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)

Preface

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 heartful 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'Alès, 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

List of Figures

List of Tables

List of Algorithms

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 *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

(Assignment) the assignment of customers to vehicles,

(Routing) the visiting order of customers who are assigned to a vehicle, and

(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.

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 perturbation. 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],

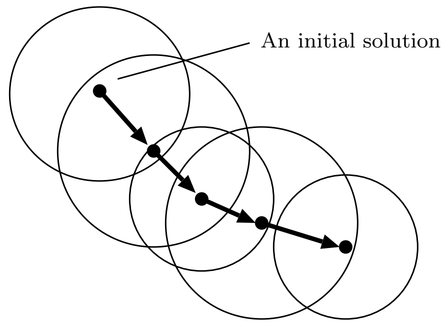


Figure 1.1: An illustration of local search

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 $(\mathcal{S}, f)$, where the solution set $\mathcal{S}$ is the set of feasible solutions and the cost function f is a mapping $f : \mathcal{S} \to \mathbb{R}$. A neighborhood function is a mapping $N : \mathcal{S} \to 2^{\mathcal{S}}$, which defines for each solution $x \in \mathcal{S}$ a set $N(x) \subseteq \mathcal{S}$ 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 ϵ-local optimality and showed that, for every $\epsilon > 0$, an ϵ-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 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 NP = 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], MAX-CUT 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 *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.

explanation of the tabu search. TS is described as Algorithm 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 (≥ 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 σ_k denote the route traveled by vehicle k, where $\sigma_k(h)$ denotes the hth customer in σ_k, and let

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

Note that each customer i is included in exactly one route σ_k, and is visited by vehicle k exactly once. We denote by n_k the number of customers in σ_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. Note that each vehicle is allowed to wait at customers before starting services.

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

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

That is, $y_{ik}(\boldsymbol{\sigma}) = 1$ 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:

$$
\begin{array}{llll}
\text{minimize} & \displaystyle\sum_{k \in M} d(\sigma_k) & & (2.2.1)\\
\text{subject to} & \displaystyle\sum_{k \in M} y_{ik}(\boldsymbol{\sigma}) = 1, & i \in V \setminus \{0\} & (2.2.2)\\
& \displaystyle\sum_{i \in V \setminus \{0\}} a_i y_{ik}(\boldsymbol{\sigma}) \leq u, & k \in M & (2.2.3)\\
& t_{0,\sigma_k(1)} \leq s_{\sigma_k(1)}, & k \in M & (2.2.4)\\
& s_{\sigma_k(i)} + t_{\sigma_k(i),\sigma_k(i+1)} \leq s_{\sigma_k(i+1)}, & 1 \leq i \leq n_k - 1,\ k \in M & (2.2.5)\\
& s_{\sigma_k(n_k)} + t_{\sigma_k(n_k),0} \leq s_k^{\mathrm{a}} \leq l_0, & k \in M & (2.2.6)\\
& e_i \leq s_i \leq l_i, & i \in V \setminus \{0\} & (2.2.7)\\
& y_{ik}(\boldsymbol{\sigma}) \in \{0, 1\}, & i \in V \setminus \{0\},\ k \in M. & (2.2.8)
\end{array}
$$

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 σ_k, since the values of $y_{ik}(\boldsymbol{\sigma})$ are automatically determined from $\boldsymbol{\sigma}$, and finding appropriate values for s_i and s_k^{a}, if any, is easy when $\boldsymbol{\sigma}$ 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 metaheuristics, and they may be used to generate an initial solution for local search and metaheuristics.

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 $d_{ik} = \min\{c_{0,i} + c_{i,i_k} + c_{i_k,0}, c_{0,i_k} + c_{i_k,i} + c_{i,0}\} - (c_{0,i_k} + c_{i_k,0})$. 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 λ-opt neighborhood

The λ-opt neighborhood was proposed by Lin [105] for the traveling salesman problem. For a given route, it removes λ edges of a route, and the resulting λ 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.

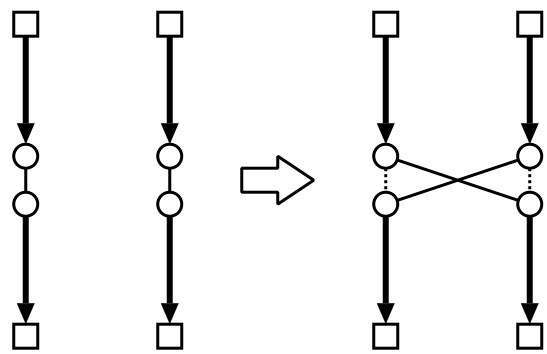


Figure 2.1: A 2-opt* neighborhood operation

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 icross exchange neighborhood is an extension where the exchanged paths can be inserted with the reverse order [22].

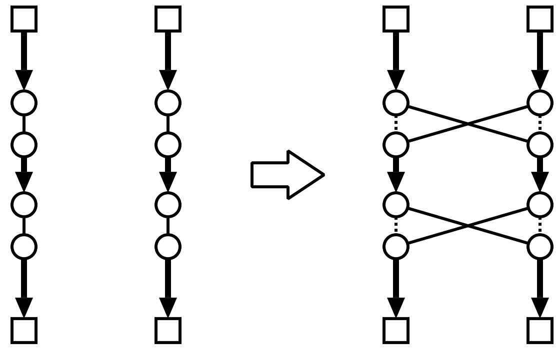


Figure 2.2: A cross exchange neighborhood operation

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

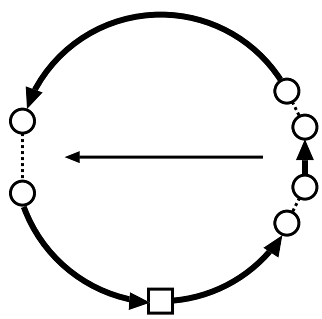


Figure 2.3: An Or-opt neighborhood operation

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 λ-transfer operation transfers λ 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 ${p_{\mathrm{cust}}}_i^\omega(X)$ for the amount X of resource ω consumed on the route from the depot to customer i,
2. a cost function ${p_{\mathrm{depot}}}_k^\omega(X)$ for the amount X of resource ω consumed on the whole route visited by vehicle k,
3. a resource demand function $\lambda_{i,j}^\omega(X)$ between edge (i, j) when the amount of resource ω consumed on the route from the depot to customer i is X, and
4. a cost function $q_{i,j}^\omega(Y, X)$ between edge (i, j) when the amount of resource ω consumed on the route from the depot to customer i is X and the amount of resource ω supplied on the edge is Y.

