<table>
<thead>
<tr>
<th>Title</th>
<th>Studies on local search-based approaches for vehicle routing and scheduling problems (Dissertation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Hashimoto, Hideki</td>
</tr>
<tr>
<td>Citation</td>
<td>Kyoto University</td>
</tr>
<tr>
<td>Issue Date</td>
<td>2008-03-24</td>
</tr>
<tr>
<td>URL</td>
<td><a href="https://doi.org/10.14989/doctor.k13987">https://doi.org/10.14989/doctor.k13987</a></td>
</tr>
<tr>
<td>Type</td>
<td>Thesis or Dissertation</td>
</tr>
<tr>
<td>Textversion</td>
<td>author</td>
</tr>
</tbody>
</table>
STUDIES ON
LOCAL SEARCH-BASED APPROACHES FOR
VEHICLE ROUTING AND SCHEDULING PROBLEMS

HIDEKI HASHIMOTO

DEPARTMENT OF APPLIED MATHEMATICS AND PHYSICS
GRADUATE SCHOOL OF INFORMATICS
KYOTO UNIVERSITY
KYOTO 606-8501, JAPAN

MARCH, 2008
Doctoral Dissertation

submitted to Graduate School of Informatics, Kyoto University

in partial fulfillment of the requirement for the degree of

DOCTOR OF INFORMATICS

(Applied Mathematics and Physics)
Vehicle routing and scheduling are problems concerning the distribution of goods between depots and final users. The standard objective is minimizing the total travel distance of a number of vehicles, under various constraints, where every customer must be visited exactly once by a vehicle. They have been intensively studied since a paper by Dantzig and Ramser appeared in 1959, and there have been hundreds of successful applications in many industries. These application successes have been aided by the growing computer power, the geographic information system (GIS) technology, and so on.

Including vehicle routing and scheduling problems, a variety of combinatorial optimization problems appear in many application fields. It is known to be difficult to obtain exact optimal solutions to them, and the difficulties were proved in the sense of NP-hardness. It is strongly believed that an NP-hard problem cannot be solved in polynomial time of the input size. In other words, solving an NP-hard problem exactly may necessitate enumerating an essential portion of the set of all solutions, whose number increases exponentially as problem size grows. However, in most practical applications, we do not need exact optimal solutions and are satisfied with sufficiently good solutions. In this sense, heuristic algorithms, which provide reasonably good solutions in practical time, have a significant benefit.

There are several representative heuristic algorithms, such as greedy methods and local search. A greedy method directly constructs a solution by successively determining the values of variables on the basis of some local information. This method can find good solutions in very short time in many cases. Local search is the method that improves the current solution iteratively. Although, in general, it is not a polynomial time algorithm, it was reported that near-optimal solutions could typically be obtained in reasonable time. More sophisticated algorithms that utilize the local search in more flexible frameworks such as iterated local search, tabu search, simulated annealing, genetic algorithm and their variants have been studied well, and applied to many NP-hard problems. Such algorithms are generically called metaheuristics.

In this thesis, we describe general models for vehicle routing and scheduling problems
and propose efficient local search-based algorithms for them incorporating mathematical programming techniques. We also propose a high-performance metaheuristic algorithm for a standard vehicle routing and scheduling problem. The aim of the thesis is to propose general models that can include various types of specific variants, and to develop high-performance algorithms.

Vehicle routing and scheduling problems are fundamental issues in human society. As information tools related to vehicle routing and scheduling (e.g., GIS, demand forecasting) have recently been enhanced, an efficient algorithm may immediately make an improvement on these issues. The author hopes that the work contained in this thesis will be helpful to advance the study in this important and interesting field.

March, 2008

Hideki Hashimoto
Acknowledgment

This thesis would not have been possible without the help of many people, whom I would like to acknowledge here.

First of all, I am heartily grateful to Professor Mutsunori Yagiura of Nagoya University for his enthusiastic guidance, discussion and persistent encouragement. He commented in detail on the whole work in the manuscript, which significantly improved the accuracy of the arguments and the quality of the expression. Without his considerable help, none of this work could have been completed.

I am deeply grateful to Professor Hiroshi Nagamochi of Kyoto University. His heartfelt and earnest guidance has encouraged me all the time. Several enthusiastic discussions with him were quite exciting and invaluable experience for me.

I am indebted to Professor Toshihide Ibaraki of Kwansei Gakuin University, Professor Koji Nonobe of Hosei University and Professor Shinji Imahori of University of Tokyo for numbers of helpful comments and suggestions.

I would like to express my sincere appreciation to Professor Michel Vasquez of Ecole des mines d’Alèes, who guided me in research and gave me many valuable opportunities during my stay at his laboratory. I also appreciate the friendship of members of his laboratory including Mr. Sylvain Boussier and Dr. Saber Aloui.

I wish to express my gratitude to Professor Liang Zhao of Kyoto University, Professor Shunji Umetani of Osaka University, Professor Kazuya Haraguchi of Ishinomaki Senshu University, Professor Takuro Fukunaga of Kyoto University, and all members in Professor Nagamochi’s laboratory for many enlightening discussions on the area of this work and their warm friendship in the period I studied at Kyoto University.

I am also thankful to Professor Masao Fukushima of Kyoto University and Professor Yoshito Ohta of Kyoto University for serving on my dissertation committee.

Finally, but not least, I would like to express my heartiest gratitude to my family for their heartfelt cooperation and encouragement.
# Contents

1 Introduction

1.1 Background ........................................................................................................... 1
1.2 Complexity of the problems .................................................................................. 2
1.3 Local search .......................................................................................................... 4
1.3.1 Overview of local search .................................................................................. 6
1.3.2 Neighborhood search ..................................................................................... 7
1.3.3 Metaheuristics .................................................................................................. 8
1.4 Overview of the thesis ............................................................................................ 10

2 Vehicle Routing and Scheduling Problem

2.1 Introduction .......................................................................................................... 13
2.2 The vehicle routing problem with time windows .................................................. 13
2.3 Construction algorithm ......................................................................................... 15
2.3.1 Insertion heuristic .......................................................................................... 15
2.3.2 Savings heuristic ............................................................................................ 16
2.3.3 Fisher and Jaikumar algorithm ....................................................................... 16
2.4 Local search .......................................................................................................... 17
2.4.1 $\lambda$-opt neighborhood .................................................................................. 17
2.4.2 2-opt* neighborhood ....................................................................................... 18
2.4.3 $\lambda$-interchange neighborhood ..................................................................... 18
2.4.4 Cross exchange neighborhood ....................................................................... 18
2.4.5 Or-opt neighborhood ....................................................................................... 19
2.4.6 Cyclic transfer .................................................................................................. 20
2.5 General model ....................................................................................................... 20
2.6 Efficient neighborhood search method ................................................................. 24
2.6.1 Basic idea ....................................................................................................... 24
2.6.2 How to apply the basic idea to evaluate solutions in neighborhoods .......... 25
## Contents

3 The Vehicle Routing Problem with Flexible Time Windows and Traveling Times 35

3.1 Introduction .................................................. 35
3.2 Problem definition ........................................... 36
3.3 Optimal start times of services .............................. 38
   3.3.1 NP-hardness ............................................. 38
   3.3.2 Pseudo polynomial time algorithm ....................... 40
   3.3.3 Polynomial time algorithm for convex traveling time cost functions 42
3.4 Local search for finding visiting orders $\sigma$ ............... 51
3.5 Efficient implementation of local search .................... 52
   3.5.1 Basic idea ............................................. 52
   3.5.2 How to apply the basic idea to the solutions in neighborhoods .... 53
3.6 Computational results ........................................ 55
3.7 Conclusion .................................................... 60

4 The Time-Dependent Vehicle Routing Problem with Time Windows 61

4.1 Introduction .................................................. 61
4.2 Problem definition ........................................... 62
4.3 Optimal start time problem ................................... 65
   4.3.1 Dynamic programming .................................. 66
   4.3.2 Algorithm and time complexity ......................... 66
   4.3.3 Remarks for the case in which condition (4.2.1) does not hold .... 69
   4.3.4 Historical notes ........................................ 70
4.4 Local search for finding visiting orders $\sigma$ ............... 71
4.5 Efficient implementation of local search .................... 73
   4.5.1 Basic idea ............................................. 73
   4.5.2 How to apply the basic idea to the solutions in neighborhoods .... 74
   4.5.3 Restriction of neighborhoods ........................... 76
4.6 Computational results ........................................ 79
   4.6.1 Effect of the restriction of the neighborhoods .......... 79
   4.6.2 The vehicle routing problem with hard time windows ........... 81
   4.6.3 Time-dependent VRPSTW ............................... 83
4.7 Conclusion .................................................... 93

5 Path Relinking Approach with an Adaptive Mechanism to Control Parameters for the Vehicle Routing Problem with Time Windows 95

5.1 Introduction .................................................. 95
5.2 Problem definition ................................................. 96
5.3 Local search ...................................................... 97
  5.3.1 Neighbor list .................................................. 98
  5.3.2 Neighborhoods ............................................... 98
5.4 Evaluation function $p(\sigma_k)$ .................................. 99
5.5 Adaptive mechanism to control parameters ....................... 101
  5.5.1 Update of the parameters $\alpha$ and $\beta$ .................. 101
  5.5.2 Update of the parameters $\gamma_i$ for each customer $i$ .... 102
5.6 Path relinking approach ........................................... 102
  5.6.1 Reference set ................................................ 102
  5.6.2 Path relinking operation .................................... 102
5.7 Computational experiments ........................................ 103
5.8 Conclusion ......................................................... 106

6 Conclusion .......................................................... 109
# List of Figures

1.1 An illustration of local search ........................................ 5

2.1 A 2-opt* neighborhood operation .................................... 18
2.2 A cross exchange neighborhood operation .......................... 19
2.3 An Or-opt neighborhood operation .................................... 19
2.4 An illustration of the search method for the 2-opt* neighborhood . 25
2.5 An illustration of the search method for the cross exchange neighborhood . 26
2.6 An illustration of the search method for the forward reverse intra neighborhood ......................................................... 29
2.7 An illustration of the search method for the forward normal intra neighborhood ................................................................. 29
2.8 An illustration of the search method for the backward reverse intra neighborhood ............................................................... 30
2.9 An illustration of the search method for the backward normal intra neighborhood ............................................................... 30

3.1 A function $g$ and the linked list that represents $g$ ................ 43
3.2 Breakpoints of $\phi_1(x_1)$ and $\phi_2(x_2)$ on the plane of $x_1$ and $x_2$ .... 45
3.3 The points that achieves $\phi(t)$ and $\phi(t + \varepsilon)$ ....................... 45
3.4 An example of the trajectory of $(x_1^1(t), x_2^1(t))$ which achieves $\phi(t)$ ........................................................................ 46
3.5 $e_l$ and $e_r$ ........................................................................ 47
3.6 An example of the lower envelope ........................................... 48
3.7 Neighborhoods in our local search ........................................ 51
3.8 The former and latter parts of a route $\sigma_k$ ............................... 52
3.9 An example of a 2-opt* operation .......................................... 54
3.10 An example of the search order in the cross exchange neighborhood ... 54

4.1 An example of $\lambda_{ij}$ which satisfies condition (4.2.1), and a function $\bar{\lambda}$ which does not satisfy condition (4.2.1) .................. 63
4.2 A function $g$ and the linked list that represents $g$ ........................................ 67
4.3 The relationship between the departure time $t_{h-1}$ and the arriving time
   $s = t_{h-1} + \lambda_{h-1,h}(t_{h-1})$ ................................................................. 68
4.4 An example of $\lambda_{ij}$ which does not satisfy condition (4.2.1) ...................... 70
4.5 Neighborhoods in our local search ............................................................... 72
4.6 The former and latter parts of a route $\sigma_k$ ............................................... 73
4.7 An example of a 2-opt* operation ............................................................... 75
4.8 An example of the search order in the cross exchange neighborhood .............. 75
4.9 The distribution of two route cost in the 2-opt* neighborhood without (left)
   and with (right) restriction ................................................................. 79
4.10 The distribution of two route cost in the cross exchange neighborhood without
   (left) and with (right) restriction ............................................................. 80
5.1 Neighborhoods in our local search ............................................................ 99
## List of Tables

3.1 The best known solutions for Solomon’s instances ........................................ 57
3.2 Computational results on Solomon’s instances ............................................. 58
3.3 Computational results with smaller number of vehicles than the best known solutions ................................................................. 59

4.1 Number of evaluations with and without restriction of neighborhood ........ 80
4.2 The results for 100-customer benchmark instances ..................................... 84
4.3 The results for 200-customer benchmark instances ..................................... 84
4.4 The results for 400-customer benchmark instances ..................................... 85
4.5 The results for 600-customer benchmark instances ..................................... 85
4.6 The results for 800-customer benchmark instances ..................................... 86
4.7 The results for 1000-customer benchmark instances ................................... 86
4.8 The detailed results for 100-customer benchmark instances ....................... 87
4.9 The detailed results for 200-customer benchmark instances ....................... 88
4.10 The detailed results for 400-customer benchmark instances ....................... 89
4.11 The detailed results for 600-customer benchmark instances ....................... 90
4.12 The detailed results for 800-customer benchmark instances ....................... 91
4.13 The detailed results for 1000-customer benchmark instances ..................... 92
4.14 Travel speed matrices for scenarios 1–3 ....................................................... 93
4.15 The results for time-dependent VRPSTW .................................................. 94

5.1 Comparison of our results with the existing methods for benchmark instances105
5.2 The detailed results of our algorithm for the 100–400-customer instances .... 107
5.3 The detailed results of our algorithm for the 600–1000-customer instances .. 108
## List of Algorithms

1. Standard local search (LS) .................................................. 5
2. Iterated local search (ILS) .................................................. 9
3. Tabu search (TS) ............................................................... 9
4. Simulated annealing (SA) .................................................. 9
5. Genetic algorithm (GA) ....................................................... 10
6. Adaptive memory programming (AMP) .......................... 10
7. Insertion heuristic ............................................................ 16
8. Savings heuristic ............................................................... 16
9. Fisher and Jaikumar heuristic ......................................... 17
10. The search method for the 2-opt* neighborhood ................ 26
11. The search method for cross exchange neighborhood ............ 27
12. The search method for the forward reverse intra neighborhood ... 28
13. The search method for the forward normal intra neighborhood ... 31
14. The search method for the backward reverse intra neighborhood ... 32
15. The search method for the backward normal intra neighborhood ... 33
16. Path relinking approach for the vehicle routing problem with time windows 104
Chapter 1

Introduction

1.1 Background

Vehicle routing and scheduling problems have been intensively studied since a paper by Dantzig and Ramser [36] appeared in 1959. They are interesting from both theoretical and practical point of view, and they attract academic and industrial people in a wide range of fields.

Vehicle routing and scheduling are problems concerning the distribution of goods between depots and final users (customers) [41, 42, 141, 149]. The standard objective is minimizing the total travel distance of a number of vehicles, under various constraints, where every customer must be visited exactly once by a vehicle. Among a number of variants of vehicle routing and scheduling problems, the capacitated vehicle routing problem (CVRP) [49], the vehicle routing problem with time windows (VRPTW) [97, 125, 133, 142], the vehicle routing problem with backhauls (VRPB) [150] and the vehicle routing problem with pickup and delivery (VRPPD) [39, 130] are classic, and the vehicle routing problem with time windows is one of the problems most intensively studied recently. In the past four decades, a variety of approaches have been applied and quite a number of exact and heuristic algorithms have been proposed. See the bibliographies by Laporte [100] and by Laporte and Osman [101]. The latter bibliography contains 500 references.

Many applications of vehicle routing and scheduling problems are described in the practical problems: soft-drink distribution [128], oil industry [55], bulk sugar delivery [153], brewing industry [46], food distribution [28], transportation of live animals [115, 137] and so on. See the survey by Golden, Assad and Wasil [69] for more real-world applications. The savings achievable by solving these problems have a significant impact on the global economic system. Indeed, according to Toth and Vigo [148], in the large number of real-world applications both in North America and in Europe, the use of computerized procedures for
the distribution process planning produces substantial savings generally from 5% to 20% in the global transportation costs. They also mentioned that the transportation process involves all stages of the production and distribution systems, and a relevant component amounts to generally from 10% to 20% of the final cost of the goods.

1.2 Complexity of the problems

A variety of combinatorial optimization problems appear in many application fields. They were known to be difficult to obtain an exact optimal solution and the difficulties were proved in the sense of NP-hardness, which was the notion proposed around 1970. In the late 1960s, the fundamental nature of algorithms was discussed; Edmonds [45] called an algorithm which runs in polynomial time of the input size a “good” algorithm. In the 1970s, Cook [31] first proved that SAT is an NP-complete problem, and in the subsequent years, the foundations for the theory of NP-completeness were established [53]. A problem is called \textit{NP-hard} when it is at least as hard as NP-complete problems. Nowadays many problems are proved to be NP-hard [13,33,53,76]. It is strongly believed that an NP-hard problem cannot be solved in polynomial time of the input size. In other words, solving an NP-hard problem exactly may necessitate enumerating an essential portion of the set of all solutions, whose number increases exponentially as problem size grows.

A solution of vehicle routing and scheduling problems basically consists of

\textbf{(Assignment)} the assignment of customers to vehicles,
\textbf{(Routing)} the visiting order of customers who are assigned to a vehicle, and
\textbf{(Scheduling)} the scheduling of service times of customers.

They usually obey some conditions; for example, the followings are typical.

- Each customer has a demand and each vehicle has a capacity, and the total load on a vehicle route cannot exceed the capacity of the assigned vehicle.

- Each travel between customers takes a cost, and the total traveling cost of all vehicles should be minimized.

- Each vehicle must start the service at each customer in the period specified by the customer.

Under these conditions, the above three components of a solution (i.e., assignment, routing and scheduling) are difficult to be determined even if some other components are fixed.
1.2 Complexity of the problems

Indeed, the following three NP-hard problems are such cases: bin packing problem (assignment), traveling salesman problem (routing) and sequencing with release times and deadlines (scheduling). They are formulated as follows.

**Bin packing problem**

**input:** A set $I$ of items, a size $a(i) \in \mathbb{Z}^+$ for each $i \in I$ and the bin capacity $b \in \mathbb{Z}^+$.

**output:** A partition of $I$ into the minimum number $k$ of disjoint subsets $I_1, I_2, \ldots, I_k$ such that the total size $\sum_{i \in I_j} a(i)$ is $b$ or less for each subset $I_j$.

**Traveling salesman problem**

**input:** A complete directed graph $G = (V, E)$ and a cost function $c : E \to \mathbb{R}^+$ ($\mathbb{R}^+$ is the set of all nonnegative real numbers).

**output:** A minimum cost tour of $G$, i.e., a directed simple cycle of $|V|$ vertices with minimum total cost.

**Sequencing with release times and deadlines**

**input:** A set $T$ of tasks and, for each task $t \in T$, a length $l(t) \in \mathbb{Z}^+$, a release time $r(t) \in \mathbb{Z}_0^+$, and a deadline $d(t) \in \mathbb{Z}^+$.

**output:** “Yes” if there is a one-processor schedule for $T$ that satisfies the release time constraints and meets all the deadlines; otherwise “No”.

All the three problems (i.e., bin packing, traveling salesman, sequencing with release times and deadlines) are known to be NP-hard in the strong sense and no pseudo-polynomial time algorithm exists unless P = NP [11, 53]. Furthermore, for the traveling salesman problem, no polynomial time algorithm guarantees the solution quality bounded by a constant times the optimal value unless P = NP [132]. Note however that, such algorithms were proposed for spacial cases of the traveling salesman problem, which are still NP-hard: for the metric traveling salesman problem by Christofides [27, 154] and for the Euclidean traveling salesman problem by Arora [11, 154].

In spite of the theoretical intractability, it may be possible to solve an NP-hard problem efficiently in the practical sense, since the NP-hardness is based on the worst case complexity. Representative methods frequently applied to this end are *branch-and-bound*, *branch-and-cut* and *dynamic programming*. Branch-and-bound and dynamic programming are methods that enumerate only promising solutions efficiently [14, 83, 84]. Although
branch-and-cut [111] is a method for the integer linear programming problem, most of
combinatorial problems can naturally be formulated as an integer linear program, which
has a mature theory [113,135]. With intensive studies on these exact algorithms and as a
result of the rapid progress of computer technology, the problem size that can be exactly
solved has been increasing. However, it is still not large enough to accommodate all the
problems arising in real applications.

Fortunately, in vehicle routing and scheduling problems as well as most applications,
we are satisfied with good solutions obtained in reasonable computation time even if we
are not able to obtain an exact optimal solution. In this thesis, we focus on heuristic
algorithms. One of the well known heuristic algorithms is the greedy method. The greedy
method directly constructs a solution by successively determining the values of variables
on the basis of some local information. This method can find optimal solutions for some
problems, which are called matroid [156], or find good solutions in many cases for other
problems in very short time. Another typical heuristic is local search. Local search is the
method that improves the current solution iteratively. Although, in general, it is not a
polynomial time algorithm, it was reported that near-optimal solutions could typically be
obtained in reasonable time (see, for example, Johnson and McGeoch [88] for the traveling
salesman problem). When more quality is needed and more computation time is available,
metaheuristics is often very effective. We will discuss local search and metaheuristics in the
next section. Among heuristic algorithms, an algorithm which guarantees the quality of
the output solution by a function of the optimal value is especially called an approximation
algorithm. Approximation algorithms have been investigated in the past two decades and
the theory of hardness of approximation have been developed [13,76,154].

1.3 Local search

In this section, we review local search in combinatorial optimization.

Local search is a universal method that starts from an initial solution and repeatedly
replaces it with a better solution in its neighborhood [2,121,158,159]. Neighborhood of a
solution is a set of solutions obtainable from the solution by applying a slight perturba-
tion. Local search terminates when it reaches a solution having no better solution in the
neighborhood. Such a solution is called locally optimal. Figure 1.1 illustrates the process
of local search. In the figure, each circle represents the neighborhood of a solution denoted
by a dot in its center and an arrow represents a move; i.e., an act of replacing solutions.
The local search has widely been used in combinatorial optimization since it provides a
robust approach to obtain good solutions. In the late 1950s and early 1960s, the first edge-
exchange algorithms for the traveling salesman problem were introduced by Croes [34],
1.3 Local search

An initial solution

Figure 1.1: An illustration of local search

Algorithm 1 Standard local search (LS)

1: Set $x := x^{(0)}$.

2: If there is a feasible solution $x' \in N(x)$ such that $f(x') < f(x)$ holds, set $x := x'$ and return to Step 2. Otherwise (i.e., $f(x) \leq f(x')$ holds for all solutions $x' \in N(x)$), output the current locally optimal solution $x$ and stop.

Lin [105] and Reister and Sherman [129]. In the subsequent years, it was also applied to scheduling problems [114, 120] and the graph partitioning problem [92]. Until now, local search algorithms have been proposed for a variety of hard optimization problems (e.g., assignment problems, packing problems, covering problems, routing problems, and so on) and have accomplished successful results in computational experiments. Furthermore, in the past few decades, some theoretical results have been developed. See the annotated bibliography by Aarts and Verhoeven [1].

An instance of a combinatorial optimization problem is a pair $(S, f)$, where the solution set $S$ is the set of feasible solutions and the cost function $f$ is a mapping $f : S \rightarrow \mathbb{R}$. A neighborhood function is a mapping $N : S \rightarrow 2^S$, which defines for each solution $x \in S$ a set $N(x) \subseteq S$ of solutions that are in some sense close to $x$. The set $N(x)$ is the neighborhood of solution $x$. A solution $x$ is locally optimal (minimal) with respect to $N$ if $f(x) \leq f(x')$ for all $x' \in N(x)$. The local search problem is the problem of finding a locally optimal solution.

The standard local search algorithm with an initial solution $x^{(0)}$, neighborhood $N(x)$ and the objective function $f(x)$ is formally described as Algorithm 1. The search procedure of finding the next solution $x'$ in Step 2 is called the neighborhood search, and the set of all solutions which may be potentially visited in a local search algorithm is called the search space. We will discuss the neighborhood search in Section 1.3.2.
1.3.1 Overview of local search

Several thousands of papers about local search have been published over the past four decades. Here we review the empirical and theoretical results of local search-based methods (i.e., it may not a result just by the standard local search algorithm) among them.

Empirical results indicate local search provides a robust approach to obtain good solutions. For the traveling salesman problem, Johnson and McGeoch [88] reported that a local optimization with the 3-opt neighborhood typically obtained solutions within 3–4% of optimal and the local search of Lin and Kernighan [106] typically obtained solutions within 1–2% for random Euclidean instances from 100 to 1,000,000 cities. Moreover, they also reported that the growth rates of their running times appeared to be subquadratic. Further successful results of local search for many problems were reported: for example, job shop scheduling by Vaessens, Aarts and Lenstra [151], vehicle routing by Gendreau, Laporte and Potvin [59] and by Kindervater and Savelsbergh [94], machine scheduling by Anderson, Glass and Potts [8], VLSI layout synthesis by Aarts et al. [3], and code design by Honkala and Östergård [81].

In spite of the very good empirical results, the standard local search algorithm is not a polynomial time algorithm even if the neighborhood search can be executed in polynomial time, because the number of improvement can be exponential. An illustrative example is the simplex algorithm for linear programming [29, 35], which can be considered as a local search algorithm. Klee and Minty [95] showed an example that the simplex algorithm takes exponential steps. Note that the complexities of a local search algorithm and a local search problem are different; e.g., though the simplex algorithm can take exponential steps, the linear programming problem can be solved in polynomial time (e.g., ellipsoid method [135], interior-point method [90], Vaidya’s algorithm [152]). Approximation in some senses may be a key to efficient algorithms for problems which have an intractable complexity. Orlin, Punnen and Schulz [117] proposed an approximate local search algorithm framework. They introduced the concept of $\epsilon$-local optimality and showed that, for every $\epsilon > 0$, an $\epsilon$-local optimum can be identified in time polynomial in the problem size and $1/\epsilon$ whenever the corresponding neighborhood can be searched in polynomial time.

Though local search usually cannot guarantee the solution quality, approximation algorithms that rely on local search have been proposed recently: facility location problems by Arya et al [12], degree-bounded minimum spanning trees by Könemann and Ravi [96], minimum vertex feedback edge set problem by Khuller, Bhatia, and Pless [93], weighted $k$-set packing problem by Arkin and Hassin [10], $k$-set cover problem by Halldórsson [72]. See the survey by Angel [9] for further results.

In the past few decades, the theory of complexity for a local search problem have
been developed. In 1988, Johnson, Papadimitriou and Yannakakis [89] introduced the complexity class \( PLS \) (for polynomial time local search) to formalize the question how easy it is to find a local optimum. A combinatorial optimization problem together with a given neighborhood function \( \mathcal{N} \) belongs to PLS if

1. for a given instance, all solutions are polynomial time recognizable and a feasible solution is computable in polynomial time,

2. for a given solution, it is decided in polynomial time whether it is locally optimal, and if not, a better neighborhood solution is computable in polynomial time.

All common local search problems are in PLS. It has been shown that a problem in PLS cannot be NP-hard unless \( \text{NP} = \text{co-NP} \) [89]. Furthermore, the concept of a PLS-reduction has been introduced and a problem in PLS is PLS-complete if any problem in PLS is PLS-reducible to it. The PLS-complete problems are the hardest ones in PLS and if one of them is shown to be solvable in polynomial time, then all the others are. The following problems are PLS-complete: graph partitioning under the swap neighborhood [134], traveling salesman problem under the \( k \)-opt neighborhood for some constant \( k \) [98], MAXCUT under the flip neighborhood [134] and MAX-2-SAT under the flip neighborhood [99]. Johnson, Papadimitriou and Yannakakis [89] showed that it is NP-hard to determine the output of the Kernighan-Lin algorithm [92] on an arbitrary instance of the graph partitioning problem, and there are instances for which it takes exponentially many iterations. Krentel [99] also showed that it is NP-hard to determine the output of the standard local search with the flip neighborhood on an arbitrary instance of the MAX-SAT problem, and there are instances for which it takes exponentially many iterations. See the extensive survey by Yannakakis [160].

1.3.2 Neighborhood search

Local search usually spends most of its computation time to search the neighborhood. Hence it is crucial to search the neighborhood efficiently in order to find a good solution in short time and handle large-scale instances.

The definition of neighborhood is an essential part in designing a local search algorithm. A locally optimal solution with a larger neighborhood is usually more accurate than that with a smaller neighborhood. On the other hand, the neighborhood search with a larger neighborhood often takes more computation time. Standard local search algorithms exhaustively search neighborhoods whose sizes are sufficiently small. In contrast, algorithms with a neighborhood whose size can be exponential are called \textit{very large-scale neighborhood search} [4,5,47,48]. Instead of searching the neighborhood exhaustively, these
algorithms solve the problem of finding a better solution in the neighborhood by using more sophisticated techniques such as heuristic methods, dynamic programming methods and network flow algorithms.

As for move strategies in local search, the first admissible move strategy and the best admissible move strategy are usually used. In the first admissible move strategy, solutions in \( N(x) \) are scanned according to a prespecified order, and the first improved solution is immediately accepted as the next solution. In the best admissible move strategy, the best solution in \( N(x) \) is chosen as the next solution. Move strategies which accept a nonimproved solution are also allowed in the metaheuristic context (e.g., simulated annealing and tabu search). By allowing such moves, they enable the local search to continue the search from locally optimal solutions and examine a wider area of solutions around them.

In addition to the definition of neighborhood and move strategy, data structure [32] plays an important role to search neighborhoods efficiently. For example, an efficient implementation of local search algorithms for the traveling salesman problem may need some sophisticated data structure (e.g., segment-tree [51], \( k \)-d tree [17], two-level tree [51], splay tree [139]).

1.3.3 Metaheuristics

Metaheuristics [20, 43, 65, 70, 82, 155, 157] first appeared in the 1980s (the term metaheuristics was coined by Glover [61] in 1986) and many metaheuristics algorithms that provide a near-optimal solution have been revealed. See the bibliography by Osman and Laporte [119], which provides a classification of a comprehensive list of 1380 references on the theory and application of metaheuristics. Gendreau and Potvin [60] provide an account of the most recent developments. In this section, we briefly review some of representative metaheuristics.

The iterated local search (ILS) [107] iterates local search many times from those initial solutions generated by slightly perturbing a good solution \( x_{seed} \) obtained so far. It is important to generate initial solutions that retain some features of solution \( x_{seed} \) and to avoid a cycling of solutions in order to improve the performance of ILS. In Algorithm 2, we describe an iterated local search algorithm which uses the best obtained solution \( x^* \) as \( x_{seed} \).

The tabu search (TS) tries to enhance local search by using the memory of previous searches. TS repeatedly replaces the current solution \( x \) with its best neighbor \( x' \in N(x) \setminus (\{x\} \cup T) \) even if \( f(x') \geq f(x) \) holds, where the set \( T \), called the tabu list, is a set of solutions which includes those solutions most recently visited. Cycling of a short period can be avoided as a result of introducing tabu list. See reference [62, 63, 66] for detailed
Algorithm 2 Iterated local search (ILS)
1: Initialize $x^*$ to be an arbitrary solution.
2: Generate a solution $x$ by slightly perturbing $x^*$
3: Improve $x$ by local search.
4: If $f(x) \leq f(x^*)$ holds, set $x^* := x$. If some stopping criterion is satisfied, output $x^*$ and halt; otherwise return to Step 2.

Algorithm 3 Tabu search (TS)
1: Generate an initial solution $x$.
2: Set $x^* := x$ and $T := \emptyset$.
3: Find the best solution $x' \in N(x) \setminus (\{x\} \cup T)$, and set $x := x'$.
4: If $f(x) < f(x^*)$ holds, set $x^* := x$. If some stopping criterion is satisfied, output the best obtained solution $x^*$ and halt; otherwise update $T$ according to some rule and return to Step 3.

The simulated annealing (SA) is a kind of probabilistic local search, in which test solutions are randomly chosen from $N(x)$ and accepted with probability that is 1 if the test solution is better than the current solution $x$, and is positive even if it is worse than $x$. The acceptance probability of moves is controlled by a parameter called temperature, whose idea stems from the physical process of annealing. SA is described as Algorithm 4. One of the simplest rules is the geometric cooling, where the temperature is updated by $t := \alpha t$ ($0 < \alpha < 1$ is a parameter) at intervals of the prespecified iterations.

The genetic algorithm (GA) [78] is inspired by the evolutionary process in nature. GA repeatedly generates a set of new solutions $Q$ by applying the operations crossover and/or mutation to the set of current solutions $P$. A crossover generates one or more new solutions by combining two or more current solutions, and a mutation generates a new solution by

Algorithm 4 Simulated annealing (SA)
1: Generate an initial solution $x$ randomly and set $x^* := x$. Determine the initial temperature $t$.
2: Generate a solution $x' \in N(x)$ randomly, and set $\Delta := f(x') - f(x)$. If $\Delta < 0$ holds (i.e., a better solution is found), set $x := x'$; otherwise set $x := x'$ with probability $e^{-\Delta/t}$.
3: If $f(x) < f(x^*)$ holds, set $x^* := x$. If some stopping criterion is satisfied, output $x^*$ and halt; otherwise update the temperature $t$ according to some rule and return to Step 2.
### Algorithm 5 Genetic algorithm (GA)

1: Generate an initial set of solutions $P$ and let $x^*$ be the best solution among $P$.
2: repeat
3: Choose two or more solutions from $P$, crossover them to generate one or more new solutions and add the generated solutions to $Q$.
4: Choose a solution from $P \cup Q$, mutate it to generate a new solution and add the generated solutions to $Q$.
5: until The set of new solutions $Q$ are obtained where the cardinality of $Q$ is prespecified
6: If there is a solution $x \in Q$ with $f(x) < f(x^*)$, choose a best solution $x \in Q$ and set $x^* := x$.
7: Select a set of solutions $P'$ (of a prespecified size) from the resulting $P \cup Q$, and set $P := P'$.
8: If some stopping criterion is satisfied, output the best obtained solution $x^*$ and halt; otherwise return to Step 2.

### Algorithm 6 Adaptive memory programming (AMP)

1: Initialize the memory.
2: while A stopping criterion is not met do
3: Generate a new provisional solution $s$ using data stored in the memory.
4: Improve $s$ by a local search.
5: Update the memory using the pieces of knowledge brought by $s$.
6: end while

slightly perturbing a current solution. GA starts from an initial set of solutions $P$ and repeatedly replaces $P$ with $P' \subseteq P \cup Q$ according to its selection rule. GA is described as Algorithm 5. In Step 7 of Algorithm 5, the following strategies to make a new set of solutions $P'$ are often used: random selection, roulette wheel selection, and elitism.

The adaptive memory programming (AMP) [64, 143] is a general framework which includes a number of metaheuristics (e.g., GA, TS). AMP is described as Algorithm 6.

### 1.4 Overview of the thesis

The thesis is organized as follows.

In Chapter 2, we explain several basic techniques for solving the standard vehicle routing and scheduling problems. We propose a general formulation which includes the standard vehicle routing and scheduling problems and then we propose an efficient neighborhood search method for the standard neighborhoods called 2-opt*, cross exchange and
Or-opt. The neighborhood search method is incorporated in the algorithms of the following chapters.

In Chapter 3, we describe a generalization of the standard vehicle routing problem by allowing soft time window and soft traveling time constraints, where both constraints can be violated and the amounts of violation are penalized by cost functions. With the proposed generalization, the problem becomes very general. In the algorithm, we use the neighborhood search method which is described in Chapter 2. In order to apply the framework, we need a dynamic programming algorithm for the problem of determining the optimal start times of services at visited customers after fixing the route of each vehicle. We show that this subproblem is NP-hard when cost functions are general, but can be efficiently solved with dynamic programming when traveling time cost functions are convex even if time window cost functions are non-convex. We deal with the latter situation in the developed iterated local search algorithm. The computational results on benchmark instances confirm the benefits of the proposed generalization.

In Chapter 4, we concern another generalization of the standard vehicle routing problem with time windows by allowing both traveling times and traveling costs to be time-dependent functions. In the algorithm, we also use the neighborhood search method. We show that the subproblem of asking an optimal time schedule of a route can be efficiently solved by dynamic programming, which is incorporated in the local search algorithm. We further propose a filtering method that restricts the search space in the neighborhoods to avoid many solutions having no prospect of improvement. The computational results of our iterated local search algorithm compared against existing methods confirm the effectiveness of the restriction of the neighborhoods and the benefits of the proposed generalization.

In Chapter 5, we describe a path relinking approach for the vehicle routing problem with time windows. The path relinking is an evolutionary mechanism that generates new solutions by combining two or more reference solutions. In our algorithm, those solutions generated by path relinking operations are improved by a local search. To make the search more efficient, we propose a neighbor list that prunes the neighborhood search heuristically. Infeasible solutions are allowed to be visited during the search, while the amount of violation is penalized. As the performance of the algorithm crucially depends on penalty weights that specify how such penalty is emphasized, we propose an adaptive mechanism to control the penalty weights. The computational results on well-studied benchmark instances with up to 1000 customers revealed that our algorithm is highly efficient especially for large instances. Moreover, it updated 41 best known solutions among 356 instances.

Finally, in Chapter 6, we summarize our study in this thesis.
Chapter 2

Vehicle Routing and Scheduling Problem

2.1 Introduction

In this chapter, we first formulate the vehicle routing problem with time windows. The problem is the most common and best-studied among a number of vehicle routing and scheduling problems and it has been a subject of intensive research focused mainly on heuristic and metaheuristics approaches [23, 24]. We briefly review classic heuristic approaches. We then formulate a general model, which includes all problems considered in this thesis. Finally we propose an efficient neighborhood search method for the general model. The proposed search method can be applied to the standard neighborhood called 2-opt*, cross exchange and Or-opt neighborhoods and can evaluate these neighborhood solutions efficiently by utilizing the information from the past dynamic programming recursion used to evaluate the current solution.

2.2 The vehicle routing problem with time windows

In this section, we formulate the standard vehicle routing problem with time windows.

Let $G = (V, E)$ be a complete directed graph with vertex set $V = \{0, 1, \ldots, n\}$ and edge set $E = \{(i, j) \mid i, j \in V, i \neq j\}$, and $M = \{1, 2, \ldots, m\}$ be a vehicle set. In this graph, vertex 0 is the depot and other vertices are customers. Each customer $i$ and each edge $(i, j) \in E$ are associated with:

1. a fixed quantity $a_i \ (\geq 0)$ of goods to be delivered to $i$,
2. a time window $[e_i, l_i]$,
3. a traveling time $t_{ij}(\geq 0)$ and a traveling distance $c_{ij}(\geq 0)$ from $i$ to $j$.

We assume $a_0 = 0$ and $e_0 = 0$ without loss of generality. Each vehicle has an identical capacity $u$.

Let $\sigma_k$ denote the route traveled by vehicle $k$, where $\sigma_k(h)$ denotes the $h$th customer in $\sigma_k$, and let

$$\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m).$$

Note that each customer $i$ is included in exactly one route $\sigma_k$, and is visited by vehicle $k$ exactly once. We denote by $n_k$ the number of customers in $\sigma_k$. For convenience, we define $\sigma_k(0) = 0$ and $\sigma_k(n_k + 1) = 0$ for all $k$ (i.e., each vehicle $k \in M$ departs from the depot and comes back to the depot). Moreover, let $s_i$ be the start time of service at customer $i$ (by exactly one of the vehicles) and $s^k_i$ be the arrival time of vehicle $k$ at the depot. Note that each vehicle is allowed to wait at customers before starting services.

Let us introduce 0-1 variables $y_{ik}(\sigma) \in \{0, 1\}$ for $i \in V \setminus \{0\}$ and $k \in M$ by

$$y_{ik}(\sigma) = 1 \iff i = \sigma_k(h) \text{ holds for exactly one } h \in \{1, 2, \ldots, n_k\}.$$ 

That is, $y_{ik}(\sigma) = 1$ holds if and only if vehicle $k$ visits customer $i$. The traveling distance of a vehicle $k$ is expressed as $d(\sigma_k) = \sum_{h=0}^{n_k} c_{\sigma_k(h), \sigma_k(h+1)}$. Then the vehicle routing problem with time windows is formulated as follows:

$$\text{minimize} \quad \sum_{k \in M} d(\sigma_k) \quad (2.2.1)$$

subject to

$$\sum_{k \in M} y_{ik}(\sigma) = 1, \quad i \in V \setminus \{0\} \quad (2.2.2)$$

$$\sum_{i \in V \setminus \{0\}} a_i y_{ik}(\sigma) \leq u, \quad k \in M \quad (2.2.3)$$

$$t_{0,\sigma_k(1)} \leq s_{\sigma_k(1)}, \quad k \in M \quad (2.2.4)$$

$$s_{\sigma_k(i)} + t_{\sigma_k(i), \sigma_k(i+1)} \leq s_{\sigma_k(i+1)}, \quad 1 \leq i \leq n_k - 1, \quad k \in M \quad (2.2.5)$$

$$s_{\sigma_k(n_k)} + t_{\sigma_k(n_k), 0} \leq s^k_0 \leq l_0, \quad k \in M \quad (2.2.6)$$

$$e_i \leq s_i \leq l_i, \quad i \in V \setminus \{0\} \quad (2.2.7)$$

$$y_{ik}(\sigma) \in \{0, 1\}, \quad i \in V \setminus \{0\}, \quad k \in M. \quad (2.2.8)$$

Constraint (2.2.2) means that every customer $i \in V \setminus \{0\}$ must be served exactly once by a vehicle. Constraint (2.2.3) means a capacity constraint for vehicle $k$. Constraints (2.2.4)–(2.2.6) require that each vehicle cannot serve a customer before arriving at the customer. Constraint (2.2.7) is a time window constraint for each customer. Note that essential decision variables in this formulation are routes $\sigma_k$, since the values of $y_{ik}(\sigma)$ are
automatically determined from $\mathbf{\sigma}$, and finding appropriate values for $s_i$ and $s_{ik}^a$, if any, is easy when $\mathbf{\sigma}$ is fixed.

