNew);
// ListsPlusReturnToWnew(wNew);
MySolution.w=MySolution.deepCopyOf(wNew);//Arrays.copyOf(wNew, wNew.length);
// try deepcopy // try lists to wnew(w)

public static void RemoveFromList(Solution soln, int customer, int movement, int m, boolean penalty){
wLists.clear();
int[][][] w = MySolution.w;
int[][][] wNew = new int[Main.V][Main.V][Main.V+3][Main.k];
rememberW = MySolution.deepCopyOf(w);

for(int i=0; i<MyMoveManager.removedCustomers.size(); i++){
    RemoveSingleFromList(MyMoveManager.removedCustomers.get(i).get(0).intValue(),
            MyMoveManager.removedCustomers.get(i).get(1).intValue());
}

// RegretInsertion only looks for simple distance between 2 points (not used anymore)
if(m==Main.k || m>2) regretMInsertion(m, penalty);
else if(m==2) regretTWOInsertion(penalty);
else if(m==1) regretONEInsertation(); //regretOne looks for regret between first and secondBest choice

private static void find2ndPair(int[][][] w, int kOld, int[] secondPair) {
//findFurthestPair(w, secondPair, kOld);
List<Integer> wListKCopy = new ArrayList<Integer>();
for (int i = 0; i < wLists.get(kOld).size(); i++) wListKCopy.add(wLists.get(kOld).get(i));
wListKCopy.add(0);

double currentHigh = Integer.MIN_VALUE;
double currentDist = 0;
if (wListKCopy.size() > 4) {
    for (int i = 1; i < wListKCopy.size(); i++) {
        for (int j = 0; j < wListKCopy.size() - 1; j++) {
            if (MyData.demand[wListKCopy.get(i)][wListKCopy.get(j)] > 0) {
                // if succeeding nods:
                if (j == i + 1) {
                    currentDist = 0.5 * MyData.distance[wListKCopy.get(i - 1)][wListKCopy.get(i)]
                        + MyData.distance[wListKCopy.get(i)][wListKCopy.get(j)]
                        + 0.5 * MyData.distance[wListKCopy.get(j)][wListKCopy.get(j + 1)];
                } else {
                    currentDist = 0.5 * MyData.distance[wListKCopy.get(i - 1)][wListKCopy.get(i)]
                        + 0.5 * MyData.distance[wListKCopy.get(i)][wListKCopy.get(i + 1)]
                        + 0.5 * MyData.distance[wListKCopy.get(j - 1)][wListKCopy.get(j)]
                        + 0.5 * MyData.distance[wListKCopy.get(j)][wListKCopy.get(j + 1)];
                }
            } else if (currentDist > currentHigh) {
                secondPair[0] = wListKCopy.get(i);
                secondPair[1] = wListKCopy.get(j);
            }
        }
    }
    else if (wListKCopy.size() > 4) {
        secondPair[0] = wListKCopy.get(1);
        secondPair[1] = wListKCopy.get(2);
    }
}

public static void operatorReverse(Solution solution) {
    MySolution.w = Arrays.copyOf(rememberW, rememberW.length);
}

public static int[] findEmptyPositions(int[][][] w) {
    //position={customer , customer, vehicle}
    for (int i = 0; i < Main.V; i++) {
        for (int j = 0; j < Main.V; j++) {
            if (w[i][j][position[0]][position[2]] == 1) {
                p3w[0] = i;
                p3w[1] = j;
            } else if (w[i][j][position[1]][position[2]] == 1) {
                p3w[2] = i;
                p3w[3] = j;
            }
        }
    }
    return p3w;
}

public static PositionValues findPositions(int[][][] w, int customer, int movement, int[] plw, int[] p2w)
```java
boolean cmpair = false;

boolean mcpair = false;

// Find positions
for (int i = 0; i < w.length; i++) {
    for (int j = 0; j < w.length; j++) {
        for (int q = 0; q < w.length; q++) {
            for (int k = 0; k < Main.k; k++) {
                if (w[i][j][q][k] == 1 && i == movement && j == customer) {
                    mcpair = true;
                    System.out.println("m" + mcpair + "w[" + i + "][" + j + "][" + q + "][" + k + "]=" + w[customer][movement][q][k]);
                }
                if (w[i][j][q][k] == 1 && i == customer && j == movement) {
                    System.out.println("c" + cmpair + "w[" + i + "][" + j + "][" + q + "][" + k + "]=" + w[i][j][q][k]);
                }
            }
        }
    }
}

//pos2 = pos1 + movement;
//int customer2 = pos1 + movement;  //Aenderung: movement ist jetzt customer 2

// find pos2
for (int i = 0; i < w.length; i++) {
    for (int j = 0; j < w.length; j++) {
        for (int q = 0; q < w.length; q++) {
            for (int k = 0; k < Main.k; k++) {
                if (w[i][customer][q][k] == 1) {
                    plw[0] = i;
                    plw[1] = customer;
                    plw[2] = q;
                    plw[3] = k;
                }
                if (w[customer][j][q][k] == 1) {
                    plw[4] = customer;
                    plw[5] = j;
                    plw[6] = q;
                    plw[7] = k;
                }
            }
        }
    }
}

PositionValues cmpair, mcpair, plw;

return new PositionValues(cmpair, mcpair, plw, p2w);
```

fBGPfindK(w, plw1, p2w1, selectionCriteria);  // determine new K
position[0]=0;
position[1]=0;

List<Integer> positionsInRoute = new ArrayList<Integer>();
int routeLength = 0;
// determine route length for position:
for(int q=1; q<w.length; q++){
  for(int i=0; i<w.length; i++){
    for(int j=0; j<Main.V; j++){
      if(w[i][j][q][position[2]]==1){
        positionsInRoute.add(q);
        routeLength++;
        //System.out.println("Position: "+positionsInRoute.get(q-1));
      }
    }
  }
}
if(positionsInRoute.size()>1 && routeLength>0){
  if(selectionCriteria == 4)  fBGPfindP(w, plw1, p2w1, selectionCriteria);
  else{
    // find position for customer1 by TW
    double lowestTWdiff = Integer.MAX_VALUE;
    double twDiff = 0;
    Q: for(int q=1; q<routeLength-1; q++;
    Q: for(int q=1; q<routeLength-1; q++){
      for(int i=0; i<Main.V; i++){
        for(int j=0; j<Main.V; j++){
          if(w[i][j][q][position[2]]==1){
            twDiff = MyData.TWend[j]-MyData.TWend[plw1];
            if(twDiff < lowestTWdiff && (MyData.TWend[j]-MyData.TWend[plw1])>=0){
              lowestTWdiff=MyData.TWend[j]-MyData.TWend[plw1];
              //System.out.println("lowestTWdiff plw1 at "+q+": "+lowestTWdiff);
              position[0]=q;
            }
            else if(twDiff > lowestTWdiff) break Q;
          }
        }
      }
    }
    //find position for customer2 by TW
    lowestTWdiff = Integer.MAX_VALUE;
    I: for(int i=1; i<Main.V; i++){
      for(int j=1; j<Main.V; j++){
        for(int q=position[0]+1; q<routeLength; q++;
        Q: for(int q=position[0]+1; q<routeLength; q++){
          if(w[i][j][q][position[2]]==1 && q != position[0]){
            twDiff = MyData.TWend[j]-MyData.TWend[p2w1];
            if(twDiff < lowestTWdiff && (MyData.TWend[j]-MyData.TWend[p2w1])>=0){
              lowestTWdiff=MyData.TWend[j]-MyData.TWend[p2w1];
              //System.out.println("lowestTWdiff p2w1 at "+q+": "+lowestTWdiff);
              position[1]=q;
            }
            else if(twDiff > lowestTWdiff) break I;
          }
        }
      }
    }
}
```java
static void findSinglePositionList(int customer, int movement, int selectionCriteria) {
    // determine sender/recipient:
    int senderPos = 0;
    int recipientPos = wLists.get(position[2]).size();
    position[0] = 0;

    P: for (int i = 0; i < wLists.get(position[2]).size(); i++) {
        if ((MyData.demand[wLists.get(position[2]).get(i)][customer] > 0) {
            senderPos = i;
            break P;
        } else if (MyData.demand[customer][wLists.get(position[2]).get(i)] > 0) {
            recipientPos = i;
        }
    }
    // System.out.println(); // by distance

    if (selectionCriteria == 2) {
        double posDist = Integer.MAX_VALUE;
        double currentDist;
        for (int i = senderPos; i < recipientPos - 1; i++) {
            currentDist = MyData.distance[wLists.get(position[2]).get(i)][customer]
                        - MyData.distance[customer][wLists.get(position[2]).get(i + 1)];
            System.out.println("current distance: "+ currentDist);
            if (currentDist < posDist) {
                posDist = currentDist;
                position[0] = i + 1;
                System.out.println("New min distance: "+ currentDist);
            }
        }
        // dist alternative -- incremental distance for inserting
        else if (selectionCriteria == 0) {
            for (int i = senderPos; i < recipientPos - 1; i++) {
                if (wLists.get(position[2]).get(i) == customer) {
                    currentPos = i;
                    System.out.println("Current position found");
                    break CP;
                }
            }
            double posDist = Integer.MAX_VALUE;
            double currentDist;
            for (int i = senderPos; i < recipientPos - 1; i++) {
                currentDist = MyData.distance[wLists.get(position[2]).get(i)][customer]
                        - MyData.distance[customer][wLists.get(position[2]).get(i + 1)];
                System.out.println("current distance: "+ currentDist);
                if (MyObjectiveFunction.customersToMove.size() > 5) previousPosition = 0;
                if (currentDist < posDist && i + 1 != currentPos && i + 1 != previousPosition) {
                    posDist = currentDist;
                    position[0] = i + 1;
                    System.out.println("New min distance: "+ currentDist);
                }
            }
        }
    }
}
```
else if(selectionCriteria == 1){
    double lowestTWdiff = Integer.MAX_VALUE;
    double currentTWdiff;
    double currentTWdiff2;
    for(int i=senderPos; i<recipientPos-1; i++){
        currentTWdiff = MyData.TWend[customer] - MyData.TWend[wLists.getPosition[2]].get(i);
        currentTWdiff2 = MyData.TWend[wLists.getPosition[2]].get(i+1) - MyData.TWend[customer];
        if(currentTWdiff>0 && currentTWdiff2 > 0 && (currentTWdiff2+currentTWdiff)<lowestTWdiff){
            lowestTWdiff = currentTWdiff2+currentTWdiff;
            position[0] = i+1;
        }
    }
}

else if(selectionCriteria == 3){
    // find current position to avoid this when selecting:
    double posDist = Integer.MAX_VALUE;
    double currentDist;
    for(int i=senderPos; i<recipientPos-1; i++){
        if(MyData.TWstart[customer] < MyData.TWend[wLists.getPosition[2]].get(i+1)){
            currentDist = (MyData.distance[wLists.getPosition[2]].get(i)][customer] + MyData.distance[customer][wLists.getPosition[2]].get(i+1))
            - MyData.distance[wLists.getPosition[2]].get(i)][wLists.getPosition[2]].get(i+1)];
            System.out.println("current distance: "+ currentDist);
            if(currentDist < posDist){
                posDist = currentDist;
                position[0] = i+1;
            }
        }
    }
}