When a vehicle travels an edge (i,j), it consumes $\lambda^{\omega}_{i,j}(X)$ units of resource for each ω 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 $q^{\omega}_{i,j}(Y,X)$.

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

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

Note that each customer i is included in exactly one route σ_k, and is visited by vehicle k exactly once. We denote by n_k the number of customers in σ_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 $y_{ik}(\boldsymbol{\sigma}) \in \{0,1\}$ for $i \in V \setminus \{0\}$ and $k \in M$ by

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

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

$$\boldsymbol{X} = \begin{pmatrix} X^1_1 & X^1_2 & \ldots & X^1_n & {X_{\mathrm{depot}}}^1_1 & {X_{\mathrm{depot}}}^1_2 & \ldots & {X_{\mathrm{depot}}}^1_m \\ X^2_1 & X^2_2 & \ldots & X^2_n & {X_{\mathrm{depot}}}^2_1 & {X_{\mathrm{depot}}}^2_2 & \ldots & {X_{\mathrm{depot}}}^2_m \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ X^{\nu}_1 & X^{\nu}_2 & \ldots & X^{\nu}_n & {X_{\mathrm{depot}}}^{\nu}_1 & {X_{\mathrm{depot}}}^{\nu}_2 & \ldots & {X_{\mathrm{depot}}}^{\nu}_m \end{pmatrix}.$$

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

$$\boldsymbol{Y} = (Y^{\omega}_{i,j}).$$

The cost $p^{\omega}(\sigma_k, \boldsymbol{X}, \boldsymbol{Y})$ of resource ω for a route σ_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:

$$\begin{aligned} p^{\omega}(\sigma_k, \boldsymbol{X}, \boldsymbol{Y}) = & \sum_{h=1}^{n_k} {p_{\mathrm{cust}}}^{\omega}_{\sigma_k(h)}(X^{\omega}_{\sigma_k(h)}) + {p_{\mathrm{depot}}}^{\omega}_k({X_{\mathrm{depot}}}^{\omega}_k) \\ & + q^{\omega}_{0,\sigma_k(1)}(Y^{\omega}_{0,\sigma_k(1)}, 0) + \sum_{h=1}^{n_k} q^{\omega}_{\sigma_k(h),\sigma_k(h+1)}(Y^{\omega}_{\sigma_k(h),\sigma_k(h+1)}, X^{\omega}_{\sigma_k(h)}). \quad (2.5.9) \end{aligned}$$

Then a general problem is formulated as follows:

$$\text{minimize} \quad \sum_{\omega \in \Omega} \sum_{k \in M} p^{\omega}(\sigma_k, \boldsymbol{X}, \boldsymbol{Y}) \tag{2.5.10}$$

$$\text{subject to} \quad \sum_{k \in M} y_{ik}(\boldsymbol{\sigma}) = 1, \qquad i \in V \setminus \{0\} \tag{2.5.11}$$

$$\lambda^{\omega}_{0,\sigma_k(1)}(0) - Y^{\omega}_{0,\sigma_k(1)} = X^{\omega}_{\sigma_k(1)}, \qquad k \in M, \omega \in \Omega \tag{2.5.12}$$

$$X^{\omega}_{\sigma_k(h)} + \lambda^{\omega}_{\sigma_k(h),\sigma_k(h+1)}(X^{\omega}_{\sigma_k(h)}) - Y^{\omega}_{\sigma_k(h),\sigma_k(h+1)} = X^{\omega}_{\sigma_k(h+1)}, \qquad 1 \leq h \leq n_k - 1, \; k \in M, \omega \in \Omega \tag{2.5.13}$$

$$X^{\omega}_{\sigma_k(n_k)} + \lambda^{\omega}_{\sigma_k(n_k),0}(X^{\omega}_{\sigma_k(n_k)}) - Y^{\omega}_{\sigma_k(n_k),0} = X_{\mathrm{depot}}{}^{\omega}_{k}, \qquad k \in M, \omega \in \Omega \tag{2.5.14}$$

$$y_{ik}(\boldsymbol{\sigma}) \in \{0,1\}, \qquad i \in V \setminus \{0\}, \; k \in M. \tag{2.5.15}$$

For example, in this model, time is considered as a resource ω; it takes $\lambda^{\omega}_{i,j}(X)$ time to travel an edge (i,j) (in other words the resource is consumed), its traveling time depends on the start time X of traveling (in other words, the consumed amount of the resource so far), and the traveling time can be shortened by Y with additional cost $q^{\omega}_{i,j}(Y,X)$ (in other words, it takes cost to supply the resource). The time window constraint of servicing customer i (resp., arrival of vehicle k at the depot) can be expressed by defining $p_{\mathrm{cust}}{}^{\omega}_{i}(X^{\omega}_{i})$ (resp., $p_{\mathrm{depot}}{}^{\omega}_{k}(X_{\mathrm{depot}}{}^{\omega}_{k})$) as 0 if X^{ω}_{i} (resp., $X_{\mathrm{depot}}{}^{\omega}_{k}$) is within its time window, otherwise ∞.