For VRPTW, even just finding a feasible schedule with a given number of vehicles is known to be NP-complete in the strong sense. Hence it may not be reasonable to restrict the search only within the feasible region of VRPTW, especially when the constraints are tight. Moreover, in real-world situation, time window and capacity constraints can be often violated to some extent. Considering these, the two constraints are often allowed to be violated. A constraint is called hard if it must be satisfied, while it is called soft if it can be violated. The violation of soft constraints is usually penalized and added to the objective function. The VRP with hard (resp., soft) time window constraints is abbreviated as VRPHTW (resp., VRPSTW).

2.3 Construction algorithm

In this section, we review construction algorithms which successively determines the values of variables (e.g., assignment of a customer to a vehicle, a route edge). In this section, the number of vehicles $m$ can also be a decision variable, and, in this case, the objective is to find a solution with the minimum vehicle number and the total traveling distance in the lexicographical order (i.e., a solution is better than another (1) if its vehicle number is smaller or (2) if the vehicle numbers are the same but the distance is smaller).

Construction algorithms run in very short time compared with local search and meta-heuristics, and they may be used to generate an initial solution for local search and meta-heuristics.

2.3.1 Insertion heuristic

An insertion heuristic was proposed and analyzed for the traveling salesman problem by Rosenkrantz, Stearns and Lewis [131] in 1977 and it was applied to the vehicle routing problem with time windows by Solomon [140] in 1987.

An insertion heuristic starts from an empty set of routes and unrouted customers. It repeats inserting an unrouted customer into a current partial route until no unrouted customer exists. An insertion heuristic is described as Algorithm 7. Important points in designing insertion heuristics are (1) the selection of an unrouted customer who is inserted for the next insertion, and (2) the position where the selected customer is inserted.

Campbell and Savelsbergh [26] discussed efficient implementations of insertion heuristics for vehicle routing and scheduling problems with complicated constraints.
Algorithm 7 Insertion heuristic
1: Let $N$ be a set of unrouted customers and $R$ be a set of routes which initially contain no customer.
2: while $N \neq \emptyset$ do
3: Select customer $i \in N$, route $r \in R$ and a position in $r$.
4: Insert $i$ at the selected position and let $N := N \setminus \{i\}$.
5: end while

2.3.2 Savings heuristic

A savings heuristic was proposed for the capacitated vehicle routing problem, which has no time window constraint, by Clarke and Wright [30] in 1964.

It begins with a solution in which every customer is visited by an individual vehicle, and the following combining procedure is repeated. Let $S_{ij}$ be a cost saving which is achieved by combining two routes, where the last customer of a route is $i$ and the first customer of the other route is $j$, i.e., $S_{ij} = c_{i0} + c_{0j} - c_{ij}$. It selects the edge $(i, j)$ which maximizes $S_{ij}$ under the condition that the combined route is feasible and combines the two routes. A savings heuristic is described as Algorithm 8.

Algorithm 8 Savings heuristic
1: Generate $n$ vehicle routes $\sigma_i = (0, i, 0)$ for $i \in V \setminus \{0\}$.
2: Compute the savings $S_{ij} = c_{i0} + c_{0j} - c_{ij}$ and order the savings in a nonincreasing fashion.
3: for Start from the top of the savings list do
4: Given a saving $S_{ij}$, determine whether there exist two routes, one containing edge $(0, j)$ and the other containing edge $(i, 0)$, that can feasibly be merged. If so, combine these two routes by deleting $(0, j)$ and $(i, 0)$ and introducing $(i, j)$.
5: end for

2.3.3 Fisher and Jaikumar algorithm

The Fisher and Jaikumar algorithm [50] was proposed for the capacitated vehicle routing problem in 1981.

The algorithm is a two-phase algorithm which is categorized as a cluster-first route-second method. In the first phase, it partitions customers to $m$ clusters by solving a generalized assignment problem (GAP), where a cluster will be visited by a vehicle. It selects a seed customer $i_k$ for each vehicle $k$, and estimates the assignment cost of customer $i$ to cluster $k$ by $d_{ik} = \min\{c_{0,i} + c_{i,i_k} + c_{i_k,0}, c_{0,i} + c_{i_k,i} + c_{i,0}, c_{0,i_k} + c_{i_k,0}\} - (c_{0,i_k} + c_{i_k,0})$. They solve
the resulting GAP by a branch-and-bound method based on a Lagrangian relaxation technique. Then, in the second phase, it determines an optimal route for each cluster by solving the corresponding traveling salesman problems (TSP). The algorithm is described as Algorithm 9.

**Algorithm 9** Fisher and Jaikumar heuristic

1. Select seed customers $i_k \in V$ for each vehicle $k$.
2. Compute the cost $d_{ik}$ of assigning customer $i$ to vehicle $k$ as $d_{ik} = \min\{c_{0,i} + c_{i,i_k} + c_{i_k,0}, c_{0,i_k} + c_{i_k,i} + c_{i,0}\} - (c_{0,i_k} + c_{i_k,0})$.
3. Solve the resulting instance of the GAP (i.e., assignment cost $d_{ik}$, customer weights $a_i$ and vehicle capacity $u$).
4. Solve instances of TSP for each assignment of vehicle $k$ corresponding the GAP solution.

Bramel and Simchi-Levi [21] and Koskosidis, Powell and Solomon [97] proposed algorithms, which are generalizations of the Fisher and Jaikumar algorithm, for the vehicle routing problem with time windows.

2.4 Local search

In this section, we review the representative neighborhoods for the traveling salesman problem, the capacitated vehicle routing problem, and the vehicle routing problem with time windows. See [4, 23, 52, 94] for more information.

In general, for routing problems, a neighborhood consists of the set of solutions that can be obtained by swapping a subset of route edges. There are two categories: One is the single-route neighborhood whose neighborhood operation is applied for a single route. The other is the multi-route neighborhood whose neighborhood operation is applied for more than one route. In figures of this thesis, squares represent the depot (which is duplicated at each end) and small circles represent customers in the routes. A thin line represents a route edge and a thick line represents a path (i.e., more than two customers may be included).

2.4.1 \(\lambda\)-opt neighborhood

The \(\lambda\)-opt neighborhood was proposed by Lin [105] for the traveling salesman problem. For a given route, it removes \(\lambda\) edges of a route, and the resulting \(\lambda\) segments are reconnected.
2.4.2 2-opt* neighborhood

The 2-opt* neighborhood was proposed by Potvin and Rousseau [126] in 1995, which is a variant of the 2-opt neighborhood. A 2-opt* operation removes two edges from two different routes (one from each) to divide each route into two parts and exchanges the second parts of the two routes. Figure 2.1 illustrates a 2-opt* neighborhood operation.

![Figure 2.1: A 2-opt* neighborhood operation](image)

2.4.3 λ-interchange neighborhood

Osman [118] proposed a λ-interchange neighborhood for the capacitated vehicle routing problem. A λ-interchange exchanges up to λ customers between two routes.

2.4.4 Cross exchange neighborhood

The cross exchange neighborhood was proposed by Taillard et al. [142] in 1997. A cross exchange operation removes two paths from two routes (one from each) of different vehicles and exchanges them. Figure 2.2 illustrates a cross exchange neighborhood operation.

The exchange neighborhood, which is the set of solutions obtainable by exchanging a customer of a route with a customer of the other, is included in the cross exchanging neighborhood. The relocate neighborhood, which is the set of solutions obtainable by removing a customer from a route and inserting the customer to another, is also included in the cross exchanging neighborhood.

The cross exchange neighborhood is an extension where the exchanged paths can be inserted with the reverse order [22].
2.4 Local search

2.4.5 Or-opt neighborhood

The Or-opt neighborhood was proposed for TSP by Or [116] in 1976. An Or-opt neighborhood operation removes a path which contains at most three customers and inserts it into another position of the same route. Figure 2.3 illustrates an Or-opt neighborhood operation.

The intra neighborhood [142] is an extension of Or-opt neighborhood that removes a path which can contain more than three customers and inserts it into another position of the same route. The intra neighborhood is also known as the iopt neighborhood [22]. An intra operation is categorized into four types:
1. the removed path is inserted forward with its order reversed,
2. the removed path is inserted forward with its order preserved,
3. the removed path is inserted backward with its order reversed, and
4. the removed path is inserted backward with its order preserved.

2.4.6 Cyclic transfer

Thompson and Psaraftis [147] proposed a cyclic transfer to vehicle routing and scheduling problems. A $k$-cyclic $\lambda$-transfer operation transfers $\lambda$ customers from each route in a circular manner among $k$ routes. However, in general, the neighborhood search problem is NP-hard [146].

2.5 General model

In this section, we formulate a general model which includes all problems considered in this thesis, and, in the next section, we describe a general neighborhood search framework for the model. Desaulniers et al. [40] made a similar attempt to provide a general solving framework. They proposed another general model, which can treat a variety of vehicle routing and scheduling problems, and presented a branch-and-bound framework.

Let $G = (V, E)$ be a complete directed graph with vertex set $V = \{0, 1, \ldots, n\}$ and edge set $E = \{(i, j) \mid i, j \in V, i \neq j\}$, $M = \{1, 2, \ldots, m\}$ be a vehicle set, and $\Omega = \{1, 2, \ldots, \nu\}$ be a resource set. In this graph, vertex 0 is the depot and other vertices are customers. Each customer $i$, each edge $(i, j) \in E$ and the depot are associated with:

1. a cost function $p_{\text{cust}}(X)_i$ for the amount $X$ of resource $\omega$ consumed on the route from the depot to customer $i$,
2. a cost function $p_{\text{depot}}(X)_k$ for the amount $X$ of resource $\omega$ consumed on the whole route visited by vehicle $k$,
3. a resource demand function $\lambda_{i,j}(X)$ between edge $(i, j)$ when the amount of resource $\omega$ consumed on the route from the depot to customer $i$ is $X$, and
4. a cost function $q_{i,j}(Y, X)$ between edge $(i, j)$ when the amount of resource $\omega$ consumed on the route from the depot to customer $i$ is $X$ and the amount of resource $\omega$ supplied on the edge is $Y$. 
When a vehicle travels an edge \((i, j)\), it consumes \(\lambda_{i,j}^\omega(X)\) units of resource for each \(\omega\) where the amount depends on the amount \(X\) consumed on the route from the depot to \(i\), and the resource can be supplied by arbitrary amount \(Y\) with additional cost \(q_{i,j}^\omega(Y, X)\).

Let \(\sigma_k\) denote the route traveled by vehicle \(k\), where \(\sigma_k(h)\) denotes the \(h\)th customer in \(\sigma_k\), and let

\[
\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m).
\]

Note that each customer \(i\) is included in exactly one route \(\sigma_k\), and is visited by vehicle \(k\) exactly once. We denote by \(n_k\) the number of customers in \(\sigma_k\). For convenience, we define \(\sigma_k(0) = 0\) and \(\sigma_k(n_k + 1) = 0\) for all \(k\) (i.e., each vehicle \(k \in M\) departs from the depot and comes back to the depot). Let us introduce 0-1 variables \(y_{ik}(\sigma) \in \{0, 1\}\) for \(i \in V \setminus \{0\}\) and \(k \in M\) by

\[
y_{ik}(\sigma) = 1 \iff i = \sigma_k(h) \text{ holds for exactly one } h \in \{1, 2, \ldots, n_k\}.
\]

That is, \(y_{ik}(\sigma) = 1\) holds if and only if vehicle \(k\) visits customer \(i\). Moreover, let \(X^i_\omega\) be the amount of resource \(\omega\) consumed on the route from the depot to \(i\) by a vehicle, let \(X^\text{depot}_k\) be the amount of resource \(\omega\) consumed on the whole route visited by vehicle \(k\), and let

\[
X = \begin{pmatrix}
X^1_1 & X^1_2 & \cdots & X^1_n & X^\text{depot}_1 & X^\text{depot}_2 & \cdots & X^\text{depot}_m \\
X^2_1 & X^2_2 & \cdots & X^2_n & X^\text{depot}_2 & X^\text{depot}_3 & \cdots & X^\text{depot}_m \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
X^\nu_1 & X^\nu_2 & \cdots & X^\nu_n & X^\text{depot}_\nu & X^\text{depot}_\nu & \cdots & X^\text{depot}_m
\end{pmatrix}.
\]

Let \(Y^\omega_{i,j}\) be the amount of resource \(\omega\) supplied on edge \((i, j)\), and let

\[
Y = (Y^\omega_{i,j}).
\]

The cost \(p^\omega(\sigma_k, X, Y)\) of resource \(\omega\) for a route \(\sigma_k\) is the sum of the cost for the amount consumed on the route from the depot to each customer \(\sigma_k(h)\) and the depot and the cost for the amounts of resource supplied on the traveled edges by vehicle \(k\):

\[
p^\omega(\sigma_k, X, Y) = \sum_{h=1}^{n_k} p_{\text{cust}, \sigma_k(h)}(X^\omega_{\sigma_k(h)}) + p_{\text{depot}, \sigma_k}(X^\text{depot}_k)
+ q^\omega_{0, \sigma_k(1)}(Y^\omega_{0, \sigma_k(1)}, 0) + \sum_{h=1}^{n_k} q^\omega_{\sigma_k(h), \sigma_k(h+1)}(Y^\omega_{\sigma_k(h), \sigma_k(h+1)}, X^\omega_{\sigma_k(h)}).
\]  

(2.5.9)
Then a general problem is formulated as follows:

\[
\begin{align*}
\text{minimize} & \quad \sum_{\omega \in \Omega} \sum_{k \in M} p^\omega(\sigma_k, X, Y) \\
\text{subject to} & \quad \sum_{k \in M} y_{ik}(\sigma) = 1, \quad i \in V \setminus \{0\} \\
& \quad \lambda_{0,\sigma_k(1)}^\omega(0) - Y_{0,\sigma_k(1)}^\omega = X_{\sigma_k(1)}^\omega, \quad k \in M, \omega \in \Omega \\
& \quad X_{\sigma_k(h)}^\omega + \lambda_{\sigma_k(h),\sigma_k(h+1)}^\omega(X_{\sigma_k(h)}^\omega) - Y_{\sigma_k(h),\sigma_k(h+1)}^\omega = X_{\sigma_k(h+1)}^\omega, \quad 1 \leq h \leq n_k - 1, \quad k \in M, \omega \in \Omega \\
& \quad X_{\sigma_k(n_k)}^\omega + \lambda_{\sigma_k(n_k),0}^\omega(X_{\sigma_k(n_k)}^\omega) - Y_{\sigma_k(n_k),0}^\omega = X_{\text{depot}_k}^\omega, \quad k \in M, \omega \in \Omega \\
& \quad y_{ik}(\sigma) \in \{0, 1\}, \quad i \in V \setminus \{0\}, \quad k \in M.
\end{align*}
\]
as follows:

\[
\begin{align*}
\text{minimize} & \quad \sum_{h=1}^{n_k+1} p_{\text{cust}}^\omega(X_h^\omega) + \sum_{h=0}^{n_k} q_{h,h+1}^\omega(Y_{h,h+1}^\omega, X_h^\omega) \\
\text{subject to} & \quad X_h^\omega + \lambda_{h,h+1}^\omega(X_h^\omega) - Y_{h,h+1}^\omega = X_{h+1}^\omega, \quad 0 \leq h \leq n_k.
\end{align*}
\] (2.5.16)

We will formulate the dynamic programming recursion for the OSP in two ways, which will be applied in the efficient neighborhood search in the next section. Note that, in general, the problem is NP-hard as described in Section 3.3.

Let \( f_h(t) \) be the minimum cost incurred on the path from the depot through customer \( h \) under the condition that the amount of resource consumed on the path is exactly \( t \) (i.e., \( X_h^\omega = t \)).

We call \( f_h(t) \) as a forward minimum cost function. Then it can be computed by the following recurrence formula of dynamic programming:

\[
\begin{align*}
f_0(t) &= \begin{cases} 
0, & t = 0 \\
+\infty, & \text{otherwise}
\end{cases} \\
f_h(t) &= p_{\text{cust}}^\omega(t) + \min_{X_{h-1}^\omega+\lambda_{h-1,h}^\omega(X_{h-1}^\omega) - Y_{h-1,h}^\omega = t} \{ f_{h-1}(X_{h-1}^\omega) + q_{h-1,h}^\omega(Y_{h-1,h}^\omega, X_{h-1}^\omega) \}, \\
&1 \leq h \leq n_k + 1, -\infty < t < +\infty.
\end{align*}
\] (2.5.18)

The optimal cost of the OSP for a route \( \sigma_k \) is given by \( \min_t f_{n_k+1}(t) \).

Let \( b_h(t) \) be the minimum cost incurred on the path from customer \( h \) through the depot under the condition that the amount of resource consumed from the depot through \( h \) is exactly \( t \) (i.e., \( X_h^\omega = t \)).

We call this a backward minimum cost function. Then, \( b_h(t) \) can be formulated as follows in a symmetric manner:

\[
\begin{align*}
b_{n_k+1}(t) &= p_{\text{cust}}^\omega(n_k+1)(t) \\
b_h(t) &= p_{\text{cust}}^\omega(h)(t) + \min_{t+\lambda_{h,h+1}^\omega(t) - Y_{h,h+1}^\omega = X_{h+1}^\omega} \{ b_{h+1}(X_{h+1}^\omega) + q_{h,h+1}^\omega(Y_{h,h+1}^\omega, t) \}, \\
&1 \leq h \leq n_k.
\end{align*}
\] (2.5.19)

Now the optimal cost of the OSP for a route \( \sigma_k \) is also given by

\[
\begin{align*}
\min_t \left\{ f_h(t) + \min_{t+\lambda_{h,h+1}^\omega(t) - Y_{h,h+1}^\omega = X_{h+1}^\omega} \{ b_{h+1}(X_{h+1}^\omega) + q_{h,h+1}^\omega(Y_{h,h+1}^\omega, t) \} \right\}
\end{align*}
\] (2.5.20)

for any \( h \) (1 \( h \) \( n_k \)).
2.6 Efficient neighborhood search method

In this section, we propose an efficient neighborhood search method for the general problem. We first describe the basic idea, and then describe how it is applied to each neighborhood search. Let $p_{\text{opt}}(\sigma_k)$ be the optimal cost of $\sigma_k$.

2.6.1 Basic idea

A key observation to the efficient computation is that each route $\sigma_k$ of a neighborhood solution is a recombination of a few paths of the current solution. Hence we consider a speeding up approach that stores some useful information of paths from the depot to customers and those from customers to the depot, among those paths of the current routes. For each customer $h$ in a new route $\sigma_k$, let $\mathcal{F}_h$ (resp., $\mathcal{B}_h$) be some data structure that contains the information of the path (of $\sigma_k$) from the depot to $h$ (resp., from $h$ to the depot). We call $\mathcal{F}_h$ as a forward data structure and $\mathcal{B}_h$ as a backward data structure. Note that $\mathcal{F}_h$ and $\mathcal{B}_h$ signify the information of the paths of the new route $\sigma_k$. For example, if $\sigma_k$ is generated by a 2-opt* operation, and the path from the depot to $h$ and the path from $h+1$ to the depot are from the current solution, then $\mathcal{F}_h$ and $\mathcal{B}_{h+1}$ are available from the stored information when they are used to compute $p_{\text{opt}}(\sigma_k)$. On the other hand, for the cross exchange and intra-route neighborhoods, $\mathcal{F}_h$ and $\mathcal{B}_h$ for customers $h$ in inserted paths need to be computed, because in the new route $\sigma_k$ the path from the depot to such an $h$ and that from $h$ to the depot are different from those in the current route. What is important in this approach is to execute the followings efficiently for a given $\sigma_k$:

1. construction of $\mathcal{F}_{h+1}$ from $\mathcal{F}_h$ (the forward computation),
2. construction of $\mathcal{B}_h$ from $\mathcal{B}_{h+1}$ (the backward computation), and
3. computation of $p_{\text{opt}}(\sigma_k)$ from $\mathcal{F}_h$ and $\mathcal{B}_{h+1}$.

It is not hard to show that each neighborhood solution can be evaluated in $O(T)$ time, if the above operations can be done in $O(T)$ time for any $h$ ($0 \leq h \leq n_k$). However, to accomplish this, the neighborhood need to be searched in an appropriate search order.

This strategy has also been used to devise efficient algorithms for a variety of vehicle routing and scheduling problems [74, 75, 85, 86]. Although, in this strategy, the optimal cost for a route is computed by connecting two paths, Kindervater and Savelsbergh [94] and Ibaraki et al. [86] proposed search strategies where the optimal cost for a route is computed by connecting more than two paths for the vehicle routing problem with time windows and the vehicle routing problem with convex time penalty functions, respectively.
2.6 Efficient neighborhood search method

2.6.2 How to apply the basic idea to evaluate solutions in neighborhoods

We now explain how to apply the above idea to evaluate solutions in the 2-opt*, cross exchange and intra neighborhoods efficiently. Here we assume that $\mathcal{F}_i$ and $\mathcal{B}_i$ for each customer $i$ are stored in memory. The time for the initial construction and an update caused by a move from the current solution can be ignored because they occur much less often than evaluations of solutions.

Below let Forward($\mathcal{F}, (i, j)$) (resp., Backward($\mathcal{B}, (i, j)$)) denote a call to construction of data structure for the forward (resp., backward) computation from $\mathcal{F}$ (resp., $\mathcal{B}$) along edge $(i, j)$ whose output is the constructed data structure, and let Connect($\mathcal{F}, \mathcal{B}, (i, j)$) denote a call to the computation of the optimal cost of the route which consists of the paths corresponding to $\mathcal{F}$ and $\mathcal{B}$ and $(i, j)$ concatenating them. We assume these procedure (i.e., Forward, Backward and Connect) can be done in $O(T)$ time. We will denote by $\langle \sigma_k(h_1) \rightarrow \sigma_k(h_2) \rangle$ the path from the $h_1$st customer to the $h_2$nd customer in route $\sigma_k$, and by $\langle \sigma_k(h_1) \rightarrow \sigma_k(h_2) \rangle - \langle \sigma_{k'}(h_3) \rightarrow \sigma_{k'}(h_4) \rangle$ the path constructed by connecting two paths $\langle \sigma_k(h_1) \rightarrow \sigma_k(h_2) \rangle$ and $\langle \sigma_{k'}(h_3) \rightarrow \sigma_{k'}(h_4) \rangle$ from routes $\sigma_k$ and $\sigma_{k'}$.

2-opt* neighborhood

Let us consider the 2-opt* operation on routes $\sigma_k$ and $\sigma_{k'}$.

In Figure 2.4, an example of a 2-opt* operation on routes $\sigma_k$ and $\sigma_{k'}$ is shown. We denote by $\sigma_k^{new}$ and $\sigma_{k'}^{new}$ the resulting two routes (i.e., $\sigma_k^{new} = \langle 0 \rightarrow \sigma_k(h_k) \rangle - \langle \sigma_k(h_k + 1) \rightarrow 0 \rangle$ and $\sigma_{k'}^{new} = \langle 0 \rightarrow \sigma_{k'}(h_{k'}) \rangle - \langle \sigma_{k'}(h_{k'} + 1) \rightarrow 0 \rangle$). Then the cost $p_{opt}(\sigma_{k'}^{new})$ (resp., $p_{opt}(\sigma_k^{new})$) of the new route is obtained by connecting $\sigma_k(h_k)$ and $\sigma_{k'}(h_{k'} + 1)$ (resp., $\sigma_{k'}(h_{k'})$ and $\sigma_k(h_k + 1)$). Hence, when a 2-opt* operation is applied to routes $\sigma_k$ and $\sigma_{k'}$, we can evaluate the cost of the resulting solution in $O(T)$ time. The procedure is described as Algorithm 10.
Algorithm 10 The search method for the 2-opt* neighborhood

1:  \textbf{procedure} 2-\textsc{opt}∗(σ_k, σ_{k'}, h_k, h_{k'})
2:  \text{Connect}(F_{σ_k(h_k)}, B_{σ_{k'}(h_{k'}+1)}, (σ_k(h_k), σ_{k'}(h_{k'}+1))) ⊳ p_{opt}(σ_{k'}^{\text{new}})
3:  \text{Connect}(F_{σ_{k'}(h_{k'}+1)}, B_{σ_k(h_k+1)}, (σ_{k'}(h_{k'}), σ_k(h_k+1))) ⊳ p_{opt}(σ_k^{\text{new}})
4:  \textbf{end procedure}

Cross exchange neighborhood

Let us consider the cross exchange operation on routes σ_k and σ_{k'}. We restrict the length (i.e., the number of customers in the path) of the exchanged path at most \(L^{\text{cross}}\) (a parameter).

To evaluate solutions in the cross exchange neighborhood efficiently, we need to search the solutions in the neighborhood in a specific order. To apply cross exchange operations on routes σ_k and σ_{k'}, we start from a solution obtainable by exchanging one customer from each route, and then extend lengths of the paths to be exchanged one by one. We denote by σ_k^{\text{new}} and σ_{k'}^{\text{new}} the resulting two routes (i.e., σ_k^{\text{new}} = \langle 0 → σ_k(h_k^1) → σ_k(h_k^1 + 1) → \cdots \rangle and σ_{k'}^{\text{new}} = \langle 0 → σ_{k'}(h_{k'}^1) → σ_{k'}(h_{k'}^1 + 1) → \cdots \rangle). Then the cost \(p_{opt}(σ_k^{\text{new}})\) (resp., \(p_{opt}(σ_{k'}^{\text{new}})\)) of the new route is obtained by connecting σ_k'(h_{k'} + l') and σ_k(h_k + l) (resp., σ_k'(h_{k'} + l') and σ_k'(h_{k'} + l') + 1)). The procedure is described as Algorithm 11. See Figure 2.5 for a help to understand the description.

![Figure 2.5](image)

Although, in Algorithm 11, at least one customer is exchanged from both routes for
Algorithm 11 The search method for cross exchange neighborhood

1: procedure Cross($\sigma_k, \sigma_{k'}, h^k_1, h'^{k'}_1$)
2:   for $l \leftarrow 1, \min\{L^\text{cross}, n_k - h^k_1\} \ do \ extender$ path $\langle \sigma_k(h^k_1 + 1) \rightarrow \sigma_k(h^k_1 + l) \rangle$
3:       if $l = 1 \ then$
4:          $\tilde{F} := \text{Forward}(\mathcal{F}'_{\sigma_k(h^k_1)}, (\sigma_k(h^k_1), \sigma_k(h^k_1 + 1)))$
5:       else
6:          $\tilde{F} := \text{Forward}(\mathcal{F}, (\sigma_k(h^k_1 + l - 1), \sigma_k(h^k_1 + l)))$
7:       end if
8:   for $l' \leftarrow 1, \min\{L^\text{cross}, n_{k'} - h'^{k'}_1\} \ do \ extender$ path $\langle \sigma_{k'}(h'^{k'}_1 + 1) \rightarrow \sigma_{k'}(h'^{k'}_1 + l') \rangle$
9:       if $l = 1 \ then$
10:          $\tilde{F}' := \text{Forward}(\mathcal{F}_k^{k'}(h'^{k'}_1), (\sigma_{k'}(h'^{k'}_1), \sigma_{k'}(h'^{k'}_1 + 1)))$
11:       else
12:          $\tilde{F}' := \text{Forward}(\mathcal{F}', (\sigma_{k'}(h^k_1 + l' - 1), \sigma_{k'}(h'^{k'}_1 + l')))$
13:       end if
14:       Connect($\tilde{F}, B^k_{\sigma_k(h^k_1 + l + 1)}(\sigma_k(h^k_1 + l'), \sigma_k(h^k_1 + l + 1))) \ extender p^\text{opt}(\sigma^\text{new}_k)$
15:       Connect($\tilde{F}', B^{k'}_{\sigma_{k'}(h'^{k'}_1 + l' + 1)}(\sigma_{k'}(h'^{k'}_1 + l'), \sigma_{k'}(h'^{k'}_1 + l' + 1))) \ extender p^\text{opt}(\sigma^\text{new}_{k'})$
16:   end for
17: end procedure

simplicity, a neighborhood operation which relocates the path from a route to the other (e.g., the relocation neighborhood) can be evaluated in $O(T)$ time. Furthermore each icross neighborhood solution can analogously be evaluated in $O(T)$ time.

Intra neighborhood

Let us consider the intra operation on route $\sigma_k$. We restrict the length of the exchanged path at most $L^\text{intra}_{\text{path}}$ (a parameter) and the position to be inserted is limited within length $L^\text{intra}_{\text{ins}}$ (a parameter) from the original position. As described before, there are four types of intra neighborhood operations:

1. the removed path is inserted forward with its order reversed,
2. the removed path is inserted forward with its order preserved,
3. the removed path is inserted backward with its order reversed, and
4. the removed path is inserted backward with its order preserved.

The procedure for the case that the removed path is inserted forward with its order reversed is described as Algorithm 12. See Figure 2.6 for a help to understand the description.
We denote by $\sigma_k^{\text{new}}$ the resulting route. In Algorithm 12, a path $(\sigma_k(h_1 + l - 1) \rightarrow \sigma_k(h_1))$

Algorithm 12 The search method for the forward reverse intra neighborhood

1: procedure INTRAForwardReverse($\sigma_k, h_1$)
2:     for $h_2 \leftarrow h_1 + 1, \min\{h_1 + L_{\text{ins}}^{\text{intra}}, n_k\}$ do
3:         $\triangleright$ Insert a path between $\sigma_k(h_2)$ and $\sigma_k(h_2 + 1)$
4:             for $l \leftarrow 1, \min\{L_{\text{path}}^{\text{intra}}, h_2 - h_1\}$ do
5:                 $\triangleright$ Extend path $(\sigma_k(h_1) \rightarrow \sigma_k(h_1 + l - 1))$
6:                     if $h_2 = h_1 + 1$ then
7:                         $\tilde{F}_l := \text{Forward}(\tilde{F}_l, (\sigma_k(h_1 - 1), \sigma_k(h_2)))$
8:                     else
9:                         $\tilde{F}_l := \text{Forward}(\tilde{F}_l, (\sigma_k(h_2 - 1), \sigma_k(h_2)))$
10:                 end if
11:             if $l = 1$ then
12:                 $\tilde{B} := \text{Backward}(\tilde{B}, (\sigma_k(h_1 + l - 1), \sigma_k(h_2 + 1)))$
13:             else
14:                 $\tilde{B} := \text{Backward}(\tilde{B}, (\sigma_k(h_1 + l - 1), \sigma_k(h_1 + l - 2)))$
15:             end if
16:     end for
17:     $\triangleright p_{\text{opt}}(\sigma_k^{\text{new}})$
18: end procedure

is removed from $\sigma_k$ and is inserted between $\sigma_k(h_2)$ and $\sigma_k(h_2 + 1)$, and the exchanged path is extended by one at each iteration. For each evaluation of the neighborhood solutions, it takes $O(T)$ time; it calls Forward, Backward and Connect once. Hence each intra neighborhood solution which inserts a path forward with its order reversed can be evaluated in $O(T)$ time.

Other cases can be treated similarly. The descriptions of search methods and figures to help understand them are as follows: Algorithm 13 and Figure 2.7 are for the case where the removed path is inserted forward with its order preserved, Algorithm 14 and Figure 2.8 are for the case the removed path is inserted backward with its order reversed, and Algorithm 15 and Figure 2.9 are for the case the removed path is inserted backward with its order preserved.
2.6 Efficient neighborhood search method

\[ \sigma_k(h_1) \quad \sigma_k(h_1 + l - 1) \]
\[ \sigma_k(h_1 - 1) \quad \sigma_k(h_1 + l) \]
\[ \sigma_k(h_1) \quad \sigma_k(h_1 + 1) \]
\[ \sigma_k(h_1 - l + 1) \quad \sigma_k(h_1) \]
\[ \sigma_k(h_1 - l) \quad \sigma_k(h_1 + 1) \]
\[ \sigma_k(h_2) \quad \sigma_k(h_2 + 1) \]

Figure 2.6: An illustration of the search method for the forward reverse intra neighborhood

Figure 2.7: An illustration of the search method for the forward normal intra neighborhood
Figure 2.8: An illustration of the search method for the backward reverse intra neighborhood

Figure 2.9: An illustration of the search method for the backward normal intra neighborhood
Algorithm 13 The search method for the forward normal intra neighborhood

1: procedure \texttt{IntraForwardNormal}(\sigma_k, h_1)
2: \hspace{1em} for \( h_2 \leftarrow h_1 + 1, \min\{ h_1 + L_{\text{intra}}^{\sigma_k}, n_k \} \) do \Comment{Insert a path between \( \sigma_k(h_2) \) and \( \sigma_k(h_2 + 1) \)}
3: \hspace{2em} for \( l \leftarrow 1, \min\{ L_{\text{path}}^{\sigma_k}, h_2 - h_1 \} \) do \Comment{Extend path \( \langle \sigma_k(h_1 - l + 1) \rightarrow \sigma_k(h_1) \rangle \)}
4: \hspace{3em} if \( h_2 = h_1 + 1 \) then
5: \hspace{4em} \tilde{F}_l := \text{Forward}(\mathcal{F}_{\sigma_k(h_1 - l)}, \langle \sigma_k(h_1 - l), \sigma_k(h_2) \rangle) \)
6: \hspace{4em} else
7: \hspace{5em} \tilde{F}_l := \text{Forward}(\tilde{F}_l, \langle \sigma_k(h_2 - 1), \sigma_k(h_2) \rangle) \)
8: \hspace{3em} end if
9: \hspace{2em} if \( l = 1 \) then
10: \hspace{3em} \tilde{B} := \text{Backward}(\mathcal{B}_{\sigma_k(h_2 + 1)}, \langle \sigma_k(h_1 - l + 1), \sigma_k(h_2 + 1) \rangle) \)
11: \hspace{3em} else
12: \hspace{4em} \tilde{B} := \text{Backward}(\tilde{B}, \langle \sigma_k(h_1 - l + 1), \sigma_k(h_1 - l + 2) \rangle) \)
13: \hspace{3em} end if
14: \hspace{1em} Connect(\tilde{F}_l, \tilde{B}, \langle \sigma_k(h_2), \sigma_k(h_1 - l + 1) \rangle) \Comment{p_{\text{opt}}(\sigma_k^{\text{new}})}
15: \hspace{1em} end for
16: \hspace{1em} end for
17: end procedure
Algorithm 14 The search method for the backward reverse intra neighborhood

1: procedure IntraBackwardReverse($\sigma_k, h_1$)
2:   for $h_2 \leftarrow h_1 - 1, \max\{h_1 - L_{\text{ins}}^{\text{intra}}, 1\}$ do ▷ Insert a path between $\sigma_k(h_2 - 1)$ and $\sigma_k(h_2)$
3:     for $l \leftarrow 1, \min\{L_{\text{path}}^{\text{intra}}, h_1 - h_2\}$ do ▷ Extend path $\langle \sigma_k(h_1 - l + 1) \rightarrow \sigma_k(h_1) \rangle$
4:         if $h_2 = h_1 - 1$ then
5:             $\tilde{B}_l$:=Backward($B_{\sigma_k(h_1-l)}$, $(\sigma_k(h_2), \sigma_k(h_1+1))$)
6:         else
7:             $\tilde{B}_l$:=Backward($\tilde{B}_l$, $(\sigma_k(h_2), \sigma_k(h_2+1))$)
8:         end if
9:         if $l = 1$ then
10:            $\tilde{F}$:=Forward($F_{\sigma_k(h_2-1)}$, $(\sigma_k(h_2-1), \sigma_k(h_1-l+1))$)
11:        else
12:            $\tilde{F}$:=Forward($\tilde{F}$, $(\sigma_k(h_1-l+2), \sigma_k(h_1-l+1))$)
13:        end if
14:     end for
15:     Connect($\tilde{F}, \tilde{B}_l$, $(\sigma_k(h_1-l+1), \sigma_k(h_2))$) ▷ $p_{\text{opt}}(\sigma_k^{\text{new}})$
16: end for
17: end procedure
Algorithm 15 The search method for the backward normal intra neighborhood

1: procedure INTRABACKWARDNORMAL($\sigma_k, h_1$)
2:   for $h_2 \leftarrow h_1 - 1, \max\{h_1 - I_{\text{ins}}^\text{intra}, 1\}$ do  \Comment{Insert a path between $\sigma_k(h_2 - 1)$ and $\sigma_k(h_2)$}
3:     for $l \leftarrow 1, \min\{I_{\text{path}}^\text{intra}, h_1 - h_2\}$ do  \Comment{Extend path $\langle \sigma_k(h_1) \rightarrow \sigma_k(h_1 + l - 1) \rangle$}
4:       if $h_2 = h_1 - 1$ then
5:         $\tilde{B}_l := \text{Backward}(\mathcal{B}_{\sigma_k(h_1 + l)}, (\sigma_k(h_2), \sigma_k(h_1 + l)))$
6:       else
7:         $\tilde{B}_l := \text{Backward}(\tilde{B}_l, (\sigma_k(h_2), \sigma_k(h_2 + 1)))$
8:     end if
9:     if $l = 1$ then
10:        $\tilde{F} := \text{Forward}(\mathcal{F}_{\sigma_k(h_2 - 1)}, (\sigma_k(h_2 - 1), \sigma_k(h_1 + l - 1)))$
11:     else
12:        $\tilde{F} := \text{Forward}(\tilde{F}, (\sigma_k(h_1 + l - 2), \sigma_k(h_1 + l - 1)))$
13:     end if
14:     Connect($\tilde{F}, \tilde{B}_l, (\sigma_k(h_1 + l - 1), \sigma_k(h_2))$)  \Comment{$p_{\text{opt}}(\sigma_k^{\text{new}})$}
15:   end for
16: end procedure
Vehicle Routing and Scheduling Problem
Chapter 3

The Vehicle Routing Problem
with Flexible Time Windows and Traveling Times

3.1 Introduction

In this chapter, in addition to soft time window constraints, we treat the traveling times between customers as variables. The difference between the start times of services at a customer $i$ and the next customer $j$ in a route is the sum of the following three components: (1) the service time of $i$, (2) the traveling time between $i$ and $j$, and (3) the waiting time at $j$. The service time and the traveling time are given as constant values, in standard VRP formulation. In practice, however, these values can be changed with some cost (e.g., the service time can be shortened by investing more work force, and the traveling time can be shortened by paying the turnpike toll). We therefore redefine the traveling time as a variable representing the difference between the start times of services at two consecutive customers, and introduce its cost function. Our goal is to find a flexible solution, whose cost is considerably small, with a little penalty if necessary.

With soft time windows and/or variable traveling times, even after fixing the order of customers for each vehicle to visit, it becomes nontrivial to determine the optimal start times of services at all customers so that the total cost of the vehicle is minimized. We first show that this problem is NP-hard when cost functions are general. We then consider a restricted problem, which is still NP-hard, and propose a dynamic programming algorithm whose time complexity is of pseudo polynomial order. Then, assuming that traveling time cost functions are convex we modify the dynamic programming into a polynomial time algorithm, which is then incorporated in the iterated local search algorithm of this chapter,
We conduct computational experiments on representative benchmark instances of VRPTW. Our algorithm can find solutions whose traveling distances are much smaller than those of the best known solutions by allowing small violations of the given time window and/or traveling time constraints. The outcomes may indicate the usefulness of the proposed generalization.

3.2 Problem definition

Here we formulate the VRP with time window and traveling time constraints. Let $G = (V, E)$ be a complete directed graph with vertex set $V = \{0, 1, \ldots, n\}$ and edge set $E = \{(i, j) \mid i, j \in V, i \neq j\}$, and $M = \{1, 2, \ldots, m\}$ be a vehicle set. In this graph, vertex 0 is the depot and other vertices are customers. Each customer $i$, each vehicle $k$ and each edge $(i, j) \in E$ are associated with:

1. a fixed quantity $a_i (\geq 0)$ of goods to be delivered to $i$,
2. a time window cost function $p_i(t)$ of the start time $t$ of the service at $i$ ($p_0(t)$ is the time window cost function of the arrival time $t$ at the depot),
3. a capacity $u_k (\geq 0)$ of $k$,
4. a distance $d_{ij} (\geq 0)$ from $i$ to $j$,
5. a traveling time cost function $q_{ij}(t)$ of the traveling time $t$ from $i$ to $j$.

We assume $a_0 = 0$ without loss of generality. The distance matrix $(d_{ij})$ is not necessarily symmetric. We assume that each time window cost function $p_i(t)$ is nonnegative, piecewise linear and lower semicontinuous (i.e., $p_i(t) \leq \lim_{\varepsilon \to 0} \min\{p_i(t + \varepsilon), p_i(t - \varepsilon)\}$ at every discontinuous point $t$). Note that $p_i(t)$ can be non-convex and discontinuous as long as it satisfies the above conditions. We also assume $p_i(t) = +\infty$ for $t < 0$ so that the start time $t$ of the service is nonnegative. Similarly, we assume that each traveling time cost function $q_{ij}(t)$ is nonnegative, piecewise linear and lower semicontinuous. We also assume $q_{ij}(t) = +\infty$ for $t < 0$ so that the traveling time $t$ between customers is nonnegative. These assumptions ensure the existence of an optimal solution. We further assume that the linear pieces of each piecewise linear function are given explicitly.

Let $\sigma_k$ denote the route traveled by vehicle $k$, where $\sigma_k(h)$ denotes the $h$th customer in $\sigma_k$, and let

$$\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m).$$

Note that each customer $i$ is included in exactly one route $\sigma_k$, and is visited by vehicle $k$ exactly once. We denote by $n_k$ the number of customers in $\sigma_k$. For convenience, we define
\( \sigma_k(0) = 0 \) and \( \sigma_k(n_k + 1) = 0 \) for all \( k \) (i.e., each vehicle \( k \in M \) leaves the depot and comes back to the depot). Moreover, let \( s_i \) be the start time of service at customer \( i \) (by exactly one of the vehicles) and \( s^a_k \) be the arrival time of vehicle \( k \) at the depot, and let

\[
s = (s_1, s_2, \ldots, s_n, s^a_1, s^a_2, \ldots, s^a_m).
\]

We assume that all vehicles have the same departure time 0 (i.e., \( s_0 = 0 \)) from the depot, and all time window cost functions of the arrival time at the depot (i.e., \( p_0(t) \)) are identical for convenience. Note however that we can consider separate time window cost functions of vehicles at the depot by introducing \( m \) dummy customers and making each vehicle visit one of the dummy customers first.