else if(selectionCriteria == 4){
    position[0] = movement;
}

else if(selectionCriteria == 5){
    // find current position to avoid this when selecting:
    int currentPos = 0;
    CP: for(int i=senderPos; i<recipientPos-1; i++){
        if(wLists.getPosition[2].get(i) == customer){
            currentPos = i;
            break CP;
        }
    }
}
0928 // } 0929 // } 0930 double posDist = Integer.MAX_VALUE; 0931 double currentDist; 0932 for(int i=senderPos; i<recipientPos-1; i++){
0933     currentDist = (MyData.distance[wLists.get(position[2]).get(i)].get(i)[customer] + MyData.distance[customer][wLists.get(position[2]).get(i)][position[2]].get(i)])/MyData.distance[wLists.get(position[2]).get(i)][wLists.get(position[2]).get(i+1)];
0934     if(currentDist < posDist && /*NEW***/ MyData.TWstart[wLists.get(position[2]).get(i)] < MyData.TWend[customer])){
0935         posDist = currentDist;
0936         position[0] = i+1;
0937     }
0938     System.out.println("New min distance: "+currentDist);
0939 }
0940 }
0941 //System.out.println("Move c " +customer+ " to position "+position[0]);
0942 }
0943 } 0944 else if(selectionCriteria == 6){
0945     double lowestTWdiff = Integer.MAX_VALUE;
0946     double currentTWdiff;
0947     double currentTWdiff2;
0948     double currentTWdiff3;
0949     double currentTWdiff4;
0950     if(MyData.TWend[customer] - MyData.TWstart[customer] < 1000){
0951         for(int i=senderPos; i<recipientPos-1; i++){
0952             currentTWdiff = MyData.TWend[customer] - MyData.TWend[wLists.get(position[2]).get(i)];
0953             lowestTWdiff = MyData.TWend[wLists.get(position[2]).get(i+1)] - MyData.TWend[customer];
0954             currentTWdiff3 = MyData.TWstart[customer] - MyData.TWstart[wLists.get(position[2]).get(i)];
0955             currentTWdiff4 = MyData.TWstart[wLists.get(position[2]).get(i+1)] - MyData.TWstart[customer];
0956             if(currentTWdiff>0 && currentTWdiff2 >0 && (currentTWdiff2+currentTWdiff)>lowestTWdiff && (MyData.distance[customer][wLists.get(position[2]).get(i)] + 
0957                 MyData.distance[wLists.get(position[2]).get(i+1)][customer]) <70)){
0958                 lowestTWdiff = currentTWdiff2+currentTWdiff;
0959                 position[0] = i+1;
0960             }
0961             if(currentTWdiff3>0 && /*currentTWdiff4 >0 && /*/currentTWdiff4/*>/<currentTWdiff2 && 
0962                 MyData.distance[customer][wLists.get(position[2]).get(i)] + 
0963                 MyData.distance[wLists.get(position[2]).get(i+1)][customer]) <70)){
0964                 lowestTWdiff = currentTWdiff2+currentTWdiff4;
0965                 position[0] = i+1;
0966             }
0967             else{//sc5 (new)
0968                 int currentPos = 0;
0969                 CP: for(int i=senderPos; i<recipientPos-1; i++){
0970                     if(wLists.get(position[2]).get(i) == customer)
0971                         currentPos = i;
0972                         break CP;
0973                 }
0974                 double posDist = Integer.MAX_VALUE;
0975                 double currentDist;
0976                 for(int i=senderPos; i<recipientPos-1; i++){
0977                     currentDist = (MyData.distance[wLists.get(position[2]).get(i)].get(i)][customer] + MyData.distance[customer][wLists.get(position[2]).get(i)][position[2]].get(i+1)) - MyData.distance[wLists.get(position[2]).get(i)][wLists.get(position[2]).get(i+1)];
0978                     if(currentDist < posDist && i+1 != currentPos && /*NEW***/ MyData.TWstart[wLists.get(position[2]).get(i)] < MyData.TWend[customer]){
0979                         posDist = currentDist;
0980                         position[0] = i+1;
0981 }
else if (selectionCriteria == 7) {
if (MyData.TWend[customer] - MyData.TWstart[customer] < 1000) {
    double currentTime = 0;
    H: for (int i = 0; i < recipientPos - 1; i++) {
        double posDist = MyData.distance[wLists.get(position[2]).get(i)][customer] - MyData.distance[customer][wLists.get(position[2]).get(i + 1)];
            position[0] = i + 1;
            System.out.println("Arrival time at c " + wLists.get(position[2]).get(i + 1) + ": "+currentTime + "+MyData.TWend[customer]);
            currentTime += MyData.distance[wLists.get(position[2]).get(i)].get(i + 1) / Main.v;
            currentDist += MyData.Service[wLists.get(position[2]).get(i)];
        }
        else {
            System.err.println("Late Arrival time at c " + wLists.get(position[2]).get(i + 1) + ": " + currentTime);
            break H;
        }
    }
    System.out.println("Time: Customer " + customer + " has been moved to position[0] +" in k: " + position[2]);
} else { // sc5 (new)
    int currentPosition = 0;
    CP: for (int i = senderPos; i < recipientPos - 1; i++) {
        if (wLists.get(position[2]).get(i) == customer) {
            currentPosition = i;
            break CP;
        }
    }
    double posDist = Integer.MAX_VALUE;
    double currentDist;
    for (int i = senderPos; i < recipientPos - 1; i++) {
        currentDist = (MyData.distance[wLists.get(position[2]).get(i)][customer] + MyData.distance[customer][wLists.get(position[2]).get(i)].get(i + 1)) - MyData.distance[wLists.get(position[2]).get(i)][wLists.get(position[2]).get(i + 1)];
        if (currentDist < posDist && i + 1 != currentPosition && MyData.TWstart[wLists.get(position[2]).get(l)] < MyData.TWend[customer]) {
            posDist = currentDist;
        }
    }
    //System.out.println("Time: Customer " + customer + " has been moved to position[0] +" in k: " + position[2]);
}
position[0] = i+1;
}
}

// System.out.println("Move customer "+customer+" to position "+position[0]+" in k: "+position[2]);
if(position[0] != 0){
    wLists.get(position[2]).add(position[0], customer);
    wLists.get(position[2]).remove(wLists.get(position[2]).size()-1);
}
else{
    // System.out.println("ALARM");
    wLists.get(position[2]).add(recipientPos-1, customer);
    wLists.get(position[2]).remove(wLists.get(position[2]).size()-1);
    error = true;
}

private static void findPositionsArrayList(int c1, int c2, int k){
    double posDist = Integer.MAX_VALUE;
    position[0] = 0;
    position[1] = 0;
    //position[3] = k;
    // find position for customer1
    double currdist;
    for(int q=1; q<wLists.get(k).size()-1; q++){  // um alle positionen vor 0
        currdist = MyData.distance[wLists.get(k).get(q)][c1]+MyData.distance[c1][wLists.get(k).get(q+1)];
        if(currdist<posDist /*&& i != p1w1*/){
            posDist=currdist;
            position[0]=q;
        }
    }
    // System.out.println("Insert customer: "+c1+" at position "+position[0]+" in vehicle "+k);
    //find position for customer2
    posDist = Integer.MAX_VALUE;
    System.out.println("MyData.distance.length: "+MyData.distance.length);
    System.out.println("wLists.get k size:: "+ wLists.get(k).size());
    System.out.println("customer "+ c1);
    System.out.println("customer 2 "+ c2);
    for(int q=position[0];q<wLists.get(k).size()-1; q++){  // for(int q=1; q<wLists.get(k).size()-1; q++){
        currdist = MyData.distance[wLists.get(k).get(q)][c1]+MyData.distance[c1][wLists.get(k).get(q+1)];
        if(currdist<posDist /*&& i != p1w1*/){
            posDist=currdist;
            position[1]=q;
        }
    }
    if(position[0] == 0){
        position[0] = 1;
        position[1] = 2;
    }
    if(position[1] == 0 ^ position[1]<position[0]) position[1] = position[0]+1 ;
    // System.out.println("Insert customer: "+c2+" at position "+position[1]+" in vehicle "+k);
}

public static void findPositionsIncrementalAndInsert(int c1, int c2, int k){
    if(wLists.get(k).size()==1){
        // System.out.println("ALARM");
        wLists.get(position[2]).add(recipientPos-1, customer);
        wLists.get(position[2]).remove(wLists.get(position[2]).size()-1);
        error = true;
    }
    // System.out.println("Insert customer: "+c2+" at position "+position[1]+" in vehicle "+k);
}
wLists.get(k).add(c1);
wLists.get(k).add(c2);
}
else{
    //find minimum distance positions for c1 & c2:
    double posDist = Integer.MAX_VALUE;
    double currentDist;
    int posC1 = 0;
    int posC2 = 0;
    // Find position by incremental distance for c1:
    for(int i=0; i<wLists.get(k).size()-1; i++){
        currentDist = (MyData.distance[wLists.get(k).get(i)][c1] + MyData.distance[wLists.get(k).get(i)][wLists.get(k).get(i+1)]) - MyData.distance[wLists.get(k).get(i)][wLists.get(k).get(i+1)];
        if(currentDist < posDist){
            posDist = currentDist;
            posC1 = i+1;
            //System.out.println("New min distance c1 :"+posC1+" + currentDist");
        }
    }
    wLists.get(k).add(posC1, c1);
    // Find position by incremental distance for c2:
    posDist = Integer.MAX_VALUE;
    for(int i=posC1; i<wLists.get(k).size()-1; i++){
        currentDist = (MyData.distance[wLists.get(k).get(i)][c2] + MyData.distance[wLists.get(k).get(i)][wLists.get(k).get(i+1)]) - MyData.distance[wLists.get(k).get(i)][wLists.get(k).get(i+1)];
        if(currentDist < posDist){
            posDist = currentDist;
            posC2 = i+1;
            //System.out.println("New min distance c2 :"+posC2+" + currentDist");
        }
    }
    wLists.get(k).add(posC2, c2);
}
}
public static void findPositionsIncrementalAndInsertAnyK(int c1, int c2){
    // if(wLists.get(k).size()==1){
    //    wLists.get(k).add(c1);
    //    wLists.get(k).add(c2);
    // }
    // Find minimum distance positions for c1 & c2:
    double posDist1 = Integer.MAX_VALUE;
    double posDist2 = Integer.MAX_VALUE;
    double Kdist = Integer.MAX_VALUE;
    double currentDist;
    int posC1 = 0;
    int posC2 = 0;
    int KtoGo = 0;
    for(int k=0; k<wLists.size();k++){
        // Find position by incremental distance for c1:
        for(int i=0; i<wLists.get(k).size()-1; i++){
            currentDist = (MyData.distance[wLists.get(k).get(i)][c1] + MyData.distance[wLists.get(k).get(i)][wLists.get(k).get(i+1)]) - MyData.distance[wLists.get(k).get(i)][wLists.get(k).get(i+1)];
            if(currentDist < posDist1){
                posDist1 = currentDist;
                posC1 = i+1;
                //System.out.println("New min distance c1 :"+posC1+" + currentDist");
            }
        }
        for(int ii=posC1; ii<wLists.get(k).size()-1; ii++){
currentDist = (MyData.distance[wLists.get(k).get(ii)][c2] + MyData.distance[wLists.get(k).get(ii+1)]) - MyData.distance[wLists.get(k).get(ii)][wLists.get(k).get(ii+1)];

//System.out.println("current distance: "+ currentDist);
if(currentDist < posDist2){
    posDist2 = currentDist;
    posC2 = ii+2/*+1*/; // one position extra as Cl is inserted before!
    //System.out.println("New min distance c2 :"+posC2+" "+ currentDist);
}
if(posDist1+posDist2 < Kdist){
    Kdist = posDist1+posDist2;
    KtoGo = k;
    posC1 = i+1;
    posC2 = ii+2;
    System.out.println("new k determined: "+KtoGo+" distance: "+ Kdist);
}
if(posDist1+posDist2 < Kdist){
    Kdist = posDist1+posDist2;
    KtoGo = k;
    posC1 = i+1;
    posC2 = ii+2;
    System.out.println("new k determined: "+KtoGo+" distance: "+ Kdist);
}
}
}

public static void regretInsertation/*greedy dist*/(){
    double lowest;
    double secondlowest;
    double posDist1;
    double posDist2;
    while(MyMoveManager.removedCustomers.size()>0){
        for(int c=0; c<MyMoveManager.removedCustomers.size(); c++){
            lowest = Integer.MAX_VALUE-1;
            secondlowest = Integer.MAX_VALUE;
            posDist1 = Integer.MAX_VALUE;
            posDist2 = Integer.MAX_VALUE;
            for(int k=0; k<wLists.size();k++){
                // Find position for c1:
                for(int i=0; i<wLists.get(k).size()-1; i++){
                    posDist1 = (MyData.distance[wLists.get(k).get(i)][MyMoveManager.removedCustomers.get(c).get(0)].intValue()) + MyData.distance[MyMoveManager.removedCustomers.get(c).get(0)].intValue()[wLists.get(k).get(i+1)]) - MyData.distance[wLists.get(k).get(i)][wLists.get(k).get(i+1)];
                }
                // Find position for c2:
                for(int ii=0; ii<wLists.get(k).size()-1; ii++){
                    posDist2 = (MyData.distance[wLists.get(k).get(ii)][MyMoveManager.removedCustomers.get(c).get(1)].intValue()) + MyData.distance[MyMoveManager.removedCustomers.get(c).get(1)].intValue()[wLists.get(k).get(ii+1)]) - MyData.distance[wLists.get(k).get(ii)][wLists.get(k).get(ii+1)];
                }
                if(posDist1+posDist2 < lowest){
                    secondlowest = lowest;
                    lowest = posDist1+posDist2;
                }
                MyMoveManager.removedCustomers.get(c).set(3, (double) (i+1));
                MyMoveManager.removedCustomers.get(c).set(4, (double) (ii+2));
                MyMoveManager.removedCustomers.get(c).set(5, (double) (k));
            }
        }
    }
}
311
    //System.out.println("New lowest determined: "+lowest);
312    }
313    else if(posDist1+posDist2 < secondlowest){
314      secondlowest = posDist1+posDist2;
315      //System.out.println("New second lowest determined: "+secondlowest);
316    }
317    }
318  }
319}
320  Collections.sort(MyMoveManager.removedCustomers, new Comparator<ArrayList<Double>>(){
321    private static final int INDEX = 2;
322    public int compare(ArrayList<Double> o1, ArrayList<Double> o2) {
323      return Double.compare(o1.get(INDEX), o2.get(INDEX));
324    }
325  });
326  //System.out.println("removedCustomers with c* in right order:");
327  //for (int is = 0; is<MyMoveManager.removedCustomers.size(); is++) {
328    System.out.println(MyMoveManager.removedCustomers.get(is).toString());
329  //}
330  //insert biggest regret
331  insertInLists(MyMoveManager.removedCustomers.get(MyMoveManager.removedCustomers.size()-
332    1).get(0).intValue(), MyMoveManager.removedCustomers.get(MyMoveManager.removedCustomers.size()-
333    1).get(3).intValue(), MyMoveManager.removedCustomers.get(MyMoveManager.removedCustomers.size()-1).get(5).intValue());
334  insertInLists(MyMoveManager.removedCustomers.get(MyMoveManager.removedCustomers.size()-
335    1).get(1).intValue(), MyMoveManager.removedCustomers.get(MyMoveManager.removedCustomers.size()-
336    1).get(4).intValue(), MyMoveManager.removedCustomers.get(MyMoveManager.removedCustomers.size()-1).get(5).intValue());
337  MyMoveManager.removedCustomers.remove(MyMoveManager.removedCustomers.size()
338    -1);
339  //System.out.println("Highest regret customers inserted and list trimmed");
340  }
341  }
342  public static void regretONEInsertation(){
343    double lowest;
344    // double secondlowest;
345    double deltaC = 0;
346    double currentCosts;
347    double newCosts;
348    while(MyMoveManager.removedCustomers.size()>0){
349      System.out.println("in regretOne loop");
350      for(int c=0; c<MyMoveManager.removedCustomers.size(); c++){
351        lowest = Integer.MAX_VALUE-1;
352        secondlowest = Integer.MAX_VALUE;
353        forDto2DListePlusReturn(MySolution.w);
354        currentCosts = MyObjectiveFunction.evaluateQuick(false);
355        for(int k=0; k<wLists.size();k++){
356          // Find position for cl:
357          for(int i=1; i<wLists.get(k).size()/2; i++){
358            // insert cl in wLists
359            insertInLists(MyMoveManager.removedCustomers.get(c).get(0).intValue

for(int ii=i+1; ii<wLists.get(k).size(); ii++) {
    // insert c2 in wLists
    insertInLists(MyMoveManager.removedCustomers.get(c).get(i).intValue(), ii, k);
}