We consider the problem of determining the optimal schedule for a given route σ_k so that the total cost is minimized. Since the route is given and the problem is independent for each resource ω, we have only to consider the following simpler problem, which we call the optimal scheduling problem (OSP). For convenience, 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., $X^{\omega}_{0} = 0$), and let customer $n_k + 1$ represent the arrival at the depot (i.e., $X^{\omega}_{n_k+1} = X_{\mathrm{depot}}{}^{\omega}_{k}$ and $p_{\mathrm{cust}}{}^{\omega}_{n_k+1}(X^{\omega}_{n_k+1}) = p_{\mathrm{depot}}{}^{\omega}_{k}(X_{\mathrm{depot}}{}^{\omega}_{k})$). Then, the OSP is described as follows:

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{h=1}^{n_k+1} p_{\text{cust}}{}_h^\omega(X_h^\omega) + \sum_{h=0}^{n_k} q_{h,h+1}^\omega(Y_{h,h+1}^\omega, X_h^\omega) & (2.5.16) \\
\text{subject to} & X_h^\omega + \lambda_{h,h+1}^\omega(X_h^\omega) - Y_{h,h+1}^\omega = X_{h+1}^\omega, \qquad 0 \le h \le n_k. & (2.5.17)
\end{array}
$$

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:

$$
\begin{aligned}
f_0(t) &= \begin{cases} 0, & t = 0 \\ +\infty, & \text{otherwise} \end{cases} \\
f_h(t) &= p_{\text{cust}}{}_h^\omega(t) + \min_{X_{h-1}^\omega + \lambda_{h-1,h}^\omega(X_{h-1}^\omega) - Y_{h-1,h}^\omega = t} \{f_{h-1}(X_{h-1}^\omega) + q_{h-1,h}^\omega(Y_{h-1,h}^\omega, X_{h-1}^\omega)\}, \\
&\qquad 1 \le h \le n_k + 1, -\infty < t < +\infty. \qquad (2.5.18)
\end{aligned}
$$

The optimal cost of the OSP for a route σ_k is given by $\min_t f_{n_k+1}(t)$. We can also formulate the dynamic programming recursion in another way.

> 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:

$$
\begin{aligned}
b_{n_k+1}(t) &= p_{\text{cust}}{}_{n_k+1}^\omega(t) \\
b_h(t) &= p_{\text{cust}}{}_h^\omega(t) + \min_{t + \lambda_{h,h+1}^\omega(t) - Y_{h,h+1}^\omega = X_{h+1}^\omega} \{b_{h+1}(X_{h+1}^\omega) + q_{h,h+1}^\omega(Y_{h,h+1}^\omega, t)\}, \\
&\qquad 1 \le h \le n_k. \qquad (2.5.19)
\end{aligned}
$$

Now the optimal cost of the OSP for a route σ_k is also given by

$$
\min_t \left\{ f_h(t) + \min_{t + \lambda_{h,h+1}^\omega(t) - Y_{h,h+1}^\omega = X_{h+1}^\omega} \{b_{h+1}(X_{h+1}^\omega) + q_{h,h+1}^\omega(Y_{h,h+1}^\omega, t)\} \right\} \qquad (2.5.20)
$$

for any h $(1 \le h \le n_k)$.

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 σ_k.

2.6.1 Basic idea

A key observation to the efficient computation is that each route σ_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 σ_k, let $\mathcal{F}_h$ (resp., $\mathcal{B}_h$) be some data structure that contains the information of the path (of σ_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 σ_k. For example, if σ_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 σ_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 σ_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 ($0 \le h \le n_k$). 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.

2.6.2 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 $\langle \sigma_k(h_1) \to \sigma_k(h_2) \rangle$ the path from the h_1st customer to the h_2nd customer in route σ_k, and by $\langle \sigma_k(h_1) \to \sigma_k(h_2) \rangle$–$\langle \sigma_{k'}(h_3) \to \sigma_{k'}(h_4) \rangle$ the path constructed by connecting two paths $\langle \sigma_k(h_1) \to \sigma_k(h_2) \rangle$ and $\langle \sigma_{k'}(h_3) \to \sigma_{k'}(h_4) \rangle$ from routes σ_k and $\sigma_{k'}$.

2-opt* neighborhood

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

In Figure 2.4, an example of a 2-opt* operation on routes σ_k and $\sigma_{k'}$ is shown. We

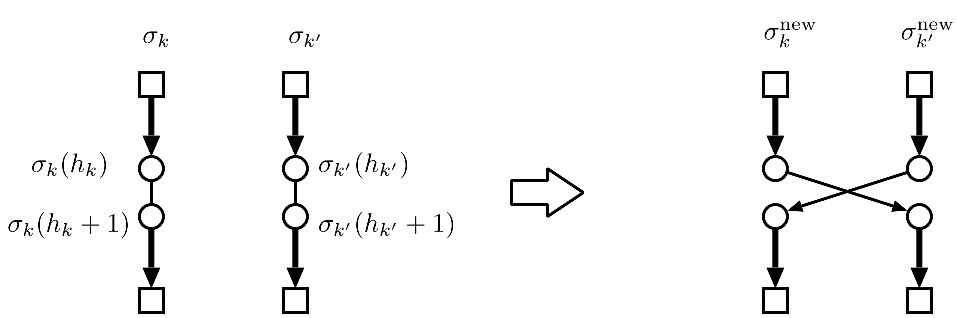


Figure 2.4: An illustration of the search method for the 2-opt* neighborhood

denote by σ_k^{new} and $\sigma_{k'}^{\mathrm{new}}$ the resulting two routes (i.e., $\sigma_k^{\mathrm{new}} = \langle 0 \to \sigma_k(h_k) \rangle$–$\langle \sigma_{k'}(h_{k'} + 1) \to 0 \rangle$ and $\sigma_{k'}^{\mathrm{new}} = \langle 0 \to \sigma_{k'}(h_{k'}) \rangle$–$\langle \sigma_k(h_k + 1) \to 0 \rangle$). Then the cost $p_{\mathrm{opt}}(\sigma_{k'}^{\mathrm{new}})$ (resp., $p_{\mathrm{opt}}(\sigma_k^{\mathrm{new}})$) 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 σ_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

procedure 2-OPT*$(\sigma_k, \sigma_{k'}, h_k, h_{k'})$
  Connect$(\mathcal{F}_{\sigma_k(h_k)}, \mathcal{B}_{\sigma_{k'}(h_{k'}+1)}, (\sigma_k(h_k), \sigma_{k'}(h_{k'}+1)))$ ▷ $p_{\mathrm{opt}}(\sigma_k^{\mathrm{new}})$
  Connect$(\mathcal{F}_{\sigma_{k'}(h_{k'})}, \mathcal{B}_{\sigma_k(h_k+1)}, (\sigma_{k'}(h_{k'}), \sigma_k(h_k+1)))$ ▷ $p_{\mathrm{opt}}(\sigma_{k'}^{\mathrm{new}})$
end procedure