Let us introduce 0-1 variables \( y_{ik}(\sigma) \in \{0, 1\} \) for \( i \in V \setminus \{0\} \) and \( k \in M \) by

\[
y_{ik}(\sigma) = 1 \iff i = \sigma_k(h) \text{ holds for exactly one } h \in \{1, 2, \ldots, n_k\}.
\]

That is, \( y_{ik}(\sigma) = 1 \) holds if and only if vehicle \( k \) visits customer \( i \). Then the total distance \( d_{\text{sum}} \) traveled by all vehicles, the total time window cost \( p_{\text{sum}} \) of all customers, the total traveling time cost \( q_{\text{sum}} \), and the total excess amount \( a_{\text{sum}} \) of capacities are expressed as

\[
d_{\text{sum}}(\sigma) = \sum_{k \in M} \sum_{h=0}^{n_k} d_{\sigma_k(h), \sigma_k(h+1)}, \quad (3.2.1)
\]

\[
p_{\text{sum}}(s) = \sum_{i \in V \setminus \{0\}} p_i(s_i) + \sum_{k \in M} p_0(s^a_k), \quad (3.2.2)
\]

\[
q_{\text{sum}}(\sigma, s) = \sum_{k \in M} \sum_{h=0}^{n_k-1} q_{\sigma_k(h), \sigma_k(h+1)}(s_{\sigma_k(h+1)} - s_{\sigma_k(h)}) + \sum_{k \in M} q_{\sigma_k(n_k), 0}(s^a_k - s_{\sigma_k(n_k)}), \quad (3.2.3)
\]

\[
a_{\text{sum}}(\sigma) = \sum_{k \in M} \max \left\{ \sum_{i \in V \setminus \{0\}} a_i y_{ik}(\sigma) - u_k, 0 \right\}. \quad (3.2.4)
\]

Then, the problem can be formulated as follows:

\[
\begin{align*}
\text{minimize} & \quad d_{\text{sum}}(\sigma) + p_{\text{sum}}(s) + a_{\text{sum}}(\sigma) + q_{\text{sum}}(\sigma, s) \\
\text{subject to} & \quad \sum_{k \in M} y_{ik}(\sigma) = 1, \quad \forall i \in V \setminus \{0\}. \quad (3.2.6)
\end{align*}
\]

In this formulation, time window, traveling time and capacity constraints are all treated as soft, and their violation is evaluated as costs \( p_{\text{sum}}(s) \), \( q_{\text{sum}}(\sigma, s) \) and \( a_{\text{sum}}(\sigma) \) in the objective function, respectively.
The standard problem VRPHTW, in which time windows $[t^i_r, t^d_i]$, service times $\tau_i$ and traveling times $t_{ij}$ are given as constant values, can be formulated in the form of (3.2.5)–(3.2.6) by using the following $p_i(t)$ and $q_{ij}(t)$:

$$p_i(t) = \begin{cases} 0, & t^r_i \leq t \leq t^d_i \\ +\infty, & \text{otherwise} \end{cases}$$

$$q_{ij}(t) = \begin{cases} +\infty, & t < \tau_i + t_{ij} \\ 0, & t \geq \tau_i + t_{ij} \end{cases}$$

### 3.3 Optimal start times of services

In this section, we consider the problem of determining the time to start services at customer $i$ in a given route $\sigma_k$ so that the total of time window and traveling time costs is minimized. (How to determine $\sigma_k$ will be discussed in Section 3.4.) We call this the optimal start time problem, and abbreviate it as OSTP. First, we prove that OSTP is NP-hard in general in Section 3.3.1. Next, in Section 3.3.2, we consider a restricted problem, which is still NP-hard, but permits a dynamic programming algorithm of pseudo polynomial time order. Then in Section 3.3.3, we show that the same dynamic programming can be implemented so that it runs in polynomial time, if each traveling time cost function is convex.

For convenience, throughout this section, we assume that vehicle $k$ visits customers $1, 2, \ldots, n_k$ in this order and let customer $n_k + 1$ represent the arrival at the depot (i.e., $s_{n_k+1} = s^a_k$ and $p_{n_k+1}(s_{n_k+1}) = p_0(s^a_k)$). Then, OSTP is formulated as follows:

$$\text{minimize} \quad f_{\text{OSTP}}(s) = \sum_{i=1}^{n_k+1} p_i(s_i) + \sum_{i=1}^{n_k+1} q_{i-1,i}(s_i - s_{i-1}) \quad (3.3.7)$$

subject to $s_0 = 0$.

#### 3.3.1 NP-hardness

**Theorem 3.3.1** The optimal start time problem (OSTP) is NP-hard if each time window cost function $p_i$ and each traveling time cost function $q_{ij}$ are general piecewise linear.

**Proof.** We reduce the 0-1 knapsack problem (abbreviated as KP), which is one of the
3.3 Optimal start times of services

representative NP-hard problems [91, 109], to OSTP. KP with \( n' \) items is defined by

\[
\text{maximize} \quad \sum_{i=1}^{n'} c_i x_i \tag{3.3.8}
\]

subject to

\[
\sum_{i=1}^{n'} w_i x_i \leq b
\]

\( x_i \in \{0, 1\}, \quad i = 1, 2, \ldots, n' \).

Note that objective function (3.3.8) is equivalent to

\[
\text{minimize} \quad \sum_{i=1}^{n'} c_i (1 - x_i) = \sum_{i=1}^{n'} c_i - \sum_{i=1}^{n'} c_i x_i. \tag{3.3.9}
\]

For a given instance of KP, we set \( n_k := n' - 1 \) and define \( p_i \) and \( q_{i-1,i} \) for \( i = 1, 2, \ldots, n_k + 1 \) as follows:

\[
p_i(t) = \begin{cases} 
0, & t \in [0, b] \\
+\infty, & t \in (b, +\infty),
\end{cases} \tag{3.3.10}
\]

\[
q_{i-1,i}(t) = \begin{cases} 
c_i, & t \in [0, w_i) \\
0, & t \in [w_i, +\infty). \tag{3.3.11}
\end{cases}
\]

We will show that this OSTP instance has the same objective value as KP with objective function (3.3.9).

Let us define a vector \( \tilde{s} = (\tilde{s}_0, \tilde{s}_1, \ldots, \tilde{s}_{n_k+1}) \) of start times, corresponding to a feasible solution \( \tilde{x} \) of the 0-1 knapsack, by \( \tilde{s}_0 = 0 \) and the following \( \tilde{s}_i \) for \( i \geq 1 \):

\[
\tilde{s}_i = \begin{cases} 
\tilde{s}_{i-1}, & \text{if } \tilde{x}_i = 0 \\
\tilde{s}_{i-1} + w_i, & \text{if } \tilde{x}_i = 1.
\end{cases} \tag{3.3.12}
\]

Then

\[
\tilde{s}_i = \sum_{j \leq i} w_j \tilde{x}_j \leq b, \quad i = 1, 2, \ldots, n_k + 1 \tag{3.3.13}
\]

holds from the feasibility of \( \tilde{x} \). The time window cost of \( \tilde{s} \) is \( \sum_{i=1}^{n_k+1} p_i(\tilde{s}_i) = 0 \) from (3.3.10) and (3.3.13), and the traveling time cost is \( \sum_{i=1}^{n_k+1} q_{i-1,i}(\tilde{s}_i - \tilde{s}_{i-1}) = \sum_{i=1}^{n_k+1} c_i (1 - \tilde{x}_i) \) from (3.3.11) and (3.3.12). Hence the objective value of \( \tilde{s} \) for the OSTP instance is equal to the objective value of \( \tilde{x} \) for the KP instance.

Conversely, let us define \( \hat{x} \) corresponding to a solution \( \hat{s} \) that has a finite objective value for the OSTP instance as follows:

\[
\hat{x}_i = \begin{cases} 
1, & \text{if } \hat{s}_i - \hat{s}_{i-1} \geq w_i \\
0, & \text{if } \hat{s}_i - \hat{s}_{i-1} < w_i.
\end{cases} \tag{3.3.14}
\]
Then we have
\[ \sum_{i=1}^{n_k+1} w_i \hat{x}_i \leq \sum_{i=1}^{n_k+1} (\hat{s}_i - \hat{s}_{i-1}) \hat{x}_i \leq \sum_{i=1}^{n_k+1} (\hat{s}_i - \hat{s}_{i-1}) = \hat{s}_{n_k+1} \leq b. \]

Note that the last inequality is derived from (3.3.10) and the fact that \( \hat{s} \) has a finite objective value. For the same reason, we have
\[ \sum_{i=1}^{n_k+1} p_i(\hat{s}_i) = 0, \]
and hence
\[ \sum_{i=1}^{n_k+1} c_i (1 - \hat{x}_i) = \sum_{i=1}^{n_k+1} p_i(\hat{s}_i) + \sum_{i=1}^{n_k+1} q_{i-1,i}(\hat{s}_i - \hat{s}_{i-1}) \]
holds from definitions (3.3.11) and (3.3.14). The optimal value of the KP instance is therefore equal to the optimal value of the OSTP instance.

As the KP instance always has a feasible solution \( \mathbf{x} = 0 \), the above discussion shows that KP is reducible to OSTP.  □

### 3.3.2 Pseudo polynomial time algorithm

We first show that OSTP defined for route \( \sigma_k \) of vehicle \( k \) can be solved by using dynamic programming.

Let \( f^k_h(t) \) be the minimum sum of the costs for customers 0, 1, 2, \ldots, \( h \) served by vehicle \( k \) in this order under the condition that customers 0, 1, \ldots, \( h-1 \) are served before time \( t \) and customer \( h \) is served exactly at time \( t \) (i.e.,
\[ \min_{s_0 = 0, s_h = t} \sum_{i=1}^{h} p_i(s_i) + \sum_{i=1}^{h} q_{i-1,i}(s_i - s_{i-1}) \]).

Throughout this chapter, we call this a forward minimum cost function. Then \( f^k_h(t) \) can be computed by the following recursive formula
\[
\begin{align*}
  f^k_0(t) &= \begin{cases} 
    +\infty, & t \neq 0 \\
    0, & t = 0
  \end{cases} \\
  f^k_h(t) &= p_h(t) + \min_{0 \leq t' \leq t} \{ f^k_{h-1}(t') + q_{h-1,h}(t-t') \},
\end{align*}
\]
\[ 1 \leq h \leq n_k + 1, -\infty < t < +\infty. \]

The optimal cost for route \( \sigma_k \) is given by \( \min_t f^k_{n_k+1}(t) \).

In this subsection, we assume that each breakpoint \( t \) (i.e., the left or right end of a linear piece) of given piecewise linear functions \( p_i(t) \) and \( q_{ij}(t) \) is integer. Note that \( p_i(t) \) and \( q_{ij}(t) \) may not be integers. In this case, it is not difficult to show the following lemma.

**Lemma 3.3.1** An OSTP instance with integer input has an optimal solution whose start times are integers.
Proof. Let \( s^* = (s^*_1, s^*_2, \ldots, s^*_n) \) be an optimal solution of the problem. We will show by induction that all \( s^*_i \) can be integers.

By definition (3.3.7), \( s^*_0 = 0 \) holds. Assume that \( s^*_i \) are integers for \( i \leq h - 1 \) but \( s^*_h \) is not an integer, where \( h \leq n_k \) holds.

If \( s^*_{h+1} - s^*_h \) is not an integer, then the gradient of \( f_{OSTP} \) for \( s^*_h \) exists and is given by

\[
\frac{\partial}{\partial s^*_h} f_{OSTP}(s^*) = \frac{\partial}{\partial s^*_h} \{q_{h-1,h}(s^*_h - s^*_{h-1}) + p_h(s^*_h) + q_{h,h+1}(s^*_{h+1} - s^*_h)\},
\]

(3.3.16)

because the breakpoints of \( q_{h-1,h}, p_h \) and \( q_{h,h+1} \) are integers but \( s^*_h - s^*_{h-1}, s^*_h \) and \( s^*_{h+1} - s^*_h \) are not integers. If the gradient (3.3.16) is not 0, we can reduce the objective value by changing \( s^*_h \) slightly, which is a contradiction. Hence the gradient is 0. We can therefore change \( s^*_h \) until it becomes an integer without increasing the objective value by choosing the direction of the change appropriately so that \( s^*_h \) becomes an integer before \( s^*_{h+1} - s^*_h \) does or both \( s^*_h \) and \( s^*_{h+1} - s^*_h \) become integers simultaneously.

If \( s^*_{h+1} - s^*_h \) is an integer, we fix the difference \( s^*_{h+1} - s^*_h \) and change the values of \( s^*_h \) and \( s^*_{h+1} \) simultaneously. For this purpose, we contract customers \( h \) and \( h + 1 \) to form a new customer \( \tilde{h} \) with \( s^*_\tilde{h} = s^*_h \) and define the time window cost function and traveling time cost functions as follows:

\[
p_{\tilde{h}}(t) = p_h(t) + p_{h+1}(t + s^*_{h+1} - s^*_h),
\]

(3.3.17)

\[
q_{h-1,\tilde{h}}(t) = q_{h-1,h}(t),
\]

(3.3.18)

\[
q_{\tilde{h},h+2}(t) = q_{h,h+1}(s^*_{h+1} - s^*_h) + q_{h+1,\tilde{h}}(t - s^*_{h+1} + s^*_h),
\]

(3.3.19)

where we define \( q_{\tilde{h},h+2}(t) \) only if \( h \leq n_k - 1 \). Then increasing the value of \( s^*_h \) by a constant \( c \) is equivalent to increasing the values of \( s^*_h \) and \( s^*_{h+1} \) by \( c \) in the original formulation. The breakpoints of \( p_{\tilde{h}}(t), q_{h-1,\tilde{h}}(t) \) and \( q_{\tilde{h},h+2}(t) \) are integers, because \( s^*_{h+1} - s^*_h \) is an integer. Hence the number of variables \( s^*_i \) we have to consider decreases.

Now assume that all \( s^*_i \) (\( i \leq n_k \)) are integers. If \( s^*_{n_k+1} \) is not an integer, the gradient of the objective function for \( s^*_{n_k+1} \)

\[
\frac{\partial}{\partial s^*_{n_k+1}} f_{OSTP}(s^*) = \frac{\partial}{\partial s^*_{n_k+1}} \{q_{n_k,n_k+1}(s^*_{n_k+1} - s^*_{n_k}) + p_{n_k+1}(s^*_{n_k+1})\}
\]

(3.3.20)

exists. Hence, by a similar argument, we can change \( s^*_{n_k+1} \) until it becomes an integer without increasing the objective value. \( \square \)

The lemma indicates that traveling times between customers are nonnegative integers, and hence we can compute \( f^k_h(t) \) by

\[
f^k_h(t) = p_h(t) + \min_{t' \in \{0,1,\ldots,t\}} \{f^k_{h-1}(t') + q_{h-1,h}(t - t')\}, \quad t = 0, 1, 2, \ldots,
\]

(3.3.21)
instead of equation (3.3.15). In order to compute equation (3.3.21), we consider a \( T \times (n_k + 1) \) table whose \((t, h)\) element is \( f^k_h(t)\), where \( T \) is the maximum time that we need to consider. We will discuss the value of \( T \) below. With this table, \( f^k_h(t) \) can be computed in \( O(T) \) time from \( f^{k-1}_{h-1}(0), f^{k-1}_{h-1}(1), \ldots, f^k_h(t) \). Hence, starting from \( f^0_0(0) = 0, f^0_0(1) = f^k_0(2) = \cdots = f^k_0(T) = +\infty \), we can compute all elements of the table in \( O(n_k T^2) \) time.

Now we consider the value of \( T \). Let \( t = P_h \) be the largest breakpoint of the piecewise linear function \( p_h \). Similarly, let \( t = Q_{h-1,h} \) be the largest breakpoint of function \( q_{h-1,h} \). Note that the gradients of the last pieces of \( p_h(t) \) and \( q_{h-1,h}(t) \) are nonnegative because of the nonnegativity of these functions, and hence the gradient of the last piece of \( f^k_h(t) \) is also nonnegative. Then,

\[
T_h = \begin{cases} 
0, & h = 0, \\
\max\{T_{h-1} + Q_{h-1,h}, P_h\}, & h \geq 1
\end{cases} \tag{3.3.22}
\]

represents the maximum time that we need to consider to compute \( f^k_h(t) \). We therefore set \( T = T_{n_k+1} \). Let \( h^* \) be the largest \( h \) that satisfy \( T_{h-1} + Q_{h-1,h} < P_h \) (if \( T_{h-1} + Q_{h-1,h} \geq P_h \) holds for all \( h = 1, 2, \ldots, n_k + 1 \), we set \( h^* = 0 \) and \( P_0 = 0 \) for convenience), i.e., \( T_{h^*} = P_{h^*} \) and \( T_h = T_{h-1} + Q_{h-1,h} \) holds for all \( h > h^* \). Then we have \( T = T_{n_k+1} = P_{h^*} + \sum_{h=h^*+1}^{n_k+1} Q_{h-1,h} \leq \max_{h \in \{1, 2, \ldots, n_k + 1\}} P_h + \sum_{h=1}^{n_k+1} Q_{h-1,h} \). Recall that these functions are all given explicitly as the input. This indicates that the time complexity \( O(n_k T^2) \) is pseudo-polynomial order.

**Theorem 3.3.2** Problem OSTP with integer input can be solved in pseudo polynomial time.

### 3.3.3 Polynomial time algorithm for convex traveling time cost functions

In this section, we propose a polynomial time algorithm for OSTP (3.3.7) in which each traveling time cost function \( q_{h-1,h} \) is convex. Here we do not assume integer input as in Section 3.3.2. Note that time window cost functions \( p_h \) can be general; i.e., \( p_h \) can be non-convex and/or discontinuous functions. If all time window cost functions are also convex, this problem can be formulated as a convex programming problem and efficient algorithms exist [19, 29]. Moreover if all coefficients are integers, this can be a special case of the convex cost integer dual network flow problem and a more general algorithm exists [6].

**Computation of **\( f^k_h \)**

Since functions \( p_h \) and \( q_{h-1,h} \) are piecewise linear, each \( f^k_h \) is also piecewise linear. Therefore we can store all functions in recursion (3.3.15) in linked lists; each cell stores the
interval and the linear function of the corresponding piece, and the cells are linked according to the order of intervals. For example, Figure 3.1 shows a piecewise linear function $g$ and its linked list. Let $\delta(g)$ be the number of linear pieces of a piecewise linear function $g$. For example, the function $g$ in Figure 3.1 has seven linear pieces (i.e., $\delta(g) = 7$). Then it is straightforward to see that summation $g + g'$ of two piecewise linear functions $g$ and $g'$ can be done in $O(\delta(g) + \delta(g'))$ time.

![Figure 3.1: A function $g$ and the linked list that represents $g$](image)

A nontrivial part in recursion (3.3.15) is the computation of $\min_{0 \leq t' \leq t} \{ f^k_{h-1}(t') + q_{h-1,h}(t - t') \}$. Even if $q_{h-1,h}$ is convex, $f^k_{h-1}$ is not necessarily convex. For convenience of explanation, we define a convex interval of a piecewise linear function to be a maximal domain interval on which the function is convex, and let $\hat{\delta}(g)$ be the number of convex intervals of $g$. Then let $S_l, l = 1, 2, \ldots, \hat{\delta}(g)$, denote all convex intervals of $g$, which are labeled in the increasing order of their contents. For example, the function $g$ in Figure 3.1 has three convex intervals $S_1 = (-\infty, 4]$, $S_2 = (4, 10)$ and $S_3 = [10, +\infty)$, and hence $\hat{\delta}(g) = 3$.

We split the entire domain of $f^k_{h-1}$ into convex intervals $S_1, S_2, \ldots, S_K$, where $K = \hat{\delta}(f^k_{h-1})$, and define the following convex functions $F_l$ for $l = 1, 2, \ldots, K$, which are exp-
tended from \( f_{h-1}^k(t) \) on \( S_l \). Let \( \text{Cl}(S_l) = [c_l^1, c_l^R] \) denote the closure of \( S_l \).

\[
F_l(t) = \begin{cases} 
+\infty, & t \notin \text{Cl}(S_l) \\
f_{h-1}^k(t), & t \in S_l \\
\lim_{\varepsilon \to 0} f_{h-1}^k(t + \varepsilon), & t = c_l^1 \\
\lim_{\varepsilon \to 0} f_{h-1}^k(t - \varepsilon), & t = c_l^R.
\end{cases}
\]

Let

\[
e_l(t) = \min_{0 \leq t' \leq t} \{ F_l(t') + q_{h-1,h}(t - t') \}.
\]

Then we have

\[
\min_{0 \leq t' \leq t} \{ f_{h-1}^k(t') + q_{h-1,h}(t - t') \} = \min_{0 \leq t' \leq t} \left\{ \min_{1 \leq l \leq K} \{ F_l(t') + q_{h-1,h}(t - t') \} \right\} = \min_{1 \leq l \leq K} \left\{ \min_{0 \leq t' \leq t} \{ F_l(t') + q_{h-1,h}(t - t') \} \right\} = \min_{1 \leq l \leq K} e_l(t).
\]

That is, \( \min_{0 \leq t' \leq t} \{ f_{h-1}^k(t' + q_{h-1,h}(t - t') \} \) is the lower envelope of functions \( e_1, e_2, \ldots, e_K \).

**Computation of \( e_l \)**

To compute \( e_l(t) \) of (3.3.23), we use the next theorem since both of \( F_l \) and \( q_{h-1,h} \) are convex piecewise linear.

**Theorem 3.3.3** Suppose that two lower semicontinuous convex piecewise linear functions \( \phi_1 \) and \( \phi_2 \) are stored in linked lists. Then we can compute function

\[
\phi(t) = \min_{0 \leq t' \leq t} \{ \phi_1(t') + \phi_2(t - t') \}
\]

in \( \mathcal{O}(\delta(\phi)) \) time, where \( \delta(\phi) \leq \delta(\phi_1) + \delta(\phi_2) \) holds. Moreover \( \phi \) is convex.

**Proof.** Let us consider the plane whose horizontal axis is \( x_1 \) and vertical axis is \( x_2 \), and consider \( \phi_1(x_1) + \phi_2(x_2) \). This is shown in Figure 3.2, where vertical and horizontal broken lines represent breakpoints of functions \( \phi_1 \) and \( \phi_2 \), respectively. Then \( \phi(t) \) is given as the minimum of \( \phi_1(x_1) + \phi_2(x_2) \) on the line \( x_1 + x_2 = t \). First, we consider the minimum point of \( \phi_1(x_1) + \phi_2(x_2) \) on the line segment AB in Figure 3.2. The shaded rectangle that contains AB corresponds to one linear piece of \( \phi_1(x_1) \) and another of \( \phi_2(x_2) \). This means that \( \phi_1(x_1) + \phi_2(x_2) = \phi_1(x_1) + \phi_2(t - x_1) = \phi_1(t - x_2) + \phi_2(x_2) \) is a linear function of \( x_1 \) (or equivalently of \( x_2 \)) on AB; hence either point A or point B achieves its minimum. Since similar argument applies to all rectangular regions of broken lines, \( \phi(t) \) is achieved on one of the points where line \( x_1 + x_2 = t \) and broken lines meet.
3.3 Optimal start times of services

Now we show the continuity of the points which achieve \( \phi(t) \). Let a point \((x_1^*(t), x_2^*(t))\) achieve \( \phi(t) \) (if there is more than one such point, we take the one whose \( x_1^*(t) \) is smallest). We then derive a contradiction from assumption \( x_1^*(t + \varepsilon) < x_1^*(t) \). Figure 3.3 shows this situation, where assume that points A and C achieve \( \phi(t) \) and \( \phi(t + \varepsilon) \), and points B and D are their projections to lines \( x_1 + x_2 = t + \varepsilon \) and \( x_1 + x_2 = t \), respectively, where \( \varepsilon \) is an arbitrarily small positive number. Then, the value \( \phi_1(x_1) + \phi_2(x_2) \) at C is less than or equal to that at B, and the value at A is exactly less than that at D. Hence the increment from A to B is greater than the increment from D to C, which is a contradiction to the convexity of function \( \phi_2 \). We then have \( x_1^*(t) \leq x_1^*(t + \varepsilon) \). Similarly we have \( x_2^*(t) \leq x_2^*(t + \varepsilon) \) (i.e., \( t - x_1^*(t) \leq t + \varepsilon - x_1^*(t + \varepsilon) \)). Then we have

\[
x_1^*(t) \leq x_1^*(t + \varepsilon) \leq x_1^*(t) + \varepsilon. \tag{3.3.24}
\]
Thus trajectory \((x_1^*(t), x_2^*(t))\) for \(t = 0 \rightarrow +\infty\) is continuous and lies on the lattice of broken lines (see Figure 3.2), i.e., it is a nondecreasing staircase curve from \((0,0)\) to its top right as shown in Figure 3.4.

![Figure 3.4: An example of the trajectory of \((x_1^*(t), x_2^*(t))\) which achieves \(\phi(t)\)](image)

In order to compute \(\phi\), we walk on the lattice of broken lines from \((0,0)\), selecting the direction (i.e., upward or rightward) with a smaller gradient at each intersection. Figure 3.4 shows such an example, where the numbers on \(x_1\) and \(x_2\) axes denote the gradients of the corresponding intervals of linear pieces of \(\phi_1\) and \(\phi_2\). Whenever we determine the direction at each intersection, we compute the data of \(\phi\) for the corresponding interval, and add it to the linked list that represents \(\phi(t)\). Note that the gradient of \(\phi\) is the same as that of the selected piece of \(\phi_i\). As it is not difficult to show that the gradients of linear pieces of \(\phi\) added to the list are nondecreasing, the computed \(\phi\) is also convex.

The time complexity of this algorithm is clearly \(O(\delta(\phi))\). It is also clear from the above argument that \(\delta(\phi) \leq \delta(\phi_1) + \delta(\phi_2)\) holds. \(\square\)

**Lower envelope of all \(e_l\)**

After obtaining convex functions \(e_l(t), l = 1, 2, \ldots, K\) by the algorithm described in Section 3.3.3, we compute their lower envelope. We show in this subsection that the time for this computation is \(O(\sum_{l=1}^{K} \delta(e_l))\). For convenience, let \(E_L(t) = \min_{1 \leq l \leq L} e_l(t)\). In general, the information of the lower envelope includes

- the set of functions \(e_l\) which appear in the lower envelope \(E_K\),
- their order, and
- the crossing point of \(e_l\) and \(e_{l'}\) for each adjacent pair.

We use the following Lemma 3.3.2 and Theorem 3.3.4 to obtain these data.
Lemma 3.3.2 If $l < l'$, then the right differential coefficient of $e_l(t)$ is greater than or equal to that of $e_{l'}(t)$ at any $t$.

**Proof.** Let us consider $\phi(t) = e_l(t)$ and the trajectory $(x_1^l(t), x_2^l(t))$ achieving $\phi(t)$ (e.g., Figure 3.4), where in this case the horizontal axis denotes start time $s_{h-1}$ and the vertical axis denotes traveling time $t_{h-1}$. Figure 3.5 illustrates the situation in which there are two trajectories corresponding to $e_l(t)$ and $e_{l'}(t)$. For a given $t$, let $t^*_l = \arg \min_{0 \leq t' \leq t} \{ F_l(t') + q_{h-1,h}(t-t') \}$, i.e., $e_l(t) = F_l(t^*_l) + q_{h-1,h}(t-t^*_l)$, and $t^*_l = \arg \min_{0 \leq t' \leq t} \{ F_{l'}(t') + q_{h-1,h}(t-t') \}$. Then $t - t^*_l \geq t - t^*_l$ holds, since $t^*_l \in \text{Cl}(S_l)$ and $t^*_l \in \text{Cl}(S_{l'})$. If $t^*_l = t^*_l$ the lemma holds, since the right differential coefficient of $e_l(t)$ is equal to that of $q_{h-1,h}(t-t^*_l)$ and the right differential coefficient of $e_{l'}(t)$ is less than or equal to that of $q_{h-1,h}(t-t^*_l)$. Then we assume $t^*_l < t^*_l$. The right differential coefficient of $e_l(t)$ is greater than or equal to that of $q_{h-1,h}(t-t^*_l - \varepsilon)$ where $\varepsilon$ is an arbitrarily small positive number. The right differential coefficient of $e_{l'}(t)$ is less than or equal to that of $q_{h-1,h}(t-t^*_l + \varepsilon)$. Since $q_{h-1,h}$ is convex and $t - t^*_l > t - t^*_l$, the right differential coefficient of $q_{h-1,h}(t-t^*_l - \varepsilon)$ is greater than or equal to that of $q_{h-1,h}(t-t^*_l + \varepsilon)$. Hence the right differential coefficient of $e_l(t)$ is greater than or equal to that of $e_{l'}(t)$ at any $t$. □

**Theorem 3.3.4** Each $e_l$ appears in the envelope $E_L$ ($l \leq L$), at most once and the order of their appearances preserves the order of their indices $l$.

**Proof.** Consider an adjacent pair of $e_l(t)$ and $e_{l'}(t)$ ($l < l'$), which appear in $E_L$. Lemma 3.3.2 implies that $e_{l'}(t) - e_l(t)$ is nonincreasing with $t$; hence $e_l$ and $e_{l'}$ cross
at most once, and if they cross, the sign of \( e_l(t) - e_l(t) \) changes from positive to negative. This tells that each \( e_l \) appears in \( E_L \) at most once and the order of their appearances is \( l \) before \( l' \). □

The computation of the lower envelope \( E_K \) proceeds as follows.

**Compute-Lower-Envelope**

*Input:* Functions \( e_1, e_2, \ldots, e_K \).

*Output:* Their lower envelope \( E_K \).

**Step 1**  Let \( L := 1 \).

**Step 2**  If \( L = K + 1 \), then halt. Otherwise, compute \( E_L(t) \) from \( E_{L-1}(t) \) and \( e_L(t) \), let \( L := L + 1 \) and return to Step 2.

In Step 2, all we have to do is to check how \( e_L \) affects \( E_{L-1}(t) \). We illustrate how to compute \( E_L(t) \) from \( E_{L-1}(t) \) with an example of Figure 3.6. Assume that \( E_{L-1}(t) \) consists of \( v \) \((\leq L-1)\) functions \( e_{l_1}, e_{l_2}, \ldots, e_{l_v} \), and \( e_{l_{r-1}} \) and \( e_{l_r} \) cross at \( t_r \) (i.e., \( e_{l_{r-1}}(t_r) = e_{l_r}(t_r) \)) for \( r = 2, 3, \ldots, v \), where we assume \( t_1 = -\infty \) and \( t_{v+1} = +\infty \) for convenience. We first find the largest \( r = r^* \) that satisfy \( e_{l_r}(t_r) \leq e_L(t_r) \) by scanning the list of \( t_r \)'s from \( r = v + 1 \) to 1 and scanning \( e_L \) from right to left. If \( r^* \) does not exist (i.e., \( e_{l_r}(t_r) > e_L(t_r) \) holds for all \( r = 1, 2, \ldots, v + 1 \)), then \( E_L(t) = e_L(t) \) holds. If \( r^* = v + 1 \), then \( E_L(t) = E_{L-1}(t) \) holds. Otherwise (i.e., \( r^* \in [2, v] \)), \( e_L \) crosses with \( e_{l_{r^*}} \). In this case, we find the point \( t^* \) where \( e_{l_{r^*}} \) and \( e_{l_{r^*}} \) cross with \( e_{l_{r^*}} \) by scanning \( e_L \) from right to left and scanning \( e_{l_{r^*}} \) to the left from the linear piece whose interval includes \( t_{r^*+1} \). Then we compute \( E_L(t) \) by concatenating \( E_{L-1}(t) \) for \( t \leq t^* \) and \( e_L(t) \) for \( t \geq t^* \). In order to execute the above computation efficiently, we keep an array that stores \( t_r \) and \( e_{l_r}(t_r) \) for all \( r \), and a pointer to the linear piece of \( e_{l_{r-1}}(t) \) whose interval includes \( t_r \).

Now we estimate the time complexity of the above algorithm. Note that, once we know that \( e_{l_r}(t_r) > e_L(t_r) \) holds, we can remove \( t_r \) from the list, because \( e_{l_r}(t) > e_L(t) \) holds for
all $t \geq t_r$ and hence $e_{t_r}(t)$ is removed from the envelope. This implies that we can keep the list of $t_r$’s as a stack, and the time complexity of maintaining the stack during the whole computation of Compute-Lower-Envelope is $O(K)$ because at most $K$ elements are inserted into the stack and an element is removed from the stack once the element next to it is scanned. It therefore takes

$$O \left( \sum_{l=1}^{K} \delta(e_l) + K \right) = O \left( \sum_{l=1}^{K} \delta(e_l) \right)$$

time to find $r^*$ for all $L = 2, 3, \ldots, K$. For the same reason, in the computation of finding the point $t^*$ where $e_L$ and $e_{t_r^*}$ cross, a linear piece of $e_{t_r^*}$ is removed from the list of the lower envelope and will not be checked again once the linear piece to its left is scanned. This implies that the total number of linear pieces scanned from $E_{L-1}(t)$ for all $L = 2, 3, \ldots, K$ is $O(\sum_{l=1}^{K-1} \delta(e_l))$. Hence the total time complexity of algorithm Compute-Lower-Envelope is $O(\sum_{l=1}^{K} \delta(e_l))$.

**Time complexity for the dynamic programming**

In order to compute $f^k_h$ from $f^k_{h-1}$ by recursion (3.3.15), we execute the following three steps:

1. Compute functions $e_1(t), e_2(t), \ldots, e_{\hat{\delta}(f^k_{h-1})}(t)$, where $\hat{\delta}(f^k_{h-1}) (= K)$ is the number of convex intervals of $f^k_{h-1}(t)$.

2. Compute their lower envelope $E_{\hat{\delta}(f^k_{h-1})}(t)$, which gives $\min \{ f^k_{h-1}(t') + q_{h-1,h}(t-t') \}$ in recursion (3.3.15).

3. Compute $p_h(t) + E_{\hat{\delta}(f^k_{h-1})}(t)$, i.e., $f^k_h(t)$.

Time complexity of these three steps is as follows.

1. Since the computation of $e_l$ takes $O(\delta(e_l))$ time for each $l$, it takes $O(\sum_{l=1}^{\hat{\delta}(f^k_{h-1})} \delta(e_l))$ time to compute all $e_1, e_2, \ldots, e_{\hat{\delta}(f^k_{h-1})}$.

2. The lower envelope $E_{\hat{\delta}(f^k_{h-1})}$ can be computed from $e_1, e_2, \ldots, e_{\hat{\delta}(f^k_{h-1})}$ in $O(\sum_{l=1}^{\hat{\delta}(f^k_{h-1})} \delta(e_l))$ time by algorithm Compute-Lower-Envelope.

3. We can add $p_h$ to the lower envelope $E_{\hat{\delta}(f^k_{h-1})}$ in $O(\delta(p_h) + \sum_{l=1}^{\hat{\delta}(f^k_{h-1})} \delta(e_l))$ time, since the lower envelope $E_{\hat{\delta}(f^k_{h-1})}$ has at most $\sum_{l=1}^{\hat{\delta}(f^k_{h-1})} \delta(e_l)$ linear pieces.
Hence, we can compute \( f^k_h \) from \( f^k_{h-1} \) in \( O(\delta(p_h) + \sum_{l=1}^{\hat{\delta}(f^k_{h-1})} \delta(e_l)) \) time. For convenience, we introduce the following notations,

\[
\Delta^k_p = \sum_{h=1}^{n_k+1} \delta(p_h), \quad \Delta^k_q = \sum_{h=1}^{n_k+1} \delta(q_{h-1,h}).
\]

For the number of convex intervals,

\[
\hat{\delta}(f^k_h) \leq \hat{\delta}(f^k_{h-1}) + \delta(p_h) - 1 \tag{3.3.25}
\]

holds, because the number of convex intervals of the lower envelope \( E_{\hat{\delta}(f^k_{h-1})} \) is less than or equal to \( \hat{\delta}(f^k_{h-1}) \). For the number of linear pieces of \( f^k_h \),

\[
\delta(f^k_h) \leq \delta(p_h) + \sum_{l=1}^{\hat{\delta}(f^k_{h-1})} \delta(e_l) \\
\leq \delta(p_h) + \sum_{l=1}^{\hat{\delta}(f^k_{h-1})} \{ \delta(F_l) + \delta(q_{h-1,h}) \} \\
\leq \delta(p_h) + \delta(f^k_{h-1}) + \hat{\delta}(f^k_{h-1}) \delta(q_{h-1,h}) \tag{3.3.26}
\]

holds. Note that the first inequality holds because \( \delta(E_{\hat{\delta}(f^k_{h-1})}) \leq \sum_{l=1}^{\hat{\delta}(f^k_{h-1})} \delta(e_l) \), and the second inequality is from Theorem 3.3.3. By applying (3.3.25) recursively, we have \( \hat{\delta}(f^k_h) = O(\Delta^k_p) \). Similarly, from (3.3.26), we have \( \delta(f^k_h) = O(\Delta^k_p + \Delta^k_q) \). Consequently, the time to compute \( f^k_h \) from \( f^k_{h-1} \) is

\[
O\left( \delta(p_h) + \sum_{l=1}^{\hat{\delta}(f^k_{h-1})} \delta(e_l) \right) = O\left( \delta(p_h) + \delta(f^k_{h-1}) + \hat{\delta}(f^k_{h-1}) \delta(q_{h-1,h}) \right) \\
= O(\Delta^k_p + \Delta^k_q) \Delta^k_q).
\]

Then the time complexity of computing all \( f^k_1, f^k_2, \ldots, f^k_{n_k+1} \) is \( O\left( n_k(\Delta^k_p + \Delta^k_q) \right) \). Since all linear pieces of input functions are explicitly given as linked lists, the input size is \( \Delta^k_p + \Delta^k_q \). Thus the above time complexity is polynomial in the input size.

In summary, for a given route \( \sigma_k \), we can compute the optimal start times of services at customers in \( O(n_k \Delta(\sigma_k)) \) time, where

\[
\Delta(\sigma_k) = \sum_{h=1}^{n_k+1} \delta(p_{\sigma_k(h)}) + \left( \sum_{h=1}^{n_k+1} \hat{\delta}(p_{\sigma_k(h)}) \right) \left( \sum_{h=1}^{n_k+1} \delta(q_{\sigma_k(h-1), \sigma_k(h)}) \right) \tag{3.3.27}
\]
3.4 Local search for finding visiting orders $\sigma$

In this section, we describe a framework of our local search (LS) for finding good visiting orders $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ that satisfy condition (3.2.6). It starts from an initial solution $\sigma$ and repeats replacing $\sigma$ with a better solution in its neighborhood $N(\sigma)$ until no better solution is found in $N(\sigma)$. We use the standard neighborhoods $N(\sigma)$ called 2-opt*, cross exchange and Or-opt neighborhoods with slight modifications (see Figure 3.7).

A 2-opt* operation removes two edges from two different routes (one from each) to divide each route into two parts and exchanges the second parts of the two routes (See Section 2.4.2). A cross exchange operation removes two paths from two routes (one from each) of different vehicles, whose length (i.e., the number of customers in the path) is at most $L_{\text{cross}}$ (a parameter), and exchanges them (See Section 2.4.4). The cross exchange and 2-opt* operations always change the assignment of customers to vehicles. We also use the intra-route neighborhood to improve individual routes. An intra-route operation removes a path of length at most $L_{\text{intra}}$ (a parameter) and inserts it into another position of the same route, where the position is limited within length $L_{\text{ins}}$ (a parameter) from the original position (See Section 2.4.5). Our LS searches the above intra-route neighborhood, 2-opt* neighborhood and cross exchange neighborhood, in this order. Whenever a better solution is found, we immediately accept it (i.e., we adopt the first admissible move strategy), and resume the search from the intra-route neighborhood.

As only one execution of LS may not be sufficient to find a good solution, we use the iterated local search (ILS) [107], which iterates LS many times from those initial solutions generated by perturbing good solutions obtained by then. We perturb a solution by applying one random cross exchange operation with no restriction on $L_{\text{cross}}$ (i.e., $L_{\text{cross}} =$

![Figure 3.7: Neighborhoods in our local search](image)
ILS is summarized as follows:

**Algorithm: Iterated Local Search (ILS)**

**Step 1** Generate an initial solution $\sigma^0$. Let $\sigma^{\text{seed}} := \sigma^0$ and $\sigma^{\text{best}} := \sigma^0$.

**Step 2** Improve $\sigma^{\text{seed}}$ by LS and let $\sigma$ be the improved solution.

**Step 3** If $\sigma$ is better than $\sigma^{\text{best}}$ then replace $\sigma^{\text{best}}$ with $\sigma$.

**Step 4** If some stopping criterion is satisfied, output $\sigma^{\text{best}}$ and halt; otherwise generate a solution $\sigma^{\text{seed}}$ by perturbing $\sigma^{\text{best}}$ and return to Step 2.

### 3.5 Efficient implementation of local search

A solution $\sigma$ is evaluated by $d_{\text{sum}}(\sigma) + a_{\text{sum}}(\sigma) + (p + q)_{\text{sum}}^*(\sigma)$, where $(p + q)_{\text{sum}}^*(\sigma)$ denotes the minimum time window and traveling time cost for $\sigma$. For this, it is important to see that dynamic programming computation of $(p + q)_{\text{sum}}^*(\sigma)$ for the solutions in neighborhoods can be sped up by using information from the previous computation. The efficient neighborhood search method in Section 2.6 can be applied. In this section, for convenience, we discuss the case in which we use the polynomial time algorithm for OSTP in Section 3.3.3. But the idea is also applicable to the case of the pseudo polynomial time algorithm in Section 3.3.2.

#### 3.5.1 Basic idea

Let us consider to compute the minimum cost of a route $\sigma_k = (\sigma_k(0), \sigma_k(1), \ldots, \sigma_k(n_k + 1))$ (where the cost is composed of the distance, the amount of capacity excess, the time window cost and the traveling time cost) by connecting its former part $\sigma_k(0) \rightarrow \sigma_k(1) \rightarrow \cdots \rightarrow \sigma_k(h)$ and latter part $\sigma_k(h + 1) \rightarrow \sigma_k(h + 2) \rightarrow \cdots \rightarrow \sigma_k(n_k + 1)$ for some $h$.