// calculate wLists costs delta ij
newCosts = MyObjectiveFunction.evaluateQuick(false);
deltaC = newCosts - currentCosts;

if(deltaC < lowest) {
    lowest = deltaC;
    MyMoveManager.removedCustomers.get(c).set(2, secondlowest - lowest);
    MyMoveManager.removedCustomers.get(c).set(3, (double) i);
    MyMoveManager.removedCustomers.get(c).set(4, (double) ii);
    MyMoveManager.removedCustomers.get(c).set(5, (double) k);
}

// remove c2 from wLists
wLists.get(k).remove(ii);

// remove c1 from wLists
wLists.get(k).remove(i);

// assign c* value to customers
MyMoveManager.removedCustomers.get(c).set(2, lowest);
}

// sort removed customer list
Collections.sort(MyMoveManager.removedCustomers, new Comparator<ArrayList<Double>>() {
    private static final INDEX = 2;
    public int compare(ArrayList<Double> o1, ArrayList<Double> o2) {
        return Double.compare(o1.get(INDEX), o2.get(INDEX)) * (-1);
    }
    System.out.println("removedCustomers with c* in right order:");
    for (int is = 0; is<MyMoveManager.removedCustomers.size(); is++) {
        System.out.println(MyMoveManager.removedCustomers.get(is).toString());
    }
    }
    public static void regretInsertationCEval(){
        double lowest;
        double secondlowest;
        double deltaC = 0;
        double currentCosts;

```java
double newCosts;
while (MyMoveManager.removedCustomers.size() > 0) {
    for (int c = 0; c < MyMoveManager.removedCustomers.size(); c++) {
        lowest = Integer.MAX_VALUE - 1;
        secondlowest = Integer.MAX_VALUE;
        // fourDto2DListePlusReturn(MySolution.w);
        currentCosts = MyObjectiveFunction.evaluateQuick(false);
        for (int k = 0; k < wLists.size(); k++) {
            // Find position for c1:
            for (int i = 1; i < wLists.get(k).size() / 2; i++) {
                // insert c1 in wList
                insertInLists(MyMoveManager.removedCustomers.get(c).get(0).intValue, i, k);
                // Find position for c2:
                for (int ii = i + 1; ii < wLists.get(k).size(); ii++) {
                    // insert c2 in wList
                    insertInLists(MyMoveManager.removedCustomers.get(c).get(1).intValue, ii, k);
                    // calculate wList costs delta ij
                    newCosts = MyObjectiveFunction.evaluateQuick(false);
                    deltaC = newCosts - currentCosts;
                    if (deltaC < lowest) {
                        secondlowest = lowest;
                        lowest = deltaC;
                        MyMoveManager.removedCustomers.get(c).set(2, secondlowest - lowest);
                    } else if (deltaC < secondlowest) {
                        secondlowest = deltaC;
                        // if(c==0)System.out.println("New lowest determined: "+lowest + " + secondlowest = "+(secondlowest - lowest));
                        if (c == 0) System.out.println("New lowest determined: "+lowest + " + secondlowest = "+(secondlowest - lowest));
                    }
                    // remove c2 from wLists
                    wLists.get(k).remove(ii);
                    } else {
                    // remove c1 from wLists
                    wLists.get(k).remove(i);
                    } // assign c* value to customers
                MyMoveManager.removedCustomers.get(c).set(2, secondlowest - lowest);
            }
        }
        // sort removed customer list
        Collections.sort(MyMoveManager.removedCustomers, new Comparator<ArrayList<Double>>() {
            private static final int INDEX = 2;
            public int compare(ArrayList<Double> o1, ArrayList<Double> o2) {
                return Double.compare(o1.get(INDEX), o2.get(INDEX));
            }
        });
        System.out.println("removedCustomers with c* in right order:");
        for (int is = 0; is < MyMoveManager.removedCustomers.size(); is++) {
            System.out.println(MyMoveManager.removedCustomers.get(is).toString());
        }
        // fourDto2DListe(MySolution.w); // recreate wLists WITHOUT return
    }
}
```
public static void regretTWOInsertion(boolean penalty) {
    double lowest;  
    double secondlowest;  
    int kLowest = Main.k - 1;  
    double deltaC = 0;  
    double currentCosts;  
    double newCosts;
    while (MyMoveManager.removedCustomers.size() > 0) {
        System.out.println("in regretTwo loop");
        for (int c = 0; c < MyMoveManager.removedCustomers.size(); c++) {
            currentCosts = MyObjectiveFunction.evaluateSingleRouteQuick(c, penalty);
            System.out.println(wLists.get(c).toString());
            for (int i = 1; i < wLists.get(c).size() - 1; i++) {
                // insert c1 in wLists
                insertInLists(MyMoveManager.removedCustomers.get(c).get(0).intValue(), i, k);
                // Find position for c2:
                for (int ii = i + 1; ii < wLists.get(c).size() - 1; ii++) {
                    // insert c2 in wLists
                    insertInLists(MyMoveManager.removedCustomers.get(c).get(1).intValue(), ii, k);
                    // System.out.println(wLists.get(k).toString());
                    // calculate wLists costs delta ij
                    newCosts = MyObjectiveFunction.evaluateSingleRouteQuick(k, penalty);
                    deltaC = newCosts - currentCosts;
                    if (deltaC < lowest) {
                        if (kLowest != k) secondlowest = lowest;
                        kLowest = k;
                        secondlowest = deltaC;
                    }
                }
            }
        }
    }
}

lowest = deltaC;
    MyMoveManager.removedCustomers.get(c).set(2, secondlowest - lowest);
    if(c==0)System.out.println("New lowest determined: "+lowest + "+ secondlowest = "+(secondlowest - lowest));
    else if(deltaC < secondlowest && kLowest != k){
        secondlowest = deltaC;
        if(c==0)System.out.println("New lowest determined: "+lowest + "+ secondlowest = "+(secondlowest - lowest));
    } else {
        MyMoveManager.removedCustomers.get(c).set(3, 1.0);
        MyMoveManager.removedCustomers.get(c).set(4, 2.0);
        MyMoveManager.removedCustomers.get(c).set(5, (double) k);
    }
    //remove c2 from wlists
    wLists.get(k).remove(ii);
    //remove c1 from wlists
    wLists.get(k).remove(i);
    } else {
    MyMoveManager.removedCustomers.get(c).set(2, secondlowest - lowest);
}
Collections.sort(MyMoveManager.removedCustomers, new Comparator<ArrayList<Double>>()
{
    private static final int INDEX = 2;
    public int compare(ArrayList<Double> o1, ArrayList<Double> o2) {
        return Double.compare(o1.get(INDEX), o2.get(INDEX));
    }
});
System.out.println("removedCustomers with c* in right order (regTwo):") ;
for (int is = 0; is<MyMoveManager.removedCustomers.size(); is++) {
    System.out.println(MyMoveManager.removedCustomers.get(is).toString());
}
public static void regretMInsertion(int m, boolean penalty) {
    double lowestTotal;
    double lowestk;
    double cStar;
    List<Double> lowestList = new ArrayList<Double>();
    double deltaC = Integer.MAX_VALUE;
    double currentCosts;
    double newCosts;
    //
    int p;
    //
    double y;
    //
    double yExpP;
    while (MyMoveManager.removedCustomers.size() > 0) {
        System.out.println("in regretM loop");
        for (int c=0; c<MyMoveManager.removedCustomers.size(); c++) {
            lowestTotal = Integer.MAX_VALUE;
            lowestk = Integer.MAX_VALUE;
            lowestList.clear();
            //initialize lowestCK array:
            for (int kk=0; kk<Main.k; kk++) {
                lowestList.add(0.0);
            }
            for (int k=0; k<wLists.size(); k++) {
                currentCosts = MyObjectiveFunction.evaluateSingleRouteQuick(k, penalty);
                //This part for empty routes:
                if (wLists.get(k).size() <= 1) {
                    insertInLists(MyMoveManager.removedCustomers.get(c).get(0).intValue (), 1, k);
                    insertInLists(MyMoveManager.removedCustomers.get(c).get(1).intValue (), 2, k);
                    wLists.get(k).add(0);
                    for (int i=0; i<wLists.get(k).size(); i++) {
                        System.out.print(wLists.get(k).get(i)+" ");
                    }
                    System.out.println();
                }
                //calculate wLists costs delta ij
                newCosts = MyObjectiveFunction.evaluateSingleRouteQuick(k, penalty);
                //
                deltaC = newCosts - currentCosts;
                if (deltaC < lowestTotal) {
                    lowestTotal = deltaC;
                    lowestList.set(k, deltaC);
                    lowestk = k;
                    MyMoveManager.removedCustomers.get(c).set(3, (double) (1));
                    MyMoveManager.removedCustomers.get(c).set(4, (double) (2));
                    MyMoveManager.removedCustomers.get(c).set(5, (double) (k));
                    //System.err.println("i: "+i+" j: "+i+" k: "+k+" cc: "+currentCosts);
                    } else if ((lowestList.get(k) == 0 || deltaC < lowestList.get(k)) && k != lowestk) {
                        lowestList.set(k, deltaC);
                        //if(c==0)System.out.println("New lowest determined: "+deltaC);
                        //System.out.println("New lowest determined: "+deltaC+" i: "+i+" k: "+k+" cc: "+currentCosts);
                    } else if ((lowestList.get(k) == 0 || deltaC < lowestList.get(k)) && k != lowestk) {
                        lowestList.set(k, deltaC);
                        //if(c==0)System.out.println("New lowest determined: "+deltaC);
                        //System.out.println("New lowest determined: "+deltaC+" i: "+i+" k: "+k+" cc: "+currentCosts);
                    }
                } else if (lowestList.get(k) == 0 || deltaC < lowestList.get(k)) && k != lowestk) {
                    lowestList.set(k, deltaC);
                    //if(c==0)System.out.println("New lowest determined: "+deltaC);
                    //System.out.println("New lowest determined: "+deltaC+" i: "+i+" k: "+k+" cc: "+currentCosts);
                }
            }
        }
    }
}
else {
    MyMoveManager.removedCustomers.get(c).set(3, 1.0);
    MyMoveManager.removedCustomers.get(c).set(4, 2.0);
    MyMoveManager.removedCustomers.get(c).set(5, (double) k));
}
//remove c2 from wlists
wLists.get(k).remove(ii);
wLists.get(k).remove(iii);
wLists.get(k).remove(iii);
} // end empty routes

else {
    // Find position for c1:
    for(int i=1; i<wLists.get(k).size(); i++) {
        // insert c1 in wLists
        insertInLists(MyMoveManager.removedCustomers.get(c).get(0).intValue(), i, k);
    }
    // Find position for c2:
    for(int ii=i+1; ii<wLists.get(k).size(); ii++) {
        // insert c2 in wLists
        insertInLists(MyMoveManager.removedCustomers.get(c).get(1).intValue(), ii, k);
    }
    //calculate wLists costs delta ij
    newCosts = MyObjectiveFunction.evaluateSingleRouteQuick(k, penalty);
    deltaC = newCosts - currentCosts;
    if(deltaC < lowestTotal) {
        lowestTotal = deltaC;
        lowestk = k;
        MyMoveManager.removedCustomers.get(c).set(3, (double) i));
        MyMoveManager.removedCustomers.get(c).set(4, (double) ii));
        MyMoveManager.removedCustomers.get(c).set(5, (double) k));
        //System.err.println("i: " + i + " j: " +ii + " k: " +k);
        //System.out.println("New lowest determined: "+deltaC+ " i: "+
i" k: "+k +" cc: " + currentCosts);
    } else if((lowestList.get(k) == 0 || deltaC < lowestList.get(k)) &&
    k!=lowestk) {
        lowestList.set(k, deltaC);
        //System.out.println("New 2lowest determined: "+deltaC);
        //System.out.println("New kowest determined: "+deltaC+ " i: "+
i" k: "+k +" cc: " + currentCosts);
    }
    //remove c2 from wlists
    wLists.get(k).remove(ii);
    } //remove c1 from wlists
    wLists.get(k).remove(iii);
    }
// sort lowest List by values:
Collections.sort(lowestList);
//System.out.println("list of lowest k values for regret calculation:");
for(int i=0; i<lowestList.size(); i++) System.out.print(lowestList.get(i)+" ");
System.out.println();

// calculate regret value c*
cStar=0;
for(int k=1; k<m; k++){
cStar+= lowestList.get(k) - lowestList.get(0);
}