Cross exchange neighborhood

Let us consider the cross exchange operation on routes σ_k and $\sigma_{k'}$. We restrict the length (i.e., the number of customers in the path) of the exchanged path at most L^{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 $\sigma_{k'}$, 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. We denote by σ_k^{new} and $\sigma_{k'}^{\mathrm{new}}$ the resulting two routes (i.e., $\sigma_k^{\mathrm{new}} = \langle 0 \to \sigma_k(h_1^k)\rangle$–$\langle\sigma_{k'}(h_1^{k'}+1) \to \sigma_{k'}(h_{k'}+l')\rangle$– $\langle\sigma_k(h_k+l+1) \to 0\rangle$ and $\sigma_{k'}^{\mathrm{new}} = \langle 0 \to \sigma_{k'}(h_1^{k'})\rangle$–$\langle\sigma_k(h_1^k+1) \to \sigma_k(h_k+l)\rangle$–$\langle\sigma_{k'}(h_{k'}+l'+1) \to 0\rangle$). Then the cost $p_{\mathrm{opt}}(\sigma_k^{\mathrm{new}})$ (resp., $p_{\mathrm{opt}}(\sigma_{k'}^{\mathrm{new}})$) of the new route is obtained by connecting $\sigma_{k'}(h_{k'}+l')$ and $\sigma_k(h_k+l+1)$ (resp., $\sigma_k(h_k+l)$ and $\sigma_{k'}(h_{k'}+l'+1)$). The procedure is described as Algorithm 11. See Figure 2.5 for a help to understand the description.

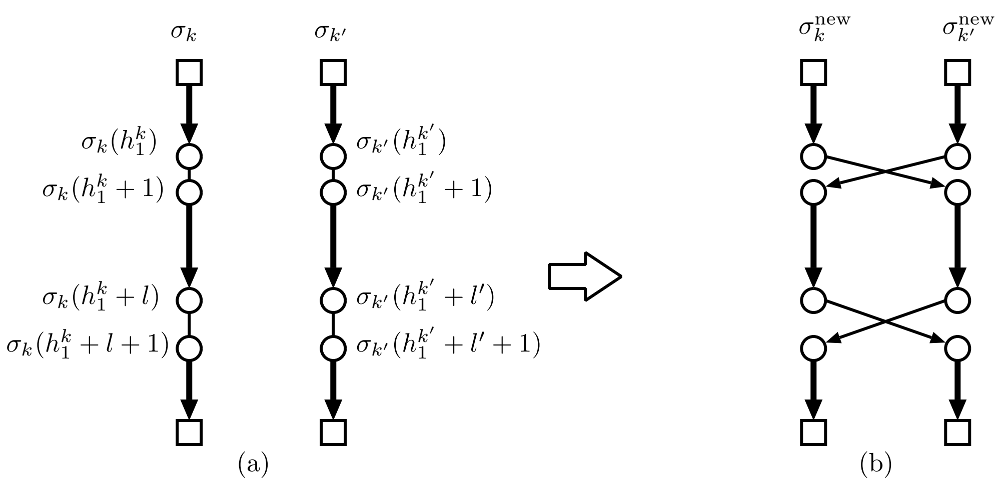


Figure 2.5: An illustration of the search method for the cross exchange neighborhood

Although, in Algorithm 11, at least one customer is exchanged from both routes for

Algorithm 11 The search method for cross exchange neighborhood

procedure Cross($\sigma_k, \sigma_{k'}, h_1^k, h_1^{k'}$)
  for $l \leftarrow 1, \min\{L^{\text{cross}}, n_k - h_1^k\}$ **do** ▷ Extend path $\langle \sigma_k(h_1^k+1) \to \sigma_k(h_1^k+l) \rangle$
    if $l = 1$ **then**
      $\tilde{\mathcal{F}}$:=Forward($\mathcal{F}^{k'}_{\sigma_{k'}(h_1^{k'})}, (\sigma_k(h_1^k), \sigma_k(h_1^k+1))$)
    else
      $\tilde{\mathcal{F}}$:=Forward($\tilde{\mathcal{F}}, (\sigma_k(h_1^k+l-1), \sigma_k(h_1^k+l))$)
    end if
    for $l' \leftarrow 1, \min\{L^{\text{cross}}, n_{k'} - h_1^{k'}\}$ **do** ▷ Extend path $\langle \sigma_{k'}(h_1^{k'}+1) \to \sigma_{k'}(h_1^{k'}+l) \rangle$
      if $l = 1$ **then**
        $\tilde{\mathcal{F}}'$:=Forward($\mathcal{F}^{k}_{\sigma_k(h_1^k)}, (\sigma_{k'}(h_1^{k'}), \sigma_{k'}(h_1^{k'}+1))$)
      else
        $\tilde{\mathcal{F}}'$:=Forward($\tilde{\mathcal{F}}', (\sigma_{k'}(h_1^{k'}+l'-1), \sigma_{k'}(h_1^{k'}+l'))$)
      end if
      Connect($\tilde{\mathcal{F}}, \mathcal{B}^k_{\sigma_k(h_1^k+l+1)}, (\sigma_{k'}(h_1^{k'}+l'), \sigma_k(h_1^k+l+1))$) ▷ $p_{\text{opt}}(\sigma_k^{\text{new}})$
      Connect($\tilde{\mathcal{F}}', \mathcal{B}^{k'}_{\sigma_{k'}(h_1^{k'}+l'+1)}, (\sigma_k(h_1^k+l), \sigma_{k'}(h_1^{k'}+l'+1))$) ▷ $p_{\text{opt}}(\sigma_{k'}^{\text{new}})$
    end for
  end for
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 σ_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 σ_k^{new} the resulting route. In Algorithm 12, a path $\langle \sigma_k(h_1+l-1) \rightarrow \sigma_k(h_1) \rangle$