![Figure 3.8: The former and latter parts of a route $\sigma_k$](image)

In this scheme, the distance of route $\sigma_k$ is computed in $O(1)$ time, from distances of its former and latter parts. The amount of capacity excess on route $\sigma_k$ is also computed in $O(1)$ time, if both $\sum_{i=1}^{h} a_{\sigma_k(i)}$ and $\sum_{i=h+1}^{n_k} a_{\sigma_k(i)}$ are known. We therefore store
\[ \sum_{i=1}^{h} d_{\sigma_k(i-1),\sigma_k(i)}, \sum_{i=h+1}^{n_k+1} d_{\sigma_k(i-1),\sigma_k(i)}, \sum_{i=1}^{h} a_{\sigma_k(i)} \text{ and } \sum_{i=h}^{n_k} a_{\sigma_k(i)} \text{ for each customer } \sigma_k(h) \text{ and vehicle } k \text{ whenever the current route is updated.} \]

Now we concentrate on the computation of the minimum cost \((p + q)^*\text{sum}(\sigma_k)\), which is the sum of time window and traveling time costs on route \(\sigma_k\). We define \(b^k_h(t)\) to be the minimum sum of the costs for customers \(\sigma_k(h),\sigma_k(h+1),\ldots,\sigma_k(n_k),\sigma_k(n_k+1)\), provided that all of them are served after time \(t\) and customer \(\sigma_k(h)\) is served exactly at time \(t\) (i.e., \(\min_{\sigma_k(h) = t} \sum_{i=h}^{n_k+1} p_{\sigma_k(i)}(s_{\sigma_k(i)}) + \sum_{i=h+1}^{n_k+1} q_{\sigma_k(i)}(s_{\sigma_k(i)} - s_{\sigma_k(i-1)})\)). We call this a backward minimum cost function. Let \(f^k_h(t)\) be the forward minimum cost function at the \(h\)th customer in route \(\sigma_k\), which was discussed in Section 3.3. Then, \(b^k_h(t)\) can be computed as follows in a symmetric manner:

\[
\begin{align*}
\sigma_k(h) + 1) + min_{t'}(b^k_{h+1}(t') + q_{h,h+1}(t' - t)), & \quad 1 \leq h \leq n_k. \\
\end{align*}
\]

We can then obtain the optimal cost of route \(\sigma_k\) by

\[ (p + q)^*\text{sum}(\sigma_k) = \min_t \left( f^k_h(t) + \min_{t'}(b^k_{h+1}(t') + q_{h,h+1}(t' - t)) \right) \quad (3.5.29) \]

for any \(h\) (\(0 \leq h \leq n_k\)). If \(f^k_h(t)\) and \(b^k_{h+1}(t)\) are already available for some \(h\), this is possible in \(O(\Delta(\sigma_k))\) time, because \(f^k_h(t)\) and \(b^k_{h+1}(t)\) consist of \(O(\Delta(\sigma_k))\) linear pieces and \(\min_{t'}(b^k_{h+1}(t') + q_{h,h+1}(t' - t))\) can be computed in \(O(\Delta(\sigma_k))\) time as explained in Section 3.3.3 (for the case of \(f^k_h(t)\)). To achieve this, we store all functions \(f^k_h(t)\) and \(b^k_h(t)\) for each customer \(\sigma_k(h)\), when they were computed in the process of LS.

In summary, we can compute the minimum cost of route \(\sigma_k\) in \(O(\Delta(\sigma_k))\) time, if we keep the data

\[
\begin{align*}
\sum_{i=1}^{h} a_{\sigma_k(i)} & \quad \text{and} \quad \sum_{i=h}^{n_k} a_{\sigma_k(i)}, \\
\sum_{i=1}^{h} d_{\sigma_k(i-1),\sigma_k(i)} & \quad \text{and} \quad \sum_{i=h+1}^{n_k+1} d_{\sigma_k(i-1),\sigma_k(i)}, \\
f^k_h(t) & \quad \text{and} \quad b^k_h(t) \\
\end{align*}
\]

(3.5.30)

(3.5.31)

(3.5.32)

for all \(h = 1,2,\ldots,n_k\) and \(k \in M\).

### 3.5.2 How to apply the basic idea to the solutions in neighborhoods

Now we explain how to apply the above idea to the solutions in neighborhoods. We only discuss the sum of time window and traveling costs since other costs can be similarly treated.
In Figure 3.9, an example of a 2-opt* operation on routes $\sigma_k$ and $\sigma_k'$ is shown. The sum of time window and traveling time costs for $\sigma_k$, after a 2-opt* operation is applied, can be computed by

$$\min_t \left( f_h^k(t) + \min_{t'} (b_{h_k+1}^{k'}(t') + q_{\sigma_k(h_k),\sigma_k'(h_k'+1)}(t' - t)) \right)$$

in $O(\Delta(\sigma_k))$ time. Similarly the cost for $\sigma_k'$ can be computed in in $O(\Delta(\sigma_k'))$ time. Hence we can evaluate the cost of the resulting solution in $O(\Delta(\sigma_k) + \Delta(\sigma_k'))$ time, when a 2-opt* operation is applied to routes $\sigma_k$ and $\sigma_k'$.

To evaluate solutions in the cross exchange neighborhood efficiently (see Figure 3.7), we need to search the solutions in the neighborhood in a specific order. To apply cross exchange operations on routes $\sigma_k$ and $\sigma_k'$, we start from a solution obtainable by exchanging
3.6 Computational results

We conducted computational experiments to evaluate the proposed algorithm ILS (see Section 3.4). The algorithm was coded in C language and run on a handmade PC (Intel Pentium 4, 2.8 GHz, 1 GB memory).

We use the benchmark instances by Solomon [140] which have been widely used in the literature. The number of customers in each instance is 100, and their locations are distributed in the square \([0, 100]^2\) in the plane. The distances between customers are measured by Euclidean distance (in double precision), and the traveling times are the same as the corresponding distances. Each customer \(i\) (including the depot) has one time window \([t_{\text{r}i}, t_{\text{d}i}]\), an amount of requirement \(a_i\) and a service time \(\tau_i\). All vehicles have an identical capacity \(u\). Both time window and capacity constraints are considered hard. For these instances, the number of vehicles \(m\) is also a decision variable, and the objective is to find a solution with the minimum \((m, d_{\text{sum}}(\sigma))\) in the lexicographical order. These benchmark instances consist of six different sets of problem instances called R1, R2, RC1, RC2, C1 and C2, respectively. Locations of customers are uniformly and randomly distributed in type R and are clustered in groups in type C, and these two types are mixed in type RC. Furthermore, for instances of type 1, the time window is narrow at the depot, and hence only a small number of customers can be served by one vehicle. Conversely, for instances of type 2, the time window is wide, and hence many customers can be served by one vehicle. Table 3.1 is the best known solutions for these instances (the data was taken as of
To evaluate our algorithm, we modified those instances by introducing time window cost function $p_i$ and traveling time cost function $q_{ij}$ as follows:

$$p_i(t) = \begin{cases} 
\alpha_1(t^r_i - t), & t < t^r_i \\
0, & t^r_i \leq t \leq t^d_i \\
\alpha_1(t - t^d_i), & t^d_i < t, 
\end{cases}$$

$$q_{ij}(t) = \begin{cases} 
+\infty, & t < 0.9(\tau_i + t_{ij}) \\
\alpha_2(\tau_i + t_{ij} - t), & 0.9(\tau_i + t_{ij}) \leq t < \tau_i + t_{ij} \\
0, & \tau_i + t_{ij} \leq t, 
\end{cases}$$

where $\alpha_1$ and $\alpha_2$ are positive parameters. For other parameters, we used $L^\text{cross} = 3$, $L^\text{intra} = 3$ and $L^\text{intra} = 20$, and set the time limit of computation to 2000 seconds (in conformity with the values in [85]). Note that, in this formulation, time window and traveling time constraints are considered soft, and they can be violated if it is advantageous from the view point of minimizing the cost functions.

Our results are shown in Tables 3.2 and 3.3. In each table, column “$P$” denotes the total deviation of start time of services from the boundaries of time windows (i.e., $P = \frac{p_{\text{sum}}(s)}{\alpha_1}$), and column “$Q$” denotes the total amount of shortened traveling time (i.e., $Q = \frac{q_{\text{sum}}(\sigma, s)}{\alpha_2}$). A number in parentheses is the number of customers (resp., edges) at which the time window (resp., traveling time) constraint is violated. A mark “∗” in columns “$d_{\text{sum}}$” and “feasible” means that the value is smaller than or equal to that of the best known solution. In Table 3.2, we set the number of vehicles to be the same as the best known solutions in Table 3.1, and set $\alpha_1 = \alpha_2 = 10$. We determined $\alpha_1$ and $\alpha_2$ after some preliminary trials so that our solutions do not violate the constraints too much. Column “feasible” shows the traveling distance of the solution if it is feasible (i.e., $p_{\text{sum}}(s) = q_{\text{sum}}(\sigma, s) = a_{\text{sum}}(\sigma) = 0$), otherwise “–“ is written, which means that our algorithm encountered no feasible solution. In Table 3.3, on the other hand, we set the number of vehicles to be smaller by one than that of the best known solution, and set $\alpha_1 = \alpha_2 = 100$. Column “$P_{\text{max}}$” denotes the maximum deviation of start time of services from time windows (i.e., $P_{\text{max}} = \max\{p_i(s_i)/\alpha_1 \mid i \in V\}$).

In Table 3.2, we observe that our algorithm could obtain the same quality as the best known solutions in almost all instances for type C. For types R and RC, there are many solutions whose $P$ and $Q$ are nonzero. But since the width of the depot’s time window is 230 for type R1, 240 for type RC1, 1000 for type R2, 960 for type RC2, the violation of time windows is less than 1% of the whole scheduling period in almost all instances. Also the percentage of the shortened traveling time against the total of original traveling times $d_{\text{sum}}$ is less than 0.5%. Hence these violations may be acceptable in many
Table 3.1: The best known solutions for Solomon’s instances

<table>
<thead>
<tr>
<th>instance</th>
<th>number of vehicles</th>
<th>distance</th>
<th>instance</th>
<th>number of vehicles</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>R101</td>
<td>19</td>
<td>1645.79</td>
<td>R201</td>
<td>4</td>
<td>1252.37</td>
</tr>
<tr>
<td>R102</td>
<td>17</td>
<td>1486.12</td>
<td>R202</td>
<td>3</td>
<td>1191.70</td>
</tr>
<tr>
<td>R103</td>
<td>13</td>
<td>1292.68</td>
<td>R203</td>
<td>3</td>
<td>939.54</td>
</tr>
<tr>
<td>R104</td>
<td>9</td>
<td>1007.24</td>
<td>R204</td>
<td>2</td>
<td>825.52</td>
</tr>
<tr>
<td>R105</td>
<td>14</td>
<td>1377.11</td>
<td>R205</td>
<td>3</td>
<td>994.42</td>
</tr>
<tr>
<td>R106</td>
<td>12</td>
<td>1251.98</td>
<td>R206</td>
<td>3</td>
<td>906.14</td>
</tr>
<tr>
<td>R107</td>
<td>10</td>
<td>1104.66</td>
<td>R207</td>
<td>2</td>
<td>893.33</td>
</tr>
<tr>
<td>R108</td>
<td>9</td>
<td>960.88</td>
<td>R208</td>
<td>2</td>
<td>726.75</td>
</tr>
<tr>
<td>R109</td>
<td>11</td>
<td>1194.73</td>
<td>R209</td>
<td>3</td>
<td>909.16</td>
</tr>
<tr>
<td>R110</td>
<td>10</td>
<td>1118.59</td>
<td>R210</td>
<td>3</td>
<td>939.34</td>
</tr>
<tr>
<td>R111</td>
<td>10</td>
<td>1096.72</td>
<td>R211</td>
<td>2</td>
<td>892.71</td>
</tr>
<tr>
<td>R112</td>
<td>9</td>
<td>982.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C101</td>
<td>10</td>
<td>828.94</td>
<td>C201</td>
<td>3</td>
<td>591.56</td>
</tr>
<tr>
<td>C102</td>
<td>10</td>
<td>828.94</td>
<td>C202</td>
<td>3</td>
<td>591.56</td>
</tr>
<tr>
<td>C103</td>
<td>10</td>
<td>828.06</td>
<td>C203</td>
<td>3</td>
<td>591.17</td>
</tr>
<tr>
<td>C104</td>
<td>10</td>
<td>824.78</td>
<td>C204</td>
<td>3</td>
<td>590.60</td>
</tr>
<tr>
<td>C105</td>
<td>10</td>
<td>828.94</td>
<td>C205</td>
<td>3</td>
<td>588.88</td>
</tr>
<tr>
<td>C106</td>
<td>10</td>
<td>828.94</td>
<td>C206</td>
<td>3</td>
<td>588.49</td>
</tr>
<tr>
<td>C107</td>
<td>10</td>
<td>828.94</td>
<td>C207</td>
<td>3</td>
<td>588.29</td>
</tr>
<tr>
<td>C108</td>
<td>10</td>
<td>828.94</td>
<td>C208</td>
<td>3</td>
<td>588.32</td>
</tr>
<tr>
<td>C109</td>
<td>10</td>
<td>828.94</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC101</td>
<td>14</td>
<td>1696.94</td>
<td>RC201</td>
<td>4</td>
<td>1406.91</td>
</tr>
<tr>
<td>RC102</td>
<td>12</td>
<td>1554.75</td>
<td>RC202</td>
<td>3</td>
<td>1365.645</td>
</tr>
<tr>
<td>RC103</td>
<td>11</td>
<td>1261.67</td>
<td>RC203</td>
<td>3</td>
<td>1049.62</td>
</tr>
<tr>
<td>RC104</td>
<td>10</td>
<td>1135.48</td>
<td>RC204</td>
<td>3</td>
<td>798.41</td>
</tr>
<tr>
<td>RC105</td>
<td>13</td>
<td>1629.44</td>
<td>RC205</td>
<td>4</td>
<td>1297.19</td>
</tr>
<tr>
<td>RC106</td>
<td>11</td>
<td>1424.73</td>
<td>RC206</td>
<td>3</td>
<td>1146.32</td>
</tr>
<tr>
<td>RC107</td>
<td>11</td>
<td>1230.48</td>
<td>RC207</td>
<td>3</td>
<td>1061.14</td>
</tr>
<tr>
<td>RC108</td>
<td>10</td>
<td>1139.82</td>
<td>RC208</td>
<td>3</td>
<td>828.14</td>
</tr>
</tbody>
</table>
Table 3.2: Computational results on Solomon’s instances

<table>
<thead>
<tr>
<th>instance</th>
<th>$d_{sum}$</th>
<th>$P$</th>
<th>$Q$</th>
<th>feasible</th>
</tr>
</thead>
<tbody>
<tr>
<td>R101</td>
<td>*1616.54</td>
<td>0.57(2)</td>
<td>0.12(1)</td>
<td>1695.80</td>
</tr>
<tr>
<td>R102</td>
<td>*1422.78</td>
<td>1.73(2)</td>
<td>0.54(2)</td>
<td>1252.37</td>
</tr>
<tr>
<td>R103</td>
<td>*1174.57</td>
<td>3.23(2)</td>
<td>1.42(1)</td>
<td>1252.37</td>
</tr>
<tr>
<td>R104</td>
<td>1018.67</td>
<td>0.08(1)</td>
<td>1019.55</td>
<td></td>
</tr>
<tr>
<td>R105</td>
<td>*1372.18</td>
<td>0.10(1)</td>
<td>*1377.11</td>
<td></td>
</tr>
<tr>
<td>R106</td>
<td>1257.96</td>
<td>0</td>
<td>1257.96</td>
<td></td>
</tr>
<tr>
<td>R107</td>
<td>1122.82</td>
<td>0.14(1)</td>
<td>1125.62</td>
<td></td>
</tr>
<tr>
<td>R108</td>
<td>967.05</td>
<td>0.34(1)</td>
<td>989.05</td>
<td></td>
</tr>
<tr>
<td>R109</td>
<td>1197.42</td>
<td>0</td>
<td>1197.42</td>
<td></td>
</tr>
<tr>
<td>R110</td>
<td>1142.81</td>
<td>0.58(1)</td>
<td>1150.28</td>
<td></td>
</tr>
<tr>
<td>R111</td>
<td>1096.73</td>
<td>0</td>
<td>1096.73</td>
<td></td>
</tr>
<tr>
<td>R112</td>
<td>986.41</td>
<td>0</td>
<td>986.41</td>
<td></td>
</tr>
<tr>
<td>C101</td>
<td>*828.94</td>
<td>0</td>
<td>*828.94</td>
<td></td>
</tr>
<tr>
<td>C102</td>
<td>*828.94</td>
<td>0</td>
<td>*828.94</td>
<td></td>
</tr>
<tr>
<td>C103</td>
<td>*828.06</td>
<td>0</td>
<td>*828.06</td>
<td></td>
</tr>
<tr>
<td>C104</td>
<td>*824.78</td>
<td>0</td>
<td>*824.78</td>
<td></td>
</tr>
<tr>
<td>C105</td>
<td>*828.94</td>
<td>0</td>
<td>*828.94</td>
<td></td>
</tr>
<tr>
<td>C106</td>
<td>*828.94</td>
<td>0</td>
<td>*828.94</td>
<td></td>
</tr>
<tr>
<td>C107</td>
<td>*828.94</td>
<td>0</td>
<td>*828.94</td>
<td></td>
</tr>
<tr>
<td>C108</td>
<td>*828.94</td>
<td>0</td>
<td>*828.94</td>
<td></td>
</tr>
<tr>
<td>C109</td>
<td>*828.94</td>
<td>0</td>
<td>*828.94</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>instance</th>
<th>$d_{sum}$</th>
<th>$P$</th>
<th>$Q$</th>
<th>feasible</th>
</tr>
</thead>
<tbody>
<tr>
<td>R201</td>
<td>*1252.37</td>
<td>0</td>
<td>0</td>
<td>*1252.37</td>
</tr>
<tr>
<td>R202</td>
<td>*1183.53</td>
<td>1.11(1)</td>
<td>0</td>
<td>1195.31</td>
</tr>
<tr>
<td>R203</td>
<td>949.80</td>
<td>0.33(1)</td>
<td>0.01(1)</td>
<td>953.89</td>
</tr>
<tr>
<td>R204</td>
<td>847.87</td>
<td>0</td>
<td>0</td>
<td>847.87</td>
</tr>
<tr>
<td>R205</td>
<td>1009.83</td>
<td>0</td>
<td>0</td>
<td>1009.83</td>
</tr>
<tr>
<td>R206</td>
<td>935.90</td>
<td>0</td>
<td>0</td>
<td>935.90</td>
</tr>
<tr>
<td>R207</td>
<td>915.60</td>
<td>0</td>
<td>0</td>
<td>915.60</td>
</tr>
<tr>
<td>R208</td>
<td>749.56</td>
<td>0</td>
<td>0</td>
<td>749.56</td>
</tr>
<tr>
<td>R209</td>
<td>945.70</td>
<td>0</td>
<td>0</td>
<td>945.70</td>
</tr>
<tr>
<td>R210</td>
<td>961.10</td>
<td>0</td>
<td>0</td>
<td>961.10</td>
</tr>
<tr>
<td>R211</td>
<td>934.27</td>
<td>0</td>
<td>0</td>
<td>934.27</td>
</tr>
<tr>
<td>C201</td>
<td>*591.56</td>
<td>0</td>
<td>0</td>
<td>*591.56</td>
</tr>
<tr>
<td>C202</td>
<td>*591.56</td>
<td>0</td>
<td>0</td>
<td>*591.56</td>
</tr>
<tr>
<td>C203</td>
<td>*591.17</td>
<td>0</td>
<td>0</td>
<td>*591.17</td>
</tr>
<tr>
<td>C204</td>
<td>601.18</td>
<td>0</td>
<td>0</td>
<td>601.18</td>
</tr>
<tr>
<td>C205</td>
<td>*588.88</td>
<td>0</td>
<td>0</td>
<td>*588.88</td>
</tr>
<tr>
<td>C206</td>
<td>*588.49</td>
<td>0</td>
<td>0</td>
<td>*588.49</td>
</tr>
<tr>
<td>C207</td>
<td>*588.29</td>
<td>0</td>
<td>0</td>
<td>*588.29</td>
</tr>
<tr>
<td>C208</td>
<td>*588.32</td>
<td>0</td>
<td>0</td>
<td>*588.32</td>
</tr>
<tr>
<td>C209</td>
<td>*588.94</td>
<td>0</td>
<td>0</td>
<td>*588.94</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>instance</th>
<th>$d_{sum}$</th>
<th>$P$</th>
<th>$Q$</th>
<th>feasible</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC101</td>
<td>*1629.99</td>
<td>0.25(1)</td>
<td>5.70(4)</td>
<td>–</td>
</tr>
<tr>
<td>RC102</td>
<td>*1442.53</td>
<td>8.39(1)</td>
<td>4.93(6)</td>
<td>–</td>
</tr>
<tr>
<td>RC103</td>
<td>*1261.67</td>
<td>0</td>
<td>*1261.67</td>
<td></td>
</tr>
<tr>
<td>RC104</td>
<td>1160.60</td>
<td>0</td>
<td>1160.60</td>
<td></td>
</tr>
<tr>
<td>RC105</td>
<td>*1506.65</td>
<td>0.42(3)</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>RC106</td>
<td>*1382.03</td>
<td>0</td>
<td>1.87(2)</td>
<td>–</td>
</tr>
<tr>
<td>RC107</td>
<td>*1212.48</td>
<td>0</td>
<td>0.76(1)</td>
<td>1232.20</td>
</tr>
<tr>
<td>RC108</td>
<td>*1133.81</td>
<td>0.42(2)</td>
<td>*1139.82</td>
<td></td>
</tr>
<tr>
<td>RC109</td>
<td>*828.94</td>
<td>0</td>
<td>0</td>
<td>*828.94</td>
</tr>
<tr>
<td>RC201</td>
<td>1414.59</td>
<td>0.06(1)</td>
<td>0.77(1)</td>
<td>1424.65</td>
</tr>
<tr>
<td>RC202</td>
<td>*1321.07</td>
<td>0.92(2)</td>
<td>0</td>
<td>1397.45</td>
</tr>
<tr>
<td>RC203</td>
<td>1058.80</td>
<td>0.01(1)</td>
<td>0</td>
<td>1061.98</td>
</tr>
<tr>
<td>RC204</td>
<td>825.24</td>
<td>0</td>
<td>0</td>
<td>825.24</td>
</tr>
<tr>
<td>RC205</td>
<td>1297.65</td>
<td>0</td>
<td>0</td>
<td>1297.65</td>
</tr>
<tr>
<td>RC206</td>
<td>*1146.30</td>
<td>0.10(1)</td>
<td>0</td>
<td>1155.33</td>
</tr>
<tr>
<td>RC207</td>
<td>1065.74</td>
<td>0.47(1)</td>
<td>0</td>
<td>1071.43</td>
</tr>
<tr>
<td>RC208</td>
<td>862.46</td>
<td>0</td>
<td>0</td>
<td>862.46</td>
</tr>
</tbody>
</table>
### Table 3.3: Computational results with smaller number of vehicles than the best known solutions

<table>
<thead>
<tr>
<th>instance</th>
<th>$d_{\text{sum}}$</th>
<th>$P$</th>
<th>$Q$</th>
<th>$a_{\text{sum}}$</th>
<th>$P_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>R101</td>
<td>*1636.28</td>
<td>0.30(1)</td>
<td>0</td>
<td>0</td>
<td>0.30</td>
</tr>
<tr>
<td>R102</td>
<td>*1473.77</td>
<td>0</td>
<td>1.65(2)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R103</td>
<td>*1268.77</td>
<td>2.82(1)</td>
<td>1.42(1)</td>
<td>0</td>
<td>2.82</td>
</tr>
<tr>
<td>R104</td>
<td>*988.16</td>
<td>38.05(14)</td>
<td>145.27(85)</td>
<td>1</td>
<td>11.23</td>
</tr>
<tr>
<td>R105</td>
<td>1495.92</td>
<td>0</td>
<td>5.90(4)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R106</td>
<td>1360.94</td>
<td>4.39(1)</td>
<td>0</td>
<td>0</td>
<td>4.39</td>
</tr>
<tr>
<td>R107</td>
<td>*1098.82</td>
<td>10.23(4)</td>
<td>41.42(30)</td>
<td>0</td>
<td>7.03</td>
</tr>
<tr>
<td>R108</td>
<td>*937.96</td>
<td>8.74(6)</td>
<td>99.97(61)</td>
<td>0</td>
<td>4.66</td>
</tr>
<tr>
<td>R109</td>
<td>1285.81</td>
<td>0</td>
<td>39.02(25)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R110</td>
<td>*1105.73</td>
<td>5.85(4)</td>
<td>68.30(42)</td>
<td>0</td>
<td>1.86</td>
</tr>
<tr>
<td>R111</td>
<td>1134.38</td>
<td>12.11(7)</td>
<td>77.00(50)</td>
<td>0</td>
<td>6.17</td>
</tr>
<tr>
<td>R112</td>
<td>*948.94</td>
<td>11.91(7)</td>
<td>118.87(70)</td>
<td>3</td>
<td>7.65</td>
</tr>
<tr>
<td>C101</td>
<td>1037.42</td>
<td>822.23(46)</td>
<td>685.48(83)</td>
<td>70</td>
<td>103.66</td>
</tr>
<tr>
<td>C102</td>
<td>1146.93</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>C103</td>
<td>967.44</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>C104</td>
<td>912.23</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>C105</td>
<td>1019.59</td>
<td>130.44(16)</td>
<td>459.37(58)</td>
<td>80</td>
<td>22.23</td>
</tr>
<tr>
<td>C106</td>
<td>1150.63</td>
<td>62.28(6)</td>
<td>366.03(45)</td>
<td>40</td>
<td>20.44</td>
</tr>
<tr>
<td>C107</td>
<td>968.71</td>
<td>21.02(3)</td>
<td>125.09(23)</td>
<td>30</td>
<td>9.20</td>
</tr>
<tr>
<td>C108</td>
<td>1112.67</td>
<td>2.40(1)</td>
<td>70.12(15)</td>
<td>30</td>
<td>2.40</td>
</tr>
<tr>
<td>C109</td>
<td>954.78</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>RC101</td>
<td>1682.87</td>
<td>3.50(3)</td>
<td>15.64(10)</td>
<td>0</td>
<td>2.68</td>
</tr>
<tr>
<td>RC102</td>
<td>*1497.87</td>
<td>8.39(1)</td>
<td>11.80(9)</td>
<td>0</td>
<td>8.39</td>
</tr>
<tr>
<td>RC103</td>
<td>1347.96</td>
<td>0</td>
<td>2.21(1)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RC104</td>
<td>1150.75</td>
<td>0.06(1)</td>
<td>16.18(12)</td>
<td>12</td>
<td>0.06</td>
</tr>
<tr>
<td>RC105</td>
<td>*1625.64</td>
<td>0</td>
<td>7.80(5)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RC106</td>
<td>*1350.07</td>
<td>21.29(7)</td>
<td>57.44(32)</td>
<td>1</td>
<td>11.44</td>
</tr>
<tr>
<td>RC107</td>
<td>1330.42</td>
<td>0</td>
<td>3.28(4)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RC108</td>
<td>1153.31</td>
<td>0</td>
<td>29.66(22)</td>
<td>19</td>
<td>0</td>
</tr>
</tbody>
</table>
practical applications. For those instances with $P > 0$ or $Q > 0$, the traveling distance of the obtained solution tends to be much smaller than that of the best known solution at the cost of small penalties. This may suggest useful benefits of searching flexible vehicle schedules with our general solver.

In Table 3.3, we conducted experiments only for type 1 instances. (Since the number of vehicles of the best known solution is already 2 or 3, reducing vehicles is impractical for type 2 instances.) We obtained solutions whose traveling distances are much smaller than that of the best known solutions with little violation of constraints (i.e., with small $P$ and $Q$) for some instances such as R101, R102, RC103, RC105 and RC107. As it is usually more important to reduce the number of vehicles than to reduce traveling distance in practical applications, it may also be worthwhile to find solutions with moderate violations such as R103, R105, C102, C103, C104 and C109.

In summary, our algorithm could obtain the same quality as the best known solutions for 20 instances, implying that the performance of our algorithm is acceptable even for Solomon’s original instances. Furthermore, we could obtain solutions with smaller number of vehicles or with much shorter traveling distances than the best known solutions by allowing a little violation of constraints. These violations should be acceptable in many practical applications, or at least it provides the information about feasibility bottlenecks. This kind of information could not be obtained by other standard approaches.

### 3.7 Conclusion

In this chapter, we generalized the traveling time constraints for the vehicle routing problem by introducing traveling cost functions. We proved that the subproblem of determining the optimal start times of services for a given route becomes NP-hard when the traveling time cost functions are general, and proposed a pseudo-polynomial time algorithm of dynamic programming. Moreover, we proposed an algorithm based on the same dynamic programming for the subproblem, which runs in polynomial time, when each traveling time cost function is convex. Then, we proposed an iterated local search algorithm, which is based on the local search using cross exchange, 2-opt∗ and Or-opt neighborhoods, in which the dynamic programming algorithm for computing optimal start times of services is incorporated. Computational experiments on modified Solomon’s benchmark instances indicate the usefulness of relaxing time window and traveling time constraints.
Chapter 4

The Time-Dependent Vehicle Routing Problem with Time Windows

4.1 Introduction

In real situations, traveling times are often dependent on the departure times and they cannot be treated as constants in such cases (e.g., rush-hour traffic jam). For TSP, the generalization with time-dependent traveling times is called the *time dependent traveling salesman problem* (TDTSP) and is well-studied [71, 108, 123]. On the contrary, to the best of our knowledge, not much has been investigated on similar generalizations of VRPTW except for a few papers. Ichoua et al. [87] considered a formulation in which each customer has only one time window. Desaulniers et al. [40] presented a branch-and-bound framework for a very general model that can handle time-dependency and various other issues. In this chapter, we introduce traveling time and cost functions between each customer, whose values are dependent on the start time of traveling. These functions can be nonconvex and/or discontinuous as long as they are piecewise linear. Although we assume some property for each traveling time function, any functions satisfying the FIFO condition considered in [87] can still be represented, and the problem is fairly general. Our model generalizes that of Ichoua et al. in that it can allow more flexible time penalty function for each customer, and that of Ibaraki et al. [85] in that it can treat time-dependent traveling time and cost.

In our algorithm, we use local search to determine the routes of vehicles. When we evaluate a neighborhood solution, we need to solve the problem of determining the optimal start times on each route. In Ichoua et al., they solve this subproblem approximately (for
their restricted formulation), but solve it exactly only for the best $M$ approximate neighborhood solutions ($M$ is a parameter). We show that this subproblem can be efficiently solved with dynamic programming. The time complexity of our dynamic programming algorithm is the same as that of Ibaraki et al. [85] in spite of its generality if each traveling time and cost are constants. This dynamic programming is incorporated in the local search algorithm. In our local search, we use the standard neighborhoods called 2-opt*, cross exchange and Or-opt with slight modifications. We can evaluate the solutions in these neighborhoods efficiently by utilizing the information from the past dynamic programming recursion. We further propose a filtering method to restrict the search in the neighborhoods to avoid many solutions having no prospect of improvement. For the 2-opt* neighborhood, even with this restriction, we will not miss a better solution in the neighborhood if there is any. We develop an iterated local search algorithm incorporating all the above ingredients. Finally we report computational results on benchmark instances, and confirm the effectiveness of the restriction of the neighborhood. We compare the performance of our iterated local search algorithm against existing methods, and discuss the benefits of the proposed generalization.

4.2 Problem definition

Here we formulate the time-dependent vehicle routing problem with time windows. Let $G = (V, E)$ be a complete directed graph with vertex set $V = \{0, 1, \ldots, n\}$ and edge set $E = \{(i, j) \mid i, j \in V, i \neq j\}$, and $M = \{1, 2, \ldots, m\}$ be a vehicle set. In this graph, vertex 0 is the depot and other vertices are customers. Each customer $i$, each vehicle $k$ and each edge $(i, j) \in E$ are associated with:

i. a fixed quantity $a_i \geq 0$ of goods to be delivered to $i$,

ii. a time window cost function $p_i(t)$ of the start time $t$ of the service at $i$ ($p_0(t)$ is the time window cost function of the arrival time $t$ at the depot),

iii. a capacity $u_k \geq 0$ of $k$,

iv. a traveling time function $\lambda_{ij}(t)$ and a traveling cost function $q_{ij}(t)$ from $i$ to $j$ when the start time is $t$.

We assume $a_0 = 0$ without loss of generality. Each time window cost function $p_i(t)$ is nonnegative, piecewise linear and lower semicontinuous (i.e., $p_i(t) \leq \lim_{\varepsilon \to 0} \min\{p_i(t + \varepsilon), p_i(t - \varepsilon)\}$ at every discontinuous point $t$). Note that $p_i(t)$ can be non-convex and discontinuous as long as it satisfies the above conditions. We also assume $p_i(t) = +\infty$ for
4.2 Problem definition

$t < 0$ so that the start time $t$ of the service is nonnegative. We assume that each traveling cost function $q_{ij}(t)$ satisfies the same conditions as $p_i(t)$ (i.e., nonnegative, piecewise linear, lower semicontinuous and $q_{ij}(t) = +\infty$ for $t < 0$). We assume that each traveling time function $\lambda_{ij}(t)$ is nonnegative, piecewise linear and lower semicontinuous. The number of linear pieces of these functions are assumed to be finite. These assumptions ensure the existence of an optimal solution. We further assume that $\lambda_{ij}(t)$ satisfies

$$t + \lambda_{ij}(t) = t' + \lambda_{ij}(t')$$

$$\Rightarrow t + \lambda_{ij}(t) = \alpha t + (1 - \alpha)t' + \lambda_{ij}(\alpha t + (1 - \alpha)t'), \quad 0 \leq \alpha \leq 1$$

(4.2.1)

unless otherwise stated (see an example in Figure 4.1). In Figure 4.1, $s = t + \lambda_{ij}(t)$ is the arriving time at $j$ when a vehicle departs from $i$ at $t$. It is known that the FIFO condition

in [87] (i.e., $t \leq t' \Rightarrow t + \lambda_{ij}(t) \leq t' + \lambda_{ij}(t')$) implies condition (4.2.1). In our problem, the linear pieces of each piecewise linear function are given explicitly (i.e., the number of linear pieces is a part of the input size).

Let $\sigma_k$ denote the route traveled by vehicle $k$, where $\sigma_k(h)$ denotes the $h$th customer in $\sigma_k$, and let

$$\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m).$$

Note that each customer $i$ is included in exactly one route $\sigma_k$, and is visited by vehicle $k$ exactly once. We denote by $n_k$ the number of customers in $\sigma_k$. For convenience, we define $\sigma_k(0) = 0$ and $\sigma_k(n_k + 1) = 0$ for all $k$ (i.e., each vehicle $k \in M$ departs from the depot and comes back to the depot). Moreover, let $s_i$ be the start time of service at customer $i$.
(by exactly one of the vehicles) and \( s_k^a \) be the arrival time of vehicle \( k \) at the depot, and let

\[
s = (s_1, s_2, \ldots, s_n, s_1^a, s_2^a, \ldots, s_m^a).
\]

We assume \( s_0 = 0 \) for convenience of explanation. Let \( t_i \) be the departure time of a vehicle from customer \( i \) and \( t_k^1 \) be the departure time of vehicle \( k \) from the depot, and let

\[
t = (t_1, t_2, \ldots, t_n, t_1^1, t_2^1, \ldots, t_m^1).
\]

Note that each vehicle is allowed to wait at customers before starting services and before traveling.

Let us introduce 0-1 variables \( y_{ik}(\sigma) \in \{0, 1\} \) for \( i \in V \setminus \{0\} \) and \( k \in M \) by

\[
y_{ik}(\sigma) = 1 \iff i = \sigma_k(h) \text{ holds for exactly one } h \in \{1, 2, \ldots, n_k\}.
\]

That is, \( y_{ik}(\sigma) = 1 \) if and only if vehicle \( k \) visits customer \( i \). Then the total traveling cost \( q_{\text{sum}} \) traveled by all vehicles, the total time window cost \( p_{\text{sum}} \) for start times of services, and the total amount \( a_{\text{sum}} \) of capacity excess are expressed as follows:

\[
\begin{align*}
p_{\text{sum}}(s) & = \sum_{i \in V \setminus \{0\}} p_i(s_i) + \sum_{k \in M} p_0(s_k^a), \\
q_{\text{sum}}(\sigma, t) & = \sum_{k \in M} q_{0,\sigma_k(1)}(t_k^1) + \sum_{k \in M} \sum_{h=1}^{n_k} q_{\sigma_k(h),\sigma_k(h+1)}(t_{\sigma_k(h)}), \\
a_{\text{sum}}(\sigma) & = \sum_{k \in M} \max \left\{ \sum_{i \in V} a_i y_{ik}(\sigma) - u_k, 0 \right\}.
\end{align*}
\]

Then the problem we consider in this chapter is formulated as follows:

\[
\begin{align*}
\text{minimize} & \quad \text{cost}(\sigma, s, t) = p_{\text{sum}}(s) + q_{\text{sum}}(\sigma, t) + a_{\text{sum}}(\sigma) \\
\text{subject to} & \quad \sum_{k \in M} y_{ik}(\sigma) = 1, \quad i \in V \setminus \{0\} \quad (4.2.3) \\
& \quad s_i \leq t_i, \quad i \in V \setminus \{0\} \quad (4.2.4) \\
& \quad t_k^1 + \lambda_{0,\sigma_k(1)}(t_k^1) \leq s_{\sigma_k(1)}, \quad k \in M \quad (4.2.5) \\
& \quad t_{\sigma_k(i)} + \lambda_{\sigma_k(i),\sigma_k(i+1)}(t_{\sigma_k(i)}) \leq s_{\sigma_k(i+1)}, \quad 1 \leq i \leq n_k - 1, \quad k \in M \quad (4.2.6) \\
& \quad t_{\sigma_k(n_k)} + \lambda_{\sigma_k(n_k),0}(t_{\sigma_k(n_k)}) \leq s_k^a, \quad k \in M \quad (4.2.7) \\
& \quad y_{ik}(\sigma) \in \{0, 1\}, \quad i \in V \setminus \{0\}, \quad k \in M. \quad (4.2.8)
\end{align*}
\]

Constraint (4.2.3) means that every customer \( i \in V \setminus \{0\} \) must be served exactly once by a vehicle. Constraint (4.2.4) requires that each vehicle must depart from customer \( i \)
after the service and constraints (4.2.5)–(4.2.7) require that each vehicle cannot serve a customer before arriving at the customer. The time window and capacity constraints are treated as soft, and their violation is evaluated as the costs $p_{\text{sum}}(s)$ and $a_{\text{sum}}(\sigma)$ in the objective function. Note that, for any solution with a finite cost, $t_k^i \geq 0$ holds because of the assumptions, and hence $s, t \geq 0$ hold.

Although we assume that each service takes no time, we can treat the case with positive constant service times by defining each traveling time and cost functions as $\lambda_{ij}(t) = \tilde{b}_i + \tilde{\lambda}_{ij}(t + \tilde{b}_i)$ and $q_{ij}(t) = \tilde{q}_{ij}(t + \tilde{b}_i)$ if the given traveling time and cost functions between customer $i$ and $j$ are $\tilde{\lambda}_{ij}$ and $\tilde{q}_{ij}$, and the service time of customer $i$ is $\tilde{b}_i$. In our formulation, a traveling cost function can be a constant function such as distance, which is a major objective function in traditional formulations, and hence our problem is a generalization of VRPSTTW and the model of Ibaraki et al. [85].

This problem is separated into $m$ scheduling problems of finding the optimal start times if vehicle routes $\sigma$ are fixed. Hence our algorithm searches $\sigma$ by local search and solve the corresponding $m$ scheduling problems for each $\sigma$ generated during the search. In Section 4.3, we discuss this scheduling problem. How to search $\sigma_k$ will be discussed in Section 4.4.

### 4.3 Optimal start time problem

In this section, we consider the problem of determining the optimal start times for a given route $\sigma_k$ so that the total cost is minimized. Since the route is given, the objective function we have to consider is the sum of the time window costs and traveling costs. We call this subproblem the TOSTP (time-dependent optimal start time problem) in this chapter.

For convenience, throughout this section, we assume that vehicle $k$ visits customers $1, 2, \ldots, n_k$ in this order. Let customer 0 represent the departure from the depot (i.e., $t_0 = t_k^1$ and $q_{0,1}(t_0) = q_{0,1}(t_k^1)$), and let customer $n_k + 1$ represent the arrival at the depot (i.e., $s_{n_k+1} = s_k^a$ and $p_{n_k+1}(s_{n_k+1}) = p_0(s_k^a)$).

Then, the TOSTP is described as follows:

\[
\text{minimize} \quad \sum_{h=1}^{n_k+1} p_h(s_h) + \sum_{h=0}^{n_k} q_{h,h+1}(t_h)
\]

\[
\text{subject to} \quad s_h \leq t_h, \quad 1 \leq h \leq n_k
\]

\[
t_h + \lambda_{h,h+1}(t_h) \leq s_{h+1}, \quad 0 \leq h \leq n_k.
\]

We can solve the TOSTP by a dynamic programming algorithm in polynomial time as will be explained in Section 4.3.1.
4.3.1 Dynamic programming

We will show that the TOSTP is solvable in polynomial time by using dynamic programming.

Let \( f_h(t) \) be the minimum sum of the time window costs for customers 0, 1, \ldots, \( h \) and the traveling costs between them under the condition that they are all served before time \( t \).