//randomizer:
p = randomGenerator.nextInt(3)+1;
y = randomGenerator.nextDouble();
yExpP = (int) Math.pow(y, p);
cStar = cStar * (1-yExpP);

// assign c* value to customer
MyMoveManager.removedCustomers.get(c).set(2, cStar);
}

// sort removed customer list
Collections.sort(MyMoveManager.removedCustomers, new Comparator<ArrayList<Double>>()
{
    private static final int INDEX = 2;
    public int compare(ArrayList<Double> o1, ArrayList<Double> o2) {
        return Double.compare(o1.get(INDEX), o2.get(INDEX));
    }
});

System.out.println("removedCustomers with c* in right order (regM):");
for (int is = 0; is<MyMoveManager.removedCustomers.size(); is++) {
    System.out.println(MyMoveManager.removedCustomers.get(is).toString());
}

// fourDto2DListe(MySolution.w); // recreate wLists WITHOUT return

// insert biggest regret
System.out.println("insert "+MyMoveManager.removedCustomers.get(MyMoveManager.removedCustomers.size()-1).intValue()+" in k "+MyMoveManager.removedCustomers.get(MyMoveManager.removedCustomers.size()-1).intValue()+" on pos "+MyMoveManager.removedCustomers.get(MyMoveManager.removedCustomers.size()-1).intValue());
insertInLists(MyMoveManager.removedCustomers.get(MyMoveManager.removedCustomers.size()-1).intValue(), MyMoveManager.removedCustomers.get(MyMoveManager.removedCustomers.size()-1).intValue(), MyMoveManager.removedCustomers.get(MyMoveManager.removedCustomers.size()-1).intValue());
if(wLists.get(MyMoveManager.removedCustomers.get(MyMoveManager.removedCustomers.size()-1).intValue()).size()==3)wLists.get(MyMoveManager.removedCustomers.get(MyMoveManager.removedCustomers.size()-1).intValue()).add(0);
MyMoveManager.removedCustomers.remove(MyMoveManager.removedCustomers.size()-1);
}

public static void FBGPfindK(int[][][][][]w, int plwl, int p2wl, int selectionCriteria){
    position[3] = 0; //currentK
    for(int j=0; j<w.length; j++){
        for(int q=1; q<=w.length; q++){
```java
for (int k = 0; k < Main.k; k++){
    if (w[p1w1][i][q][k] == 1) position[3] = k;  // current k
}
}
// finde route mit der kuerzesten durchschnittlichen entfernung zum naechsten customer:
if (selectionCriteria == 0) {  // find greedy time route:
double currentLowest = Integer.MAX_VALUE;
for (int k = 0; k < Main.k; k++){
    if (k != position[3]){
        double lowestAvDist = 0;
        double customersInRoute = 0;
        for (int q = 1; q <= w.length; q++){
            for (int i = 1; i < w.length; i++){
                for (int j = 1; j < Main.V; j++){
                    if (w[i][j][q][k] == 1) {
                        customersInRoute++;
                        lowestAvDist += MyData.distance[p1w1][j];
                        lowestAvDist += MyData.distance[p2w1][j];
                    }
                }
            }
        }
        if ((lowestAvDist / customersInRoute) < currentLowest) {
            currentLowest = lowestAvDist / customersInRoute;
            position[2] = k;
        }
    }
}
// find greedy time route:
else if (selectionCriteria == 1) {
    double lowestTWdiff = Integer.MAX_VALUE;
    for (int k = 0; k < Main.k; k++){
        if (k != position[3]){
            for (int q = 1; q <= w.length; q++){
                for (int i = 0; i < w.length; i++){
                    for (int j = 0; j < Main.V; j++){
                        if ((MyData.TWend[j] - MyData.TWend[p1w1]) < lowestTWdiff &
                                                 (MyData.TWend[j] - MyData.TWend[p1w1]) >= 0 && j != plw1) {
                            lowestTWdiff = MyData.TWend[j] - MyData.TWend[p1w1];
                        }
                    }
                }
            }
        }
    } // System.out.println("lowestTWdiff: "+lowestTWdiff);
    position[2] = k;
}
else if (selectionCriteria == 2) {
    boolean found = false;
    for (int k = 0; k < Main.k; k++){
        if (k != position[3]){
            if (w[0][0][i][k] == 1) {
                position[2] = k;
                found = true;
            }
        }
    }
    if (found == false) selectionCriteria = 1;
    // System.out.println("Empty pos found and filled at "+position[2]);
}
else if (selectionCriteria == 3 || selectionCriteria == 4) {
    // put pair into t
```
he shortest route:
1710  int shortestRouteLength=Integer.MAX_VALUE;
1711  for(int k=0; k<Main.K; k++){
1712      int length=0;
1713      for(int q=1; q<Main.W; q++){
1714          for(int i=0; i<Main.V; i++){
1715              if(w[i][j][q][k]==1) length++;
1716          }
1717      }
1718
1719      if(length<shortestRouteLength && k!=position[3]) position[2] = k;
1720  }
1721  }
1722  }
1723  }
1724  public static void fBGFindP(int[][][][]w, int p1w1, int p2w1, int selectionCriteria){
1725      List<Integer> positionsInRoute = new ArrayList<Integer>();
1726      positionsInRoute.clear();
1727      //find random position:
1728      if(selectionCriteria==4){
1729          for(int q=1; q<Main.W; q++){
1730              for(int i=0; i<Main.V; i++){
1731                  for(int j=0; j<Main.V; j++){
1732                      if(w[i][j][q][position[2]]==1){
1733                          positionsInRoute.add(q);
1734                          //System.out.print(q +", ");
1735                      }
1736                  }
1737              }
1738          }
1739          // System.out.println();
1740          if(positionsInRoute.size()>2){
1741              position[0] = positionsInRoute.get(randomGenerator.nextInt(positionsInRoute.size()-1));
1742              position[1] = position[0]+1;
1743          } else position[0] = 0;
1744          System.out.println("p1w1: "+p1w1+p2w1: "+p2w1+" position[0] =" +position[0]+" position[1] = " +position[1]);
1745      }
1746      }
1747      else{
1748          // determine route length for position:
1749          for(int q=1; q<Main.W; q++){
1750              for(int i=0; i<Main.V; i++){
1751                  for(int j=0; j<Main.V; j++){
1752                      if(w[i][j][q][position[2]]==1){
1753                          positionsInRoute.add(q);
1754                          //System.out.println("Position: "+positionsInRoute.get(q-1));
1755                      }
1756                  }
1757          }
1758          double posDist = 1000000;
1759          // find position for customer1
1760          for(int q=1; q<positionsInRoute.size()-1; q++){
1761              for(int i=0; i<Main.V; i++){
1762                  for(int j=0; j<Main.V; j++){
1763                      if(w[i][j][q][position[2]]==1){
1764                          if((MyData.distance[i][p1w1]+MyData.distance[p1w1][j])<posDist && i != p1w1){
1765                              posDist=MyData.distance[i][p1w1]+MyData.distance[p1w1][j];
1766                      }
1767                  }
1768          }
1769      }
1770      }
1771  }
//find position for customer2
posDist = 1000000;
for(int i=1; i<Main.V; i++){
    for(int j=1; j<Main.V; j++){
        int q=position[0]+1;<positionsInRoute.size(); q++;
        if(w[i][j][q][position[2]]==1 && q != position[0]){
            if((MyData.distance[i][p2w1]+MyData.distance[p2w1][j])<posDist)
                position[1]=q;
            posDist=MyData.distance[i][p2w1]+MyData.distance[p2w1][j];
        }
    }
}
else{
    position[0]=1;
    position[1]=2;
}
if(position[1]<=position[0]) position[1]=position[0]+1;
}

public static int[] findGreedyInRoute(int[][][] w, int plw1/**, int[] position**/), int selectionCriteria, int movement){
    int currentK = 0;
    int currentQ = 1;
    for(int i=0; i<w.length; i++){
        for(int q=1; q<w.length; q++){
            for(int k=0; k<Main.k; k++){
                if(w[i][plw1][q][k]==1){
                    currentK=k;
                    currentQ=q;
                    // Find and set current vehicle
                }
            }
        }
    }
    List<Integer> positionsInRoute = new ArrayList<Integer>();
    // determine route length for position:
    MyreipientPos = 0;
    for(int q=1; q<w.length; q++){
        for(int i=0; i<w.length; i++){
            for(int j=0; j<Main.V; j++){
                if(w[i][j][q][currentK]==1){
                    MyreipientPos++;
                    positionsInRoute.add(q);
                }
            }
        }
        //System.out.println("Position: "+positionsInRoute.get(q-1));
    }
    // determine customers where shippings come from and go to:
    int receivingFrom=0;
    int sendingTo =0;
    for(int i=0; i<Main.V; i++){
        if(MyData.demand[plw1][i]>0)sendingTo=i;
if (MyData.demand[i][plw1] > 0) receivingFrom = i;

int MysenderPos = 0;
for (int i = 0; i < Main.V; i++) { // find S & R positions:
  for (int j = 0; j < Main.V; j++) {
    for (Integer element : positionsInRoute) {
      if (sendingTo != 0 && w[i][sendingTo][element][currentK] == 1) MyrecipientPos = element;
      if (receivingFrom != 0 && w[i][receivingFrom][element][currentK] == 1) MysenderPos = element;
    }
  }
}

// determine positions of senders and recipients:
int MysenderPos = 0;
for (int i = 0; i < Main.V; i++) {
  for (int j = 0; j < Main.V; j++) {
    for (Integer element : positionsInRoute) {
      if (sendingTo != 0 && w[i][sendingTo][element][currentK] == 1) MyrecipientPos = element;
      if (receivingFrom != 0 && w[i][receivingFrom][element][currentK] == 1) MysenderPos = element;
    }
  }
}

// create list with positions allowed in route:
List<Integer> positionsAllowed = new ArrayList<Integer>();
for (Integer q : positionsInRoute) {
  if (q > MysenderPos && q < MyrecipientPos && q != currentQ) positionsAllowed.add(q);
}

position[0] = currentQ;
int position[2] = currentK;
if (MysenderPos + 1 >= MyrecipientPos) {
  position[0] = currentQ;
  for (int j = 1; j < Main.V; j++) {
    if (w[plw1][j][currentQ + 1][currentK] == 1) position[1] = j;
  }
}

// Falls Fehler auftreten:
int position[2] = currentK;
if (MyData.demand[i][plw1] > 0) receivingFrom = i;
else if (selectionCriteria == 0) fGIRsc0/*@*/(currentK, currentQ, plw1, MysenderPos, MyrecipientPos, positionsInRoute, w, positionsAllowed);
else if (selectionCriteria == 1) fGIRsc1(currentK, currentQ, plw1, MysenderPos, MyrecipientPos, positionsInRoute, w, positionsAllowed);
else if (selectionCriteria == 2) fGIRsc2(currentK, currentQ, plw1, MysenderPos, MyrecipientPos, positionsInRoute, w, movement, positionsAllowed);
if (position[0] == currentQ) position[0] = 0;
return position;

static void RemoveSingleFromList/*replaces findSinglePosition*/(int customer) {
{
  for (int k = 0; k < wLists.size(); k++) {
    for (int i = 0; i < wLists.get(k).size(); i++) {
      if (wLists.get(k).get(i) == customer) {
        previousPosition = i;
        wLists.get(k).remove(i);
        position[2] = k;
        break K;
      }
    }
  }
}

static void RemovePairFromList(int customer, int movement) {
{
  for (int k = 0; k < wLists.size(); k++) {
    for (int i = wLists.get(k).size() - 2; i > 0; i--) {
      if (wLists.get(k).get(i) == movement) {
        wLists.get(k).remove(i);
        if (wLists.get(k).get(i) == customer) {
          wLists.get(k).remove(i);
          break K;
        }
      }
    }
  }
}

322
static int[] removePairFromListnReturnPos(int customer, int movement, int k) {
    // for(int l=0; l<wLists.get(k).size(); l++) {
    //    System.out.print(wLists.get(k).get(l) + " ");
    // }
    // System.out.println();
    int[] positions = new int[2];
    positions[1] = wLists.get(k).size()-1;
    I: for(int i=wLists.get(k).size()-1; i>0; i--){
        if(wLists.get(k).get(i) == movement){
            positions[1] = i;
            // System.out.println("remove movement: "+wLists.get(k).get(i)+" from position "+i);
            wLists.get(k).remove(i);
        } else if(wLists.get(k).get(i) == customer) {
            positions[0] = i;
            // System.out.println("remove customer: "+wLists.get(k).get(i)+" from position "+i);
            wLists.get(k).remove(i);
            break I;
        }
    }
    if(positions[0]==0) System.out.println("positions[0] == 0");
    if(positions[1]==0) System.out.println("positions[1] == 0");
    return positions;
}

public static int[] findSinglePosition(int[][][] w, int customer, int[] plw ){
    for(int i = 0; i < w.length; i++) {
        for(int j=0; j<w.length; j++){
            for(int q=0; q<w.length; q++){
                for(int k=0; k<Main.k; k++){
                    if( w[i][customer][q][k] == 1 ) {
                        plw[0] = i;
                        plw[1] = customer;
                        plw[2] = q;
                        plw[3] = k;
                    }
                }
            }
        }
    }
    return plw;
}