Algorithm 12 The search method for the forward reverse intra neighborhood

---

procedure $\textsc{IntraForwardReverse}(\sigma_k, h_1)$
    for $h_2 \leftarrow h_1+1, \min\{h_1 + L_{\text{ins}}^{\text{intra}}, n_k\}$ **do**   ▷ Insert a path between $\sigma_k(h_2)$ and $\sigma_k(h_2+1)$
        for $l \leftarrow 1, \min\{L_{\text{path}}^{\text{intra}}, h_2 - h_1\}$ **do**   ▷ Extend path $\langle \sigma_k(h_1) \rightarrow \sigma_k(h_1+l-1) \rangle$
            if $h_2 = h_1 + 1$ **then**
                $\tilde{\mathcal{F}}_l$:=Forward$(\tilde{\mathcal{F}}_l, (\sigma_k(h_1-1), \sigma_k(h_2)))$
            else
                $\tilde{\mathcal{F}}_l$:=Forward$(\tilde{\mathcal{F}}_l, (\sigma_k(h_2-1), \sigma_k(h_2)))$
            end if
            if $l = 1$ **then**
                $\tilde{\mathcal{B}}$:=Backward$(\mathcal{B}, (\sigma_k(h_1+l-1), \sigma_k(h_2+1)))$
            else
                $\tilde{\mathcal{B}}$:=Backward$(\tilde{\mathcal{B}}, (\sigma_k(h_1+l-1), \sigma_k(h_1+l-2)))$
            end if
            Connect$(\tilde{\mathcal{F}}_l, \tilde{\mathcal{B}}, (\sigma_k(h_2), \sigma_k(h_1+l-1)))$ ▷ $p_{\text{opt}}(\sigma_k^{\text{new}})$
        end for
    end for
end procedure

---

is removed from σ_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.

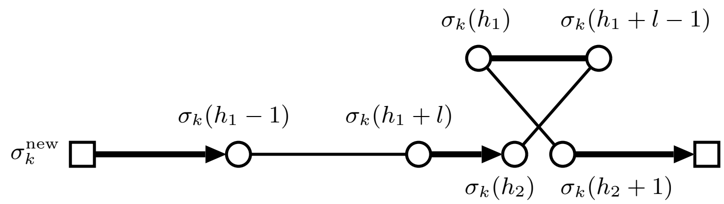


Figure 2.6: An illustration of the search method for the forward reverse intra neighborhood

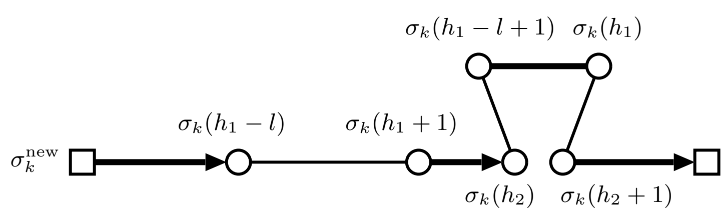


Figure 2.7: An illustration of the search method for the forward normal intra neighborhood

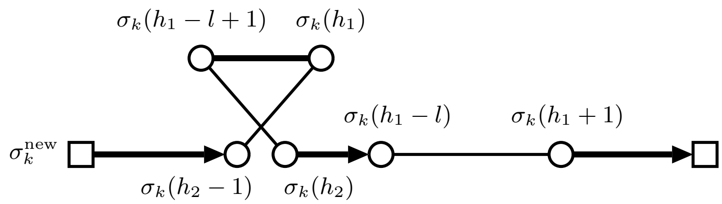


Figure 2.8: An illustration of the search method for the backward reverse intra neighborhood

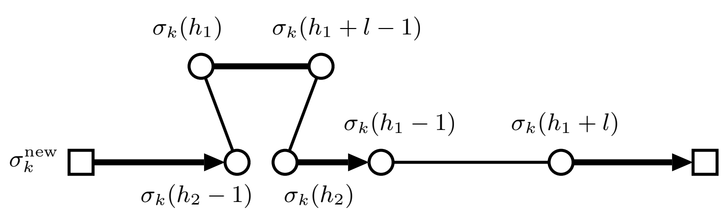


Figure 2.9: An illustration of the search method for the backward normal intra neighborhood

Algorithm 13 The search method for the forward normal intra neighborhood

procedure IntraForwardNormal(σ_k, h_1)
    for $h_2 \leftarrow h_1 + 1, \min\{h_1 + L_{\text{ins}}^{\text{intra}}, n_k\}$ **do** ▷ Insert a path between $\sigma_k(h_2)$ and $\sigma_k(h_2 + 1)$
        for $l \leftarrow 1, \min\{L_{\text{path}}^{\text{intra}}, h_2 - h_1\}$ **do** ▷ Extend path $\langle \sigma_k(h_1 - l + 1) \rightarrow \sigma_k(h_1) \rangle$
            if $h_2 = h_1 + 1$ **then**
                $\tilde{\mathcal{F}}_l$:=Forward($\mathcal{F}_{\sigma_k(h_1-l)}, (\sigma_k(h_1 - l), \sigma_k(h_2))$)
            else
                $\tilde{\mathcal{F}}_l$:=Forward($\tilde{\mathcal{F}}_l, (\sigma_k(h_2 - 1), \sigma_k(h_2))$)
            end if
            if $l = 1$ **then**
                $\tilde{\mathcal{B}}$:=Backward($\mathcal{B}_{\sigma_k(h_2+1)}, (\sigma_k(h_1 - l + 1), \sigma_k(h_2 + 1))$)
            else
                $\tilde{\mathcal{B}}$:=Backward($\tilde{\mathcal{B}}, (\sigma_k(h_1 - l + 1), \sigma_k(h_1 - l + 2))$)
            end if
            Connect($\tilde{\mathcal{F}}_l, \tilde{\mathcal{B}}, (\sigma_k(h_2), \sigma_k(h_1 - l + 1))$) ▷ $p_{\text{opt}}(\sigma_k^{\text{new}})$
        end for
    end for
end procedure