We call \( f_h(t) \) as a forward minimum cost function. Then it can be computed by the following recurrence formula of dynamic programming:

\[
\begin{align*}
  f_0(t) &= \begin{cases} +\infty, & t < 0 \\ 0, & t \geq 0 \end{cases} \\
  f_h(t) &= \min_{s_h \leq t} \left\{ p_h(s_h) + \min_{t_{h-1} + \lambda_{h-1,h}(t_{h-1}) \leq s_h} \left\{ f_{h-1}(t_{h-1}) + q_{h-1,h}(t_{h-1}) \right\} \right\}, \\
  & \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad 1 \leq h \leq n_k + 1, -\infty < t < +\infty. 
\end{align*}
\]

The optimal cost of the TOSTP for a route \( \sigma_k \) is given by \( \min_t f_{n_k+1}(t) \).

4.3.2 Algorithm and time complexity

In this subsection, we consider the data structure and algorithm for computing forward minimum cost functions \( f_h \) in the recurrence formula (4.3.9). Since all functions of the input are piecewise linear, each \( f_h \) is also piecewise linear. We can therefore store all functions in linked lists; each cell stores the interval and the linear function of the corresponding piece, and the cells are linked according to the order of intervals. For example, Figure 4.2 shows a piecewise linear function \( g \) and the corresponding linked list.

Let \( \delta(g) \) be the sum of the number of linear pieces and the number of discontinuous points of a piecewise linear function \( g \) (i.e., the number of pieces of the polygonal line of \( g \)). For example, the function \( g \) in Figure 4.2 has seven pieces and two discontinuous points, and hence \( \delta(g) = 9 \). Then it is straightforward to see that the summation \( g + g' \) of two piecewise linear functions \( g \) and \( g' \) can be computed in \( O(\delta(g) + \delta(g')) \) time and the resulting function satisfies \( \delta(g + g') \leq \delta(g) + \delta(g') \). It is also easy to see that function \( \phi(t) = \min_{x \leq t} g(x) \) can be computed in \( O(\delta(g)) \) time and the resulting function \( \phi \) satisfies \( \delta(\phi) \leq \delta(g) \). When \( g \) is an increasing (resp., decreasing) piecewise linear function, i.e, \( t < t' \Rightarrow g(t) < g(t') \) (resp., \( t < t' \Rightarrow g(t) > g(t') \)), we can compute the composite function \( g' \circ g \) (i.e., \( g' \circ g(t) = g'(g(t)) \)) for a piecewise linear function \( g' \) in \( O(\delta(g) + \delta(g')) \) time since we can compute \( g'(g(t)) \) by increasing \( g(t) \) gradually. In this case, the resulting function satisfies \( \delta(g' \circ g) \leq \delta(g) + \delta(g') \). We can also compute the inverse of \( g \) in \( O(\delta(g)) \) time, and the inverse \( g^{-1} \) satisfies \( \delta(g^{-1}) = \delta(g) \).
4.3 Optimal start time problem

We define

$$\gamma_h(s) = \begin{cases} \min\{f_{h-1}(t) + q_{h-1,h}(t) \mid t + \lambda_{h-1,h}(t) = s\}, & \text{if } \{t \mid t + \lambda_{h-1,h}(t) = s\} \neq \emptyset, \\ \infty, & \text{otherwise}, \end{cases}$$

and reformulate the above recurrence formula (4.3.9) as

$$f_h(t) = \min_{s_h \leq t} \left\{ p_h(s_h) + \min_{s \leq s_h} \gamma_h(s) \right\}. \quad (4.3.10)$$

To compute $f_h$ by the recurrence formula (4.3.10), we must compute the function $\gamma_h(s)$ first. Let us consider the plane whose horizontal axis corresponds to the start time $t_{h-1}$ of traveling and the vertical axis corresponds to the arriving time $s = t_{h-1} + \lambda_{h-1,h}(t_{h-1})$. This is illustrated in Figure 4.3. Then $\gamma_h(s')$ for a fixed $s = s'$ is the minimum value of $f_{h-1}(t_{h-1}) + q_{h-1,h}(t_{h-1})$ among the points that satisfy

$$t_{h-1} + \lambda_{h-1,h}(t_{h-1}) = s',$$ \quad (4.3.11)

if such a point exists. In order to compute $\gamma_h(s)$, we split the domain of $t_{h-1}$ into increasing, constant and decreasing continuous parts and denote the closures of split intervals as $D_1, D_2, \ldots, D_L$ (see $D_1, D_2, \ldots, D_5$ in Figure 4.3). Then, for each $l = 1, 2, \ldots, L$, function $t + \lambda_{ij}(t)$ on domain $D_l$ admits the inverse or is a constant function. Let $R_l$ be the range of $t + \lambda_{ij}(t)$ on domain $D_l$ (i.e., $R_l = \{t + \lambda_{ij}(t) \mid t \in D_l\}$). By condition (4.2.1) and the definition of $D_l$, $R_l \cap R_{l'}$ contains at most one point for any $l \neq l'$. Hence we can compute
The Time-Dependent VRPTW

\[ s = t_{h-1} + \lambda_{h-1,h}(t_{h-1}) \]

Figure 4.3: The relationship between the departure time \( t_{h-1} \) and the arriving time \( s = t_{h-1} + \lambda_{h-1,h}(t_{h-1}) \)

\[ \gamma_h(s) \] partially for each domain \( D_l \) except for \( s \in \bigcup_{l \neq l'} R_l \cap R_{l'} \). Then \( \gamma_h(s) \) is completed by merging them and taking the minimums for \( s \in \bigcup_{l \neq l'} R_l \cap R_{l'} \). Note that if \( s \notin \bigcup_{l=1}^L R_l \), we define \( \gamma_h(s) = \infty \). We have to arrange all \( R_l \)'s in the increasing order when we merge them, because the order of the appearance of \( R_l \) may be different from that of \( D_l \). In Figure 4.3, the order of the appearance of \( R_l \) is \((R_1, R_2, R_3, R_5, R_4)\). However, the order of \( R_l \)'s can be determined by \( \lambda_{ij} \) alone and need be computed only once before a search. This computation is negligible in comparison with the whole computation time. Hence we assume that the order of \( R_l \)'s for each \( \lambda_{ij} \) is given as a part of input.

We can now compute \( \gamma_h(s) \) by the following steps:

i. Compute \( \gamma_h|R_i \) for each domain \( R_1, R_2, \ldots, R_L \) by increasing \( t_{h-1} \) gradually and computing the corresponding \( \gamma_h(s) \).

ii. Merge \( \gamma_h|R_i \) for \( l = 1, 2, \ldots, L \) and add linear pieces for the intervals with \( s \notin \bigcup_{l=1}^L R_l \).

For computing each \( \gamma_h|R_i \) for \( l = 1, 2, \ldots, L \), we need either to take the minimum (i.e., \( \gamma_h(s) = \min\{f_{h-1}(t_{h-1}) + q_{h-1,h}(t_{h-1}) \mid t_{h-1} \in D_l \}\)), or to calculate the composite function (i.e., \( \gamma_h(s) = f_{h-1}(t_{h-1}) + q_{h-1,h}(t_{h-1}) \) where \( t_{h-1} + \lambda_{h-1,h}(t_{h-1}) = s \) holds. We can compute \( t_{h-1} \) from \( s \) by taking the inverse of \( t_{h-1} + \lambda_{h-1,h}(t_{h-1}) \)). In both cases, the time complexity is linear to the number of the corresponding linear pieces of \( f_{h-1}, q_{h-1,h} \) and \( \lambda_{h-1,h} \). During the whole computation of (i), we need to scan (the linked lists representing)
the functions \( \lambda_{h-1,h}(t_{h-1}) \) and \( f_{h-1}(t_{h-1}) + q_{h-1,h}(t_{h-1}) \) only once from left to right. For completing \( \gamma_h \) by merging \( \gamma_h|_R \) for \( l = 1, 2, \ldots, L \) in (ii), it is straightforward to see that the time complexity is \( O(L) \). Hence the time complexity of computing function \( \gamma_h(s) \) is \( O(\delta(f_{h-1}) + \delta(q_{h-1,h}) + \delta(\lambda_{h-1,h})) \), and \( \delta(\gamma_h) \leq \delta(f_{h-1}) + \delta(q_{h-1,h}) + \delta(\lambda_{h-1,h}) \) holds.

Now we can compute \( f_h \) by the recurrence formula (4.3.10) in \( O(\delta(p_h) + \delta(\gamma_h)) \) time, where the number of pieces of \( f_h \) is at most \( \delta(p_h) + \delta(\gamma_h) \). Hence we can compute \( f_h \) from \( f_{h-1} \) in

\[
O(\delta(f_{h-1}) + \delta(q_{h-1,h}) + \delta(\lambda_{h-1,h})) + O(\delta(p_h) + \delta(\gamma_h))
\]

\[
= O(\delta(f_{h-1}) + \delta(p_h) + \delta(q_{h-1,h}) + \delta(\lambda_{h-1,h}))
\]

time and we have

\[
\delta(f_h) \leq \delta(p_h) + \delta(\gamma_h)
\]
\[
\leq \delta(f_{h-1}) + \delta(p_h) + \delta(q_{h-1,h}) + \delta(\lambda_{h-1,h}).
\]

Hence we have

\[
\delta(f_h) \leq \sum_{h' = 1}^{h} \delta(p_{h'}) + \delta(q_{h'-1,h'}) + \delta(\lambda_{h'-1,h'}).\]

Using this, the time complexity of computing \( f_h \) from \( f_{h-1} \) is evaluated as

\[
O \left( \sum_{h' = 1}^{h} \delta(p_{h'}) + \delta(q_{h'-1,h'}) + \delta(\lambda_{h'-1,h'}) \right).
\]

In summary, given a route \( \sigma_k \), we can compute the forward minimum cost function of a customer from that of the previous customer in \( O(\Delta(\sigma_k)) \) time, where

\[
\Delta(\sigma_k) = \sum_{h=1}^{n_k+1} \delta(p_{\sigma_k(h)}) + \delta(q_{\sigma_k(h-1),\sigma_k(h)}) + \delta(\lambda_{\sigma_k(h-1),\sigma_k(h)}).
\]

We can then obtain the optimal cost of \( \sigma_k \) in \( O(n_k \Delta(\sigma_k)) \) time by computing the forward minimum cost functions of \( n_k \) customers in \( \sigma_k \), and taking the minimum of the forward minimum cost function of the depot. Note that \( \Delta(\sigma_k) \) is the same as the input size of the TOSTP. If traveling cost and time functions are constant functions (i.e., if there is no time-dependency), this time complexity of the dynamic programming algorithm becomes the same as that of Ibaraki et al. [85].

4.3.3 Remarks for the case in which condition (4.2.1) does not hold

Even if condition (4.2.1) does not hold, we can compute \( \gamma_h(s) \) in a similar manner as in Section 4.3.2. Figure 4.4 shows the same situation as Figure 4.3 in which condition (4.2.1)
does not hold. In order to compute $\gamma_h(s)$, we split the domain of $t_{h-1}$ into the intervals $D_1, D_2, \ldots, D_L$ as before, compute the functions

$$\tilde{\gamma}^l_h(s) = f_{h-1}(t_{h-1}) + q_{h-1,h}(t_{h-1}),$$

where $t_{h-1} + \lambda_{h-1,h}(t_{h-1}) = s$ and $t_{h-1} \in D_l$, for each $R_l$, $l = 1, 2, \ldots, L$, and complete $\gamma_h$ by taking the lower envelope of them.

In general, the complexity of the lower envelope of $n$ segments is $\theta(\alpha(n)n)$, where $\alpha$ denotes the inverse of Ackermann’s function [73]. Using this fact, letting $\Delta = \delta(f_{h-1}) + \delta(q_{h-1,h}) + \delta(\lambda_{h-1,h})$, the complexity of $\gamma_h$ is bounded by $O(\alpha(\Delta)\Delta)$. However, this upper bound is not small enough to prove that the complexity of $f_h$ is of polynomial order. Hence our dynamic programming algorithm may require exponential time. Whether the algorithm runs in polynomial time or not, and whether the TOSTP itself is NP-hard or not are both open.

4.3.4 Historical notes

The TOSTP without time-dependency (i.e., $q_{h,h+1}(t)$ and $\lambda_{h,h+1}(t)$ are constant functions) has been intensively studied in the literature especially when the time window cost functions $p_h(s)$ are convex. Below is a brief summary of such results.

Special cases of convex time window cost functions were considered in the literature of VRPSTW and scheduling problems. In Taillard et al. [142], the time window cost for each customer is $+\infty$ for earliness and linear for tardiness, and an $O(1)$ time algorithm to approximately compute the optimal time window cost of a solution in the neighbor-
4.4 Local search for finding visiting orders \( \sigma \)

In this section, we describe the framework of our local search (LS) for finding good visiting orders \( \sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m) \) that satisfy condition (4.2.3). It starts from an initial solution \( \sigma \) and repeats replacing \( \sigma \) with a better solution in its neighborhood \( N(\sigma) \) until no better solution is found in \( N(\sigma) \). As \( N(\sigma) \) we use the standard neighborhoods called 2-opt*, cross exchange and Or-opt with slight modifications (see Figure 4.5). In this figure, squares represent the depot (which is duplicated at each end) and small circles represent customers in the routes. A thin line represents a route edge and a thick line represents a path (i.e., more than two customers may be included).
A 2-opt* operation removes two edges from two different routes (one from each) to divide each route into two parts and exchanges the second parts of the two routes (See Section 2.4.2). A cross exchange operation removes two paths from two routes (one from each) of different vehicles, whose length (i.e., the number of customers in the path) is at most $L_{\text{cross}}$ (a parameter), and exchanges them (See Section 2.4.4). The cross exchange and 2-opt* operations always change the assignment of customers to vehicles. We also use the intra-route neighborhood to improve individual routes. An intra-route operation removes a path of length at most $L_{\text{path}}$ (a parameter) and inserts it into another position of the same route, where the position is limited within length $L_{\text{ins}}$ (a parameter) from the original position (See Section 2.4.5). Our LS searches the above intra-route neighborhood, 2-opt* neighborhood and cross exchange neighborhood, in this order. Whenever a better solution is found, we immediately accept it (i.e., we adopt the first admissible move strategy), and resume the search from the intra-route neighborhood.

As only one execution of LS may not be sufficient to find a good solution, we use the iterated local search (ILS), which iterates LS many times from those initial solutions generated by perturbing good solutions obtained by then. We perturb a solution by applying one random cross exchange operation with no restriction on $L_{\text{cross}}$ (i.e., $L_{\text{cross}} = n$). ILS is summarized as follows:

**Algorithm: Iterated Local Search (ILS)**

**Step 1** Generate an initial solution $\sigma^0$. Let $\sigma_{\text{seed}} := \sigma^0$ and $\sigma_{\text{best}} := \sigma^0$.

**Step 2** Improve $\sigma_{\text{seed}}$ by LS and let $\sigma$ be the improved solution.

**Step 3** If $\sigma$ is better than $\sigma_{\text{best}}$, then replace $\sigma_{\text{best}}$ with $\sigma$. 

Figure 4.5: Neighborhoods in our local search
Step 4 If some stopping criterion is satisfied, output $\sigma_{\text{best}}$ and halt; otherwise generate a solution $\sigma_{\text{seed}}$ by perturbing $\sigma_{\text{best}}$ and return to Step 2.

4.5 Efficient implementation of local search

A solution $\sigma$ is evaluated by $(p + q)\sum(\sigma) + a_{\text{sum}}(\sigma)$, where $(p + q)\sum(\sigma)$ denotes the minimum time window and traveling time cost for computed $\sigma$ by dynamic programming in Section 4.3. (Actually in our algorithm, we split each traveling cost function into the constant part and the time-dependent part, and compute them separately to improve the efficiency.) For this, it is important to see that dynamic programming computation of $(p + q)\sum(\sigma)$ for the solutions in neighborhoods can be sped up by using information from the previous computation. The efficient neighborhood search method in Section 2.6 can be applied. Below we will denote by $\langle \sigma_k(h_1) \rightarrow \sigma_k(h_2) \rangle$ the path from the $h_1$-th customer to the $h_2$-th customer in route $\sigma_k$, and by $\langle \sigma_k(h_1) \rightarrow \sigma_k(h_2) \rangle$ the path constructed by connecting two paths $\langle \sigma_k(h_1) \rightarrow \sigma_k(h_2) \rangle$ and $\langle \sigma_k(h_3) \rightarrow \sigma_k(h_4) \rangle$ from routes $\sigma_k$ and $\sigma_k'$.

4.5.1 Basic idea

Consider the computation of the minimum cost (including the amount of capacity excess, the time window cost and the traveling time cost) for a given route $\sigma_k = (\sigma_k(0), \sigma_k(1), \ldots, \sigma_k(n_k + 1))$ when it is obtained by connecting its former part $\langle 0 \rightarrow \sigma_k(h) \rangle$ and the latter part $\langle \sigma_k(h + 1) \rightarrow 0 \rangle$ for some $h$ as illustrated in Figure 4.6. The amount of capacity excess for route $\sigma_k$ is computed in $O(1)$ time, if both $\sum_{i=1}^{h} a_{\sigma_k(i)}$ and $\sum_{i=h+1}^{n_k} a_{\sigma_k(i)}$ are known. We therefore store $\sum_{i=1}^{h} a_{\sigma_k(i)}$ and $\sum_{i=h+1}^{n_k} a_{\sigma_k(i)}$ for each customer $\sigma_k(h)$ and vehicle $k$ whenever the current route is updated.

Now we concentrate on the computation of $(p + q)\sum(\sigma_k)$, which is the minimum sum of time window and traveling time costs on route $\sigma_k$.

Let $b^k_h(t)$ be the minimum sum of the time window costs for customers $\sigma_k(h), \sigma_k(h+1), \ldots, \sigma_k(n_k)$.
1), \ldots, \sigma_k(n_k + 1) and the traveling costs between them provided that all of them are served after time $t$.

We call this a \textit{backward minimum cost function}. Then, $b^k_h(t)$ can be computed as follows, which is symmetric to the forward minimum cost computation discussed in Section 4.3:

\begin{equation}
\begin{align*}
b^k_{n_k+1}(t) &= \min_{s \geq t} p_0(s), \\
b^k_h(t) &= \min_{s \geq t} \{ p_{\sigma_k(h)}(s) \\
&\quad + \min_{t' \geq s} \{ b^k_{h+1}(t' + \lambda \sigma_k(h), \sigma_k(h+1)(t')) + q_{\sigma_k(h), \sigma_k(h+1)}(t') \} \},
\end{align*}
\end{equation}

(4.5.12)

Let $f^k_h(t)$ be the forward minimum cost function at the $h$th customer in route $\sigma_k$. We can then obtain the optimal cost $(p + q)^* \text{sum}(\sigma_k)$ by

\[ \min_t \left\{ b^k_h(t) + \min_{t' + \lambda \sigma_k(h-1), \sigma_k(h)(t') \leq t} f^k_{h-1}(t') + q_{\sigma_k(h-1), \sigma_k(h)}(t') \right\} \]

for any $h$ ($1 \leq h \leq n_k + 1$). If $f^k_{h-1}(t)$ and $b^k_h(t)$ are already available for some $h$, this is possible in $O(\Delta(\sigma_k))$ time (since the computation of (4.5.13) is similar to that of (4.3.9)). To achieve this, we store all functions $f^k_h(t)$ and $b^k_h(t)$ for each customer $\sigma_k(h)$, when these were computed in the process of LS.

In summary, we can compute the minimum cost of route $\sigma_k$ in $O(\Delta(\sigma_k))$ time, if we keep the data $\sum_{i=1}^h a_{\sigma_k(i)}$, $\sum_{i=h}^{n_k} a_{\sigma_k(i)}$, $f^k_h(t)$ and $b^k_h(t)$ for all $h = 1, 2, \ldots, n_k$ and $k \in M$.

In our algorithm, the number of pieces in forward and backward minimum cost functions is closely linked to the speed of our algorithm. Hence we consider its reduction. Since $f^k_h(t)$ (resp., $b^k_h(t)$) is nonincreasing (resp., nondecreasing), there are usually many pieces with considerably large values in $f^k_h(t)$ (resp., in $b^k_h(t)$) for small (resp., large) $t$. Such pieces will not be used in evaluating improved solutions. We therefore shrink those pieces whose minimum values over their intervals are larger than the objective value of the current solution, into one piece.

### 4.5.2 How to apply the basic idea to the solutions in neighborhoods

We now explain how to apply the above idea to evaluate solutions in the neighborhoods efficiently. We only discuss the sum of time window and traveling costs, since the amount of capacity excess can be similarly treated. Recall that we can compute the forward minimum cost function (resp., the backward minimum cost function) of a customer from that of the previous customer (resp., the next customer) in $O(\Delta(\sigma_k))$ time, and that we can evaluate the route cost by connecting the forward and backward minimum cost functions in $O(\Delta(\sigma_k))$ time.
4.5 Efficient implementation of local search

Figure 4.7: An example of a 2-opt* operation

In Figure 4.7, an example of a 2-opt* operation on routes $\sigma_k$ and $\sigma_{k'}$ is shown. We denote by $\sigma^\text{new}_k$ and $\sigma^\text{new}_{k'}$ the resulting two routes (i.e., $\sigma^\text{new}_k = \langle 0 \rightarrow \sigma_k(h_k) \rangle \cdots \langle \sigma_k(h_k + 1) \rightarrow 0 \rangle$ and $\sigma^\text{new}_{k'} = \langle 0 \rightarrow \sigma_{k'}(h_{k'}) \rangle \cdots \langle \sigma_{k'}(h_{k'} + 1) \rightarrow 0 \rangle$). Then, the sum of time window and traveling time costs for $\sigma^\text{new}_k$ can be computed by

$$\min_t \left\{ b^t_{h_k+1} + \min_{t'+\lambda_{\sigma_k(h_k),\sigma_{k'}(h_{k'}+1)}} \tilde{f}^t_{h_k} + q_{\sigma_k(h_k),\sigma_{k'}(h_{k'}+1)}(t') \right\}$$

in $O(\Delta(\sigma^\text{new}_k))$ time. Similarly the cost for $\sigma^\text{new}_{k'}$ can be computed in $O(\Delta(\sigma^\text{new}_{k'}))$ time.

Hence, when a 2-opt* operation is applied to routes $\sigma_k$ and $\sigma_{k'}$, we can evaluate the cost of the resulting solution in $O(\Delta(\sigma^\text{new}_k) + \Delta(\sigma^\text{new}_{k'}))$ time.

Figure 4.8: An example of the search order in the cross exchange neighborhood

To evaluate solutions in the cross exchange neighborhood efficiently, we need to search the solutions in the neighborhood in a specific order. To apply cross exchange operations...
on routes $\sigma_k$ and $\sigma_{k'}$, we start from a solution obtainable by exchanging one customer from each route, and then extend lengths of the paths to be exchanged one by one. Figure 4.8 explains the situation. We denote by $\sigma_k^{\text{tmp}}$ and $\sigma_{k'}^{\text{tmp}}$ the routes obtained by applying a cross exchange operation on the current routes $\sigma_k$ and $\sigma_{k'}$ (see Figure 4.8 (a)) and by $\sigma_k^{\text{new}}$ and $\sigma_{k'}^{\text{new}}$ the routes generated next (see Figure 4.8 (b)). In Figure 4.8 (a), backward minimum cost functions $b^k_{h_k}$, $b^k_{h_k'}$, and $b^k_{h_{k'+1}}$ of the current routes $\sigma_k$ and $\sigma_{k'}$ are available, and we have already computed the forward minimum cost functions $\tilde{f}^k_1, \tilde{f}^k_2, \ldots, \tilde{f}^k_l$ (resp., $\tilde{f}^{k'}_1, \tilde{f}^{k'}_2, \ldots, \tilde{f}^{k'}_{l'}$) on the partial paths in $\sigma_k^{\text{tmp}}$ (resp., $\sigma_{k'}^{\text{tmp}}$) in the process of computing $(p+q)^*_{\text{sum}}(\sigma_k^{\text{tmp}})$ (resp., $(p+q)^*_{\text{sum}}(\sigma_{k'}^{\text{tmp}})$). We then compute $\tilde{f}^k_{l+1}$ from $\tilde{f}^k_l$ by recursion of the dynamic programming in $O(\Delta(\sigma_k^{\text{new}}))$ time, and evaluate $(p+q)^*_{\text{sum}}(\sigma_k^{\text{new}}) + (p+q)^*_{\text{sum}}(\sigma_{k'}^{\text{new}})$ in $O(\Delta(\sigma_k^{\text{new}}) + \Delta(\sigma_{k'}^{\text{new}}))$ time (Figure 4.8 (b)). Thus, we can compute the change in the cost after a cross exchange operation in $O(\Delta(\sigma_k^{\text{new}}) + \Delta(\sigma_{k'}^{\text{new}}))$ time.

Similarly, the change in the cost for an intra-route operation of route $\sigma_k$ can be computed in $O(\Delta(\sigma_k^{\text{new}}))$ time, by searching solutions in a specific order, where $\sigma_k^{\text{new}}$ denotes the route generated by an intra-route operation. Actually, this case is slightly more complicated than the case of cross exchange neighborhood, but the search order described in Section 2.6 also works for our problem.

### 4.5.3 Restriction of neighborhoods

In searching neighborhoods, we find that there are many solutions which have no prospects of improvements. In order to avoid evaluating such solutions, we propose a rule to restrict the search.

For a constant $U$, let

$$F^k_h(U) = \begin{cases} 
\min \{ t \mid f^k_h(t) \leq U \}, & \text{if } \min f^k_h(t) \leq U \\
+\infty, & \text{otherwise.}
\end{cases}$$

This $F^k_h(U)$ gives the earliest departure time of vehicle $k$ from customer $\sigma_k(h)$ in order to keep the sum of the time window cost of customers $\sigma_k(1)$, $\sigma_k(2), \ldots$, $\sigma_k(h)$ and the traveling cost between them below $U$. In other words

$$t \geq F^k_h(U) \iff f^k_h(t) \leq U$$

holds. As mentioned in Section 4.3.2, we store $f^k_h(t)$ in a linked list; however, we can also store $f^k_h(t)$ in an array without sacrificing the time complexity. Using this the array data structure, we can compute $F^k_h(U)$ for a given $U$ in $O(\log(\delta(f^k_h)))$ time because $f^k_h$ is a nonincreasing function. (In our program, however, we did not implement the array structure, and use $O(\delta(f^k_h))$ time to compute $F^k_h(U)$, because this does not seem to be a bottle neck of computation.)
Similarly let
\[ B^k_h(U) = \begin{cases} \max\{t \mid b^k_h(t) \leq U\}, & \text{if } \min_t b^k_h(t) \leq U \\ -\infty, & \text{otherwise.} \end{cases} \]

\( B^k_h(U) \) is the latest arrival time of vehicle \( k \) at customer \( \sigma_k(h) \) in order to keep the time window cost of customers \( \sigma_k(h), \sigma_k(h+1), \ldots, \sigma_k(n_k+1) \) and the traveling cost between them below \( U \). Note also that

\[ t \leq B^k_h(U) \iff b^k_h(t) \leq U \]

holds, because \( b^k_h \) is a nondecreasing function, and we can compute \( B^k_h(U) \) in \( O(\log(\delta(b^k_h))) \) time. Then, if

\[ F^k_{h-1}(U) + \lambda^\text{min}_{k,h} > B^k_h(U) \]

holds for \( \lambda^\text{min}_{k,h} = \min_t \lambda_{\sigma_k(h-1), \sigma_k(h)}(t) \), the cost of route \( \sigma_k \) must be larger than \( U \). This fact is utilized to restrict the search in the 2-opt* and cross exchange neighborhoods, whose details are explained in Sections 4.5.3 and 4.5.3.

Furthermore, for any nonnegative nonincreasing function \( f \) and any nonnegative nondecreasing function \( b \), we can obtain lower and upper bounds of \( \min_t \{f(t) + b(t)\} \) by the following observation. Let a point \( \hat{t} \) satisfy that \( t \geq \hat{t} \implies b(t) \geq f(\hat{t}) \) and \( t \leq \hat{t} \implies f(t) \geq b(\hat{t}) \), and we call this the switch point of \( f \) and \( b \). Then the switch point \( \hat{t} \) satisfies

\[ \max\{f(\hat{t}), b(\hat{t})\} \leq \min_t \{f(t) + b(t)\} \leq f(\hat{t}) + b(\hat{t}) \leq 2 \max\{f(\hat{t}), b(\hat{t})\}. \]

If there is no switch point, either \( f(t) > b(t) \) or \( f(t) < b(t) \) holds for all \( t \). In this case if \( f(t) > b(t) \) holds for all \( t \), then

\[ f(\bar{t}) \leq \min_t \{f(t) + b(t)\} \leq f(\hat{t}) + b(\hat{t}) \leq 2f(\hat{t}) \]

holds, where \( \bar{t} = \arg \min_t f(t) \). When \( f \) and \( b \) are continuous, the switch point is the intersecting point of \( f \) and \( b \).\(^1\) From this property, for any \( h \in \{1, 2, \ldots, n_k+1\} \), if \( \lambda_{\sigma_k(h-1), \sigma_k(h)}(t) \) is a constant function and \( q_{\sigma_k(h-1), \sigma_k(h)} = 0 \), we can obtain a lower bound on the TOSTP from the switch point of \( f^k_{h-1}(t) \) and \( b^k_h(t + \lambda_{\sigma_k(h-1), \sigma_k(h)}(t)) \), and the schedule induced by the switch point becomes a 2-approximate schedule for \( \sigma_k \) (i.e., the cost of the schedule is at most \( 2(p+q)^\ast_{\text{sum}}(\sigma_k) \)), where cost can be computed in \( O(\log(\delta(f^k_{h-1})) + \log(\delta(b^k_h))) = O(\log(\Delta(\sigma_k))) \) time. Even if \( \lambda_{\sigma_k(h-1), \sigma_k(h)} \) and \( q_{\sigma_k(h-1), \sigma_k(h)} \) are time-dependent, the switch point of \( f^k_{h-1}(t) \) and \( b^k_h(t + \lambda^\text{min}_{k,h}) \) gives a lower bound on the optimal cost. Hence we can skip solving the TOSTP optimally in the search of neighborhoods if its lower bound tells that it cannot improve the current \( \sigma \).

\(^1\)In our implementation, we just took an intersecting point instead of a switch point, because all functions of the tested instances are continuous.
2-opt* neighborhood

Consider to evaluate a solution in the 2-opt* neighborhood obtained by reconnecting two routes $\sigma_k$ and $\sigma_{k'}$ (see Figure 4.7). We set a threshold $U$, and avoid evaluating the routes if we can conclude $(p+q)_{\text{sum}}(\sigma_k^{\text{new}}) + (p+q)_{\text{sum}}(\sigma_{k'}^{\text{new}}) > U$. As our purpose is to obtain a better solution than the current one, we can set $U$ as the total cost of the current routes $\sigma_k$ and $\sigma_{k'}$. Our first idea is based on the following fact: The solution obtained by connecting $\sigma_k(h_k)$ and $\sigma_{k'}(h_{k'}+1)$ will have a cost larger than $U$ if $F^k_{h_k}(U) > B^k_{h_k+1}(U)$ holds. Let

$$P^{kk'}_{\text{valid}}(U) = \left\{ (\sigma_k(h_k), \sigma_{k'}(h_{k'})) \mid F^k_{h_k}(U) \leq B^k_{h_k+1}(U) \text{ and } F^{k'}_{h_{k'}}(U) \leq B^{k'}_{h_{k'}+1}(U) \right\}.$$ 

Then we can compute $P^{kk'}_{\text{valid}}(U)$ in $O(n_k + n_{k'} + |P^{kk'}_{\text{valid}}(U)|)$ time if $F^k_{h_k}(U)$, $B^k_{h_k}(U)$, $F^{k'}_{h_{k'}}(U)$ and $B^{k'}_{h_{k'}+1}(U)$ are available for all $h_k$ and $h_{k'}$. It takes $O(n_k \Delta(\sigma_k) + n_{k'} \Delta(\sigma_{k'}))$ time for the preprocessing ($O(n_k \log(\Delta(\sigma_k)) + n_{k'} \log(\Delta(\sigma_{k'})))$ time if we use the array structure). Such preprocessing is necessary only when the current solution is changed (i.e., when an improved solution is found or a perturbation is applied), and is usually dominated by the evaluation time of solutions.

Let

$$P_{\text{valid}}(U) = \bigcup_{k \neq k'} P^{kk'}_{\text{valid}}(U).$$

Then we can compute $P_{\text{valid}}(U)$ in $O(nm + |P_{\text{valid}}(U)|)$ time. Any solution cannot be better than the current solution unless it is induced from $P_{\text{valid}}(U)$.

Hence we restrict the 2-opt* neighborhood only to the solutions induced from $P_{\text{valid}}(U)$. Although the size of the 2-opt* neighborhood is reduced from $O(n^2)$ to $O(|P_{\text{valid}}(U)|)$ by this modification, we miss no better solution in the 2-opt* neighborhood.

Cross exchange neighborhood

Consider the search in the cross exchange neighborhood. The size of the cross exchange neighborhood is $O(n^2(L^{\text{cross}})^2)$, and is largest in the standard neighborhoods used in this chapter. Here we consider a restriction of the cross exchange neighborhood. In order to keep the change in the time window costs and traveling time costs small, it seems preferable to keep the arriving times at customers in the generated solution close to those of the current solution. Based on this intuition, we restrict the paths to be exchanged to those which satisfy $\sigma_k(h^k_1), \sigma_{k'}(h^k_{1'}) \in P^{kk'}_{\text{valid}}(U)$, where $\sigma_k(h^k_1)$ and $\sigma_{k'}(h^k_{1'})$ are the first customers of the paths. Note that, different from the case of 2-opt* neighborhood, this restriction has a possibility of missing a better solution in the cross exchange neighborhood.
4.6 Computational results

We conducted computational experiments to evaluate the proposed algorithm ILS. The algorithm was coded in C language and run on a handmade PC (Intel Pentium 4, 2.8 GHz, 1 GB memory). We used $L_{\text{cross}} = 3$, $L_{\text{path}}^{\text{intra}} = 3$ and $L_{\text{ins}}^{\text{intra}} = 15$ in the experiments.

4.6.1 Effect of the restriction of the neighborhoods

We first consider the effect of the restriction of the neighborhoods discussed in Section 4.5.3. We run our local search algorithm with the 2-opt* neighborhood only, from a random solution and a locally optimum solution, both with and without restriction. We use the same random solution and the same locally optimal solution for the initial solutions of the runs with and without restriction. In a similar manner, we also test our local search algorithm with the cross exchange neighborhood only. For these tests, we used the instance r201 in Solomon’s benchmark list, whose details will be described in Section 4.6.2.

Figures 4.9 shows the results with the 2-opt* neighborhood without (left) and with (right) restriction, respectively, whose the vertical axis gives the total cost of every two routes constructed as neighborhood solutions, while the horizontal axis shows the cost before the neighborhood operation is applied (i.e., the current solution). Namely, each point in the figure corresponds to the two route cost of a neighborhood solution. Similarly, Figure 4.10 shows the results with the cross exchange neighborhood.

![Figure 4.9: The distribution of two route cost in the 2-opt* neighborhood without (left) and with (right) restriction](image)

From these figures, we observe that the proposed restriction succeeds in avoiding the evaluations of solutions whose costs are much larger than that of the current solutions. We can also observe that the restriction becomes more effective when the cost of the current solution is small.
Table 4.1: Number of evaluations with and without restriction of neighborhood

<table>
<thead>
<tr>
<th>neighborhood</th>
<th>2-opt*</th>
<th>cross exchange</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial solution</td>
<td>random</td>
<td>locally optimal</td>
</tr>
<tr>
<td>without restriction</td>
<td>227746 (11016.59)</td>
<td>4035 (1253.23)</td>
</tr>
<tr>
<td>with restriction</td>
<td>146440 (17906.10)</td>
<td>164 (1253.23)</td>
</tr>
</tbody>
</table>

Then Table 4.1 shows the number of cost evaluations needed to obtain a locally optimal solution, with and without restriction of neighborhood. Column “random” (resp., “locally optimal”) shows the number of evaluations during the local search when the initial solution is a random (resp., locally optimal) solution. Note that, in the case of the locally optimal initial solution, no improvement is achieved as a result of local search. The two rows correspond to the cases with and without restriction, respectively, where the values in parenthesis are the objective values of the obtained locally optimal solutions.

From Table 4.1, we can confirm the effectiveness of the restriction. In the neighborhood of a random solution we can reduce the number of evaluations to almost a half, and in the neighborhood of a locally optimal solution, we can reduce it to only a few percent. In our restriction of 2-opt* neighborhood, the solution quality is basically the same since the restriction is guaranteed not to miss any improved neighborhood solution. However, the objective values are different in the table, because we use random numbers when we determine the search order in neighborhood. Although we may miss improved solutions in the case of the cross exchange neighborhood with restriction, the output solution happens to be slightly better than that obtained without restriction in this particular case. Since each run of local search resumes from a solution generated by applying a small perturbation
4.6 Computational results

on a good locally optimal solution in our ILS algorithm, the effect of the restriction is expected to be significant.

4.6.2 The vehicle routing problem with hard time windows

We used Solomon’s benchmark instances [140] and Gehring and Homberger’s benchmark instances [80], which have been widely used in the literature. We first explain Solomon’s instances. The number of customers in each instance is 100, and their locations are distributed in the square $[0, 100]^2$ in the plane. The distances between customers are measured by Euclidean distances (in double precision), and the traveling times are the same as the corresponding distances. Each customer $i$ (including the depot) has one time window $[r_i, d_i]$, an amount of requirement $a_i$ and a service time $b_i$. All vehicles have an identical capacity $u$. Both time window and capacity constraints are considered hard. For these instances, the number of vehicles $m$ is also a decision variable, and the objective is to find a solution with the minimum vehicle number and the total traveling distance in the lexicographical order. These benchmark instances consist of six different sets of problem instances called R1, R2, RC1, RC2, C1 and C2, respectively. Locations of customers are uniformly and randomly distributed in type R and are clustered into groups in type C, and these two types are mixed in type RC. Furthermore, for the instances of type 1, the time window is narrow at the depot, and hence only a small number of customers can be served by one vehicle. On the contrary, for the instances of type 2, the time window at the depot is wide, and many customers can be served by one vehicle. Recently, 300 instances with larger number of customers are added by Gehring and Homberger [80], which are divided into five groups by the number of customers, 200, 400, 600, 800 and 1000, where each group has 10 instances for each of six types (i.e., R1, R2, RC1, RC2, C1 and C2).

In order to handle the above instances by our algorithm, we define time window cost function $p_i$, traveling cost functions $q_{ij}$ and traveling time functions $\lambda_{ij}$ as follows:

$$
p_i(t) = \begin{cases} 
\alpha(r_i - t), & t < r_i \\
0, & r_i \leq t \leq d_i \\
\alpha(t - d_i), & d_i < t,
\end{cases}
$$

$$
q_{ij}(t) = c_{ij},
$$

$$
\lambda_{ij}(t) = b_i + c_{ij},
$$

where $\alpha$ is a positive parameter and $c_{ij}$ is the distance (as well as the traveling time) between customers $i$ and $j$. Note that, in this formulation, the time window constraint is considered as soft, and can be violated if it is advantageous from the view point of minimizing the cost function. We set the number of vehicles in each instance to what is used in [86].
We first conduct preliminary experiments to determine parameter value $\alpha$ for each instance. We run the algorithm with some values of $\alpha$ from $\{1, 5, 10, 50, 100, 500, \ldots\}$ in the manner of binary search, where the time limit for each $\alpha$ was within 10% of the time limit reported in the tables in this section. Then we use the best $\alpha$ among them and the adjacent values in both directions (e.g., if the best results was obtained with $\alpha = 50$, we use 10, 50 and 100 for $\alpha$), run the algorithm by using the three values of $\alpha$ with 100% of the time limit, and report the best result below. If we cannot find a feasible solution, which satisfies the hard time window and capacity constraints, even with $\alpha = 1000000$, we increase the number of vehicles by one. Actually we could find a feasible solution for every instance except for six instances, and, for the six instances, we could find feasible solutions with one more vehicle.

We then compare the solutions obtained by our experiments with those obtained by existing methods. For Solomon’s instances, the time limit of our algorithm is 1000 seconds for each instance. The results are shown in Table 4.2. In this table, “CNV” represents the cumulative number of vehicles, and “CTD” represents the cumulative total distance, which are usually used in the literature to compare the results on Solomon’s instances. The upper (resp., lower) part of each cell in the table shows the mean number of vehicles (resp., the mean total distance) with respect to all instances for the type. Columns “IIKMUY”, “HG99”, “GH02”, “BBB”, “B”, “BVH”, “HG03”, “IINSUY” and “ILS” are the results of Ibaraki et al. [85], Homberger and Gehring [79], Gehring and Homberger [58], Berger et al. [18], Bräysy [22], Bent and Van Hentenryck [15], Homberger and Gehring [80], Ibaraki et al. [86] and our ILS algorithm, respectively. The bottom rows describe the computer, the average CPU time and the number of independent runs for each method reported by the author, where “P” and “SU” mean Pentium and SUN Ultra, respectively. The row “Computer” contains the clock frequency of the computer (e.g., “P 200” means a computer whose CPU is Pentium 200MHz). Computation time of “HG03” is not clearly stated in [80]. To make a fair comparison of the performance of various algorithms, we estimate the total computation time for each experiment by using the SPEC data presented in the web page of SPEC (http://www.specbench.org/). The row “Estimated time” represents this estimated time. An asterisk “*” in rows of the mean number of vehicles indicates that the value is the best among all the algorithms in the table and there is no tie. When there are ties for the number of vehicles, we give an asterisk “*” on the corresponding distance value that is the smallest among those ties. In the row CNV, all results with the smallest value get “*”.

The results for Gehring and Homberger’s instances are given in Tables 4.3–4.7. The time limit of our algorithm for 200, 400, 600, 800 and 1000-customer instances are 2000, 4000, 6000, 8000 and 10000 seconds, respectively. Columns “GH99”, “GH01”, “BVH”, 
“LL”, “LC”, “BHD”, “MB” and “IINSUY” are the results by Gehring and Homberger [56], Gehring and Homberger [58], Bent and Van Hentenryck [15], Li and Lim [104], Le Bouthillier and Crainic [102], Bräysy et al. [25], Mester and Bräysy [110] and Ibaraki et al. [86] respectively. “AMD” in the row “Computer” means Advanced Micro Devices and “n/a” in the row “CPU(min)” and “Runs” means that the data is not available.