/** finds position optimised distance wise **/
public static void fGIRsc0(int currentK, int currentQ, int plw1, int MysenderPos, int MyrecipientPos, List<Integer> positionsInRoute, int[][][][] w, List<Integer> positionsAllowed){
    double posDist = Integer.MAX_VALUE;
    // System.out.println("MysenderPos= "+MysenderPos+" "+ "CurrentPos= "+ currentQ+" "+ MyrecipientPos);
    for(Integer q : positionsAllowed){
        // System.out.println("positionsAllowed = "+q);
        for(Integer q : positionsAllowed){
            for(int i=0; i<Main.V; i++){
                //...
for(int j=0; j<Main.V; j++){
    // In diesem Fall wird die Distanz zum folgenden Kunden minimiert:
    if(w[i][j][q][currentK]==1 & & q != currentQ){
        if((MyData.TWend[p1w1]+MyData.distance[p1w1][j])<posDist & & i
            != plw1){
            posDist=(MyData.distance[i][plw1]+MyData.distance[plw1][j]);
            position[0]=q;
        }
    }
}
}

// Determine plw1's Future Follower after finding the best position:
for(int i=0; i<Main.V; i++){
    for(int j=0; j<Main.V; j++){
        if(w[i][j][position[0]+1][currentK]==1)position[1]=j;
    }
}

/** finds random position**/
public static void fGLRsc2(int currentK, int currentQ, int plw1, int Mysender Pos, int MyrecipientPos, List<Integer> positionsInRoute, int[][][][] w, List<Integer> positionsAllowed){
    double lowestTWdiff = 10000000;
    for(int q : positionsAllowed){
        for(int i=0; i<Main.V; i++){
            for(int j=0; j<Main.V; j++){
                // In diesem Fall wird die Distanz zum folgenden Kunden minimiert:
                if(w[i][j][position[0]+1][currentK]==1 & & q != currentQ){
                    lowestTWdiff=lowestTWdiff & MyData.TWend[j]-
                    MyData.TWend[plw1])<lowestTWdiff & (MyData.TWend[j]-
                    MyData.TWend[plw1])>0 & & j != plw1{
                        lowestTWdiff=lowestTWdiff & MyData.TWend[j]-
                        MyData.TWend[plw1];
                        // System.out.println("lowestTWdiff: "+lowestTWdiff);
                        position[0]=q;
                    }
                }
            }
        }
    }

    // Determine plw1's Future Follower after finding the best position:
    for(int i=0; i<Main.V; i++){
        for(int j=0; j<Main.V; j++){
            if(w[i][j][position[0]+1][currentK]==1)position[1]=j;
        }
    }
}

/** finds random position**/
public static void fGIRsc3(int currentK, int currentQ, int plwl, int MysenderPos, int MyrecieipientPos, List<Integer> positionsInRoute, int[][][][] w, List<Integer> positionsAllowed) {
    double posDist = Integer.MAX_VALUE;
    double currentDist;
    double currentTimeDist;
    double currentDist2;
    double currentTimeDist2;
    for (Integer q : positionsAllowed) {
        for (int i = 0; i < Main.V; i++) {
            for (int j = 0; j < Main.V; j++) {
                if (w[i][j][q][currentK] == 1 && q != currentQ) {
                    currentDist = MyData.distance[i][plwl];
                    currentTimeDist = MyData.TWstart[plwl] - MyData.TWend[i];
                    if (currentTimeDist > 0) currentDist -= currentTimeDist;
                    currentDist2 = MyData.distance[plwl][j];
                    currentTimeDist2 = MyData.TWstart[j] - MyData.TWend[plwl];
                    if (currentTimeDist2 > 0) currentDist2 -= currentTimeDist2;
                    if ((Math.abs(currentDist) + Math.abs(currentDist2)) < posDist && i != plwl) {
                        posDist = Math.abs(currentDist) + Math.abs(currentDist2);
                        position[0] = q;
                        position[1] = j;
                    }
                }
            }
        }
    }
    // Determine plwl's future follower after finding the best position:
    for (int i = 0; i < Main.V; i++) {
        for (int j = 0; j < Main.V; j++) {
            if (w[i][j][position[0] + 1][currentK] == 1) position[1] = j;
        }
    }

    public static SinglePositionValues findSinglePosition2(int[][][] w, int customer, int[] plw) {
        for (int i = 0; i < w.length; i++) {
            for (int j = 0; j < w.length; j++) {
                for (int k = 0; k < Main.k; k++) {
                    if (w[i][customer][q][k] == 1) {
                        plw[0] = i;
                        plw[1] = customer;
                        plw[2] = q;
                        plw[3] = k;
                    }
                }
            }
        }
    }
if (w[customer][j][q][k] == 1) {
    plw[4] = customer;
    plw[5] = j;
    plw[6] = q;
    plw[7] = k;
}
return new SinglePositionValues(plw);
}

public static void findFurthestPair(/*int i, int j, boolean ij,*/ int[][][] w, int[] secondPair, int k){
    int[] currenthighest=Integer.MIN_VALUE;
    double currentDist = 0;
    for (int q=1; q<w[0][0][0].length; q++){
        for (int ii=0; ii<w.length; ii++){
            if (w[ii][jj][q][k]==1){
                currentDist = MyData.distance[ii][jj];
            }
            if (currentDist>currenthighest){
                currenthighest=currentDist;
                secondPair[0] = jj;
                for (int iii=0; iii<Main.V; iii++){
                    if (MyData.demand[iii][jj]>0){
                        secondPair[1] = secondPair[0];
                        secondPair[0] = iii;
                    }
                    else if (MyData.demand[jj][iii]>0) secondPair[1]=iii;
                }
            }
        }
    }
    System.out.println("Remove customer:");
    System.out.println(secondPair[0] + " and " + secondPair[1] + " from vehicle "+ k);
}

private static void localRouteOpt(int cpos, int mpos, int k, List<List<Integer>> wLists){
    List<Integer> thisRoute = new ArrayList<Integer>();
    List<Integer> selection = new ArrayList<Integer>();
    System.out.println("Sub-route for optimisation:");
    for (int i=cpos+1; i<=mpos; i++){
        selection.add(wLists.get(k).get(i));
        System.out.println(wLists.get(k).get(i)+", ");
    }
    System.out.println();
    //copy relevant tour section
    System.out.println("Sub-route for optimisation:");
    for (int i=cpos+1; i<=mpos; i++){
        selection.add(wLists.get(k).get(i));
        System.out.println(wLists.get(k).get(i)+", ");
    }
    //System.out.println();
    thisRoute.add(wLists.get(k).get(cpos));
    thisRoute.add(0);
    int c1;
    int c2;
    while(selection.size()>0){
        double currentFurthest= Integer.MIN_VALUE;
        double currentDist;
        double[] currentRouteCentre = new double[2];
c1=0;
c2=0;
// determine current route centre:
for (int i=0; i < thisRoute.size(); i++) {
currentRouteCentre[0]+=MyRMcontrol.fileMatrixCurrent.get(thisRoute.get(i))[1];
currentRouteCentre[1]+=MyRMcontrol.fileMatrixCurrent.get(thisRoute.get(i))[2];
}
currentRouteCentre[0]/= thisRoute.size();
currentRouteCentre[1]/= thisRoute.size();
//find furthest customer from route centre:
for (int r=0; r < selection.size(); r++) {
currentDist = MyRMcontrol.norm(currentRouteCentre[0], currentRouteCentre[1], MyRMcontrol.fileMatrixCurrent.get(selection.get(r))[1], MyRMcontrol.fileMatrixCurrent.get(selection.get(r))[2]);
if (currentDist>currentFurthest){
currentFurthest = currentDist;
c1=selection.get(r);
c2=MyRMcontrol.fileMatrixCurrent.get(selection.get(r))[8].intValue();
} else {c1=selection.get(r);}
}
//System.err.println("c1: "+c1+" c2: "+c2);
//remove c1 & c2 from route list:
for (int i=0; i < selection.size(); i++) {
if (selection.get(i) == c1) {
selection.remove(i);
i--;
} else if (selection.get(i) == c2 && c2 != 0) {
selection.remove(i);
}
}
//find minimum distance positions for c1 & c2:
double posDist = Integer.MAX_VALUE;
int posC1 = 0;
int posC2 = 0;
//reuse:
for (int i=0; i < thisRoute.size()-1; i++) {
currentDist = (MyData.distance[thisRoute.get(i)][c1] + MyData.distance[c1][thisRoute.get(i+1)]) - MyData.distance[thisRoute.get(i)][thisRoute.get(i+1)];
System.out.println("current distance: "+c1+" "+ c2+" currentDist");
if (currentDist < posDist) {
posDist = currentDist;
posC1 = i+1;
System.out.println("New min distance c1 : "+c1+" "+ currentDist);
}
thisRoute.add(posC1, c1);
System.out.println("added customer: "+c1+" at pos "+posC1);
if (c2!=0){
posDist = Integer.MAX_VALUE;
for (int i=posC1; i < thisRoute.size()-1; i++) {
currentDist = (MyData.distance[thisRoute.get(i)][c2] + MyData.distance[c2][thisRoute.get(i+1)]) - MyData.distance[thisRoute.get(i)][thisRoute.get(i+1)];
    
    //System.out.println("current distance: "+ currentDist);
    if (currentDist < posDist){
        posDist = currentDist;
        posC2 = i+1;
        //System.out.println("New min distance c2 : "+ currentDist);
    }
    
    thisRoute.add(posC2, c2);
    //System.out.println("added customer: "+c2 +" at pos "+posC2);
    }
}

System.out.println();

/*************************/
// replace old sub-route by new sub-route:
thisRoute.remove(thisRoute.size()-1);
for(int i=cpos; i<=mpos; i++){
    wLists.get(k).remove(cpos);
    }
    wLists.get(k).addAll(cpos, thisRoute);
}

//System.out.println("Vehicle route after local optimisation:");
for(int i=0; i< wLists.get(k).size();i++){
    System.out.print(wLists.get(k).get(i)+" ");
    System.out.println();
}
private static void reSortSelection2(int cpos, int mpos, int k, List<List<Integer>> wLists){
    List<Integer> selectionNew = new ArrayList<Integer>();
    List<Integer> selection = new ArrayList<Integer>();
    // System.out.println("cpos : "+cpos+" mpos : "+ mpos + " k: ":"+k);
    //copy relevant tour section
    if(cpos+1-mpos-1>1){
        int counter = cpos+1;
        for(int i=cpos+1; i<mpos; i++){
            //if(mpos < wLists.get(k).size()){
                selection.add(wLists.get(k).get(counter));
                System.out.print(wLists.get(k).get(counter)+" ");
                wLists.get(k).remove(counter);
            //}
        }
    }
    
    //find furtherst form 1st and last:
    double furthestDist = Integer.MIN_VALUE;
    int furthestPos = 0;
    double dist = 0;
    for(int i=0; i<selection.size(); i++){
        dist = MyData.distance[wLists.get(k).get(cpos)][selection.get(i)] + MyData.distance[selection.get(i)][wLists.get(k).get(cpos+1)] - MyData.distance[wLists.get(k).get(cpos)][wLists.get(k).get(cpos+1)];
        if(dist > furthestDist){
            furthestDist = dist;
            furthestPos = i;
        }
    }
    // insert furthest in selNew + delete from sel:
//System.out.println("set c "+selection.get(furthestPos)+ " on first pos in list");
selection.add(0, selection.get(furthestPos));
selection.remove(furthestPos+1);

//find best positions for all customers:
int c=0;
double nearestDist;
int nearestPos = 0;
while(selection.size()>0){
    nearestDist = Integer.MAX_VALUE;
c=selection.get(0);
    for(int i =1; i<wLists.get(k).size(); i++){
        dist = MyData.distance[wLists.get(k).get(i-1)][c] + MyData.distance[c][wLists.get(k).get(i)] - MyData.distance[wLists.get(k).get(i-1)][wLists.get(k).get(i)];
        if(dist < nearestDist){
            nearestDist = dist;
            nearestPos = i;
        }
    }
    System.out.println("shortest for c "+c+" is between "+wLists.get(k).get(nearestPos-1)+" and "+wLists.get(k).get(nearestPos));
wLists.get(k).add(nearestPos, c);
selection.remove(0);
}