Algorithm 14 The search method for the backward reverse intra neighborhood

procedure IntraBackwardReverse(σ_k, h_1)
    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 $\sigma_k(h_2)$
        for $l \leftarrow 1, \min\{L_{\text{path}}^{\text{intra}}, h_1 - h_2\}$ **do** ▷ Extend path $\langle \sigma_k(h_1 - l + 1) \rightarrow \sigma_k(h_1) \rangle$
            if $h_2 = h_1 - 1$ **then**
                $\tilde{\mathcal{B}}_l$:=Backward($\mathcal{B}_{\sigma_k(h_1 - l)}, (\sigma_k(h_2), \sigma_k(h_1 + 1))$)
            else
                $\tilde{\mathcal{B}}_l$:=Backward($\tilde{\mathcal{B}}_l, (\sigma_k(h_2), \sigma_k(h_2 + 1))$)
            end if
            if $l = 1$ **then**
                $\tilde{\mathcal{F}}$:=Forward($\mathcal{F}_{\sigma_k(h_2 - 1)}, (\sigma_k(h_2 - 1), \sigma_k(h_1 - l + 1))$)
            else
                $\tilde{\mathcal{F}}$:=Forward($\tilde{\mathcal{F}}, (\sigma_k(h_1 - l + 2), \sigma_k(h_1 - l + 1))$)
            end if
            Connect($\tilde{\mathcal{F}}, \tilde{\mathcal{B}}_l, (\sigma_k(h_1 - l + 1), \sigma_k(h_2))$) ▷ $p_{\text{opt}}(\sigma_k^{\text{new}})$
        end for
    end for
end procedure

Algorithm 15 The search method for the backward normal intra neighborhood

procedure IntraBackwardNormal(σ_k, h_1)
  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 $\sigma_k(h_2)$
    for $l \leftarrow 1, \min\{L_{\text{path}}^{\text{intra}}, h_1 - h_2\}$ **do** ▷ Extend path $\langle \sigma_k(h_1) \rightarrow \sigma_k(h_1 + l - 1) \rangle$
      if $h_2 = h_1 - 1$ **then**
        $\tilde{\mathcal{B}}_l$:=Backward($\mathcal{B}_{\sigma_k(h_1+l)}, (\sigma_k(h_2), \sigma_k(h_1 + l))$)
      else
        $\tilde{\mathcal{B}}_l$:=Backward($\tilde{\mathcal{B}}_l, (\sigma_k(h_2), \sigma_k(h_2 + 1))$)
      end if
      if $l = 1$ **then**
        $\tilde{\mathcal{F}}$:=Forward($\mathcal{F}_{\sigma_k(h_2-1)}, (\sigma_k(h_2 - 1), \sigma_k(h_1 + l - 1))$)
      else
        $\tilde{\mathcal{F}}$:=Forward($\tilde{\mathcal{F}}, (\sigma_k(h_1 + l - 2), \sigma_k(h_1 + l - 1))$)
      end if
      Connect($\tilde{\mathcal{F}}, \tilde{\mathcal{B}}_l, (\sigma_k(h_1 + l - 1), \sigma_k(h_2))$) ▷ $p_{\text{opt}}(\sigma_k^{\text{new}})$
    end for
  end for
end procedure

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 $V = \{0, 1, \dots, n\}$ and edge set $E = \{(i, j) \mid i, j \in V, i \neq j\}$, and $M = \{1, 2, \dots, m\}$ 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 (≥ 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 (≥ 0) of k,
4. a distance d_{ij} (≥ 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 σ_k denote the route traveled by vehicle k, where $\sigma_k(h)$ denotes the hth customer in σ_k, and let

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

Note that each customer i is included in exactly one route σ_k, and is visited by vehicle k exactly once. We denote by n_k the number of customers in σ_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_k^{a} be the arrival time of vehicle k at the depot, and let

$$\boldsymbol{s} = (s_1, s_2, \ldots, s_n, s_1^{\mathrm{a}}, s_2^{\mathrm{a}}, \ldots, s_m^{\mathrm{a}}).$$

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 $y_{ik}(\boldsymbol{\sigma}) \in \{0,1\}$ for $i \in V \setminus \{0\}$ and $k \in M$ by

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

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

$$d_{\mathrm{sum}}(\boldsymbol{\sigma}) = \sum_{k \in M} \sum_{h=0}^{n_k} d_{\sigma_k(h), \sigma_k(h+1)}, \tag{3.2.1}$$

$$p_{\mathrm{sum}}(\boldsymbol{s}) = \sum_{i \in V \setminus \{0\}} p_i(s_i) + \sum_{k \in M} p_0(s_k^{\mathrm{a}}), \tag{3.2.2}$$

$$\begin{aligned} q_{\mathrm{sum}}(\boldsymbol{\sigma}, \boldsymbol{s}) &= \sum_{k \in M} \sum_{h=0}^{n_k-1} q_{\sigma_k(h), \sigma_k(h+1)}\big(s_{\sigma_k(h+1)} - s_{\sigma_k(h)}\big) \\ &\quad + \sum_{k \in M} q_{\sigma_k(n_k), 0}\big(s_k^{\mathrm{a}} - s_{\sigma_k(n_k)}\big), \end{aligned} \tag{3.2.3}$$

$$a_{\mathrm{sum}}(\boldsymbol{\sigma}) = \sum_{k \in M} \max \left\{ \sum_{i \in V \setminus \{0\}} a_i y_{ik}(\boldsymbol{\sigma}) - u_k, 0 \right\}. \tag{3.2.4}$$