In Table 4.2, our ILS obtained CNV 405 and the smallest CTD among all the tested algorithms. According to a recent survey by Bräysy and Gendreau [24], not many algorithms achieved CNV 407 or less, and only those algorithms cited in Table 4.2 achieved CNV 405. In Tables 4.3–4.7, we could also obtain the smallest CNV among the tested algorithms. The computation time of our ILS is reasonable compared to others especially for larger instances. These results indicate that our ILS is highly efficient to solve the vehicle routing problem with time windows, in spite of its high generality. As for the Solomon’s instances, the results are shown in Table 4.8. As for the Gehring and Homberger’s instances, the results are shown in Tables 4.9, 4.10, 4.11, 4.12 and 4.13. Each row of these tables represents a problem instance. “m” represents the number of vehicles, “d_{sum}” represents the total travel distance value, and “LS” represents the total number of local search procedure called in our iterated local search algorithm. Here it should be noted that we had to determine parameter $\alpha$ by preliminary experiments to achieve the above results, since the performance is crucially dependent on the value. Though the time spent for such tuning was not so large, it is one of the important directions of our future research to develop a mechanism to find a good value of $\alpha$ automatically.

### 4.6.3 Time-dependent VRPSTW

Our algorithm ILS is designed for more general problem than the standard VRPSTW, i.e., time-dependent VRPSTW. In order to test the performance of ILS, we generated 56 instances of time-dependent VRPSTW by modifying Solomon’s instances, as suggested by Ichoua et al. [87].

In these instances, all traveling between customers are categorized into three types, and the scheduling horizon (i.e., the time window at the depot) consists of morning, daytime and evening. The travel speed of a vehicle depends on the category and the period of scheduling horizon, which is further classified into three scenarios as shown in Table 4.14. Time-dependency is small, medium and large in scenarios 1, 2 and 3, respectively. Note that the average speed in each scenario is approximately 1, and the difficulty of time windows constraints is similar to Solomon’s instances. We define the time window cost
### Table 4.2: The results for 100-customer benchmark instances

<table>
<thead>
<tr>
<th>IIKMUY</th>
<th>HG09</th>
<th>GH02</th>
<th>BBB</th>
<th>B</th>
<th>BVH</th>
<th>HG03</th>
<th>IINSUY</th>
<th>ILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>10.00</td>
<td>10.00</td>
<td>10.00</td>
<td>10.00</td>
<td>10.00</td>
<td>10.00</td>
<td>10.00</td>
<td>10.00</td>
</tr>
<tr>
<td></td>
<td>828.38*</td>
<td>828.38*</td>
<td>828.63</td>
<td>828.48</td>
<td>828.38*</td>
<td>828.38*</td>
<td>828.38*</td>
<td>828.38*</td>
</tr>
<tr>
<td>C2</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
</tr>
<tr>
<td></td>
<td>589.86*</td>
<td>589.86*</td>
<td>590.33</td>
<td>589.93</td>
<td>589.86*</td>
<td>589.86*</td>
<td>589.86*</td>
<td>589.86*</td>
</tr>
<tr>
<td>R1</td>
<td>11.92</td>
<td>11.92</td>
<td>12.00</td>
<td>11.92</td>
<td>11.92</td>
<td>11.92</td>
<td>11.92</td>
<td>12.00</td>
</tr>
<tr>
<td></td>
<td>1217.40</td>
<td>1228.06</td>
<td>1217.57</td>
<td>1221.10</td>
<td>1222.12</td>
<td>1213.25</td>
<td>1212.73*</td>
<td>1217.99</td>
</tr>
<tr>
<td>R2</td>
<td>2.73</td>
<td>2.73</td>
<td>2.73</td>
<td>2.73</td>
<td>2.73</td>
<td>2.73</td>
<td>2.73</td>
<td>2.73</td>
</tr>
<tr>
<td></td>
<td>959.11</td>
<td>969.95</td>
<td>961.29</td>
<td>975.43</td>
<td>975.12</td>
<td>966.37</td>
<td>955.03*</td>
<td>967.97</td>
</tr>
<tr>
<td>RC1</td>
<td>11.50</td>
<td>11.63</td>
<td>11.50</td>
<td>11.50</td>
<td>11.50</td>
<td>11.50</td>
<td>11.50</td>
<td>11.63</td>
</tr>
<tr>
<td></td>
<td>1391.03</td>
<td>1392.57</td>
<td>1395.15</td>
<td>1389.89</td>
<td>1389.58</td>
<td>1384.22*</td>
<td>1386.44</td>
<td>1384.67</td>
</tr>
<tr>
<td></td>
<td>1122.79</td>
<td>1144.43</td>
<td>1139.37</td>
<td>1159.37</td>
<td>1128.38</td>
<td>1141.24</td>
<td>1123.17</td>
<td>1128.77</td>
</tr>
<tr>
<td>CNV</td>
<td>405*</td>
<td>406</td>
<td>406</td>
<td>405*</td>
<td>405*</td>
<td>405*</td>
<td>405*</td>
<td>405*</td>
</tr>
<tr>
<td>CTD</td>
<td>57444</td>
<td>57876</td>
<td>57641</td>
<td>57952</td>
<td>57710</td>
<td>57567</td>
<td>57309</td>
<td>57545</td>
</tr>
<tr>
<td></td>
<td>57282*</td>
<td>57282*</td>
<td>57282*</td>
<td>57282*</td>
<td>57282*</td>
<td>57282*</td>
<td>57282*</td>
<td>57282*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Computer</th>
<th>CPU (min)</th>
<th>Runs</th>
<th>Estimated time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P 1GHz</td>
<td>250.0</td>
<td>1</td>
<td>108.7</td>
</tr>
<tr>
<td>P 200</td>
<td>13.8</td>
<td>10</td>
<td>43.6</td>
</tr>
<tr>
<td>P 400</td>
<td>30.0</td>
<td>5</td>
<td>9.4</td>
</tr>
</tbody>
</table>

### Table 4.3: The results for 200-customer benchmark instances

<table>
<thead>
<tr>
<th>GH99</th>
<th>GH01</th>
<th>BVH</th>
<th>LL</th>
<th>LC</th>
<th>BHD</th>
<th>MB</th>
<th>IINSUY</th>
<th>ILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>18.9</td>
<td>18.9</td>
<td>18.9</td>
<td>19.1</td>
<td>18.9</td>
<td>18.9</td>
<td>18.8*</td>
<td>18.9</td>
</tr>
<tr>
<td></td>
<td>2782</td>
<td>2842.08</td>
<td>2726.63</td>
<td>2728.6</td>
<td>2743.66</td>
<td>2749.83</td>
<td>2717.21</td>
<td>2732.03</td>
</tr>
<tr>
<td>C2</td>
<td>1846</td>
<td>1856.99</td>
<td>1860.17</td>
<td>1854.9</td>
<td>1836.1</td>
<td>1842.65</td>
<td>1833.57*</td>
<td>1834.83</td>
</tr>
<tr>
<td></td>
<td>3705</td>
<td>3855.03</td>
<td>3677.96</td>
<td>3736.2</td>
<td>3676.95</td>
<td>3718.3</td>
<td>3618.68*</td>
<td>3665.77</td>
</tr>
<tr>
<td>R1</td>
<td>18.2</td>
<td>18.2</td>
<td>18.2</td>
<td>18.2</td>
<td>18.2</td>
<td>18.2</td>
<td>18.2</td>
<td>18.2</td>
</tr>
<tr>
<td></td>
<td>3055</td>
<td>3032.49</td>
<td>3023.62</td>
<td>3023</td>
<td>2986.01</td>
<td>3014.28</td>
<td>2942.92*</td>
<td>2965.64</td>
</tr>
<tr>
<td>R2</td>
<td>4.3</td>
<td>4.4</td>
<td>4.5</td>
<td>4.9</td>
<td>4.3</td>
<td>4.4</td>
<td>4.3</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td>2675</td>
<td>2671.34</td>
<td>2603.08</td>
<td>2518.7</td>
<td>2613.75</td>
<td>2585.89</td>
<td>2519.79</td>
<td>2562.56*</td>
</tr>
<tr>
<td>RC1</td>
<td>4.3</td>
<td>4.4</td>
<td>4.5</td>
<td>4.9</td>
<td>4.3</td>
<td>4.4</td>
<td>4.3</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td>2675</td>
<td>2671.34</td>
<td>2603.08</td>
<td>2518.7</td>
<td>2613.75</td>
<td>2585.89</td>
<td>2519.79</td>
<td>2562.56*</td>
</tr>
<tr>
<td>RC2</td>
<td>4.3</td>
<td>4.4</td>
<td>4.5</td>
<td>4.9</td>
<td>4.3</td>
<td>4.4</td>
<td>4.3</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td>2675</td>
<td>2671.34</td>
<td>2603.08</td>
<td>2518.7</td>
<td>2613.75</td>
<td>2585.89</td>
<td>2519.79</td>
<td>2562.56*</td>
</tr>
<tr>
<td>CNV</td>
<td>694*</td>
<td>696</td>
<td>697</td>
<td>707</td>
<td>694*</td>
<td>695</td>
<td>694*</td>
<td>694*</td>
</tr>
<tr>
<td>CTD</td>
<td>176180</td>
<td>179328</td>
<td>171715</td>
<td>172472</td>
<td>173061</td>
<td>172406</td>
<td>168573*</td>
<td>170484</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Computer</th>
<th>CPU (min)</th>
<th>Runs</th>
<th>Estimated time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P 200</td>
<td>4×10</td>
<td>1</td>
<td>2.4</td>
</tr>
<tr>
<td>P 400</td>
<td>3×2.1</td>
<td>3</td>
<td>3.8</td>
</tr>
<tr>
<td>SU 10</td>
<td>n/a</td>
<td>1</td>
<td>112.4</td>
</tr>
<tr>
<td>P 545</td>
<td>182.1</td>
<td>3</td>
<td>21.7</td>
</tr>
<tr>
<td>P 933</td>
<td>5×10</td>
<td>1</td>
<td>2.1</td>
</tr>
<tr>
<td>AMD 700</td>
<td>2.4</td>
<td>1</td>
<td>5.9</td>
</tr>
<tr>
<td>P 2GHz</td>
<td>8</td>
<td>3</td>
<td>33.3</td>
</tr>
<tr>
<td>P 2.8GHz</td>
<td>33.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P 2.8GHz</td>
<td>33.3</td>
<td>3</td>
<td>30.0</td>
</tr>
</tbody>
</table>
### Table 4.4: The results for 400-customer benchmark instances

<table>
<thead>
<tr>
<th></th>
<th>GH99</th>
<th>GH01</th>
<th>BVH</th>
<th>LL</th>
<th>LC</th>
<th>BHD</th>
<th>MB</th>
<th>IINSUY</th>
<th>ILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>37.9</td>
<td>37.9</td>
<td>37.9</td>
<td>37.9*</td>
<td>37.6*</td>
</tr>
<tr>
<td></td>
<td>7584</td>
<td>7855.82</td>
<td>7220.96</td>
<td>7181.4</td>
<td>7447.09</td>
<td>7230.48</td>
<td>7148.27</td>
<td>7282.15</td>
<td>7444.06</td>
</tr>
<tr>
<td>C2</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>11.8*</td>
</tr>
<tr>
<td></td>
<td>3935</td>
<td>3940.19</td>
<td>4154.4</td>
<td>4017.1</td>
<td>3940.87</td>
<td>3894.48</td>
<td>3840.85</td>
<td>3851.96</td>
<td>3982.50</td>
</tr>
<tr>
<td>R1</td>
<td>36.4</td>
<td>36.4</td>
<td>36.4</td>
<td>36.6</td>
<td>36.5</td>
<td>36.4</td>
<td>36.4*</td>
<td>36.4</td>
<td>36.4</td>
</tr>
<tr>
<td></td>
<td>8925</td>
<td>9478.22</td>
<td>8713.37</td>
<td>8912.4</td>
<td>8839.28</td>
<td>8692.17</td>
<td>8530.03</td>
<td>8746.94</td>
<td>8998.63</td>
</tr>
<tr>
<td>R2</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>6502</td>
<td>6650.28</td>
<td>6959.75</td>
<td>6610.6</td>
<td>6437.68</td>
<td>6382.63</td>
<td>6209.94*</td>
<td>6269.9</td>
<td>6258.00</td>
</tr>
<tr>
<td>RC1</td>
<td>36.1</td>
<td>36.1</td>
<td>36.1</td>
<td>36.5</td>
<td>36.</td>
<td>36.</td>
<td>36.</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>8763</td>
<td>9294.99</td>
<td>8330.98</td>
<td>8377.9</td>
<td>8652.01</td>
<td>8305.55</td>
<td>8066.44*</td>
<td>8405.32</td>
<td>8572.11</td>
</tr>
<tr>
<td>RC2</td>
<td>8.6</td>
<td>8.8</td>
<td>8.9</td>
<td>9.5</td>
<td>8.6</td>
<td>8.9</td>
<td>8.8</td>
<td>8.6</td>
<td>8.5*</td>
</tr>
<tr>
<td></td>
<td>5518</td>
<td>5629.43</td>
<td>5631.7</td>
<td>5466.2</td>
<td>5511.22</td>
<td>5407.87</td>
<td>5243.06</td>
<td>5337.5</td>
<td>5355.59</td>
</tr>
<tr>
<td>CNV</td>
<td>1390</td>
<td>1392</td>
<td>1393</td>
<td>1414</td>
<td>1390</td>
<td>1391</td>
<td>1389</td>
<td>1387</td>
<td>1383*</td>
</tr>
<tr>
<td>CTD</td>
<td>412270</td>
<td>428489</td>
<td>410112</td>
<td>406566</td>
<td>408281</td>
<td>399132</td>
<td>390386</td>
<td>398938</td>
<td>406109</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Computer</th>
<th>P 200</th>
<th>P 400</th>
<th>SU 10</th>
<th>P 545</th>
<th>P 933</th>
<th>AMD 700</th>
<th>P 2GHz</th>
<th>P 2.8GHz</th>
<th>P 2.8GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (min)</td>
<td>4×20</td>
<td>4×7.1</td>
<td>n/a</td>
<td>5×20</td>
<td>7.9</td>
<td>17</td>
<td>66.6</td>
<td>66.6</td>
<td></td>
</tr>
<tr>
<td>Runs</td>
<td>1</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Estimated time</td>
<td>4.8</td>
<td>13.0</td>
<td>n/a</td>
<td>221.8</td>
<td>43.3</td>
<td>6.8</td>
<td>12.5</td>
<td>66.6</td>
<td>66.6</td>
</tr>
</tbody>
</table>

### Table 4.5: The results for 600-customer benchmark instances

<table>
<thead>
<tr>
<th></th>
<th>GH99</th>
<th>GH01</th>
<th>BVH</th>
<th>LL</th>
<th>LC</th>
<th>BHD</th>
<th>MB</th>
<th>IINSUY</th>
<th>ILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>57.9</td>
<td>57.7</td>
<td>57.8</td>
<td>58.2</td>
<td>57.9</td>
<td>57.8</td>
<td>57.8</td>
<td>57.5</td>
<td>57.5</td>
</tr>
<tr>
<td></td>
<td>14792</td>
<td>14817.25</td>
<td>14357.11</td>
<td>14267.30</td>
<td>14205.58</td>
<td>14165.90</td>
<td>14003.09</td>
<td>14116.97*</td>
<td>14296.96</td>
</tr>
<tr>
<td>C2</td>
<td>17.9</td>
<td>17.8</td>
<td>17.8</td>
<td>18.2</td>
<td>17.9</td>
<td>18</td>
<td>17.8</td>
<td>17.4</td>
<td>17.4</td>
</tr>
<tr>
<td></td>
<td>7787</td>
<td>7889.96</td>
<td>8259.04</td>
<td>8202.60</td>
<td>7743.92</td>
<td>7528.73</td>
<td>7455.83</td>
<td>7945.56*</td>
<td>7960.138</td>
</tr>
<tr>
<td>R1</td>
<td>54.5</td>
<td>54.5</td>
<td>55</td>
<td>55.2</td>
<td>54.8</td>
<td>54.5</td>
<td>54.5</td>
<td>54.5</td>
<td>54.5</td>
</tr>
<tr>
<td></td>
<td>20854</td>
<td>21864.47</td>
<td>19308.62</td>
<td>19744.80</td>
<td>19869.82</td>
<td>19901.18</td>
<td>18358.68*</td>
<td>19844.39</td>
<td>20363.15</td>
</tr>
<tr>
<td>R2</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11.1</td>
<td>11.2</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>13335</td>
<td>13656.15</td>
<td>14855.43</td>
<td>13592.40</td>
<td>13093.97</td>
<td>13054.83</td>
<td>12703.52</td>
<td>12539.78*</td>
<td>13047.18</td>
</tr>
<tr>
<td>RC1</td>
<td>55.1</td>
<td>55</td>
<td>55.1</td>
<td>55.5</td>
<td>55.2</td>
<td>55</td>
<td>55</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>18411</td>
<td>19114.02</td>
<td>17035.91</td>
<td>17320.00</td>
<td>17678.13</td>
<td>16994.22</td>
<td>16418.63*</td>
<td>17278.81</td>
<td>17764.33</td>
</tr>
<tr>
<td>RC2</td>
<td>11.8</td>
<td>11.9</td>
<td>12.4</td>
<td>13</td>
<td>11.8</td>
<td>12.1</td>
<td>12.1</td>
<td>11.6</td>
<td>11.5*</td>
</tr>
<tr>
<td></td>
<td>11522</td>
<td>11670.29</td>
<td>11987.89</td>
<td>11204.90</td>
<td>11034.71</td>
<td>11212.36</td>
<td>10677.46</td>
<td>10791.70</td>
<td>11315.28</td>
</tr>
<tr>
<td>CNV</td>
<td>2082</td>
<td>2079</td>
<td>2091</td>
<td>2112</td>
<td>2088</td>
<td>2084</td>
<td>2082</td>
<td>2070</td>
<td>2069*</td>
</tr>
<tr>
<td>CTD</td>
<td>867010</td>
<td>890121</td>
<td>850804</td>
<td>843320</td>
<td>836261</td>
<td>820372</td>
<td>796172</td>
<td>825172</td>
<td>847470</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Computer</th>
<th>P 200</th>
<th>P 400</th>
<th>SU 10</th>
<th>P 545</th>
<th>P 933</th>
<th>AMD 700</th>
<th>P 2GHz</th>
<th>P 2.8GHz</th>
<th>P 2.8GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (min)</td>
<td>4×30</td>
<td>4×12.9</td>
<td>n/a</td>
<td>399.8</td>
<td>5×30</td>
<td>16.2</td>
<td>40</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Runs</td>
<td>1</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Estimated time</td>
<td>7.1</td>
<td>23.5</td>
<td>n/a</td>
<td>246.4</td>
<td>65.0</td>
<td>13.9</td>
<td>29.4</td>
<td>100.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>
### Table 4.6: The results for 800-customer benchmark instances

<table>
<thead>
<tr>
<th></th>
<th>GH99</th>
<th>GH01</th>
<th>BVH</th>
<th>LL</th>
<th>LC</th>
<th>BHD</th>
<th>MB</th>
<th>INSUY</th>
<th>ILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>76.7</td>
<td>76.1</td>
<td>76.1</td>
<td>77.4</td>
<td>76.3</td>
<td>76.3</td>
<td>76.2</td>
<td>75.7</td>
<td>75.6*</td>
</tr>
<tr>
<td></td>
<td>26528</td>
<td>26936.68</td>
<td>25391.67</td>
<td>25337.02</td>
<td>25668.82</td>
<td>25170.88</td>
<td>25132.27</td>
<td>25487.55</td>
<td>25915.59</td>
</tr>
<tr>
<td>C2</td>
<td>24</td>
<td>23.7</td>
<td>24.4</td>
<td>24.4</td>
<td>24.1</td>
<td>24.2</td>
<td>23.7</td>
<td>23.4</td>
<td>23.4</td>
</tr>
<tr>
<td></td>
<td>12451</td>
<td>11847.92</td>
<td>14253.83</td>
<td>11956.60</td>
<td>11985.11</td>
<td>11648.92</td>
<td>11352.29</td>
<td>11860.90*</td>
<td>11942.54</td>
</tr>
<tr>
<td>R1</td>
<td>72.8</td>
<td>72.8</td>
<td>72.7*</td>
<td>73</td>
<td>73.1</td>
<td>72.8</td>
<td>72.8</td>
<td>72.8</td>
<td>72.8</td>
</tr>
<tr>
<td></td>
<td>34586</td>
<td>34633.88</td>
<td>33337.91</td>
<td>33806.34</td>
<td>33552.40</td>
<td>32748.06</td>
<td>31918.47</td>
<td>33275.72</td>
<td>34095.04</td>
</tr>
<tr>
<td>R2</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>21697</td>
<td>21672.85</td>
<td>24554.63</td>
<td>21709.39</td>
<td>21157.15</td>
<td>21170.15</td>
<td>20295.28</td>
<td>20209.92*</td>
<td>20810.51</td>
</tr>
<tr>
<td>RC1</td>
<td>72.4</td>
<td>72.3</td>
<td>73</td>
<td>73.2</td>
<td>72.3</td>
<td>73</td>
<td>73</td>
<td>72.4</td>
<td>72.3</td>
</tr>
<tr>
<td></td>
<td>38509</td>
<td>40532.35</td>
<td>30500.15</td>
<td>31282.54</td>
<td>37722.62</td>
<td>30005.95</td>
<td>30731.07</td>
<td>34621.63</td>
<td>34358.45*</td>
</tr>
<tr>
<td>RC2</td>
<td>16.1</td>
<td>16.1</td>
<td>16.6</td>
<td>17.1</td>
<td>15.8</td>
<td>15.8</td>
<td>15.7</td>
<td>15.6*</td>
<td>17.13759</td>
</tr>
<tr>
<td></td>
<td>17741</td>
<td>17941.23</td>
<td>18940.84</td>
<td>17561.22</td>
<td>17441.60</td>
<td>16986.46</td>
<td>16421.92*</td>
<td>16266.76</td>
<td>17173.59</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNV</td>
<td>2770</td>
<td>2760</td>
<td>2778</td>
<td>2802</td>
<td>2766</td>
<td>2776</td>
<td>2765</td>
<td>2750</td>
<td>2747*</td>
</tr>
<tr>
<td>CTD</td>
<td>1515</td>
<td>153584.9</td>
<td>1469790</td>
<td>1416531</td>
<td>1475281</td>
<td>1384306</td>
<td>1361586</td>
<td>1421225</td>
<td>1442957</td>
</tr>
<tr>
<td>Computer</td>
<td>P 200</td>
<td>P 400</td>
<td>SU 10</td>
<td>P 545</td>
<td>P 933</td>
<td>AMD 700</td>
<td>P 2GHz</td>
<td>P 2.8GHz</td>
<td>P 2.8GHz</td>
</tr>
<tr>
<td>CPU (min)</td>
<td>4×40</td>
<td>4×23.2</td>
<td>n/a</td>
<td>5×40</td>
<td>26.2</td>
<td>145</td>
<td>133.3</td>
<td>133.3</td>
<td>133.3</td>
</tr>
<tr>
<td>Runs</td>
<td>1</td>
<td>3</td>
<td>n/a</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Estimated time</td>
<td>9.5</td>
<td>42.3</td>
<td>n/a</td>
<td>316.1</td>
<td>86.6</td>
<td>22.5</td>
<td>106.5</td>
<td>133.3</td>
<td>133.3</td>
</tr>
</tbody>
</table>

### Table 4.7: The results for 1000-customer benchmark instances

<table>
<thead>
<tr>
<th></th>
<th>GH99</th>
<th>GH01</th>
<th>BVH</th>
<th>LL</th>
<th>LC</th>
<th>BHD</th>
<th>MB</th>
<th>INSUY</th>
<th>ILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>96</td>
<td>95.4</td>
<td>95.1</td>
<td>96.3</td>
<td>95.3</td>
<td>95.8</td>
<td>95.1</td>
<td>94.5</td>
<td>94.4*</td>
</tr>
<tr>
<td></td>
<td>43237</td>
<td>43392.59</td>
<td>42505.35</td>
<td>42428.50</td>
<td>43283.92</td>
<td>42086.77</td>
<td>41589.67</td>
<td>42459.35</td>
<td>43066.89</td>
</tr>
<tr>
<td>C2</td>
<td>30.2</td>
<td>20.7</td>
<td>30.3</td>
<td>30.8</td>
<td>29.9</td>
<td>30.6</td>
<td>29.7</td>
<td>20.4</td>
<td>29.4</td>
</tr>
<tr>
<td></td>
<td>17570</td>
<td>17574.72</td>
<td>18546.13</td>
<td>17294.90</td>
<td>17443.50</td>
<td>17035.88</td>
<td>16698.46</td>
<td>16822.82*</td>
<td>16822.82*</td>
</tr>
<tr>
<td>R1</td>
<td>91.9</td>
<td>91.9</td>
<td>92.8</td>
<td>92.7</td>
<td>92.2</td>
<td>92.1</td>
<td>92.1</td>
<td>91.9</td>
<td>91.9</td>
</tr>
<tr>
<td></td>
<td>57186</td>
<td>58069.61</td>
<td>51193.47</td>
<td>50990.80</td>
<td>50176.95</td>
<td>50025.64</td>
<td>49281.48</td>
<td>53360.10*</td>
<td>54149.50</td>
</tr>
<tr>
<td>R2</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>31930</td>
<td>31873.62</td>
<td>36736.97</td>
<td>31990.90</td>
<td>30919.77</td>
<td>31458.23</td>
<td>29860.32</td>
<td>29546.10*</td>
<td>30626.04</td>
</tr>
<tr>
<td>RC1</td>
<td>90</td>
<td>90.1</td>
<td>90.2</td>
<td>90.4</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>50668</td>
<td>50550.14</td>
<td>48634.15</td>
<td>48892.40</td>
<td>49711.36</td>
<td>46736.92</td>
<td>45796.41*</td>
<td>48725.70</td>
<td>49378.71</td>
</tr>
<tr>
<td>RC2</td>
<td>19</td>
<td>18.5</td>
<td>19.4</td>
<td>19.8</td>
<td>18.5</td>
<td>19</td>
<td>18.7</td>
<td>18.3</td>
<td>18.3</td>
</tr>
<tr>
<td></td>
<td>27012</td>
<td>27175.98</td>
<td>29079.78</td>
<td>26042.30</td>
<td>26001.11</td>
<td>25994.12</td>
<td>25063.51</td>
<td>24904.08*</td>
<td>26248.81</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNV</td>
<td>3461</td>
<td>3446</td>
<td>3468</td>
<td>3490</td>
<td>3451</td>
<td>3465</td>
<td>3446</td>
<td>3431</td>
<td>3430*</td>
</tr>
<tr>
<td>CTD</td>
<td>2276390</td>
<td>2290367</td>
<td>2266959</td>
<td>2176398</td>
<td>2225366</td>
<td>2133376</td>
<td>2078110</td>
<td>2155374</td>
<td>2204728</td>
</tr>
<tr>
<td>Computer</td>
<td>P 200</td>
<td>P 400</td>
<td>SU 10</td>
<td>P 545</td>
<td>P 933</td>
<td>AMD 700</td>
<td>P 2GHz</td>
<td>P 2.8GHz</td>
<td>P 2.8GHz</td>
</tr>
<tr>
<td>CPU (min)</td>
<td>4×50</td>
<td>4×30.1</td>
<td>n/a</td>
<td>606.3</td>
<td>5×50</td>
<td>39.6</td>
<td>600</td>
<td>166.7</td>
<td>166.7</td>
</tr>
<tr>
<td>Runs</td>
<td>1</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Estimated time</td>
<td>11.9</td>
<td>54.9</td>
<td>n/a</td>
<td>373.5</td>
<td>108.3</td>
<td>34.0</td>
<td>440.8</td>
<td>166.7</td>
<td>166.7</td>
</tr>
</tbody>
</table>
### Table 4.8: The detailed results for 100-customer benchmark instances

<table>
<thead>
<tr>
<th>Inst</th>
<th>m</th>
<th>α</th>
<th>$d_{sum}$</th>
<th>LS</th>
<th>$d_{sum}$</th>
<th>Inst</th>
<th>m</th>
<th>α</th>
<th>$d_{sum}$</th>
<th>LS</th>
<th>$d_{sum}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>c101</td>
<td>10</td>
<td>5</td>
<td>828.94</td>
<td>10</td>
<td>828.94</td>
<td>c201</td>
<td>3</td>
<td>5</td>
<td>591.56</td>
<td>3</td>
<td>591.56</td>
</tr>
<tr>
<td>c102</td>
<td>10</td>
<td>5</td>
<td>829.94</td>
<td>10</td>
<td>829.94</td>
<td>c202</td>
<td>3</td>
<td>5</td>
<td>591.56</td>
<td>3</td>
<td>591.56</td>
</tr>
<tr>
<td>c103</td>
<td>10</td>
<td>5</td>
<td>828.06</td>
<td>10</td>
<td>828.06</td>
<td>c203</td>
<td>3</td>
<td>5</td>
<td>591.17</td>
<td>3</td>
<td>591.17</td>
</tr>
<tr>
<td>c104</td>
<td>10</td>
<td>5</td>
<td>824.78</td>
<td>10</td>
<td>824.78</td>
<td>c204</td>
<td>3</td>
<td>5</td>
<td>590.60</td>
<td>3</td>
<td>590.60</td>
</tr>
<tr>
<td>c105</td>
<td>10</td>
<td>5</td>
<td>829.94</td>
<td>10</td>
<td>829.94</td>
<td>c205</td>
<td>3</td>
<td>5</td>
<td>588.88</td>
<td>3</td>
<td>588.88</td>
</tr>
<tr>
<td>c106</td>
<td>10</td>
<td>5</td>
<td>828.94</td>
<td>10</td>
<td>828.94</td>
<td>c206</td>
<td>3</td>
<td>5</td>
<td>588.49</td>
<td>3</td>
<td>588.49</td>
</tr>
<tr>
<td>c107</td>
<td>10</td>
<td>5</td>
<td>828.94</td>
<td>10</td>
<td>828.94</td>
<td>c207</td>
<td>3</td>
<td>5</td>
<td>588.29</td>
<td>3</td>
<td>588.29</td>
</tr>
<tr>
<td>c108</td>
<td>10</td>
<td>5</td>
<td>828.94</td>
<td>10</td>
<td>828.94</td>
<td>c208</td>
<td>3</td>
<td>5</td>
<td>588.32</td>
<td>3</td>
<td>588.32</td>
</tr>
<tr>
<td>c109</td>
<td>10</td>
<td>5</td>
<td>828.94</td>
<td>10</td>
<td>828.94</td>
<td>r101</td>
<td>19</td>
<td>100</td>
<td>1650.80</td>
<td>19</td>
<td>1645.79</td>
</tr>
<tr>
<td>r102</td>
<td>17</td>
<td>100</td>
<td>1486.12</td>
<td>17</td>
<td>1486.12</td>
<td>r103</td>
<td>13</td>
<td>50</td>
<td>1292.68</td>
<td>13</td>
<td>1292.68</td>
</tr>
<tr>
<td>r104</td>
<td>9</td>
<td>500000</td>
<td>1007.31</td>
<td>9</td>
<td>1007.24</td>
<td>r105</td>
<td>14</td>
<td>5</td>
<td>1377.11</td>
<td>14</td>
<td>1377.11</td>
</tr>
<tr>
<td>r106</td>
<td>12</td>
<td>5</td>
<td>1251.98</td>
<td>12</td>
<td>1251.98</td>
<td>r107</td>
<td>10</td>
<td>50</td>
<td>1104.66</td>
<td>10</td>
<td>1104.66</td>
</tr>
<tr>
<td>r108</td>
<td>9</td>
<td>10</td>
<td>963.99</td>
<td>9</td>
<td>960.88</td>
<td>r109</td>
<td>11</td>
<td>10</td>
<td>1205.36</td>
<td>11</td>
<td>1194.73</td>
</tr>
<tr>
<td>r110</td>
<td>10</td>
<td>10</td>
<td>1129.47</td>
<td>10</td>
<td>1118.59</td>
<td>r111</td>
<td>10</td>
<td>50</td>
<td>1096.73</td>
<td>10</td>
<td>1096.72</td>
</tr>
<tr>
<td>r112</td>
<td>9</td>
<td>100</td>
<td>991.85</td>
<td>9</td>
<td>982.14</td>
<td>rc101</td>
<td>14</td>
<td>100</td>
<td>1696.95</td>
<td>14</td>
<td>1696.94</td>
</tr>
<tr>
<td>rc102</td>
<td>12</td>
<td>100</td>
<td>1554.75</td>
<td>12</td>
<td>1554.75</td>
<td>rc103</td>
<td>11</td>
<td>10</td>
<td>1262.02</td>
<td>11</td>
<td>1261.67</td>
</tr>
<tr>
<td>rc104</td>
<td>10</td>
<td>5</td>
<td>1135.83</td>
<td>10</td>
<td>1135.48</td>
<td>rc105</td>
<td>13</td>
<td>100</td>
<td>1629.44</td>
<td>13</td>
<td>1629.44</td>
</tr>
<tr>
<td>rc106</td>
<td>11</td>
<td>100</td>
<td>1424.73</td>
<td>11</td>
<td>1424.73</td>
<td>rc107</td>
<td>11</td>
<td>50</td>
<td>1230.48</td>
<td>11</td>
<td>1230.48</td>
</tr>
<tr>
<td>rc108</td>
<td>10</td>
<td>50</td>
<td>1139.82</td>
<td>10</td>
<td>1139.82</td>
<td>rc109</td>
<td>10</td>
<td>5</td>
<td>828.94</td>
<td>10</td>
<td>828.94</td>
</tr>
</tbody>
</table>
Table 4.9: The detailed results for 200-customer benchmark instances

<table>
<thead>
<tr>
<th>Inst</th>
<th>m</th>
<th>α</th>
<th>$d_{sum}$</th>
<th>LS</th>
<th>$d_{sum}$</th>
<th>Inst</th>
<th>m</th>
<th>α</th>
<th>$d_{sum}$</th>
<th>LS</th>
<th>$d_{sum}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>c101</td>
<td>20</td>
<td>5</td>
<td>2704.57</td>
<td>22870</td>
<td>20 2704.57</td>
<td>c201</td>
<td>6</td>
<td>5</td>
<td>1931.44</td>
<td>4168</td>
<td>6 1931.44</td>
</tr>
<tr>
<td>c102</td>
<td>18</td>
<td>10</td>
<td>2917.89</td>
<td>11949</td>
<td>18 2917.89</td>
<td>c202</td>
<td>6</td>
<td>5</td>
<td>1863.16</td>
<td>5011</td>
<td>6 1863.16</td>
</tr>
<tr>
<td>c103</td>
<td>18</td>
<td>1</td>
<td>2707.35</td>
<td>14031</td>
<td>18 2708.08</td>
<td>c203</td>
<td>6</td>
<td>1</td>
<td>1786.39</td>
<td>5869</td>
<td>6 1775.11</td>
</tr>
<tr>
<td>c104</td>
<td>18</td>
<td>1</td>
<td>2649.99</td>
<td>14891</td>
<td>18 2644.61</td>
<td>c204</td>
<td>6</td>
<td>5</td>
<td>1733.40</td>
<td>7505</td>
<td>6 1720.09</td>
</tr>
<tr>
<td>c105</td>
<td>20</td>
<td>5</td>
<td>2702.05</td>
<td>18817</td>
<td>20 2702.05</td>
<td>c205</td>
<td>6</td>
<td>1</td>
<td>1878.85</td>
<td>6071</td>
<td>6 1878.85</td>
</tr>
<tr>
<td>c106</td>
<td>20</td>
<td>5</td>
<td>2701.04</td>
<td>20996</td>
<td>20 2701.04</td>
<td>c206</td>
<td>6</td>
<td>1</td>
<td>1857.35</td>
<td>7177</td>
<td>6 1857.35</td>
</tr>
<tr>
<td>c107</td>
<td>20</td>
<td>5</td>
<td>2701.04</td>
<td>20453</td>
<td>20 2701.04</td>
<td>c207</td>
<td>6</td>
<td>1</td>
<td>1849.46</td>
<td>6975</td>
<td>6 1849.46</td>
</tr>
<tr>
<td>c108</td>
<td>19</td>
<td>50</td>
<td>2793.58</td>
<td>17304</td>
<td>18 2769.19</td>
<td>c208</td>
<td>6</td>
<td>1</td>
<td>1820.53</td>
<td>8309</td>
<td>6 1820.59</td>
</tr>
<tr>
<td>c109</td>
<td>18</td>
<td>10</td>
<td>2693.99</td>
<td>14666</td>
<td>18 2642.82</td>
<td>c209</td>
<td>6</td>
<td>1</td>
<td>1832.43</td>
<td>7834</td>
<td>6 1830.18</td>
</tr>
<tr>
<td>c110</td>
<td>18</td>
<td>5</td>
<td>2647.92</td>
<td>16678</td>
<td>18 2649.26</td>
<td>c210</td>
<td>6</td>
<td>1</td>
<td>1806.58</td>
<td>9139</td>
<td>6 1806.60</td>
</tr>
<tr>
<td>r101</td>
<td>20</td>
<td>1000</td>
<td>4795.04</td>
<td>17905</td>
<td>19 5024.65</td>
<td>r201</td>
<td>4</td>
<td>1000000</td>
<td>4520.81</td>
<td>1389</td>
<td>4 4501.80</td>
</tr>
<tr>
<td>r102</td>
<td>18</td>
<td>1000</td>
<td>4157.01</td>
<td>13533</td>
<td>18 4054.44</td>
<td>r202</td>
<td>4</td>
<td>100</td>
<td>3667.70</td>
<td>1949</td>
<td>4 3645.38</td>
</tr>
<tr>
<td>r103</td>
<td>18</td>
<td>50</td>
<td>3458.01</td>
<td>12306</td>
<td>18 3382.65</td>
<td>r203</td>
<td>4</td>
<td>1000</td>
<td>2891.23</td>
<td>3010</td>
<td>4 2932.44</td>
</tr>
<tr>
<td>r104</td>
<td>18</td>
<td>50</td>
<td>3088.56</td>
<td>12037</td>
<td>18 3067.93</td>
<td>r204</td>
<td>4</td>
<td>1</td>
<td>1988.23</td>
<td>4373</td>
<td>4 1981.29</td>
</tr>
<tr>
<td>r105</td>
<td>18</td>
<td>100</td>
<td>4190.21</td>
<td>11595</td>
<td>18 4112.88</td>
<td>r205</td>
<td>4</td>
<td>5</td>
<td>3367.53</td>
<td>2354</td>
<td>4 3367.55</td>
</tr>
<tr>
<td>r106</td>
<td>18</td>
<td>50</td>
<td>3719.57</td>
<td>12072</td>
<td>18 3599.84</td>
<td>r206</td>
<td>4</td>
<td>10</td>
<td>2914.76</td>
<td>2754</td>
<td>4 2914.56</td>
</tr>
<tr>
<td>r107</td>
<td>18</td>
<td>50</td>
<td>3195.05</td>
<td>13352</td>
<td>18 3151.42</td>
<td>r207</td>
<td>4</td>
<td>50</td>
<td>2456.05</td>
<td>4100</td>
<td>4 2453.62</td>
</tr>
<tr>
<td>r108</td>
<td>18</td>
<td>1</td>
<td>2982.37</td>
<td>9881</td>
<td>18 2963.90</td>
<td>r208</td>
<td>4</td>
<td>5</td>
<td>1849.98</td>
<td>6700</td>
<td>4 1849.87</td>
</tr>
<tr>
<td>r109</td>
<td>18</td>
<td>50</td>
<td>3909.27</td>
<td>12480</td>
<td>18 3784.33</td>
<td>r209</td>
<td>4</td>
<td>10</td>
<td>3115.72</td>
<td>2722</td>
<td>4 3111.41</td>
</tr>
<tr>
<td>r110</td>
<td>18</td>
<td>50</td>
<td>3408.31</td>
<td>13821</td>
<td>18 3307.78</td>
<td>r210</td>
<td>4</td>
<td>50</td>
<td>2666.82</td>
<td>3506</td>
<td>4 2657.00</td>
</tr>
<tr>
<td>rc101</td>
<td>18</td>
<td>100</td>
<td>3769.86</td>
<td>11424</td>
<td>18 3691.99</td>
<td>rc201</td>
<td>6</td>
<td>100</td>
<td>3125.75</td>
<td>5693</td>
<td>6 3103.48</td>
</tr>
<tr>
<td>rc102</td>
<td>18</td>
<td>50</td>
<td>3379.01</td>
<td>12974</td>
<td>18 3298.68</td>
<td>rc202</td>
<td>5</td>
<td>500</td>
<td>2829.45</td>
<td>4029</td>
<td>5 2827.45</td>
</tr>
<tr>
<td>rc103</td>
<td>18</td>
<td>50</td>
<td>3110.69</td>
<td>14202</td>
<td>18 3025.90</td>
<td>rc203</td>
<td>4</td>
<td>500</td>
<td>2618.23</td>
<td>3056</td>
<td>4 2617.90</td>
</tr>
<tr>
<td>rc104</td>
<td>18</td>
<td>5</td>
<td>2917.42</td>
<td>12541</td>
<td>18 2879.40</td>
<td>rc204</td>
<td>4</td>
<td>1</td>
<td>2103.47</td>
<td>3528</td>
<td>4 2055.97</td>
</tr>
<tr>
<td>rc105</td>
<td>18</td>
<td>100</td>
<td>3685.57</td>
<td>13026</td>
<td>18 3419.81</td>
<td>rc205</td>
<td>4</td>
<td>5</td>
<td>2933.33</td>
<td>2103</td>
<td>4 2912.57</td>
</tr>
<tr>
<td>rc106</td>
<td>18</td>
<td>50</td>
<td>3474.90</td>
<td>13496</td>
<td>18 3393.09</td>
<td>rc206</td>
<td>4</td>
<td>100</td>
<td>2889.42</td>
<td>2315</td>
<td>4 3138.02</td>
</tr>
<tr>
<td>rc107</td>
<td>18</td>
<td>10</td>
<td>3471.25</td>
<td>12592</td>
<td>18 3266.48</td>
<td>rc207</td>
<td>4</td>
<td>5</td>
<td>2557.40</td>
<td>3247</td>
<td>4 2550.56</td>
</tr>
<tr>
<td>rc108</td>
<td>18</td>
<td>10</td>
<td>3259.56</td>
<td>13221</td>
<td>18 3115.82</td>
<td>rc208</td>
<td>4</td>
<td>1</td>
<td>2361.34</td>
<td>3348</td>
<td>4 2317.80</td>
</tr>
<tr>
<td>rc109</td>
<td>18</td>
<td>10</td>
<td>3251.53</td>
<td>13826</td>
<td>18 3083.41</td>
<td>rc209</td>
<td>4</td>
<td>5</td>
<td>2198.52</td>
<td>4182</td>
<td>4 2175.61</td>
</tr>
<tr>
<td>rc110</td>
<td>18</td>
<td>5</td>
<td>3130.30</td>
<td>12097</td>
<td>18 3038.85</td>
<td>rc210</td>
<td>4</td>
<td>5</td>
<td>2029.88</td>
<td>4841</td>
<td>4 2015.60</td>
</tr>
</tbody>
</table>
### Table 4.10: The detailed results for 400-customer benchmark instances