//System.out.println("selection re-sorted by looking at the overall k-route");

static void insertPairInList(int c, int m, List<Integer> wList){
double posDist = Integer.MAX_VALUE;
double currentDist;
double currentDist2;
double currentDist3;
double currentDist4 = Integer.MAX_VALUE;
//find position for m:
int posBest2 = posBest;
for(int i=0; i<wList.size()-1; i++){
    if(MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPosition, wList.get(i+1)) == false){
        currentDist = MyData.distance[wList.get(i)][c];
currentDist2 = MyData.distance[c][wList.get(i+1)];
        J: for(int j=i; j<wList.size()-1; j++){
            if(wList.get(j+1) == 0 || MyObjectiveFunction.doesArrayContain(MyRMcontrol.startingPositionRem, wList.get(j+1))){
                //System.out.println("continue with next i pos, current j ID: "+wList.get(j+1));
                break J;
            }
        }
        currentDist3 = MyData.distance[wList.get(j)][m];
currentDist4 = MyData.distance[m][wList.get(j+1)];
        if((Math.abs(currentDist)+Math.abs(currentDist2)+Math.abs(currentDist3)+Math.abs(currentDist4))<posDist){
            posDist=Math.abs(currentDist)+Math.abs(currentDist2)+Math.abs(currentDist3)+Math.abs(currentDist4);
        }
    } else{
        currentDist3 = MyData.distance[wList.get(j)][m];
currentDist4 = MyData.distance[m][wList.get(j+1)];
        if((Math.abs(currentDist)+Math.abs(currentDist2)+Math.abs(currentDist3)+Math.abs(currentDist4))<posDist){
            posDist=Math.abs(currentDist)+Math.abs(currentDist2)+Math.abs(currentDist3)+Math.abs(currentDist4);
        }
    }
}
ntDist3)+Math.abs(currentDist4);
2326     posBest = i+1;
2327     posBest2 = j+1;
2328     //System.out.println("Pair insert. min dist: "+posDist+" i: "+posBest+" j: "+posBest2);
2329 }
2330 }
2331 }
2332 }
2333 }
2334 // System.out.println("Insert customer "+c+" at pos "+posBest);
2335 // System.out.println("Insert customer "+m+" at pos "+posBest2);
2336 
2337 // Insert customers in wList:
2338 wList.add(posBest2, m);
2339 wList.add(posBest, c);
2340 
2341 }
2342 
2343 private static void checkForPDerror(int cpos, int mpos, int k, List<List<Integer>> wLists) {
2344     int cToMove;
2345     for(int i = cpos+1; i<mpos-1; i++){
2346         if(MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(i))[7] > 0){
2347             // If this node receives a delivery
2348             for(int j = i+1; j<mpos; j++){//find sender in succeeding positions
2349                 if(wLists.get(k).get(j) == MyRMcontrol.fileMatrixCurrent.get(Operator.wLists.get(k).get(i))[7].intValue()){
2350                     cToMove = wLists.get(k).get(j);
2351                     wLists.get(k).remove(j);
2352                 }
2353             }
2354             cToMove = wLists.get(k).get(i);
2355             double nearestDist = Integer.MAX_VALUE;
2356             double dist;
2357             int nearestPos = i;
2358             for(int l=cpos+1; l<=i; l++){
2359                 dist = MyData.distance[wLists.get(k).get(l-1)][cToMove] + MyData.distance[cToMove][wLists.get(k).get(l)];
2360             }
2361         } else {
2362             nearestDist = dist;
2363             nearestPos = l;
2364         }
2365     }
2366     wLists.get(k).add(nearestPos, cToMove);
2367 }
import java.util.Arrays;

public class PositionValues {
  // Variablen fuer methoden:
  
  private boolean cmpair;
  public boolean isCmpair() {
    return cmpair;
  }

  public boolean isMcpair() {
    return mcpair;
  }

  public boolean mcpair;

  public int[] plw;
  public int[] p2w;

  public PositionValues(boolean cm, boolean mc, int[] arr1, int[] arr2) {
    cmpair = cm;
    mcpair = mc;
    this.plw = Arrays.copyOf(arr1, arr1.length);
    this.p2w = Arrays.copyOf(arr2, arr2.length);
  }

  public int[] getArray1() {
    return plw;
  }

  public int[] getArray2() {
    return p2w;
  }

  public void setArray1(int[] array) {
    this.plw = array;
  }

  public void setArray2(int[] array) {
    this.p2w = array;
  }
}
E.17. ReadSINTEFwriteH class

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class readSINTEFwriteH {

    static FileWriter fw;
    static BufferedWriter bw;
    static DecimalFormat df = new DecimalFormat("#.00");
    static int kkk;
    static int VVV;
    static int vvv;
    static int[][] fileMatrix;
    static double[][] extractedFileMatrix;
    static double[][] customerMatrix;
    static double[][] demandMatrix;
    static String readFile;
    static String writeFile;

    public static void main(String[] args) throws IOException{
        for (int i = 1; i <= 8; i++) {
            readFile = new String(MyRMcontrol.directory + "02 VRPPD test files\02 SINEF_pdp_100\pdp_100\lrc20" + i + ".txt");
            writeFile = new String(MyRMcontrol.directory + "02 VRPPD test files\04 My RM test files\DVRPPDCE Solomon\lrc20+" + i + "DVRPPDCE_FC.txt");
            List<Integer> customers = Arrays.asList(81, 78, 104, 76, 71, 70, 73, 77, 79, 80, 57, 55, 53, 56, 58, 59, 45, 48);
            // erzeugt neue datei mit anzahl von pickups
            //splitFileWriteH(customers, 2, 10);
            //splitFileInSets(customers, 6, 5, 5);
            //splitFileInSetsPlus1(customers, 6, 5, 5);
            //splitFileExplicitWriteH(customers, 6, 5, 5);
            //readSintefCreateRM();
            //writeOriginalInRMFormat();
        }
    }

    private static void writeOriginalInRMFormat() throws FileNotFoundException, IOException{
        readFile();
        double[][] newFileMatrix = new double[VVV][10];
    }
}
```
List<Double[]> newFileMatrix2 = new ArrayList<Double[]>();
List<Double[]> newFileMatrix3 = new ArrayList<Double[]>();
System.out.println();
Double[] row = new Double[10];
for (int j=0; j<9; j++){
    row[j] = (double) fileMatrix[0][j];
    System.out.print(row[j] + "\t");
}
newFileMatrix2.add(row);
System.out.println();
for (int i=1; i<VVV; i++){
    if(fileMatrix[i][0] > 0 || fileMatrix[i][1] > 0){
        Double[] row1 = new Double[10];
        for (int j=0; j<9; j++){
            row1[j] = (double) fileMatrix[i][j];
            System.out.print(row1[j] + "\t");
        }
        line ++;
        newFileMatrix2.add(row1);
        System.out.println();
    }
}
System.err.println("SIZEe "+newFileMatrix3.size());

// add request occurring time:
for (int i=0; i<newFileMatrix2.size(); i++){
    newFileMatrix2.get(i)[9] = 0.0;
}
System.out.println();
for (int i=0; i<newFileMatrix2.size(); i++){
    for (int j=0; j<10; j++){
        System.out.print(newFileMatrix2.get(i)[j]+"\t");
    }
    System.out.println();
}
writeFileSintef(newFileMatrix2, 3);
System.err.println("SIZE "+newFileMatrix2.size());

private static void readSintefCreateRM() throws IOException {
    readFile();
    double[][] newFileMatrix = new double[VVV][10];
    List<Double[]> newFileMatrix2 = new ArrayList<Double[]>();
    List<Double[]> newFileMatrix3 = new ArrayList<Double[]>();
    System.out.println();
    Double[] row = new Double[10];
    for (int j=0; j<9; j++){
        newFileMatrix[0][j] = fileMatrix[0][j];
        row[j] = (double) fileMatrix[0][j];
        System.out.print(newFileMatrix[0][j] + "\t");
    }
    newFileMatrix2.add(row);
    System.out.println();
    int line=1;
    for (int i=1; i<VVV; i++){
        if(fileMatrix[i][3] > 0){
            Double[] row1 = new Double[10];
            for (int j=0; j<9; j++){
                //newFileMatrix[line][j] = fileMatrix[i][j];
                row1[j] = (double) fileMatrix[i][j];
                System.out.print(row1[j] + "\t");
            }
            line ++;
            newFileMatrix2.add(row1);
            System.out.println();
        }
    }
}
Double[] row2 = new Double[10];
System.out.println();
for(int j=0; j<9; j++){
    //newFileMatrix[line+1][j] = fileMatrix[fileMatrix[i][8]][j];
    row2[j]=(double) fileMatrix[fileMatrix[i][8]][j];
    System.out.print(row2[j] + "\t");
}
line +=2;
newFileMatrix2.add(row1);
newFileMatrix2.add(row2);
System.out.println();
}

//sortiere nach TW end && nummeriere neu:
double nummer = 1;
newFileMatrix3.add(newFileMatrix2.get(0));
while(newFileMatrix2.size()>1){
    double lowestTW = Double.MAX_VALUE;
    int position=0;
    for(int i=1; i<newFileMatrix2.size(); i+=2){
        if(newFileMatrix2.get(i)[5]<lowestTW){
            lowestTW = newFileMatrix2.get(i)[5];
            position =i;
        }
    }
    newFileMatrix3.add(newFileMatrix2.get(position));
    newFileMatrix3.add(newFileMatrix2.get(position+1));

    /***** nummerierung:******/
    newFileMatrix3.get(newFileMatrix3.size()-2)[0] = nummer;
    newFileMatrix3.get(newFileMatrix3.size()-1)[0] = nummer+1;
    newFileMatrix3.get(newFileMatrix3.size()-2)[8] = nummer+1;
    newFileMatrix3.get(newFileMatrix3.size()-1)[7] = nummer;
    nummer+=2;
    /***********************/
    newFileMatrix2.remove(position+1);
    newFileMatrix2.remove(position);
}

//manipuliere TW:
//adjust TW to 120:
//for(int i=1; i<newFileMatrix3.size(); i++){
//    double TWsize = newFileMatrix3.get(0)[5] / 3;
//    if(newFileMatrix3.get(i)[5] > newFileMatrix3.get(0)[5]){*
//        double diff = newFileMatrix3.get(i)[5] - newFileMatrix3.get(0)[5];
//        newFileMatrix3.get(i)[5] =3390.0;
//        newFileMatrix3.get(i)[4] = 0;
//}
//}*
//service time:
//newFileMatrix3.get(i)[6] = 1.0;

//manipuliere CAP:
//for(int i=1; i<newFileMatrix3.size(); i++){
//    newFileMatrix3.get(i)[3] = newFileMatrix3.get(i)[3]/1000;
//}

//for(int i=0; i<newFileMatrix3.size(); i++){
//    newFileMatrix3.get(i)[4] = 0.0;
//    newFileMatrix3.get(i)[5] =120.0;
//    newFileMatrix3.get(i)[6] = 0.0;
//}
add request occurring time:

```java
for(int i=0; i<5; i++)
    newFileMatrix3.get(i)[9] = -13.0 /*0.0*/;
```

```java
for(int i=0; i<newFileMatrix3.size(); i++)
    newFileMatrix3.get(i)[9] = 0.0;
```

```java
double arrivalTime = -12 /*0*/;
for(int i=5; i<newFileMatrix3.size(); i+=2)
    newFileMatrix3.get(i)[9] = arrivalTime;
    newFileMatrix3.get(i+1)[9] = arrivalTime;
    arrivalTime+=1;
```

```java
System.out.println();
for(int i=0; i<newFileMatrix3.size(); i++)
    for(int j=0; j<10; j++)
        System.out.print(newFileMatrix3.get(i)[j] + "t");
    System.out.println();
writeFileSintef(newFileMatrix3, 25);
```
```cpp
int position = 0;
for (int i = 1; i < newFileMatrix2.size(); i += 2) {
    if (newFileMatrix2.get(i)[5] < lowestTW) {
        lowestTW = newFileMatrix2.get(i)[5];
        position = i;
    }
}
newFileMatrix3.add(newFileMatrix2.get(position));
newFileMatrix3.add(newFileMatrix2.get(position + 1));

//nummerierung: 
newFileMatrix3.get(newFileMatrix3.size() - 2)[0] = nummer;
newFileMatrix3.get(newFileMatrix3.size() - 1)[0] = nummer + 1;
newFileMatrix3.get(newFileMatrix3.size() - 2)[8] = nummer + 1;
newFileMatrix3.get(newFileMatrix3.size() - 1)[7] = nummer;
ummer += 2;

newFileMatrix2.remove(position + 1);
newFileMatrix2.remove(position);

//manipuliere TW:
//adjust TW to 120:
for (int i = 1; i < newFileMatrix3.size(); i++) {
    double TWsize = newFileMatrix3.get(0)[5] / 3;
    if (newFileMatrix3.get(i)[5] > newFileMatrix3.get(0)[5]) {
        double diff = newFileMatrix3.get(i)[5] - newFileMatrix3.get(0)[5];
        newFileMatrix3.get(i)[5] = 3390.0;
        newFileMatrix3.get(i)[4] = 0;
    }
    service time:
    newFileMatrix3.get(i)[6] = 1.0;
}

//manipuliere CAP:
for (int i = 1; i < newFileMatrix3.size(); i++) {
    newFileMatrix3.get(i)[3] = newFileMatrix3.get(i)[3] / 1000;
}

// add request occurring time:
for (int i = 0; i < 5; i++) {
    newFileMatrix3.get(i)[9] = -13.0; /*0.0*/;
}
for (int i = 0; i < newFileMatrix3.size(); i++) {
    newFileMatrix3.get(i)[9] = 0.0;
}

double arrivalTime = -12; /*0*/;
for (int i = 1; i < newFileMatrix3.size(); i += 2) {
    newFileMatrix3.get(i)[9] = arrivalTime;
    newFileMatrix3.get(i + 1)[9] = arrivalTime;
    arrivalTime += 1;
}