Then, the problem can be formulated as follows:

$$\begin{array}{lll} \text{minimize} & d_{\mathrm{sum}}(\boldsymbol{\sigma}) + p_{\mathrm{sum}}(\boldsymbol{s}) + a_{\mathrm{sum}}(\boldsymbol{\sigma}) + q_{\mathrm{sum}}(\boldsymbol{\sigma}, \boldsymbol{s}) & \qquad (3.2.5) \\ \text{subject to} & \displaystyle\sum_{k \in M} y_{ik}(\boldsymbol{\sigma}) = 1, \qquad i \in V \setminus \{0\}. & \qquad (3.2.6) \end{array}$$

In this formulation, time window, traveling time and capacity constraints are all treated as soft, and their violation is evaluated as costs $p_{\mathrm{sum}}(\boldsymbol{s})$, $q_{\mathrm{sum}}(\boldsymbol{\sigma}, \boldsymbol{s})$ and $a_{\mathrm{sum}}(\boldsymbol{\sigma})$ in the objective function, respectively.

The standard problem VRPHTW, in which time windows $[t_i^{\mathrm{r}}, t_i^{\mathrm{d}}]$, service times τ_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 $q_{ij}(t)$:

$$p_i(t) = \begin{cases} 0, & t_i^{\mathrm{r}} \le t \le t_i^{\mathrm{d}} \\ +\infty, & \text{otherwise,} \end{cases}$$

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

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 σ_k so that the total of time window and traveling time costs is minimized. (How to determine σ_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., $s_{n_k+1} = s_k^{\mathrm{a}}$ and $p_{n_k+1}(s_{n_k+1}) = p_0(s_k^{\mathrm{a}})$). Then, OSTP is formulated as follows:

$$\begin{aligned} &\text{minimize} && f_{\mathrm{OSTP}}(\boldsymbol{s}) = \sum_{i=1}^{n_k+1} p_i(s_i) + \sum_{i=1}^{n_k+1} q_{i-1,i}(s_i - s_{i-1}) \\ &\text{subject to} && s_0 = 0. \end{aligned} \tag{3.3.7}$$

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

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n'} c_i x_i && (3.3.8) \\ \text{subject to} \quad & \sum_{i=1}^{n'} w_i x_i \le b \\ & x_i \in \{0,1\}, \qquad i = 1,2,\dots,n'. \end{aligned}$$

Note that objective function (3.3.8) is equivalent to

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

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

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

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

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 $\tilde{\boldsymbol{s}} = (\tilde{s}_0, \tilde{s}_1, \dots, \tilde{s}_{n_k+1})$ of start times, corresponding to a feasible solution $\tilde{\boldsymbol{x}}$ of the 0-1 knapsack, by $\tilde{s}_0 = 0$ and the following $\tilde{s}_i$ for $i \ge 1$:

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

Then

$$\tilde{s}_i = \sum_{j \le i} w_j \tilde{x}_j \le b, \qquad i = 1, 2, \dots, n_k + 1 \tag{3.3.13}$$

holds from the feasibility of $\tilde{\boldsymbol{x}}$. The time window cost of $\tilde{\boldsymbol{s}}$ 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 $\tilde{\boldsymbol{s}}$ for the OSTP instance is equal to the objective value of $\tilde{\boldsymbol{x}}$ for the KP instance.

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

$$\hat{x}_i = \begin{cases} 1, & \text{if } \hat{s}_i - \hat{s}_{i-1} \ge w_i \\ 0, & \text{if } \hat{s}_i - \hat{s}_{i-1} < w_i. \end{cases} \tag{3.3.14}$$

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 $\hat{\boldsymbol{s}}$ 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 $\boldsymbol{x} = \boldsymbol{0}$, the above discussion shows that KP is reducible to OSTP. $\square$

3.3.2 Pseudo polynomial time algorithm

We first show that OSTP defined for route σ_k of vehicle k can be solved by using dynamic programming.

> Let $f_h^k(t)$ be the minimum sum of the costs for customers 0, 1, 2, ..., 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 $f_h^k(t)$ can be computed by the following recursive formula

$$\begin{aligned} f_0^k(t) &= \begin{cases} +\infty, & t \neq 0 \\ 0, & t = 0 \end{cases} \\ f_h^k(t) &= p_h(t) + \min_{0 \leq t' \leq t} \{f_{h-1}^k(t') + q_{h-1,h}(t-t')\}, \qquad (3.3.15) \\ &\qquad\qquad 1 \leq h \leq n_k + 1, -\infty < t < +\infty. \end{aligned}$$

The optimal cost for route σ_k is given by $\min_t f_{n_k+1}^k(t)$.

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 $\boldsymbol{s}^* = (s_1^*, s_2^*, \ldots, s_{n_k+1}^*)$ 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

$$\frac{\partial}{\partial s_h^*} f_{\mathrm{OSTP}}(\boldsymbol{s}^*) = \frac{\partial}{\partial s_h^*}\{q_{h-1,h}(s_h^* - s_{h-1}^*) + p_h(s_h^*) + q_{h,h+1}(s_{h+1}^* - s_h^*)\}, \qquad (3.3.16)$$

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+1}^* and s_h^* 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:

$$\begin{aligned} p_{\tilde{h}}(t) &= p_h(t) + p_{h+1}(t + s_{h+1}^* - s_h^*) && (3.3.17)\\ q_{h-1,\tilde{h}}(t) &= q_{h-1,h}(t) && (3.3.18)\\ q_{\tilde{h},h+2}(t) &= q_{h,h+1}(s_{h+1}^* - s_h^*) + q_{h+1,h+2}(t - s_{h+1}^* + s_h^*), && (3.3.19) \end{aligned}$$

where we define $q_{\tilde{h},h+2}(t)$ only if $h \leq n_k - 1$. Then increasing the value of $s_{\tilde{h}}^*$ 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 $s_{n_k+1}^*$

$$\frac{\partial}{\partial s_{n_k+1}^*} f_{\mathrm{OSTP}}(\boldsymbol{s}^*) = \frac{\partial}{\partial s_{n_k+1}^*}\{q_{n_k,n_k+1}(s_{n_k+1}^* - s_{n_k}^*) + p_{n_k+1}(s_{n_k+1}^*)\} \qquad (3.3.20)$$

exists. Hence, by a similar argument, we can change $s_{n_k+1}^*$ until it becomes an integer without increasing the objective value. □