<table>
<thead>
<tr>
<th>Inst</th>
<th>m</th>
<th>α</th>
<th>$d_{sum}$</th>
<th>LS</th>
<th>$d_{sum}$</th>
<th>Inst</th>
<th>m</th>
<th>α</th>
<th>$d_{sum}$</th>
<th>LS</th>
<th>$d_{sum}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>c101</td>
<td>40</td>
<td>1</td>
<td>7152.06</td>
<td>11893</td>
<td>40</td>
<td>7152.02</td>
<td>c201</td>
<td>12</td>
<td>5</td>
<td>4116.14</td>
<td>3296</td>
</tr>
<tr>
<td>c102</td>
<td>36</td>
<td>5000</td>
<td>7856.66</td>
<td>6817</td>
<td>37</td>
<td>7357.45</td>
<td>c202</td>
<td>12</td>
<td>1</td>
<td>3930.45</td>
<td>4605</td>
</tr>
<tr>
<td>c103</td>
<td>36</td>
<td>5</td>
<td>7363.31</td>
<td>6264</td>
<td>36</td>
<td>7151.17</td>
<td>c203</td>
<td>12</td>
<td>1</td>
<td>3779.77</td>
<td>4978</td>
</tr>
<tr>
<td>c104</td>
<td>36</td>
<td>1</td>
<td>6869.50</td>
<td>6508</td>
<td>36</td>
<td>6822.18</td>
<td>c204</td>
<td>11</td>
<td>1000000</td>
<td>4350.20</td>
<td>3120</td>
</tr>
<tr>
<td>c105</td>
<td>40</td>
<td>5</td>
<td>7152.06</td>
<td>9751</td>
<td>40</td>
<td>7152.02</td>
<td>c205</td>
<td>12</td>
<td>1</td>
<td>3938.69</td>
<td>5564</td>
</tr>
<tr>
<td>c106</td>
<td>40</td>
<td>1</td>
<td>7153.45</td>
<td>10622</td>
<td>40</td>
<td>7153.41</td>
<td>c206</td>
<td>12</td>
<td>1</td>
<td>3875.94</td>
<td>6480</td>
</tr>
<tr>
<td>c107</td>
<td>39</td>
<td>5000</td>
<td>7505.24</td>
<td>8914</td>
<td>39</td>
<td>8043.18</td>
<td>c207</td>
<td>12</td>
<td>10</td>
<td>3897.70</td>
<td>6325</td>
</tr>
<tr>
<td>c108</td>
<td>37</td>
<td>10000</td>
<td>7882.36</td>
<td>8587</td>
<td>38</td>
<td>7113.40</td>
<td>c208</td>
<td>12</td>
<td>1</td>
<td>3798.66</td>
<td>6732</td>
</tr>
<tr>
<td>c109</td>
<td>36</td>
<td>1000</td>
<td>8086.45</td>
<td>7491</td>
<td>36</td>
<td>7524.32</td>
<td>c209</td>
<td>12</td>
<td>10</td>
<td>3879.83</td>
<td>6728</td>
</tr>
<tr>
<td>c110</td>
<td>36</td>
<td>5</td>
<td>7419.52</td>
<td>6683</td>
<td>36</td>
<td>6907.26</td>
<td>c210</td>
<td>11</td>
<td>5000</td>
<td>4257.64</td>
<td>4662</td>
</tr>
<tr>
<td>r101</td>
<td>40</td>
<td>5000</td>
<td>10547.11</td>
<td>9961</td>
<td>38</td>
<td>11084.00</td>
<td>r201</td>
<td>8</td>
<td>5</td>
<td>9319.21</td>
<td>1860</td>
</tr>
<tr>
<td>r102</td>
<td>36</td>
<td>5000</td>
<td>9610.16</td>
<td>5937</td>
<td>36</td>
<td>9161.26</td>
<td>r202</td>
<td>8</td>
<td>1000</td>
<td>7662.25</td>
<td>2408</td>
</tr>
<tr>
<td>r103</td>
<td>36</td>
<td>5000</td>
<td>8513.14</td>
<td>5744</td>
<td>36</td>
<td>7941.53</td>
<td>r203</td>
<td>8</td>
<td>1000</td>
<td>6044.85</td>
<td>2811</td>
</tr>
<tr>
<td>r104</td>
<td>36</td>
<td>10</td>
<td>7649.41</td>
<td>4952</td>
<td>36</td>
<td>7332.93</td>
<td>r204</td>
<td>8</td>
<td>5</td>
<td>4348.34</td>
<td>4269</td>
</tr>
<tr>
<td>r105</td>
<td>36</td>
<td>50</td>
<td>10270.00</td>
<td>5412</td>
<td>36</td>
<td>9512.25</td>
<td>r205</td>
<td>8</td>
<td>10</td>
<td>7191.03</td>
<td>2733</td>
</tr>
<tr>
<td>r106</td>
<td>36</td>
<td>100</td>
<td>9197.03</td>
<td>5940</td>
<td>36</td>
<td>8534.05</td>
<td>r206</td>
<td>8</td>
<td>10</td>
<td>6246.39</td>
<td>2917</td>
</tr>
<tr>
<td>r107</td>
<td>36</td>
<td>50</td>
<td>8089.12</td>
<td>5630</td>
<td>36</td>
<td>7710.41</td>
<td>r207</td>
<td>8</td>
<td>5</td>
<td>5140.19</td>
<td>3631</td>
</tr>
<tr>
<td>r108</td>
<td>36</td>
<td>5</td>
<td>7701.29</td>
<td>4659</td>
<td>36</td>
<td>7398.68</td>
<td>r208</td>
<td>8</td>
<td>5</td>
<td>4124.64</td>
<td>5357</td>
</tr>
<tr>
<td>r109</td>
<td>36</td>
<td>10</td>
<td>9660.98</td>
<td>5075</td>
<td>36</td>
<td>8878.19</td>
<td>r209</td>
<td>8</td>
<td>5</td>
<td>6486.50</td>
<td>2795</td>
</tr>
<tr>
<td>r110</td>
<td>36</td>
<td>50</td>
<td>8784.10</td>
<td>5799</td>
<td>36</td>
<td>8227.49</td>
<td>r210</td>
<td>8</td>
<td>10</td>
<td>6016.55</td>
<td>3598</td>
</tr>
<tr>
<td>rc101</td>
<td>36</td>
<td>1000</td>
<td>9769.63</td>
<td>6505</td>
<td>36</td>
<td>8960.82</td>
<td>rc201</td>
<td>11</td>
<td>5</td>
<td>6770.44</td>
<td>3784</td>
</tr>
<tr>
<td>rc102</td>
<td>36</td>
<td>5000</td>
<td>8820.22</td>
<td>6916</td>
<td>36</td>
<td>8174.27</td>
<td>rc202</td>
<td>9</td>
<td>1000</td>
<td>6419.16</td>
<td>1990</td>
</tr>
<tr>
<td>rc103</td>
<td>36</td>
<td>50</td>
<td>7973.35</td>
<td>7037</td>
<td>36</td>
<td>7737.99</td>
<td>rc203</td>
<td>8</td>
<td>5</td>
<td>5048.38</td>
<td>2590</td>
</tr>
<tr>
<td>rc104</td>
<td>36</td>
<td>10</td>
<td>7551.31</td>
<td>5666</td>
<td>36</td>
<td>7411.02</td>
<td>rc204</td>
<td>8</td>
<td>10</td>
<td>3700.01</td>
<td>5338</td>
</tr>
<tr>
<td>rc105</td>
<td>36</td>
<td>100</td>
<td>8948.55</td>
<td>7427</td>
<td>36</td>
<td>8499.15</td>
<td>rc205</td>
<td>9</td>
<td>50</td>
<td>6047.21</td>
<td>2712</td>
</tr>
<tr>
<td>rc106</td>
<td>36</td>
<td>10</td>
<td>8873.07</td>
<td>6521</td>
<td>36</td>
<td>8304.99</td>
<td>rc206</td>
<td>8</td>
<td>50</td>
<td>5998.19</td>
<td>2102</td>
</tr>
<tr>
<td>rc107</td>
<td>36</td>
<td>10</td>
<td>8898.66</td>
<td>6386</td>
<td>36</td>
<td>8051.71</td>
<td>rc207</td>
<td>8</td>
<td>100</td>
<td>5570.20</td>
<td>3172</td>
</tr>
<tr>
<td>rc108</td>
<td>36</td>
<td>5</td>
<td>8493.26</td>
<td>5002</td>
<td>36</td>
<td>7917.68</td>
<td>rc208</td>
<td>8</td>
<td>1</td>
<td>4916.86</td>
<td>2885</td>
</tr>
<tr>
<td>rc109</td>
<td>36</td>
<td>5</td>
<td>8282.77</td>
<td>5030</td>
<td>36</td>
<td>7890.45</td>
<td>rc209</td>
<td>8</td>
<td>50</td>
<td>4657.48</td>
<td>4810</td>
</tr>
<tr>
<td>rc110</td>
<td>36</td>
<td>50</td>
<td>8110.30</td>
<td>7046</td>
<td>36</td>
<td>7716.32</td>
<td>rc210</td>
<td>8</td>
<td>5</td>
<td>4427.98</td>
<td>4326</td>
</tr>
</tbody>
</table>
Table 4.11: The detailed results for 600-customer benchmark instances

| Inst | m  | α  | \(d_{\text{sum}}\) | LS | \(d_{\text{sum}}\) | Inst | m  | α  | \(d_{\text{sum}}\) | LS | \(d_{\text{sum}}\) |
|------|----|----|-------------------|----|-------------------|------|----|-------------------|----|-------------------|
| c101 | 60 | 5  | 14095.64         | 7222 | 60 14095.64      | c201 | 18 | 1   | 7774.16         | 3407 | 18 7774.16       |
| c102 | 56 | 500 | 14209.47        | 4840 | 56   14325.96    | c202 | 17 | 1000 | 8784.11        | 2208 | 18 7486.88       |
| c103 | 56 | 5  | 13934.96         | 4264 | 56  13898.99     | c203 | 17 | 17  | 7977.15         | 2401 | 17 8371.07       |
| c104 | 56 | 10 | 13864.79         | 4479 | 56   13610.66    | c204 | 17 | 5   | 7474.75         | 3129 | 17 7216.45       |
| c105 | 60 | 10 | 14085.72         | 6278 | 60  14085.70     | c205 | 18 | 5   | 7576.44         | 4415 | 18 7576.35       |
| c106 | 60 | 500 | 14209.47        | 4840 | 56  14325.96     | c206 | 18 | 1   | 7479.48         | 5428 | 18 7478.63       |
| c107 | 59 | 10000 | 14580.31     | 6403 | 59   14659.74    | c207 | 18 | 1   | 7535.05         | 4797 | 18 7560.53       |
| c108 | 56 | 10000 | 15437.77     | 6056 | 57   14976.88    | c208 | 17 | 100 | 8169.53         | 4031 | 18 7352.42       |
| c109 | 56 | 100 | 14543.28        | 4323 | 56  13733.56     | c209 | 17 | 1000 | 9168.68        | 3193 | 18 7350.94       |
| c110 | 56 | 1   | 14127.95        | 4164 | 56   13758.19    | c210 | 17 | 1   | 7662.03         | 3967 | 17 7523.34       |
| r101 | 59 | 1000 | 21857.78       | 6509 | 59   21131.09    | r201 | 11 | 50  | 19060.46       | 1242 | 11 18325.60      |
| r102 | 54 | 1000 | 21734.84       | 5076 | 54   19603.70    | r202 | 11 | 500 | 15473.66       | 1465 | 11 15346.42      |
| r103 | 54 | 50  | 19475.54        | 3600 | 54   17400.60    | r203 | 11 | 1000 | 12045.89       | 1728 | 11 11663.06      |
| r104 | 54 | 50  | 17391.78        | 2954 | 54   15993.80    | r204 | 11 | 10  | 8503.38        | 2299 | 11 8386.64       |
| r105 | 54 | 1000 | 22962.20      | 4538 | 54   20395.00    | r205 | 11 | 1000 | 15871.96       | 1791 | 11 15640.60      |
| r106 | 54 | 100 | 21178.18        | 3578 | 54   18620.26    | r206 | 11 | 50  | 13565.21       | 1775 | 11 12937.47      |
| r107 | 54 | 50  | 18857.78        | 3460 | 54   17107.91    | r207 | 11 | 50  | 10565.35       | 2120 | 11 10536.84      |
| r108 | 54 | 50  | 17033.88        | 2912 | 54   15725.86    | r208 | 11 | 10  | 8254.00        | 2700 | 11 8023.64       |
| r109 | 54 | 1000 | 22315.90      | 4404 | 54   19372.96    | r209 | 11 | 50  | 14245.68       | 1839 | 11 13567.84      |
| r110 | 54 | 50  | 20823.64        | 3557 | 54   18235.57    | r210 | 11 | 1000 | 12886.16      | 2303 | 11 12607.09      |
| rc101 | 55 | 1000 | 19365.42     | 4426 | 55   17454.39    | rc201 | 14 | 100 | 13753.35       | 2216 | 15 13275.93      |
| rc102 | 55 | 500 | 17752.43       | 4475 | 55   16208.24    | rc202 | 12 | 1000 | 11756.29       | 1789 | 12 12071.40      |
| rc103 | 55 | 10  | 16461.69       | 4178 | 55   15524.33    | rc203 | 11 | 50  | 10248.58       | 1409 | 11 9978.25       |
| rc104 | 55 | 10  | 15546.46       | 3652 | 55   15180.72    | rc204 | 11 | 10  | 7894.73        | 2070 | 11 7349.88       |
| rc105 | 55 | 500 | 18828.10       | 4433 | 55   17468.57    | rc205 | 12 | 10000 | 12757.40     | 2071 | 13 11919.72      |
| rc106 | 55 | 1000 | 18583.52      | 4331 | 55   17248.87    | rc206 | 11 | 1000 | 13396.83       | 1202 | 12 11411.08      |
| rc107 | 55 | 1000 | 18167.54      | 4495 | 55   16454.79    | rc207 | 11 | 1000 | 11808.20       | 1709 | 11 11687.04      |
| rc108 | 55 | 50  | 17863.08       | 4216 | 55   16462.49    | rc208 | 11 | 10  | 10978.22       | 1784 | 11 10474.95      |
| rc109 | 55 | 50  | 17653.90       | 4004 | 55   16153.00    | rc209 | 11 | 10  | 10593.09       | 2094 | 11 10113.82      |
| rc110 | 55 | 10  | 17421.19       | 3584 | 55   16030.86    | rc210 | 11 | 5   | 9966.14         | 1987 | 11 9339.41       |
### Table 4.12: The detailed results for 800-customer benchmark instances

<table>
<thead>
<tr>
<th>Inst m</th>
<th>$\alpha$</th>
<th>$d_{sum}$</th>
<th>LS</th>
<th>Inst m</th>
<th>$\alpha$</th>
<th>$d_{sum}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>c101 80</td>
<td>5</td>
<td>25184.38</td>
<td>5074</td>
<td>80</td>
<td>25030.36</td>
<td>c201 24</td>
</tr>
<tr>
<td>c102 74</td>
<td>1000000</td>
<td>26114.66</td>
<td>4387</td>
<td>75</td>
<td>25518.17</td>
<td>c202 23</td>
</tr>
<tr>
<td>c103 72</td>
<td>5</td>
<td>26213.54</td>
<td>2472</td>
<td>72</td>
<td>25438.60</td>
<td>c203 23</td>
</tr>
<tr>
<td>c104 72</td>
<td>1</td>
<td>24719.93</td>
<td>2225</td>
<td>72</td>
<td>24040.47</td>
<td>c204 23</td>
</tr>
<tr>
<td>c105 80</td>
<td>100</td>
<td>25166.28</td>
<td>4669</td>
<td>80</td>
<td>25166.30</td>
<td>c205 24</td>
</tr>
<tr>
<td>c106 80</td>
<td>100</td>
<td>25160.85</td>
<td>4728</td>
<td>80</td>
<td>25160.90</td>
<td>c206 24</td>
</tr>
<tr>
<td>c107 79</td>
<td>500</td>
<td>25538.54</td>
<td>3758</td>
<td>79</td>
<td>25518.85</td>
<td>c207 24</td>
</tr>
<tr>
<td>c108 75</td>
<td>10000</td>
<td>26243.46</td>
<td>4672</td>
<td>76</td>
<td>25379.85</td>
<td>c208 23</td>
</tr>
<tr>
<td>c109 72</td>
<td>1000</td>
<td>27827.13</td>
<td>3729</td>
<td>73</td>
<td>24713.38</td>
<td>c209 23</td>
</tr>
<tr>
<td>c110 72</td>
<td>10</td>
<td>26987.10</td>
<td>2845</td>
<td>72</td>
<td>29536.81</td>
<td>c210 23</td>
</tr>
<tr>
<td>r101 80</td>
<td>1000000</td>
<td>38056.29</td>
<td>4884</td>
<td>79</td>
<td>39612.20</td>
<td>r201 15</td>
</tr>
<tr>
<td>r102 72</td>
<td>5000</td>
<td>35999.87</td>
<td>3367</td>
<td>72</td>
<td>33548.54</td>
<td>r202 15</td>
</tr>
<tr>
<td>r103 72</td>
<td>100</td>
<td>32529.49</td>
<td>2361</td>
<td>72</td>
<td>30151.90</td>
<td>r203 15</td>
</tr>
<tr>
<td>r104 72</td>
<td>10</td>
<td>30303.52</td>
<td>1990</td>
<td>72</td>
<td>26838.04</td>
<td>r204 15</td>
</tr>
<tr>
<td>r105 72</td>
<td>50</td>
<td>38055.58</td>
<td>2496</td>
<td>72</td>
<td>34741.53</td>
<td>r205 15</td>
</tr>
<tr>
<td>r106 72</td>
<td>10</td>
<td>34546.53</td>
<td>2321</td>
<td>72</td>
<td>31737.47</td>
<td>r206 15</td>
</tr>
<tr>
<td>r107 72</td>
<td>10</td>
<td>31537.02</td>
<td>2196</td>
<td>72</td>
<td>29538.40</td>
<td>r207 15</td>
</tr>
<tr>
<td>r108 72</td>
<td>5</td>
<td>29662.64</td>
<td>1892</td>
<td>72</td>
<td>28342.64</td>
<td>r208 15</td>
</tr>
<tr>
<td>r109 72</td>
<td>100</td>
<td>35986.98</td>
<td>2505</td>
<td>72</td>
<td>34231.38</td>
<td>r209 15</td>
</tr>
<tr>
<td>r110 72</td>
<td>10</td>
<td>34272.51</td>
<td>2299</td>
<td>72</td>
<td>31730.45</td>
<td>r210 15</td>
</tr>
<tr>
<td>rc101 73</td>
<td>100</td>
<td>33711.89</td>
<td>3387</td>
<td>73</td>
<td>31590.23</td>
<td>rc201 19</td>
</tr>
<tr>
<td>rc102 72</td>
<td>50</td>
<td>35112.82</td>
<td>3455</td>
<td>72</td>
<td>39696.20</td>
<td>rc202 16</td>
</tr>
<tr>
<td>rc103 72</td>
<td>100</td>
<td>33015.08</td>
<td>3261</td>
<td>72</td>
<td>35577.87</td>
<td>rc203 15</td>
</tr>
<tr>
<td>rc104 72</td>
<td>10</td>
<td>30085.34</td>
<td>2717</td>
<td>72</td>
<td>32654.10</td>
<td>rc204 15</td>
</tr>
<tr>
<td>rc105 73</td>
<td>100</td>
<td>32344.49</td>
<td>3481</td>
<td>73</td>
<td>30454.15</td>
<td>rc205 16</td>
</tr>
<tr>
<td>rc106 73</td>
<td>50</td>
<td>32782.06</td>
<td>3286</td>
<td>73</td>
<td>29674.68</td>
<td>rc206 15</td>
</tr>
<tr>
<td>rc107 72</td>
<td>500</td>
<td>39643.15</td>
<td>3244</td>
<td>72</td>
<td>43829.43</td>
<td>rc207 15</td>
</tr>
<tr>
<td>rc108 72</td>
<td>100</td>
<td>36512.04</td>
<td>3059</td>
<td>72</td>
<td>43694.60</td>
<td>rc208 15</td>
</tr>
<tr>
<td>rc109 72</td>
<td>50</td>
<td>35660.83</td>
<td>3038</td>
<td>72</td>
<td>41816.70</td>
<td>rc209 15</td>
</tr>
<tr>
<td>rc110 72</td>
<td>10</td>
<td>34716.75</td>
<td>2504</td>
<td>72</td>
<td>41182.44</td>
<td>rc210 15</td>
</tr>
</tbody>
</table>

4.6 Computational results
Table 4.13: The detailed results for 1000-customer benchmark instances

<table>
<thead>
<tr>
<th>Inst</th>
<th>m</th>
<th>α</th>
<th>$d_{sum}$</th>
<th>LS</th>
<th>$m$</th>
<th>$d_{sum}$</th>
<th>Inst</th>
<th>m</th>
<th>α</th>
<th>$d_{sum}$</th>
<th>LS</th>
<th>$m$</th>
<th>$d_{sum}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>c101</td>
<td>100</td>
<td>5</td>
<td>42478.95</td>
<td>5050</td>
<td>100</td>
<td>42478.95</td>
<td>c201</td>
<td>30</td>
<td>5</td>
<td>16879.24</td>
<td>2797</td>
<td>30</td>
<td>16879.24</td>
</tr>
<tr>
<td>c102</td>
<td>90</td>
<td>1000</td>
<td>45854.82</td>
<td>2666</td>
<td>92</td>
<td>42920.70</td>
<td>c202</td>
<td>29</td>
<td>50</td>
<td>17473.93</td>
<td>2307</td>
<td>29</td>
<td>17228.82</td>
</tr>
<tr>
<td>c103</td>
<td>90</td>
<td>5</td>
<td>42218.92</td>
<td>2401</td>
<td>90</td>
<td>40934.87</td>
<td>c203</td>
<td>29</td>
<td>50</td>
<td>17043.13</td>
<td>2626</td>
<td>29</td>
<td>16367.59</td>
</tr>
<tr>
<td>c104</td>
<td>90</td>
<td>1</td>
<td>41575.26</td>
<td>2112</td>
<td>90</td>
<td>40410.58</td>
<td>c204</td>
<td>29</td>
<td>5</td>
<td>16896.99</td>
<td>2624</td>
<td>29</td>
<td>17153.19</td>
</tr>
<tr>
<td>c105</td>
<td>100</td>
<td>1000</td>
<td>42469.18</td>
<td>4984</td>
<td>100</td>
<td>42469.20</td>
<td>c205</td>
<td>30</td>
<td>5</td>
<td>16568.73</td>
<td>369</td>
<td>30</td>
<td>16586.46</td>
</tr>
<tr>
<td>c106</td>
<td>100</td>
<td>1000</td>
<td>42471.28</td>
<td>5135</td>
<td>100</td>
<td>42471.30</td>
<td>c206</td>
<td>30</td>
<td>5</td>
<td>16348.20</td>
<td>4508</td>
<td>30</td>
<td>16371.65</td>
</tr>
<tr>
<td>c107</td>
<td>99</td>
<td>1000</td>
<td>42821.17</td>
<td>3984</td>
<td>99</td>
<td>42711.39</td>
<td>c207</td>
<td>30</td>
<td>10000</td>
<td>16827.81</td>
<td>4416</td>
<td>31</td>
<td>16578.42</td>
</tr>
<tr>
<td>c108</td>
<td>94</td>
<td>10000</td>
<td>43555.1</td>
<td>4238</td>
<td>96</td>
<td>42170.31</td>
<td>c208</td>
<td>29</td>
<td>5</td>
<td>16532.88</td>
<td>3500</td>
<td>29</td>
<td>18662.10</td>
</tr>
<tr>
<td>c109</td>
<td>91</td>
<td>1000</td>
<td>42755.59</td>
<td>3254</td>
<td>91</td>
<td>45386.93</td>
<td>c209</td>
<td>29</td>
<td>10000</td>
<td>17462.68</td>
<td>4022</td>
<td>30</td>
<td>16651.96</td>
</tr>
<tr>
<td>c110</td>
<td>90</td>
<td>1000</td>
<td>44468.65</td>
<td>3344</td>
<td>90</td>
<td>40894.38</td>
<td>c210</td>
<td>29</td>
<td>5</td>
<td>16194.56</td>
<td>3545</td>
<td>29</td>
<td>16178.26</td>
</tr>
<tr>
<td>r101</td>
<td>100</td>
<td>1000</td>
<td>55922.77</td>
<td>3756</td>
<td>100</td>
<td>54145.31</td>
<td>r201</td>
<td>19</td>
<td>500</td>
<td>43554.40</td>
<td>1779</td>
<td>19</td>
<td>42922.56</td>
</tr>
<tr>
<td>r102</td>
<td>91</td>
<td>1000</td>
<td>56975.89</td>
<td>2929</td>
<td>91</td>
<td>56367.45</td>
<td>r202</td>
<td>19</td>
<td>50</td>
<td>35416.79</td>
<td>1562</td>
<td>19</td>
<td>34918.49</td>
</tr>
<tr>
<td>r103</td>
<td>91</td>
<td>100</td>
<td>51259.61</td>
<td>2645</td>
<td>91</td>
<td>46621.19</td>
<td>r203</td>
<td>19</td>
<td>50</td>
<td>26396.47</td>
<td>1787</td>
<td>19</td>
<td>25689.62</td>
</tr>
<tr>
<td>r104</td>
<td>91</td>
<td>50</td>
<td>47116.49</td>
<td>2306</td>
<td>91</td>
<td>43461.84</td>
<td>r204</td>
<td>19</td>
<td>50</td>
<td>19026.43</td>
<td>2214</td>
<td>19</td>
<td>18858.24</td>
</tr>
<tr>
<td>r105</td>
<td>91</td>
<td>50</td>
<td>61437.30</td>
<td>2672</td>
<td>91</td>
<td>70838.01</td>
<td>r205</td>
<td>19</td>
<td>100</td>
<td>38162.84</td>
<td>2040</td>
<td>19</td>
<td>37265.32</td>
</tr>
<tr>
<td>r106</td>
<td>91</td>
<td>50</td>
<td>55707.67</td>
<td>2783</td>
<td>91</td>
<td>49059.80</td>
<td>r206</td>
<td>19</td>
<td>100</td>
<td>31990.34</td>
<td>1745</td>
<td>19</td>
<td>30725.20</td>
</tr>
<tr>
<td>r107</td>
<td>91</td>
<td>100</td>
<td>50834.66</td>
<td>2640</td>
<td>91</td>
<td>45847.84</td>
<td>r207</td>
<td>19</td>
<td>50</td>
<td>24603.46</td>
<td>1942</td>
<td>19</td>
<td>24363.83</td>
</tr>
<tr>
<td>r108</td>
<td>91</td>
<td>50</td>
<td>46612.45</td>
<td>2290</td>
<td>91</td>
<td>42767.77</td>
<td>r208</td>
<td>19</td>
<td>5</td>
<td>18950.37</td>
<td>2220</td>
<td>19</td>
<td>18185.38</td>
</tr>
<tr>
<td>r109</td>
<td>91</td>
<td>500</td>
<td>59344.93</td>
<td>2652</td>
<td>91</td>
<td>51391.80</td>
<td>r209</td>
<td>19</td>
<td>10</td>
<td>35737.18</td>
<td>1939</td>
<td>19</td>
<td>33777.76</td>
</tr>
<tr>
<td>r110</td>
<td>91</td>
<td>50</td>
<td>56283.20</td>
<td>2731</td>
<td>91</td>
<td>49348.36</td>
<td>r210</td>
<td>19</td>
<td>5</td>
<td>32422.07</td>
<td>2046</td>
<td>19</td>
<td>31599.84</td>
</tr>
<tr>
<td>rc101</td>
<td>90</td>
<td>1000</td>
<td>52084.03</td>
<td>2572</td>
<td>90</td>
<td>47143.90</td>
<td>rc201</td>
<td>21</td>
<td>100</td>
<td>30585.71</td>
<td>2440</td>
<td>22</td>
<td>30320.41</td>
</tr>
<tr>
<td>rc102</td>
<td>90</td>
<td>500</td>
<td>49503.47</td>
<td>2435</td>
<td>90</td>
<td>44906.58</td>
<td>rc202</td>
<td>18</td>
<td>1000000</td>
<td>29525.90</td>
<td>746</td>
<td>19</td>
<td>26592.40</td>
</tr>
<tr>
<td>rc103</td>
<td>90</td>
<td>1000</td>
<td>46038.46</td>
<td>2484</td>
<td>90</td>
<td>43782.57</td>
<td>rc203</td>
<td>18</td>
<td>500</td>
<td>22185.99</td>
<td>1217</td>
<td>18</td>
<td>20588.38</td>
</tr>
<tr>
<td>rc104</td>
<td>90</td>
<td>5</td>
<td>43998.71</td>
<td>1883</td>
<td>90</td>
<td>41917.14</td>
<td>rc204</td>
<td>18</td>
<td>10</td>
<td>17645.94</td>
<td>1478</td>
<td>18</td>
<td>16480.17</td>
</tr>
<tr>
<td>rc105</td>
<td>90</td>
<td>50</td>
<td>51822.47</td>
<td>2569</td>
<td>90</td>
<td>47632.31</td>
<td>rc205</td>
<td>18</td>
<td>100</td>
<td>30231.94</td>
<td>1116</td>
<td>18</td>
<td>29383.27</td>
</tr>
<tr>
<td>rc106</td>
<td>90</td>
<td>500</td>
<td>51377.57</td>
<td>2518</td>
<td>90</td>
<td>46391.60</td>
<td>rc206</td>
<td>18</td>
<td>50</td>
<td>29668.22</td>
<td>1225</td>
<td>18</td>
<td>27003.30</td>
</tr>
<tr>
<td>rc107</td>
<td>90</td>
<td>500</td>
<td>50657.17</td>
<td>2489</td>
<td>90</td>
<td>46391.60</td>
<td>rc207</td>
<td>18</td>
<td>50</td>
<td>27928.74</td>
<td>1341</td>
<td>18</td>
<td>26161.91</td>
</tr>
<tr>
<td>rc108</td>
<td>90</td>
<td>1000</td>
<td>50099.58</td>
<td>2175</td>
<td>90</td>
<td>45585.08</td>
<td>rc208</td>
<td>18</td>
<td>5</td>
<td>26631.77</td>
<td>1113</td>
<td>18</td>
<td>24995.00</td>
</tr>
<tr>
<td>rc109</td>
<td>90</td>
<td>5</td>
<td>50320.45</td>
<td>2004</td>
<td>90</td>
<td>45405.54</td>
<td>rc209</td>
<td>18</td>
<td>50</td>
<td>25381.34</td>
<td>1716</td>
<td>18</td>
<td>23582.89</td>
</tr>
<tr>
<td>rc110</td>
<td>90</td>
<td>1000</td>
<td>47885.23</td>
<td>2208</td>
<td>90</td>
<td>45041.64</td>
<td>rc210</td>
<td>18</td>
<td>5</td>
<td>24502.58</td>
<td>1319</td>
<td>18</td>
<td>22481.03</td>
</tr>
</tbody>
</table>
function as
\[
p_i(t) = \begin{cases} 
  r_i - t, & t < r_i \\
  0, & r_i \leq t \leq d_i \\
  t - d_i, & d_i < t,
\end{cases}
\]

and we construct each traveling time function \( \lambda_{ij}(t) \) from the distance and travel speed in Table 4.14. We then define \( q_{ij}(t) = \lambda_{ij}(t) \).

**Table 4.14: Travel speed matrices for scenarios 1–3**

<table>
<thead>
<tr>
<th>scenario1</th>
<th>scenario2</th>
<th>scenario3</th>
</tr>
</thead>
<tbody>
<tr>
<td>morning</td>
<td>daytime</td>
<td>evening</td>
</tr>
<tr>
<td>category 1</td>
<td>0.54</td>
<td>0.81</td>
</tr>
<tr>
<td>category 2</td>
<td>0.81</td>
<td>1.22</td>
</tr>
<tr>
<td>category 3</td>
<td>1.22</td>
<td>1.82</td>
</tr>
</tbody>
</table>

Table 4.15 shows the computational results of our algorithm ILS for these instances. Column “time-dependent” represents the results of our algorithm devised for the time-dependent instances. Column “const” represents the results obtained by the following method, which was tested for comparison purposes: We solved the instances with our algorithm after replacing the time-dependent traveling time with the fixed constant determined by taking the average of the traveling time in the whole periods. Note that, in the case of “const”, even though the constant traveling times were used during the search, the final costs output by this method were evaluated exactly under the time-dependent environment, and the table shows the results under the exact evaluation. Each row gives the instance type, and the average values of \( p_{\text{sum}} \) and \( q_{\text{sum}} \) with respect to all instances of the same type. We omitted \( a_{\text{sum}} \), since \( a_{\text{sum}} \) were always 0.

In Table 4.15, we can observe that both \( p_{\text{sum}} \) and \( q_{\text{sum}} \) in column “time-dependent” are smaller than those in column “const”. The deference becomes larger as the instances become more time-dependent (i.e., from scenarios 1 to 3). This indicates the usefulness of our algorithm that can accept time-dependency.

**4.7 Conclusion**

We generalized the standard vehicle routing problem with time windows by allowing both traveling times and traveling costs to be time-dependent functions, and proposed an iterated local search algorithm. Our generalization can treat time-dependent traveling times
Table 4.15: The results for time-dependent VRPSTW

<table>
<thead>
<tr>
<th></th>
<th>scenario1</th>
<th></th>
<th>scenario2</th>
<th></th>
<th>scenario3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time-dependent</td>
<td></td>
<td>const</td>
<td></td>
<td>time-dependent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$p_{\text{sum}}$</td>
<td></td>
<td>$q_{\text{sum}}$</td>
<td></td>
<td>$p_{\text{sum}}$</td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td>20.35</td>
<td></td>
<td>855.27</td>
<td></td>
<td>35.47</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>2.31</td>
<td></td>
<td>669.94</td>
<td></td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>32.47</td>
<td></td>
<td>1061.93</td>
<td></td>
<td>50.60</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>5.04</td>
<td></td>
<td>904.40</td>
<td></td>
<td>19.62</td>
<td></td>
</tr>
<tr>
<td>RC1</td>
<td>50.10</td>
<td></td>
<td>1103.62</td>
<td></td>
<td>59.50</td>
<td></td>
</tr>
<tr>
<td>RC2</td>
<td>19.69</td>
<td></td>
<td>976.53</td>
<td></td>
<td>25.09</td>
<td></td>
</tr>
</tbody>
</table>

and costs such as rush-hour traffic jam, and includes various interesting problems such as parallel machine scheduling problems as its special cases.

In our local search procedure, for each vehicle route generated during the search, we must compute an optimal schedule for the route. We showed that this subproblem can be efficiently solved by dynamic programming. We further proposed a filtering method that restricts the size of neighborhoods, based on the fact that there are many solutions having no prospect of improvement. We developed an iterated local search algorithm incorporating all the above ingredients. The computational results on representative benchmark instances indicate that the proposed algorithm is highly efficient. Artificially generated instances of the time-dependent vehicle routing problem with time windows were also solved to show the usefulness of our algorithm having high generality.
Chapter 5

Path Relinking Approach with an Adaptive Mechanism to Control Parameters for the Vehicle Routing Problem with Time Windows

5.1 Introduction

We propose a path relinking approach for the VRPTW. The path relinking [67, 68] is an evolutionary mechanism that generates new solutions by combining two or more reference solutions. Our algorithm invokes a path relinking operation for generating new candidate solutions, which are then improved by a local search whose neighborhood consists of slight modifications of the representative neighborhoods called 2-opt*, cross exchange and Or-opt. To reduce the computation time for searching these neighborhoods, we propose a neighbor list that prunes the neighborhood search heuristically. In our algorithm, infeasible solutions are allowed to be visited during the search, while the amount of violation is penalized. The amount of violation for the capacity constraint is estimated by the amount of capacity excess. To estimate the amount of violation of time window constraints of each route, we consider the total amount of traveling time to be shortened to satisfy the constraints. We also incorporate in our algorithm a frequency-based penalty, in which a customer who often appears in an infeasible route of locally optimal solutions is penalized to direct the search to make those routes with many heavily penalized customers feasi-
Path Relinking Approach for the VRPTW

As the evaluation of these penalties takes time if naively implemented, we propose an efficient algorithm, which enables us to evaluate each neighborhood solution in $O(1)$ time. We also propose an adaptive mechanism to control the weights of these penalties. Finally we report computational results on well-studied benchmark instances with up to 1000 customers. The results show the high competence of our algorithm against existing methods; it updates 41 best known results among 356 instances within a reasonable amount of computation time.

The chapter is organized as follows. In Section 5.2, we give the formulation of the vehicle routing problem with time windows. In Section 5.3, our local search, the neighbor list, and the neighborhoods are discussed. Section 5.4 describes the criterion we adopted to evaluate vehicle routes, and an efficient algorithm to evaluate solutions in the neighborhood. In Section 5.5, we will discuss an adaptive mechanism to control the penalty weights. Then, in Section 5.6, our path relinking approach is explained. Finally, in Section 5.7, we report the computational results of our algorithm and compare them against existing methods.

5.2 Problem definition

Here we formulate the vehicle routing problem with time windows. Let $G = (V, E)$ be a complete directed graph with vertex set $V = \{0, 1, \ldots, n\}$ and edge set $E = \{(i, j) \mid i, j \in V, i \neq j\}$, and $M = \{1, 2, \ldots, m\}$ be a vehicle set. In this graph, vertex 0 is the depot and other vertices are customers. Each customer $i$ and each edge $(i, j) \in E$ are associated with:

i. a fixed quantity $a_i (\geq 0)$ of goods to be delivered to $i$,

ii. a time window $[e_i, l_i]$,

iii. a traveling time $t_{ij}(\geq 0)$ and a traveling distance $c_{ij}(\geq 0)$ from $i$ to $j$.

We assume $a_0 = 0$ and $e_0 = 0$ without loss of generality. Each vehicle has an identical capacity $u$.

Let $\sigma_k$ denote the route traveled by vehicle $k$, where $\sigma_k(h)$ denotes the $h$th customer in $\sigma_k$, and let

$$\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m).$$

Note that each customer $i$ is included in exactly one route $\sigma_k$, and is visited by vehicle $k$ exactly once. We denote by $n_k$ the number of customers in $\sigma_k$. For convenience, we define $\sigma_k(0) = 0$ and $\sigma_k(n_k + 1) = 0$ for all $k$ (i.e., each vehicle $k \in M$ departs from the depot and comes back to the depot). Moreover, let $s_i$ be the start time of service at customer $i$.
by exactly one of the vehicles) and \( s_k^n \) be the arrival time of vehicle \( k \) at the depot. Note that each vehicle is allowed to wait at customers before starting services.

Let us introduce 0-1 variables \( y_{ik}(\sigma) \in \{0, 1\} \) for \( i \in V \setminus \{0\} \) and \( k \in M \) by

\[
y_{ik}(\sigma) = 1 \iff i = \sigma_k(h) \text{ holds for exactly one } h \in \{1, 2, \ldots, n_k\}.
\]

That is, \( y_{ik}(\sigma) = 1 \) holds if and only if vehicle \( k \) visits customer \( i \). The traveling distance of a vehicle \( k \) is expressed as

\[
d(\sigma_k) = \sum_{h=0}^{n_k} c_{\sigma_k(h), \sigma_k(h+1)}.
\]

Then the problem we consider in this chapter is formulated as follows:

\[
\text{minimize} \quad \sum_{k \in M} d(\sigma_k) \quad (5.2.1)
\]

subject to

\[
\sum_{k \in M} y_{ik}(\sigma) = 1, \quad i \in V \setminus \{0\} \quad (5.2.2)
\]

\[
\sum_{i \in V \setminus \{0\}} a_i y_{ik}(\sigma) \leq u, \quad k \in M \quad (5.2.3)
\]

\[
t_{0,\sigma_k(1)} \leq s_{\sigma_k(1)}, \quad k \in M \quad (5.2.4)
\]

\[
s_{\sigma_k(i)} + t_{\sigma_k(i), \sigma_k(i+1)} \leq s_{\sigma_k(i+1)}, \quad 1 \leq i \leq n_k-1, \quad k \in M \quad (5.2.5)
\]

\[
s_{\sigma_k(n_k)} + t_{\sigma_k(n_k), 0} \leq s_k^a \leq l_0, \quad k \in M \quad (5.2.6)
\]

\[
e_i \leq s_i \leq l_i, \quad i \in V \setminus \{0\} \quad (5.2.7)
\]

\[
y_{ik}(\sigma) \in \{0, 1\}, \quad i \in V \setminus \{0\}, \quad k \in M. \quad (5.2.8)
\]

Constraint (5.2.2) means that every customer \( i \in V \setminus \{0\} \) must be served exactly once by a vehicle. Constraint (5.2.3) means a capacity constraint for vehicle \( k \). Constraints (5.2.4)–(5.2.6) require that each vehicle cannot serve a customer before arriving at the customer. Constraint (5.2.7) is a time window constraint for each customer. Note that essential decision variables in this formulation are routes \( \sigma_k \), since the values of \( y_{ik}(\sigma) \) are automatically determined from \( \sigma \), and finding appropriate values for \( s_i \) and \( s_k^a \), if any, is easy when \( \sigma \) is fixed.