// Create Revenues:
newFileMatrix3.get(0)[10] = 0.0;
for (int i = 1; i < newFileMatrix3.size(); i += 2) {
```
newFileMatrix3.get(i)[10] = simulateRevenues(newFileMatrix3.get(0)[1], newFileMatrix3.get(0)[2], newFileMatrix3.get(i)[1], newFileMatrix3.get(i)[2], newFileMatrix3.get(i+1)[1], newFileMatrix3.get(i+1)[2]);

while(newFileMatrix3.get(i)[10] <= 0) newFileMatrix3.get(i)[10] = simulateRevenues(newFileMatrix3.get(0)[1], newFileMatrix3.get(0)[2], newFileMatrix3.get(i)[1], newFileMatrix3.get(i)[2], newFileMatrix3.get(i+1)[1], newFileMatrix3.get(i+1)[2]);

newFileMatrix3.get(i+1)[10] = 0.0;

System.out.println();

for(int i=0; i<newFileMatrix3.size(); i++){
    System.out.print(newFileMatrix3.get(i).length + "t");
    System.out.println();
}

writeFileSintef(newFileMatrix3, 25);

private static void writeFileSintef(List<Double[]> newFileMatrix3, int kkk) {
    try {
        fw = new FileWriter(writeFile); //C:\Users\21576959\Workspace\02 VRP PD test files\02 SINTEF_pdp_100\lc101H10.txt);
        bw = new BufferedWriter(fw);
        bw.write("kkk/3 + "t");
        bw.write(VVV + "t";
        bw.newLine();
        for(int i=0; i<newFileMatrix3.size(); i++){
            System.out.println("i: " + i + " j: " + newFileMatrix3.get(i)[j])
            bw.newLine();
        }
        bw.close();
    } catch (java.io.IOException ex) {
        System.out.println("IO Error: " + ex);
    }
}

private static void splitFileInSetsPlus1(List<Integer> pickUps, int numberOfPickups, int startAfterCounter, int customerToAdd) throws FileNotFoundException, IOException{
    readFile();
    System.out.println();
    int startAfterTotal = 2;
    int startAfterCustomer = customerToAdd;
    int number = 1;

    while(startAfterCustomer+numberOfPickups<fileMatrix.length){
        startAfterCounter=startAfterTotal;
        extractedFileMatrix = new double[3+numberOfPickups*2][9];
        for(int j=0; j<9; j++){
            extractedFileMatrix[0][j]=fileMatrix[0][j];
            extractedFileMatrix[1][j]=fileMatrix[customerToAdd][j];
            System.out.print(fileMatrix[0][j]+ " ");
        }
    }

System.out.println();
List<Integer> recipients = new ArrayList<Integer>();
recipients.add(fileMatrix[customerToAdd][8]);
int ii=2;
for(int i=customerToAdd+1; i<fileMatrix.length; i++){
    if(i>startAfterCustomer && startAfterCounter>=0 && fileMatrix[i][3]>0 && ii <=numberOfPickups+1) {
        //System.out.println( "Customer "+i+" added to the file" );
        System.out.println(ii+" ");
        //System.out.print("request added to file: ",i+" ");
        for(int j=1; j<8; j++){
            extractedFileMatrix[ii][j] = fileMatrix[i][j];
            System.out.print(fileMatrix[i][j]+" ");
        }
        extractedFileMatrix[ii][8]=numberOfPickups+ii;
        System.out.print(numberOfPickups+ii);
        ii++;
        recipients.add(fileMatrix[i][8]);
        System.out.println();
        startAfterCustomer=i;
        } else if(fileMatrix[i][3]>0){
            startAfterCounter--;
            //System.out.println("Startaftercounter = ",startAfterCounter);
        }
    }
    //Haenge recipients an:
    for(int recipient : recipients){
        extractedFileMatrix[ii][0]=ii;
        System.out.println(ii+" ");
        extractedFileMatrix[ii][j] = fileMatrix[recipient][j];
        System.out.print(fileMatrix[recipient][j]+" ");
    }
    extractedFileMatrix[ii][7]=i-(numberOfPickups+1);
    System.out.print(ii-(numberOfPickups+1));
    ii++;
    System.out.println();
} //Haenge recipients anew:
    for(int i = 0; i < extractedFileMatrix.length; i++) {
        for( int j = i+1; j < extractedFileMatrix.length; j++ )
            customermatrix[i][j] = customermatrix[j][i] = norm(extractedFileMatrix[i][1], extractedFileMatrix[i][2], extractedFileMatrix[j][1], extractedFileMatrix[j][2]);
    }
    //Dieser teil wurde geaendert: Wenn ein standort belieft wird...
    demandMatrix = new double[extractedFileMatrix.length][extractedFileMatrix.length];
    for( int i = 0; i < extractedFileMatrix.length; i++) {
        if(extractedFileMatrix[i][3]<0) {
            demandMatrix[(int) extractedFileMatrix[i][7]][i] = (extractedFileMatrix[i][3]*(-1));
        }
    }
    //Wirte in File:
    writeFile(extractedFileMatrix.length, extractedFileMatrix, writeFile +number+"+.txt");
    startAfterTotal+=numberOfPickups;
private static void splitFileInSets(List<Integer> pickUps, int numberOfPickups, int startAfterCounter, int startAfterCustomer) throws FileNotFoundException, IOException {
    readFile();
    System.out.println();
    int startAfterTotal = 2;
    //int startAfterCustomer = 2;
    int number = 1;
    while (startAfterCustomer + numberOfPickups < fileMatrix.length) {
        startAfterCounter = startAfterTotal;
        extractedFileMatrix = new double[1 + numberOfPickups * 2][9];
        for (int j = 0; j < 9; j++) {
            extractedFileMatrix[0][j] = fileMatrix[0][j];
            System.out.print(fileMatrix[0][j] + " ");
        }
        System.out.println();
        List<Integer> recipients = new ArrayList<Integer>();
        int ii = 1;
        for (int i = startAfterCustomer + 1; i < fileMatrix.length; i++) {
            if (startAfterCounter >= 0 && fileMatrix[i][3] > 0 && ii <= numberOfPickups) {
                System.out.println("Customer " + i + " added to the file");
                System.out.println("request added to file: " + i + " ");
                for (int j = 1; j < 8; j++) {
                    extractedFileMatrix[ii][j] = fileMatrix[i][j];
                    System.out.print(fileMatrix[i][j] + " ");
                }
                extractedFileMatrix[ii][8] = numberOfPickups + ii;
                System.out.println(numberOfPickups + ii);
                ii++;
                recipients.add(fileMatrix[i][8]);
                System.out.println();
                startAfterCustomer = i;
            } else if (fileMatrix[i][3] > 0) {
                startAfterCounter--;
                System.out.println("Startaftercounter = "+startAfterCounter);
            }
        }
        //Haenge recipients an:
        for (int recipient : recipients) {
            extractedFileMatrix[ii][0] = ii;
            System.out.print(ii + " ");
            for (int j = 1; j < 7; j++) {
                extractedFileMatrix[ii][j] = fileMatrix[recipient][j];
                System.out.print(fileMatrix[recipient][j] + " " );
            }
            extractedFileMatrix[ii][7] = ii + numberOfPickups;
            System.out.println(ii + numberOfPickups);
            ii++;
            System.out.println();
        }
        ///Create Matrix
customermatrix = new double[extractedFileMatrix.length][extractedFileMatrix.length];
        for (int i = 0; i < extractedFileMatrix.length; i++)
            for (int j = i + 1; j < extractedFileMatrix.length; j++)
                }
513  customermatrix[i][j] = customermatrix[j][i] = norm(extractedFileMatrix[i][1], extractedFileMatrix[i][2], extractedFileMatrix[j][1], extractedFileMatrix[j][2]);
514
515  // Dieser teil wurde geaendert: Wenn ein standort beliefert wird...
516  demandMatrix = new double[extractedFileMatrix.length][extractedFileMatrix.length];
517  
518  for (int i = 0; i < extractedFileMatrix.length; i++) {
519    if (extractedFileMatrix[i][3] < 0) {
520      demandMatrix[(int) extractedFileMatrix[i][7]][i] = (extractedFileMatrix[i][3] * (-1));
521    }
522  }
523
524  // Wirte in File:
525  writeFile(extractedFileMatrix.length, extractedFileMatrix, writeFile + number + ".txt");
526  startAfterTotal += numberOfPickups;
527  number++;
528  }
529  }
530  
531  private static void splitFileExplicitWriteH(List<Integer> pickUps, int numberOfPickups) throws FileNotFoundException, IOException{
532    readFile();
533    System.out.println();
534    //neue matrix wird erstellt:
535    extractedFileMatrix = new double[1+pickUps.size()][9];
536    // depot wird kopiert:  
537    for(int j=0; j<9; j++){
538      extractedFileMatrix[0][j] = fileMatrix[0][j];
539      System.out.println(fileMatrix[0][j] + "");
540    }
541    System.out.println();
542    List<Integer> receipients = new ArrayList<Integer>();
543    
544    int ii=1;
545    for(int i=0; i<fileMatrix.length; i++){
546      if(fileMatrix[i][3] > 0 && pickUps.contains(fileMatrix[i][0])){ //ii <=numberOfPickups}
547        extractedFileMatrix[ii][0]=ii;
548        System.out.print(ii + "");
549        for(int j=1; j<8; j++){
550          extractedFileMatrix[ii][j] = fileMatrix[i][j];
551          System.out.print(fileMatrix[i][j] + "");
552        }
553        extractedFileMatrix[ii][8] = (pickUps.size() / 2) + ii;
554        System.out.print((pickUps.size() / 2) + ii);
555        ii++;
556        receipients.add(fileMatrix[i][8]);
557        System.out.println();
558      }
559    }
560    //Haenge receipients an:
561    for(int recipient : receipients){
562      extractedFileMatrix[ii][0]=ii;
563      System.out.print(ii + "");
564      for(int j=1; j<7; j++){
565        extractedFileMatrix[ii][j] = fileMatrix[recipient][j];
566        System.out.print(fileMatrix[recipient][j] + "");
567    }
568    }
569    extractedFileMatrix[ii][7] = ii - (pickUps.size() / 2);
570    System.out.print(ii - (pickUps.size() / 2));
571    ii++;
572    System.out.println();
573  }
Create Matrix

customermatrix = new double[extractedFileMatrix.length][extractedFileMatrix.length];

for (int i = 0; i < extractedFileMatrix.length; i++)
    for (int j = i+1; j < extractedFileMatrix.length; j++)
        customermatrix[i][j] = customermatrix[j][i] = norm(extractedFileMatrix[i][1], extractedFileMatrix[i][2], extractedFileMatrix[j][1], extractedFileMatrix[j][2]);

Dieser Teil wurde geändert: Wenn ein Standort beliefert wird...

demandMatrix = new double[extractedFileMatrix.length][extractedFileMatrix.length];

for (int i = 0; i < extractedFileMatrix.length; i++)
    if(extractedFileMatrix[i][3]<0){
        demandMatrix[(int) extractedFileMatrix[i][7]][i] = (extractedFileMatrix[i][3]*(-1));
    }

for (int i=0; i<9; i++)
{
    System.out.print(fileMatrix[0][j]+" ");
}

List<Integer> receipients = new ArrayList<Integer>();

for(int i=0; i<fileMatrix.length; i++)
{
    if(startAfterCounter<=0 && fileMatrix[i][3]>0 && ii <=numberOfPickups)
    {
        extractedFileMatrix[ii][0]=ii;
        System.out.print(ii+" ");
        for(int j=1; j<8; j++)
        {
            extractedFileMatrix[ii][j] = fileMatrix[i][j];
            System.out.print(fileMatrix[i][j]+" ");
        }
        System.out.println();
        receipients.add(fileMatrix[i][8]);
        System.out.println();
        if(fileMatrix[i][3]>0){
            startAfterCounter--;
            System.out.println("Startaftercounter = "+startAfterCounter);
        }else if(fileMatrix[i][3]<0){
        }
    }else if(fileMatrix[i][3]>0){
        startAfterCounter--;
        System.out.println("Startaftercounter = "+startAfterCounter);
    }else if(fileMatrix[i][3]<0){
    }
}

// Haenge receipients an:
for (int recipient : receipients){
    extractedFileMatrix[ii][0]=ii;
    System.out.print(ii+" ");
    for(int j=1; j<7; j++)
    {
        extractedFileMatrix[ii][j] = fileMatrix[recipient][j];
        System.out.print(fileMatrix[recipient][j]+" ");
    }
    System.out.println();
}

extractedFileMatrix[ii][7]=ii-numberOfPickups;
System.out.print(ii-numberOfPickups);
ii++;
System.out.println();
}

/////Create Matrix
customermatrix = new double[extractedFileMatrix.length][extractedFileMatrix.length];

for( int i = 0; i < extractedFileMatrix.length; i++)
for( int j = i+1; j < extractedFileMatrix.length; j++)
customermatrix[i][j] = customermatrix[j][i] = norm(extractedFileMatrix[i][1], extractedFileMatrix[i][2], extractedFileMatrix[j][1], extractedFileMatrix[j][2]);

// Dieser teil wurde geaendert: Wenn ein standort beliefert wird...
demandMatrix = new double[extractedFileMatrix.length][extractedFileMatrix.length];

for( int i = 0; i < extractedFileMatrix.length; i++)
if(extractedFileMatrix[i][3]<0){
demandMatrix[(int)extractedFileMatrix[i][7)][i] = (extractedFileMatrix[i][3]*(-1));
}

public static void readfilewriteH1(int numberOfLocations) throws FileNotFoundException, IOException{
readFile();
System.out.println();
exttractedFileMatrix = new double[numberOfLocations][9];

for(int i=0; i<extractedFileMatrix.length; i++){
for( int j=0; j<9; j++)
extractedFileMatrix[i][j] = fileMatrix[i][j];
System.out.println(fileMatrix[i][j] + " ");
}
System.out.println("READFILEMETHOD");

/////Create Matrix
customermatrix = new double[extractedFileMatrix.length][extractedFileMatrix.length];

for( int i = 0; i < extractedFileMatrix.length; i++)
for( int j = i+1; j < extractedFileMatrix.length; j++)
customermatrix[i][j] = customermatrix[j][i] = norm(extractedFileMatrix[i][1], extractedFileMatrix[i][2], extractedFileMatrix[j][1], extractedFileMatrix[j][2]);

System.out.println();

/////Create Matrix
customermatrix = new double[extractedFileMatrix.length][extractedFileMatrix.length];

for( int i = 0; i < extractedFileMatrix.length; i++)
for( int j = 0; j < extractedFileMatrix.length; j++)
System.out.println(df.format(customermatrix[i][j]) + "\t");

System.out.println();

// Dieser teil wurde geaendert: Wenn ein standort beliefert wird...
demandMatrix = new double[extractedFileMatrix.length][extractedFileMatrix.length];

for( int i = 0; i < extractedFileMatrix.length; i++)
if(extractedFileMatrix[i][3]<0){
demandMatrix[(int)extractedFileMatrix[i][7]][i] = (extractedFileMatrix[i][3]*(-1));
public static void writeFile(int VVV, double[][] extractedFileMatrix2, String writeFileName) {
    try {
        fw = new FileWriter(writeFileName); //C: \Users\21576959\Workspace\02 VRRPD test files\02 SINTEF_pdp_100\lc101H10.txt);
        bw = new BufferedWriter(fw);
        System.out.println();
        System.out.println("distance={"); bw.write("[");
        for (int i = 0; i < VVV; i++) {
            if (i == 0) {
                System.out.println("t [");
                bw.write("[");
            } else {
                System.out.println("\t ");
                bw.write("[");
            }
            for (int j = 0; j < VVV; j++) {
                if (i == j) {
                    System.out.println(customermatrix[i][j]); //"0.0");
                    bw.write("0.0");
                } else {
                    System.out.print( df.format(customermatrix[i][j])); //df.format(r.nextDouble()*100));
                    bw.write( df.format(customermatrix[i][j])); //r.nextDouble()*100));
                    bw.newLine();
                }
            }
            if (j < VVV - 1) {
                System.out.println( ", ");
                bw.write( ", ");
            }
            if (i < VVV - 1) {
                System.out.println("]");
                bw.write("]");
            } else {
                System.out.println("]");
                bw.write("]");
            } System.out.println("\n");bw.newLine();
        }
        /////// WRITE DEMAND:
        boolean pair=true;
        int demand;
        System.out.println("demand= {"); bw.write("[");
        for (int i = 0; i < VVV; i++) {
            if (i == 0) {
                System.out.println("[");
                bw.write("[");
            }
        } else {
```java
System.out.print("\t [");
bw.write("[");
}
for (int j = 0; j < VVV; j++) {
    if (i==j){
        System.out.print(demandMatrix[i][j]);//"0.0");
bw.write("0.0");
    }
else{
        System.out.print(df.format(demandMatrix[i][j]));//df.format
        bw.write(df.format(demandMatrix[i][j]));//r.nextDouble()*10
0));
    }
    if(j<VVV-1){
        System.out.print("","");
bw.write("", "");
    }
    if(i<VVV-1){
        System.out.println("\n");bw.newLine();
    }
    for( int i = 0; i < VVV; i++) {
        System.out.print(extractedFileMatrix2[i][4]);
        TW= fileMatrix[i][4];
bw.write(df.format(extractedFileMatrix2[i][4]));
        if(i< VVV-1){
            System.out.print ("","");
bw.write ("", ");
        }
else{
            System.out.print("\n");
bw.newLine();
    }
    System.out.println("\n");bw.newLine();
    System.out.print("TWstart= (");bw.write("[");
    int TW;
    for( int i = 0; i < VVV; i++) {
        System.out.print(extractedFileMatrix2[i][5]);
        bw.write(df.format(extractedFileMatrix2[i][5]));
        if(i< VVV-1){
            System.out.print ("","");
bw.write ("", ");
        }
else{
            System.out.print("\n");
bw.newLine();
    }
    System.out.println("\n");bw.newLine();
    System.out.print("TWend= (");bw.write("[");
    for( int i = 0; i < VVV; i++) {
        System.out.print(extractedFileMatrix2[i][6]);
        bw.write(df.format(extractedFileMatrix2[i][6]));
    }
```

if(i<VVV-1){
    System.out.print(" ", "");
    bw.write(" ", "");
} else{
    System.out.print(" ");
    bw.write(" ");
}
System.out.println("\n"); bw.newLine(); bw.newLine();

// print out x-y coordinates:
System.out.println("Coordinates = "); bw.write(" ");
for(int i=0; i<VVV; i++){
    System.out.println("+extractedFileMatrix2[i][1]+", \"+extractedFileMatrix2[i][2]+\")");
    if(i<VVV-1){
        bw.write(" ");
        bw.newLine();
    } else bw.write(" ");
}
System.out.print(" ");
bw.write(" ");
bw.close();
}
catch (java.io.IOException ex) {
    System.out.println("IO Error: " + ex);
}
}
public static void readFile() throws FileNotFoundException, IOException{
    BufferedReader br = new BufferedReader(new FileReader(readFile));
    Scanner scanner = new Scanner(br); //C:\Users\21576959\Workspace\02 VRPPD test files\02 SINTEF_pdp_100\pdp_100\lc101.txt"));
    try {
        @SuppressWarnings("resource")
        Scanner scanner = new Scanner(new File(readFile)); //C:\Users\21576959\Workspace\02 VRPPD test files\02 SINTEF_pdp_100\pdp_100\lc101.txt"));

        //
        kkk = scanner.nextInt();
        VVV = scanner.nextInt();
        vvv = scanner.nextInt();
        //
        fileMatrix = new int [104][9];
        int line=0;
        // System.out.print(kkk+" ");
        System.out.println(VVV++ " ");
        System.out.println(vvv++ " ");
        scanner.nextLine();
        //
        //for(int i=0; i<VVV; i++){
        //    while(scanner.hasNextLine()){
        //        for(int j=0; j<9; j++){
        //            fileMatrix[line][j] = scanner.nextInt();
        //        System.out.print(fileMatrix[line][j]+" ");
        //    }
        //}
        // System.out.println();
        // line++;
        // scanner.nextLine();
        //}
        //VVV=line;
        kkk = scanner.nextInt();

    } catch (Exception e) {
        System.out.println("Error: " + e);
    }
}
```
VVV = scanner.nextInt();
vvv = scanner.nextInt();