### 5.3 Local search

In this section, we describe our local search (LS). Our LS searches a visiting order \( \sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m) \), which can be infeasible with respect to the capacity and time window constraints. The algorithm evaluates each route \( \sigma_k \) by a function \( p(\sigma_k) \), which is the sum of its traveling distance \( d(\sigma_k) \) and the penalty for violation of constraints if \( \sigma_k \) is infeasible, and it evaluates a solution \( \sigma \) by \( \sum_{k \in M} p(\sigma_k) \). The details of function \( p(\sigma_k) \) will be discussed in Section 5.4. Our LS starts from an initial solution \( \sigma \) and repeats replacing
Path Relinking Approach for the VRPTW

σ with a better solution (with respect to \( \sum_{k \in M} p(\sigma_k) \)) in its neighborhood \( N(\sigma) \) until no better solution is found in \( N(\sigma) \). To define the neighborhood \( N(\sigma) \), we use the 2-opt*, cross exchange and Or-opt neighborhoods with slight modifications. For the 2-opt* and cross exchange neighborhoods, we propose a neighbor list to prune the neighborhood search heuristically. A similar technique was successfully applied to the traveling salesman and vehicle routing problems [88, 122], in which the list is determined only on the basis of distance; therefore it is not appropriate to apply the existing method directly to the VRPTW. In Section 5.3.1, we describe the neighbor lists that take into account the time windows, and in Section 5.3.2, the details of the neighborhoods are described.

5.3.1 Neighbor list

We consider a neighbor list for each customer \( i \), which is a set of customers preferable to visit immediately after \( i \). Each customer \( j \) that can be visited after \( i \) (i.e., \( e_i + t_{ij} \leq l_j \)) is evaluated by \( \max\{t_{ij}, e_j - l_i\} \). When a vehicle visits \( j \) immediately after \( i \), it takes at least \( \max\{t_{ij}, e_j - l_i\} \) time between the start times of \( i \) and \( j \). Hence, if this value is small, it is preferable to visit \( j \) immediately after \( i \). The algorithm computes these values once at the beginning and stores the best \( N_{\text{nilist}} \) (a parameter) customers as a neighbor list of \( i \). We set \( N_{\text{nilist}} = 20 \) in the experiments.

5.3.2 Neighborhoods

We use the 2-opt*, cross exchange and Or-opt neighborhoods with slight modifications, wherein we restrict the 2-opt* and cross exchange neighborhoods by using the neighbor lists.

A 2-opt* operation removes two edges from two different routes (one from each) to divide each route into two parts and exchanges the second parts of the two routes (See Section 2.4.2). Our algorithm searches only those solutions obtainable by a 2-opt* operation in which at least one of the newly added edges is in the neighbor list. The size of this neighborhood is \( O(N_{\text{nilist}} n) \).

A cross exchange operation removes two paths from two routes (one from each) of different vehicles, whose length (i.e., the number of customers in the path) is at most \( L^{\text{cross}} \) (a parameter), and exchanges them (See Section 2.4.4). Our algorithm searches only those solutions obtainable by a cross exchange operation in which a newly added edge linking the former part of a route and the path from another route is in the neighbor list. The size of this neighborhood is \( O((L^{\text{cross}})^2 N_{\text{nilist}} n) \). We set \( L^{\text{cross}} = 3 \) in the experiments.

The cross exchange and 2-opt* operations always change the assignment of customers to vehicles. We also use an intra-route neighborhood to improve individual routes. An
intra-route operation removes a path of length at most $L_{\text{path}}^{\text{intra}}$ (a parameter) and inserts it into another position of the same route, where the position is limited within length $L_{\text{ins}}^{\text{intra}}$ (a parameter) from the original position (See Section 2.4.5). The size of the intra-route neighborhood is $O(L_{\text{path}}^{\text{intra}}L_{\text{ins}}^{\text{intra}}n)$. We set $L_{\text{path}}^{\text{intra}} = 3$ and $L_{\text{ins}}^{\text{intra}} = 10$ in the experiments.

Figure 5.1: Neighborhoods in our local search

Figure 5.1 is an illustration of the neighborhoods. In Figure 5.1, squares represent the depot (which is duplicated at each end) and small circles represent customers in the routes. A thin line represents a route edge and a thick line represents a path (i.e., more than two customers may be included). The dotted boxes mean that edges in them are in the neighbor lists.

Our LS searches the above intra-route, 2-opt* and cross exchange neighborhoods, in this order. Whenever a better solution is found, the LS immediately accepts it (i.e., we adopt the first admissible move strategy) and resumes the search from the intra-route neighborhood.

5.4 Evaluation function $p(\sigma_k)$

We first define the function $p(\cdot)$ to evaluate a route $\sigma_k$. For convenience, throughout this section, we assume that vehicle $k$ visits customers $1, 2, \ldots, n_k$ in this order and let customer $n_k + 1$ represent the arrival at the depot (i.e., $s_{n_k+1} = s_k^n$). The function we adopt is

\[
p(\sigma_k) = \begin{cases} 
d(\sigma_k), & \text{if } \sigma_k \text{ is feasible} \\
d(\sigma_k) + \alpha p_c(\sigma_k) + \beta p_t(\sigma_k) + \sum_{h=1}^{n_k} \gamma_h, & \text{otherwise}, \end{cases}
\] (5.4.9)
where \( p_c(\sigma_k) \) is the amount of capacity excess (i.e., \( p_c(\sigma_k) = \max\{0, \sum_{i=1}^{n_k} a_i - u\} \)) and \( p_t(\sigma_k) \) is the minimum total amount of traveling times to be shortened to satisfy the constraints; i.e.,

\[
p_t(\sigma_k) = \min \left\{ \sum_{h=1}^{n_k+1} \tau_h \left| \begin{array}{l}
s_0 \geq 0, \quad s_{h-1} + t_{h-1,h} - \tau_h \leq s_h, \\
\tau_h \geq 0, \quad e_h \leq s_h \leq l_h, \quad h = 1, \ldots, n_k + 1 \end{array} \right. \right\}.
\]

In function \( p, \alpha, \beta \) and \( \gamma_i \) for each \( i \in V \) are parameters, which are controlled adaptively (see Section 5.5). In this estimation, each traveling time can be shortened by an arbitrary amount (i.e., the resulting traveling time \( t_{h-1,h} - \tau_h \) can be negative) to satisfy time window constraints while the shortened amount is penalized as \( p_t(\sigma_k) \). This idea of defining \( p_t \) was proposed by Nagata [112]. The algorithm computes \( p(\sigma_k) \) by each term separately.

In the rest of this section, we focus on the computation of \( p_t(\sigma_k) \), since the other terms can be efficiently computed by using standard data structures (e.g., [74, 75, 85, 86]).

A key observation to the efficient computation is that each route \( \sigma_k \) of a neighborhood solution is a recombination of a few paths of the current solution. Hence we consider a speeding up approach that stores some useful information of paths from the depot to customers and those from customers to the depot, among those paths of the current routes. For each customer \( h \) in a new route \( \sigma_k \), let \( F_h \) (resp., \( B_h \)) be some data structure that contains the information of the paths (of \( \sigma_k \)) from the depot to \( h \) (resp., from \( h \) to the depot). Note that \( F_h \) and \( B_h \) signify the information of the paths of the new route \( \sigma_k \).

For example, if \( \sigma_k \) is generated by a 2-opt* operation, and the path from the depot to \( h \) and the path from \( h + 1 \) to the depot are from the current solution, then \( F_h \) and \( B_{h+1} \) are available from the stored information when they are used to compute \( p(\sigma_k) \). On the other hand, for the cross exchange and intra-route neighborhoods, \( F_h \) and \( B_h \) for customers \( h \) in inserted paths need to be computed, because in the new route \( \sigma_k \) the path from the depot to such an \( h \) and that from \( h \) to the depot are different from those in the current route.

What is important in this approach is to execute the followings efficiently for a given \( \sigma_k \):

1. construction of \( F_{h+1} \) from \( F_h \) (the forward computation),
2. construction of \( B_h \) from \( B_{h+1} \) (the backward computation), and
3. computation of \( p_t(\sigma_k) \) from \( F_h \) and \( B_{h+1} \).

It is not hard to show that each neighborhood solution can be evaluated in \( O(T) \) time, if the above operations can be done in \( O(T) \) time for any \( h \) (0 \( \leq h \leq n_k \)). However, to accomplish this, the neighborhood need to be searched in an appropriate search order. The detailed description of such a search order is explained in Section 2.6. Below we show that the forward and backward computation can be done in \( O(1) \) time and the computation of
5.5 Adaptive mechanism to control parameters

In this section, we describe an adaptive mechanism to control the parameters $\alpha$, $\beta$ and $\gamma_i$ for each customer $i$. The algorithm (in which the local search (LS) is executed many times) updates these parameters whenever the LS outputs a locally optimal solution. We set their initial values to $\alpha = 1000$, $\beta = 1000$ and $\gamma_i = 100$ in the experiments.

5.5.1 Update of the parameters $\alpha$ and $\beta$

Let $p^{\text{sum}}_c(\sigma) = \sum_{k \in M} p_c(\sigma_k)$ and $p^{\text{sum}}_t(\sigma) = \sum_{k \in M} p_t(\sigma_k)$, and let $p^{\text{min}}_c$ (resp., $p^{\text{min}}_t$) be the minimum $p^{\text{sum}}_c(\sigma)$ (resp., $p^{\text{sum}}_t(\sigma)$) of the solutions in the current reference set $R$ of good solutions, where rules for maintaining $R$ are described in Section 5.6. Let $R_c$ (resp., $R_t$) be the number of moves, during the last call to the LS, to a solution $\sigma$ whose $p^{\text{sum}}_c(\sigma)$ (resp.,
Path Relinking Approach for the VRPTW

$p^\text{sum}(\sigma)$ is less than $p^\text{min}_c$ (resp., $p^\text{min}_t$) or equals to 0. Let $N_\text{total}$ be the total number of moves during the last call to LS, and let $N_c = N_\text{total} - P_c$ and $N_t = N_\text{total} - P_t$. We use parameters $\delta_\text{inc}$, $\delta_\text{dec}$, $\delta_\text{inc}^\text{cust}$ and $\delta_\text{dec}^\text{cust}$, and in the experiments, we set $\delta_\text{inc} = 0.05$, $\delta_\text{dec} = 0.1$, $\delta_\text{inc}^\text{cust} = 0.1$ and $\delta_\text{dec}^\text{cust} = 0.01$. If the LS found, during last call, a solution $\sigma$ that satisfied $p^\text{sum}_c(\sigma) < p^\text{min}_c$ and $p^\text{sum}_t(\sigma) < p^\text{min}_t$, the parameters $\alpha$ and $\beta$ are decreased by

$$\alpha := \left(1 - \frac{P_c}{\max\{P_c, P_t\}} \delta_\text{dec}\right) \alpha, \quad \beta := \left(1 - \frac{P_t}{\max\{P_c, P_t\}} \delta_\text{dec}\right) \beta.$$ 

Even if the LS did not find such a solution, if $N_c = 0$ (resp., $N_t = 0$) holds, $\alpha$ (resp., $\beta$) is decreased by the same equation. Otherwise they are increased by

$$\alpha := \left(1 + \frac{N_c}{\max\{N_c, N_t\}} \delta_\text{inc}\right) \alpha, \quad \beta := \left(1 + \frac{N_t}{\max\{N_c, N_t\}} \delta_\text{inc}\right) \beta.$$ 

### 5.5.2 Update of the parameters $\gamma_i$ for each customer $i$

In the locally optimal solution, if a route violates the capacity or time window constraint, $\gamma_i$ of each customer $i$ in the route is increased by $\gamma_i := (1 + \delta_\text{inc}^\text{cust}) \gamma_i$. For each customer $i$ who is in a feasible route, $\gamma_i$ is decreased by $\gamma_i := (1 - \delta_\text{dec}^\text{cust}) \gamma_i$.

### 5.6 Path relinking approach

#### 5.6.1 Reference set

Let $R$ be a reference set of solutions. Initially $R$ is prepared by applying the LS to randomly generated solutions. Then it is updated by reflecting outcome of the LS. During the search, the algorithm always keeps the size of $R$ to $\rho$ (a parameter). We set $\rho = 10$ in the experiments. Good solutions with respect to $p$ are kept in $R$, excluding at most two solutions: One which achieves $p^\text{min}_c$ and the other which achieves $p^\text{min}_t$. After a feasible solution is found (i.e., $p^\text{min}_c = 0$ and $p^\text{min}_t = 0$), the best feasible solution is always stored as a member of $R$. Other solutions in $R$ are maintained as follows. Whenever the LS stops, the locally optimal solution $\sigma^\text{lopt}$ is exchanged with the worst (with respect to $p$) solution $\sigma^\text{worst}$ in $R$ (excluding the above solutions), provided that $\sigma^\text{lopt}$ is not worse than $\sigma^\text{worst}$ and is different from all solution in $R$.

#### 5.6.2 Path relinking operation

A path relinking operation is applied to two solutions $\sigma_A$ (initiating solution) and $\sigma_B$ (guiding solution) randomly chosen from $R$, where a random perturbation is applied to $\sigma_B$ with probability 1/2 before applying the path relinking (for the purpose of keeping the diversity of the search), and the resulting solution is redefined to be $\sigma_B$. We use a cyclic
5.7 Computational experiments

We conducted computational experiments to evaluate the proposed algorithm. The algorithm was coded in C and run on a PC (Xeon, 2.8 GHz, 1 GB memory).

We used Solomon’s benchmark instances [140] and Gehring and Homberger’s benchmark instances [80]. There are 356 instances in total, and all of them have been widely used in the literature. In Solomon’s instances, the number of customers is 100, and in Gehring and Homberger’s instances, which are the extended instances from Solomon’s instances, the number of customers is from 200 to 1000. The customers are distributed in the plane and the distances between customers are measured by Euclidean distances. For these instances, the number of vehicles \( m \) is also a decision variable, and the objective is to find a solution with the minimum vehicle number and the total traveling distance in the lexicographical order (i.e., a solution is better than another (1) if its vehicle number is smaller or (2) if the vehicle numbers are the same but the distance is smaller).

As our algorithm deals with the problem with a fixed number of vehicles, we first set the number of vehicles in each instance to the known smallest number to the best of our
Algorithm 16 Path relinking approach for the vehicle routing problem with time windows

1: Construct the neighbor lists.
2: Let $R$ be $\rho$ randomly generated solutions. For each $\sigma \in R$, let $\sigma_{\text{lopt}} := \text{LS}(\sigma)$ and then let $R := (R \setminus \{\sigma\}) \cup \sigma_{\text{lopt}}$.
3: Let $S := \emptyset$.
4: while the stopping criterion is not satisfied do
5: \hspace{0.5cm} while $S = \emptyset$ do
6: \hspace{1cm} Randomly choose two solutions $\sigma_A$ and $\sigma_B$ from $R$ ($\sigma_A \neq \sigma_B$).
7: \hspace{1cm} With probability $1/2$, apply a cyclic operation to $\sigma_B$.
8: \hspace{1cm} Apply the path relinking operation to $\sigma_A$ and $\sigma_B$, and then let $S$ be the set of best $\pi$ locally minimal solutions in the generated sequence.
9: \hspace{0.5cm} end while
10: Randomly choose $\sigma \in S$, and let $S := S \setminus \{\sigma\}$ and $\sigma_{\text{lopt}} := \text{LS}(\sigma)$.
11: Update the penalty weights.
12: Choose the worst $\sigma_{\text{worst}} \in R$ among those that satisfy (1) $\sigma_{\text{worst}}$ is not the unique feasible solution in $R$, (2) $\exists \sigma_c \in R \setminus \{\sigma_{\text{worst}}\}$, $p^\text{sum}_c(\sigma_c) \leq p^\text{sum}_c(\sigma_{\text{worst}})$ and (3) $\exists \sigma_t \in R \setminus \{\sigma_{\text{worst}}\}$, $p^\text{sum}_t(\sigma_t) \leq p^\text{sum}_t(\sigma_{\text{worst}})$.
13: if $p(\sigma_{\text{lopt}}) \leq p(\sigma_{\text{worst}})$ and $\sigma_{\text{lopt}}$ is different from all solutions in $R$ then
14: \hspace{0.5cm} $R := (R \setminus \{\sigma_{\text{worst}}\}) \cup \sigma_{\text{lopt}}$
15: end if
16: end while
17: Output the incumbent solution and stop.

knowledge, and repeat the followings. If the algorithm found a feasible solution and the number of vehicles is larger than a lower bound $\lceil \sum_{i \in V} a_i / u \rceil$, we run the algorithm again after decrementing the number of vehicles by one. On the other hand, if the algorithm was not able to find a feasible solution, we run the algorithm again after incrementing the number of vehicles by one. Among the 356 instances, the algorithm found a feasible solution in the first run for every instance except for six instances. Among the remaining six instances, it was able to find feasible solutions with one more vehicle for five instances and with two more vehicles for the one. The time limit for each run of the algorithm for 100, 200, 400, 600, 800 and 1000-customer instances are 1000, 2000, 4000, 6000, 8000 and 10000 seconds, respectively. This setting of the time limit is the same with [75, 86].

Table 5.1 shows the comparison of our results with those obtained by existing methods. A number in the first row shows the number of customers. Our results are denoted by “Ours.” For each method, we provide the cumulative number of vehicles (CNV), the cumulative total distance (CTD), the CPU, and the average computation time in minutes
Table 5.1: Comparison of our results with the existing methods for benchmark instances

<table>
<thead>
<tr>
<th>References</th>
<th>CNV</th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>600</th>
<th>800</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hashimoto et al. (in press) [75]</td>
<td>CTD 57,282</td>
<td>17</td>
<td>33</td>
<td>67</td>
<td>100</td>
<td>133</td>
<td>167</td>
</tr>
<tr>
<td>(in press) [86]</td>
<td>P4 2.8GHz</td>
<td>57,454</td>
<td>17</td>
<td>33</td>
<td>67</td>
<td>100</td>
<td>133</td>
</tr>
<tr>
<td>Ibaraki et al.</td>
<td>CTD 57,454</td>
<td>17</td>
<td>33</td>
<td>67</td>
<td>100</td>
<td>133</td>
<td>167</td>
</tr>
<tr>
<td>(in press) [86]</td>
<td>P4 2.8GHz</td>
<td>–</td>
<td>–</td>
<td>695</td>
<td>1391</td>
<td>2084</td>
<td>2776</td>
</tr>
<tr>
<td>Prescott-Gagnon et al. (2007) [127]</td>
<td>CTD 57,240</td>
<td>5×30</td>
<td>5×53</td>
<td>5×89</td>
<td>5×105</td>
<td>5×129</td>
<td>5×162</td>
</tr>
<tr>
<td>Pisinger and Ropke (2007) [124]</td>
<td>CTD 57,322</td>
<td>10×2</td>
<td>10×8</td>
<td>5×16</td>
<td>5×18</td>
<td>5×23</td>
<td>5×27</td>
</tr>
<tr>
<td>Mester and Bräysy (2005) [110]</td>
<td>CTD 168,573</td>
<td>–</td>
<td>8</td>
<td>17</td>
<td>40</td>
<td>145</td>
<td>600</td>
</tr>
<tr>
<td>Le Bouthillier et al. (2005) [103]</td>
<td>CTD 57,360</td>
<td>5×8</td>
<td>5×16</td>
<td>5×18</td>
<td>5×23</td>
<td>5×27</td>
<td></td>
</tr>
<tr>
<td>P 2GHz</td>
<td>–</td>
<td>8</td>
<td>17</td>
<td>40</td>
<td>145</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>Le Bouthillier and Crainic (2005) [102]</td>
<td>CTD 57,412</td>
<td>5×3</td>
<td>3×2</td>
<td>3×7</td>
<td>3×13</td>
<td>3×23</td>
<td>3×30</td>
</tr>
<tr>
<td>Opt 2.3GHz</td>
<td>–</td>
<td>5×8</td>
<td>5×16</td>
<td>5×18</td>
<td>5×23</td>
<td>5×27</td>
<td></td>
</tr>
<tr>
<td>Mester and Bräysy (2005) [110]</td>
<td>CTD 168,573</td>
<td>–</td>
<td>8</td>
<td>17</td>
<td>40</td>
<td>145</td>
<td>600</td>
</tr>
<tr>
<td>Le Bouthillier and Crainic (2005) [102]</td>
<td>CTD 57,412</td>
<td>5×3</td>
<td>3×2</td>
<td>3×7</td>
<td>3×13</td>
<td>3×23</td>
<td>3×30</td>
</tr>
<tr>
<td>Opt 2.3GHz</td>
<td>–</td>
<td>5×8</td>
<td>5×16</td>
<td>5×18</td>
<td>5×23</td>
<td>5×27</td>
<td></td>
</tr>
<tr>
<td>Gehring and Homberger (2001) [57]</td>
<td>CTD 57,641</td>
<td>4×14</td>
<td>3×2</td>
<td>3×7</td>
<td>3×13</td>
<td>3×23</td>
<td>3×30</td>
</tr>
<tr>
<td>Gehring and Homberger (2001) [57]</td>
<td>P 400MHz</td>
<td>4×14</td>
<td>3×2</td>
<td>3×7</td>
<td>3×13</td>
<td>3×23</td>
<td>3×30</td>
</tr>
<tr>
<td>Gehring and Homberger (2001) [57]</td>
<td>CTD 56,946</td>
<td>5×8</td>
<td>5×16</td>
<td>5×18</td>
<td>5×23</td>
<td>5×27</td>
<td></td>
</tr>
<tr>
<td>Gehring and Homberger (2001) [57]</td>
<td>P 400MHz</td>
<td>5×8</td>
<td>5×16</td>
<td>5×18</td>
<td>5×23</td>
<td>5×27</td>
<td></td>
</tr>
<tr>
<td>Gehring and Homberger (2005) [80]</td>
<td>CTD 57,422</td>
<td>5×8</td>
<td>5×16</td>
<td>5×18</td>
<td>5×23</td>
<td>5×27</td>
<td></td>
</tr>
<tr>
<td>Homberger and Gehring (2005) [80]</td>
<td>P 400MHz</td>
<td>5×8</td>
<td>5×16</td>
<td>5×18</td>
<td>5×23</td>
<td>5×27</td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>CTD 57,484</td>
<td>17</td>
<td>33</td>
<td>67</td>
<td>100</td>
<td>133</td>
<td>167</td>
</tr>
<tr>
<td>Xeon 2.8GHz</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

for solving an instance. In the notation of the CPU, “P,” “P4,” and “Opt” mean Pentium, Pentium 4 and Opteron, respectively. Marks “×” in the second column mean the number of CPUs (e.g., “4×P 200MHz” means four CPUs of Pentium 200MHz), and those in other columns mean the number of runs (e.g., “5×30” means five runs each with 30 minutes of computation time). A number in bold in rows CNV indicates that the value is the best among all the algorithms in the table and there is no tie. When there are ties for the best
CNV, the corresponding distance value that is the smallest among those ties is indicated by boldface.

From Table 5.1, the CNV obtained by our algorithm is much smaller than those of the other methods for large instances with 600 customers or more, and the computation time spent by our algorithm seems to be reasonable; e.g., for instances with $n = 1000$, the computation times spent by recent algorithms by Hashimoto, Yagiura and Ibaraki [75], Ibaraki et al. [86], Prescott-Gagnon, Desaulniers and Rousseau [127], Pisinger and Ropke [124], and Mester and Bräysy [110] are similar to or sometimes larger than ours even if the difference of CPUs are taken into consideration. Moreover, our algorithm updated 41 best known solutions among the 356 instances.\footnote{We compared our solutions with those reported in the papers [16, 75, 86, 102, 112, 127] and on the SINTEF website (www.top.sintef.no/vrp/benchmarks.html) [138], as of November 11, 2007. (Note that the SINTEF website includes the results of [57, 110, 124].)}

Tables 5.2 and 5.3 show the solution values obtained by our algorithm for Solomon’s benchmark instances and Gehring and Homberger’s benchmark instances. A value in boldface is a new best known solution. This indicates that our algorithm is highly efficient.

## 5.8 Conclusion

We proposed a path relinking approach for the vehicle routing problem with time windows with an adaptive mechanism to control parameters. The generated solutions in the path relinking are improved by a local search. In the local search, each neighborhood solution is evaluated in $O(1)$ time and the neighborhood search is pruned heuristically by the neighbor list. During the search, infeasible solutions are allowed to be visited while the amount of violation is penalized. We also proposed an adaptive mechanism to control the penalty weights. The computational results on representative benchmark instances indicate that the proposed algorithm is highly efficient, and furthermore, the algorithm updated 41 best known solutions among 356 instances.
Table 5.2: The detailed results of our algorithm for the 100–400-customer instances

<table>
<thead>
<tr>
<th>r1/100</th>
<th>r2/100</th>
<th>c1/100</th>
<th>c2/100</th>
<th>rc1/100</th>
<th>rc2/100</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/19/1650.80</td>
<td>01/4/1253.23</td>
<td>01/10/828.94</td>
<td>01/3/591.56</td>
<td>01/14/1696.95</td>
<td>01/4/1413.52</td>
</tr>
<tr>
<td>02/17/1486.12</td>
<td>02/3/1191.70</td>
<td>02/10/828.94</td>
<td>02/3/591.56</td>
<td>02/12/1554.75</td>
<td>02/3/1367.00</td>
</tr>
<tr>
<td>03/13/1292.68</td>
<td>03/3/949.66</td>
<td>03/10/828.06</td>
<td>03/3/591.17</td>
<td>03/11/1261.67</td>
<td>03/3/1068.60</td>
</tr>
<tr>
<td>04/9/1107.31</td>
<td>04/2/844.63</td>
<td>04/10/824.78</td>
<td>04/3/590.60</td>
<td>04/10/1135.48</td>
<td>04/3/816.33</td>
</tr>
<tr>
<td>05/14/1377.11</td>
<td>05/3/994.43</td>
<td>05/10/828.94</td>
<td>05/3/588.88</td>
<td>05/13/1629.44</td>
<td>05/4/1297.65</td>
</tr>
<tr>
<td>06/12/1252.03</td>
<td>06/3/929.49</td>
<td>06/10/828.94</td>
<td>06/3/588.49</td>
<td>06/11/1424.73</td>
<td>06/3/1207.75</td>
</tr>
<tr>
<td>07/10/1109.88</td>
<td>07/2/911.14</td>
<td>07/10/828.94</td>
<td>07/3/588.29</td>
<td>07/11/1230.48</td>
<td>07/3/1094.95</td>
</tr>
<tr>
<td>08/9/969.30</td>
<td>08/2/727.69</td>
<td>08/10/828.94</td>
<td>08/3/588.32</td>
<td>08/10/1139.82</td>
<td>08/3/841.18</td>
</tr>
<tr>
<td>09/11/1194.73</td>
<td>09/3/913.32</td>
<td>09/10/828.94</td>
<td>09/3/588.32</td>
<td>09/10/1139.82</td>
<td>09/3/841.18</td>
</tr>
<tr>
<td>10/10/1131.27</td>
<td>10/3/966.90</td>
<td>11/10/1096.73</td>
<td>11/2/891.89</td>
<td>12/9/1032.47</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r1/200</th>
<th>r2/200</th>
<th>c1/200</th>
<th>c2/200</th>
<th>rc1/200</th>
<th>rc2/200</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/20/4784.11</td>
<td>01/4/4504.88</td>
<td>01/20/2704.57</td>
<td>01/6/1931.44</td>
<td>01/18/3667.40</td>
<td>01/6/3102.30</td>
</tr>
<tr>
<td>02/18/4045.32</td>
<td>02/4/3655.26</td>
<td>02/18/2817.89</td>
<td>02/6/1863.16</td>
<td>02/18/3249.65</td>
<td>02/5/2827.43</td>
</tr>
<tr>
<td>03/18/3395.72</td>
<td>03/4/2945.24</td>
<td>03/18/2707.35</td>
<td>03/6/1777.56</td>
<td>03/18/3011.09</td>
<td>03/4/2623.02</td>
</tr>
<tr>
<td>04/18/3080.53</td>
<td>04/4/2825.06</td>
<td>04/18/2564.42</td>
<td>04/6/1716.20</td>
<td>04/18/3270.29</td>
<td>04/9/2164.55</td>
</tr>
<tr>
<td>05/18/3123.47</td>
<td>05/4/3400.49</td>
<td>05/20/2702.05</td>
<td>05/6/1878.85</td>
<td>05/18/3379.51</td>
<td>05/4/2911.46</td>
</tr>
<tr>
<td>06/18/3642.30</td>
<td>06/2/2954.85</td>
<td>06/20/2701.04</td>
<td>06/6/1857.35</td>
<td>06/18/3306.31</td>
<td>06/4/2880.06</td>
</tr>
<tr>
<td>07/18/3152.45</td>
<td>07/4/2476.50</td>
<td>07/20/2701.04</td>
<td>07/6/1849.46</td>
<td>07/18/3215.33</td>
<td>07/9/2566.46</td>
</tr>
<tr>
<td>08/18/3009.65</td>
<td>08/4/1987.98</td>
<td>08/19/2775.48</td>
<td>08/6/1820.53</td>
<td>08/18/3104.40</td>
<td>08/4/2325.73</td>
</tr>
<tr>
<td>09/18/3773.41</td>
<td>09/4/3125.55</td>
<td>09/18/2687.83</td>
<td>09/6/1830.05</td>
<td>09/18/3088.57</td>
<td>09/4/2270.31</td>
</tr>
<tr>
<td>10/18/3321.50</td>
<td>10/4/2694.35</td>
<td>10/18/2645.08</td>
<td>10/6/1806.58</td>
<td>10/18/3105.06</td>
<td>10/4/2057.02</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r1/400</th>
<th>r2/400</th>
<th>c1/400</th>
<th>c2/400</th>
<th>rc1/400</th>
<th>rc2/400</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/40/10407.99</td>
<td>01/8/9297.61</td>
<td>01/40/7152.06</td>
<td>01/12/4116.14</td>
<td>01/36/8925.01</td>
<td>01/11/6682.37</td>
</tr>
<tr>
<td>02/36/9198.06</td>
<td>02/8/7602.52</td>
<td>02/36/7921.43</td>
<td>02/12/3930.05</td>
<td>02/36/8073.32</td>
<td>02/9/6407.92</td>
</tr>
<tr>
<td>03/36/7921.12</td>
<td>03/8/6190.56</td>
<td>03/36/7921.43</td>
<td>03/12/3774.30</td>
<td>03/36/7631.08</td>
<td>03/8/5054.14</td>
</tr>
<tr>
<td>04/36/7368.95</td>
<td>04/8/4329.59</td>
<td>04/36/6803.26</td>
<td>04/11/3939.40</td>
<td>04/36/7428.82</td>
<td>04/8/3648.30</td>
</tr>
<tr>
<td>05/36/9554.74</td>
<td>05/8/7160.08</td>
<td>05/40/7152.06</td>
<td>05/12/3943.03</td>
<td>05/36/8312.17</td>
<td>05/9/6005.94</td>
</tr>
<tr>
<td>06/36/8623.44</td>
<td>06/8/6215.73</td>
<td>06/40/7153.45</td>
<td>06/12/3875.94</td>
<td>06/36/8297.14</td>
<td>06/8/6045.96</td>
</tr>
<tr>
<td>07/36/7719.15</td>
<td>07/8/5153.95</td>
<td>07/39/7461.24</td>
<td>07/12/3894.16</td>
<td>07/36/8093.52</td>
<td>07/8/5558.01</td>
</tr>
<tr>
<td>08/36/7391.15</td>
<td>08/4/4113.46</td>
<td>08/37/7419.34</td>
<td>08/12/3792.76</td>
<td>08/36/7876.78</td>
<td>08/8/4946.25</td>
</tr>
<tr>
<td>09/36/8977.98</td>
<td>09/8/6500.77</td>
<td>09/36/7107.59</td>
<td>09/12/3870.80</td>
<td>09/36/7853.69</td>
<td>09/8/4569.14</td>
</tr>
<tr>
<td>10/36/8255.40</td>
<td>10/8/5999.98</td>
<td>10/36/6889.23</td>
<td>10/11/3964.56</td>
<td>10/36/7687.41</td>
<td>10/8/4350.64</td>
</tr>
</tbody>
</table>
Table 5.3: The detailed results of our algorithm for the 600–1000-customer instances

<table>
<thead>
<tr>
<th>r1/600</th>
<th>r2/600</th>
<th>c1/600</th>
<th>c2/600</th>
<th>rc1/600</th>
<th>rc2/600</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/59/21713.75</td>
<td>01/11/18774.16</td>
<td>01/55/17586.75</td>
<td>01/14/13691.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02/54/19493.96</td>
<td>02/11/15106.93</td>
<td>02/55/16233.10</td>
<td>02/12/11763.41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03/54/17471.73</td>
<td>03/11/11599.34</td>
<td>03/55/15413.87</td>
<td>03/11/9807.61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04/54/16037.12</td>
<td>04/11/8285.24</td>
<td>04/55/15000.79</td>
<td>04/11/7377.47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05/54/20681.01</td>
<td>05/11/14085.72</td>
<td>05/55/17045.97</td>
<td>05/12/12405.59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/54/18616.76</td>
<td>06/11/12779.53</td>
<td>06/55/17136.28</td>
<td>06/11/12491.87</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07/54/17114.55</td>
<td>07/11/10389.22</td>
<td>07/55/16511.30</td>
<td>07/11/10928.52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>08/54/15884.43</td>
<td>08/11/7969.11</td>
<td>08/55/16237.07</td>
<td>08/11/10436.44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09/54/19387.64</td>
<td>09/11/13871.90</td>
<td>09/55/16325.51</td>
<td>09/11/10096.96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/54/18383.25</td>
<td>10/11/12572.23</td>
<td>10/55/17280.70</td>
<td>10/11/9413.25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r1/800</th>
<th>r2/800</th>
<th>c1/800</th>
<th>c2/800</th>
<th>rc1/800</th>
<th>rc2/800</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/80/37400.64</td>
<td>01/15/28839.78</td>
<td>01/80/25184.38</td>
<td>01/24/11662.08</td>
<td>01/18/21154.34</td>
<td></td>
</tr>
<tr>
<td>02/72/33573.64</td>
<td>02/15/23157.11</td>
<td>02/72/27012.87</td>
<td>02/23/12460.76</td>
<td>02/16/18799.25</td>
<td></td>
</tr>
<tr>
<td>03/72/30349.76</td>
<td>03/15/18265.81</td>
<td>03/72/24779.24</td>
<td>03/23/11770.80</td>
<td>03/15/14939.88</td>
<td></td>
</tr>
<tr>
<td>04/72/28459.74</td>
<td>04/15/13759.46</td>
<td>04/72/21779.24</td>
<td>04/23/11160.68</td>
<td>04/15/11410.24</td>
<td></td>
</tr>
<tr>
<td>05/72/31798.47</td>
<td>05/15/20775.72</td>
<td>05/72/25166.28</td>
<td>05/24/11428.66</td>
<td>05/15/34173.25</td>
<td></td>
</tr>
<tr>
<td>06/72/29655.26</td>
<td>06/15/17102.33</td>
<td>06/72/26003.16</td>
<td>06/24/11370.84</td>
<td>06/15/19497.33</td>
<td></td>
</tr>
<tr>
<td>07/72/28349.07</td>
<td>07/15/13059.96</td>
<td>07/72/25844.26</td>
<td>07/23/11301.27</td>
<td>07/15/16239.34</td>
<td></td>
</tr>
<tr>
<td>08/72/33468.43</td>
<td>08/15/23017.30</td>
<td>08/72/24793.22</td>
<td>08/23/11835.04</td>
<td>08/15/15556.86</td>
<td></td>
</tr>
<tr>
<td>09/72/31871.25</td>
<td>09/15/21074.60</td>
<td>09/72/24522.58</td>
<td>09/23/11163.36</td>
<td>09/15/14726.07</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r1/1000</th>
<th>r2/1000</th>
<th>c1/1000</th>
<th>c2/1000</th>
<th>rc1/1000</th>
<th>rc2/1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/100/54955.74</td>
<td>01/19/43054.76</td>
<td>01/100/42478.95</td>
<td>01/30/16879.24</td>
<td>01/90/48248.59</td>
<td>01/20/30912.50</td>
</tr>
<tr>
<td>02/91/52208.39</td>
<td>02/19/4293.42</td>
<td>02/90/43355.70</td>
<td>02/29/17452.36</td>
<td>02/90/45344.22</td>
<td>02/19/26597.14</td>
</tr>
<tr>
<td>03/91/46574.56</td>
<td>03/19/25934.52</td>
<td>03/90/40548.32</td>
<td>03/28/17519.99</td>
<td>03/90/43261.35</td>
<td>03/18/20698.99</td>
</tr>
<tr>
<td>04/91/43669.02</td>
<td>04/19/18269.76</td>
<td>04/90/39908.25</td>
<td>04/28/16783.84</td>
<td>04/90/42625.25</td>
<td>04/18/16402.21</td>
</tr>
<tr>
<td>05/91/55002.93</td>
<td>05/19/37357.94</td>
<td>05/100/42469.18</td>
<td>05/30/16563.10</td>
<td>05/90/47427.58</td>
<td>05/18/27715.20</td>
</tr>
<tr>
<td>06/91/50124.16</td>
<td>06/19/30879.06</td>
<td>06/100/42471.28</td>
<td>06/29/17491.11</td>
<td>06/90/46651.17</td>
<td>06/18/28256.73</td>
</tr>
<tr>
<td>07/91/46193.38</td>
<td>07/19/24075.04</td>
<td>07/90/43867.54</td>
<td>07/29/18277.92</td>
<td>07/90/46061.01</td>
<td>07/18/25763.52</td>
</tr>
<tr>
<td>08/91/43264.54</td>
<td>08/19/18229.89</td>
<td>08/90/43120.86</td>
<td>08/28/16839.40</td>
<td>08/90/45524.59</td>
<td>08/18/24454.85</td>
</tr>
<tr>
<td>09/91/53848.20</td>
<td>09/19/34224.22</td>
<td>09/90/42731.02</td>
<td>09/29/16680.50</td>
<td>09/90/45508.55</td>
<td>09/18/23802.84</td>
</tr>
<tr>
<td>10/91/50639.84</td>
<td>10/19/31390.03</td>
<td>10/90/40624.36</td>
<td>10/28/16584.54</td>
<td>10/90/44918.74</td>
<td>10/18/22365.07</td>
</tr>
</tbody>
</table>
Chapter 6

Conclusion

Throughout this thesis, we have considered local search-based algorithms for vehicle routing and scheduling problems. The results in this thesis are summarized as follows.

First, in Chapter 1, we described background of vehicle routing and scheduling problems and their underlying complexities. Vehicle routing and scheduling problems are difficult to obtain exact optimal solutions, and hence heuristic algorithms are important in practice. We reviewed recent research results for local search and gave brief descriptions of some representative metaheuristics.

In Chapter 2, we explained several basic techniques for solving the standard vehicle routing and scheduling problems. These techniques use the characteristics specific to each problem and cannot be applied to other problems directly. To deal with wider range of problems under a unified framework, we proposed a general formulation that includes the standard vehicle routing and scheduling problems. Under this general model, we proposed an efficient neighborhood search method for the standard neighborhoods called 2-opt*, cross exchange and Or-opt. The neighborhood search method was then incorporated in the algorithms of the following chapters.

In Chapter 3, we described the generalization of the standard vehicle routing problem by allowing soft time window and soft traveling time constraints, where both constraints can be violated and the amounts of violation are penalized by cost functions. In the algorithm, we used the neighborhood search method which was described in Chapter 2. In order to apply the method, we need to solve the problem of determining the optimal start times of services at visited customers after fixing the route of each vehicle. We showed that this subproblem is NP-hard when cost functions are general, but can be efficiently solved with dynamic programming when traveling time cost functions are convex even if time window cost functions are non-convex. The computational results on benchmark instances confirmed the benefits of the proposed generalization.
In Chapter 4, we considered another generalization of the standard vehicle routing problem with time windows by allowing both traveling times and traveling costs to be time-dependent functions. We showed that the subproblem of asking an optimal time schedule of a route can be efficiently solved and the algorithm could be applied in the neighborhood search method. We further proposed a filtering method that restricts the search in the neighborhoods to a small portion by avoiding solutions having no prospect of improvement. The computational results of our algorithm compared against existing methods confirmed the effectiveness of the restriction of the neighborhoods and the benefits of the proposed generalization.

In Chapter 5, we described a path relinking approach for the vehicle routing problem with time windows. The path relinking is an evolutionary mechanism that generates new solutions by combining two or more reference solutions. In our algorithm, those solutions generated by path relinking operations are improved by a local search. To make the search more efficient, we proposed a neighbor list that prunes the neighborhood search heuristically. We proposed an adaptive mechanism for parameters which control the search direction. In the mechanism, those parameters are updated using information from the last call to the local search. The computational results on well-studied benchmark instances with up to 1000 customers revealed that our algorithm is highly efficient especially for large instances. Moreover, it updated 41 best known solutions among 356 instances.

Vehicle routing and scheduling problems are fundamental issues in human society. As information tools related to vehicle routing and scheduling (e.g., GIS, demand forecasting) have recently been enhanced, an efficient algorithm may immediately make an improvement on these issues. The author hopes that the study in this thesis will be helpful for the developments of such fabulous algorithms.


A List of Author’s Work

Journals


International Conferences with Review


**Unpublished Manuscripts**