fileMatrix = new int [VVV+1][9];
int line=0;
System.out.print(kkk+ " ");
System.out.print(VVV+ " ");
System.out.println(vvv+ " ");
scanner.nextLine();

//for(int i=0; i<VVV; i++)
while(scanner.hasNextLine()){
    for(int j=0; j<9; j++){
        fileMatrix[line][j] = scanner.nextInt();
        System.out.println(fileMatrix[line][j]+ " ");
    }
    System.out.println();
    line++;
    scanner.nextLine();
}
//VVV=line;

}finally {
    br.close();
}

private static double norm( double x1, double y1, double x2, double y2 ){
    double xDiff = x2 - x1;
    double yDiff = y2 - y1;
    return Math.sqr(xDiff*xDiff + yDiff*yDiff );
}

private static double simulateRevenues(double xD, double yD, double x1, double y1, double x2, double y2) {
    double revenues = 0;
    double directDistance = norm(x1, y1, x2, y2);
    double roundtripDistance = norm(xD, yD, x1, y1)
                        + norm(xD, yD, x2, y2)
                        + directDistance;

    //marketPrice = directDistance + randomGenerator.nextDouble() * (roundtripDistance-directDistance);/** roundtripDistance+1;*/
    revenues = MyDynamicPricing.randomGenerator.nextDouble()*(roundtripDistance-directDistance)/2+directDistance+(roundtripDistance-(roundtripDistance-directDistance)/2));

    return revenues;
}

private static int intValue(Double double1) {
    return double1.intValue();
}
```
E.18. **SinglePositionValues class**

```
import java.util.Arrays;

guard class SinglePositionValues {
    public int[] plw;

    public int[] getPlw() {
        return plw;
    }

    public void setPlw(int[] plw) {
        this.plw = plw;
    }

    public SinglePositionValues(int[] arr1) {
        this.plw = Arrays.copyOf(arr1, arr1.length);
    }
}
```
### E.19. WriteExcel class

```java
//package writer;
import java.io.File;
import java.io.IOException;
import java.util.Locale;
import jxl.CellView;
import jxl.Workbook;
import jxl.WorkbookSettings;
import jxl.format.UnderlineStyle;
import jxl.write.Formula;
import jxl.write.Label;
import jxl.write.Number;
import jxl.write.WritableCellFormat;
import jxl.write.WritableFont;
import jxl.write.WritableSheet;
import jxl.write.WritableWorkbook;
import jxl.write.WriteException;
import jxl.write.biff.RowsExceededException;

public class WriteExcel {

    private WritableCellFormat timesBoldUnderline;
    private static WritableCellFormat times;
    private String inputFile;

    public void setOutputFile(String inputFile) {
        this.inputFile = inputFile;
    }

    public void write() throws IOException, WriteException {
        File file = new File(inputFile);
        WorkbookSettings wbSettings = new WorkbookSettings();
        wbSettings.setLocale(new Locale("en", "EN"));

        WritableWorkbook workbook = Workbook.createWorkbook(file, wbSettings);
        workbook.createSheet("Report", 0);
        WritableSheet excelSheet = workbook.getSheet(0);
        createLabel(excelSheet);
        createContent(excelSheet);
        workbook.write();
        workbook.close();
    }

    private void createLabel(WritableSheet sheet) throws WriteException {
        WritableFont times10pt = new WritableFont(WritableFont.TIMES, 10);
        WritableCellFormat times = new WritableCellFormat(times10pt);
        times.setWrap(false);

        WritableFont times10ptBoldUnderline = new WritableFont(WritableFont.TIMES, 10, WritableFont.BOLD, false,
                UnderlineStyle.SINGLE);
        // Lets automatically wrap the cells
        timesBoldUnderline.setWrap(true);

        CellView cv = new CellView();
    }
```
private void createContent(WritableSheet sheet) throws WriteException, RowsExceededException {
    // Write a few headers
    addCaption(sheet, 0, 0, "Collection");
    addCaption(sheet, 1, 0, "Delivery");
    addCaption(sheet, 2, 0, "Estimated Distance");
    addCaption(sheet, 3, 0, "Estimated Time");
    addCaption(sheet, 4, 0, "Customer");
    addCaption(sheet, 5, 0, "Date Listed");
    addCaption(sheet, 6, 0, "# of Quotes");
    addCaption(sheet, 7, 0, "Loading Location");
    addCaption(sheet, 8, 0, "Listing ID");
    addCaption(sheet, 9, 0, "Expires");
    addCaption(sheet, 10, 0, "Lowest Quote");
    addCaption(sheet, 11, 0, "Unloading Location");

    // Write a few number
    for (int i = 1; i < 10; i++) {
        // First column
        addNumber(sheet, 0, i, i + 10);
        // Second column
        addNumber(sheet, 1, i, 1 * i);
    }

    // Lets calculate the sum of it
    StringBuffer buf = new StringBuffer();
    buf.append("SUM(A2:A10)\n")
    // Formula f = new Formula(0, 10, buf.toString());
    sheet.addCell(f);
    buf = new StringBuffer();
    buf.append("SUM(B2:B10)\n")
    // for(int i = 12; i < 20; i++) {
    // First column
    addLabel(sheet, 0, i, "Boring text " + i);
    // Second column
    addLabel(sheet, 1, i, "Another text");
    
    // for(int i=l; i<(Ausführung.instanzEnde-Ausführung.instanzStart+2); i++){
    // //First column
    
    //addNumberdouble(sheet, 1, i,Ausführung.excelAusgabe[0][i]);
    //addNumberdouble(sheet, 2, i, Ausführung.excelAusgabe[1][i]);
    //addNumberdouble(sheet, 3, i, Ausführung.excelAusgabe[2][i]);
    //addNumberdouble(sheet, 4, i, Ausführung.excelAusgabe[3][i]);
    //addNumberdouble(sheet, 5, i, Ausführung.excelAusgabe[4][i]);
    //addNumberdouble(sheet, 6, i, (Ausführung.excelAusgabe[0][i-1] -
    //addNumberdouble(sheet, 7, i, Ausführung.excelAusgabe[5][i]);
    //}
    
    //addLabel(sheet, 0,1, Test1.liste[0][1][0]);
    //addLabel(sheet, 1,1, Test1.liste[1][1][0]);
    //addLabel(sheet, 2,1, Test1.liste[2][1][0]);
    //addLabel(sheet, 3,1, Test1.liste[3][1][0]);
    //addLabel(sheet, 4,1, Test1.liste[0][1][1]);
    //addLabel(sheet, 5,1, Test1.liste[1][1][1]);
    //addLabel(sheet, 6,1, Test1.liste[2][1][1]);

}
private void addCaption(WritableSheet sheet, int column, int row, String s)
throws RowsExceededException, WriteException {
    Label label;
    label = new Label(column, row, s, timesBoldUnderline);
    sheet.addCell(label);
}

private void addNumber(WritableSheet sheet, int column, int row,
    Integer integer) throws WriteException, RowsExceededException {
    Number number;
    number = new Number(column, row, integer, times);
    sheet.addCell(number);
}

private void addNumberdouble(WritableSheet sheet, int column, int row,
    double doub) throws WriteException, RowsExceededException {
    Number number;
    number = new Number(column, row, doub, times);
    sheet.addCell(number);
}

static void addLabel(WritableSheet sheet, int column, int row, String s)
throws WriteException, RowsExceededException {
    Label label;
    label = new Label(column, row, s, times);
    sheet.addCell(label);
}

public static void main(String[] args) throws WriteException, IOException {
    WriteExcel test = new WriteExcel();
    test.setOutputFile("C:\Users\21576959\Workspace\lars.xls");
    test.write();
    System.out.println("Please check the result file under c:/temp/lars.xls ");
}