The lemma indicates that traveling times between customers are nonnegative integers, and hence we can compute $f_h^k(t)$ by

$$f_h^k(t) = p_h(t) + \min_{t' \in \{0,1,\ldots,t\}} \{f_{h-1}^k(t') + q_{h-1,h}(t - t')\}, \ t = 0, 1, 2, \ldots, \qquad (3.3.21)$$

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 $f_h^k(t)$, where T is the maximum time that we need to consider. We will discuss the value of T below. With this table, $f_h^k(t)$ can be computed in $O(T)$ time from $f_{h-1}^k(0), f_{h-1}^k(1), \ldots, f_{h-1}^k(t)$. Hence, starting from $f_0^k(0) = 0, f_0^k(1) = f_0^k(2) = \cdots = f_0^k(T) = +\infty$, 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 $f_h^k(t)$ 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 $f_h^k(t)$. 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_h^k

Since functions p_h and $q_{h-1,h}$ are piecewise linear, each f_h^k 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.

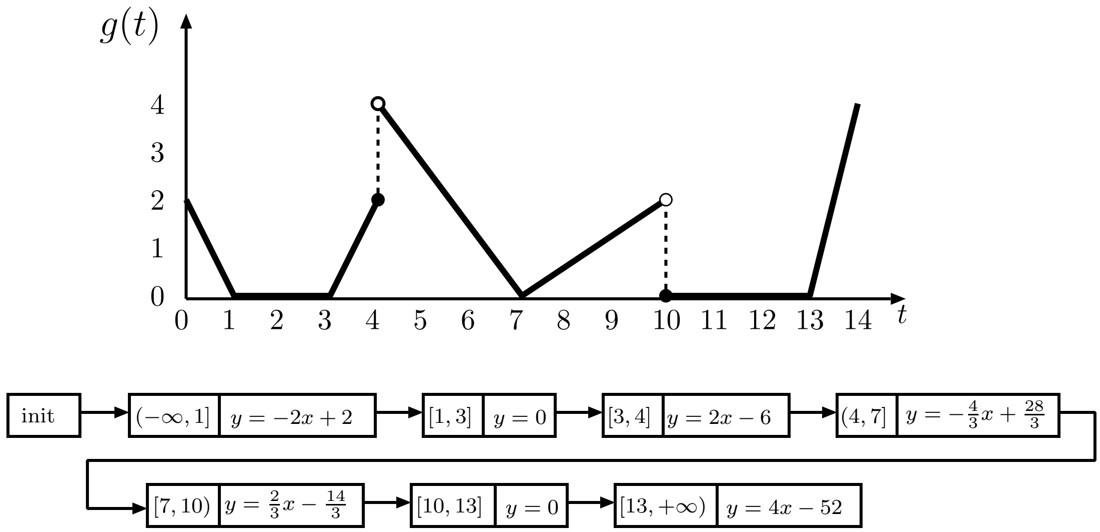


Figure 3.1: A function g and the linked list that represents g

A nontrivial part in recursion (3.3.15) is the computation of $\min_{0 \le t' \le t}\{f^k_{h-1}(t') + q_{h-1,h}(t - t')\}$.[1] 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 $\widehat{\delta}(g)$ be the number of convex intervals of g. Then let S_l, $l = 1, 2, \ldots, \widehat{\delta}(g)$, 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 $\widehat{\delta}(g) = 3$.

We split the entire domain of f^k_{h-1} into convex intervals $S_1, S_2, \ldots, S_K$, where $K = \widehat{\delta}(f^k_{h-1})$, and define the following convex functions F_l for $l = 1, 2, \ldots, K$, which are ex-

[1] Note that this computation is similar to that of Minkowski sum of a convex polygon and a nonconvex polygon [38].

tended from $f^k_{h-1}(t)$ on S_l. Let $\mathrm{Cl}(S_l) = [c^{\mathrm{L}}_l, c^{\mathrm{R}}_l]$ denote the closure of S_l.

$$F_l(t) \quad = \quad \begin{cases} +\infty, & t \notin \mathrm{Cl}(S_l) \\ f^k_{h-1}(t), & t \in S_l \\ \lim_{\varepsilon \to +0} f^k_{h-1}(t+\varepsilon), & t = c^{\mathrm{L}}_l \\ \lim_{\varepsilon \to +0} f^k_{h-1}(t-\varepsilon), & t = c^{\mathrm{R}}_l. \end{cases}$$

Let

$$e_l(t) = \min_{0 \le t' \le t} \{F_l(t') + q_{h-1,h}(t-t')\}. \tag{3.3.23}$$

Then we have

$$\begin{aligned} \min_{0 \le t' \le t} \{f^k_{h-1}(t') + q_{h-1,h}(t-t')\} &= \min_{0 \le t' \le t} \left\{ \min_{1 \le l \le K} \{F_l(t') + q_{h-1,h}(t-t')\} \right\} \\ &= \min_{1 \le l \le K} \left\{ \min_{0 \le t' \le t} \{F_l(t') + q_{h-1,h}(t-t')\} \right\} \\ &= \min_{1 \le l \le K} e_l(t). \end{aligned}$$

That is, $\min_{0 \le t' \le t}\{f^k_{h-1}(t') + q_{h-1,h}(t-t')\}$ 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 ϕ_1 and ϕ_2 are stored in linked lists. Then we can compute function*

$$\phi(t) = \min_{0 \le t' \le t} \{\phi_1(t') + \phi_2(t-t')\}$$

in $O(\delta(\phi))$ time, where $\delta(\phi) \le \delta(\phi_1) + \delta(\phi_2)$ holds. Moreover ϕ 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 ϕ_1 and ϕ_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.

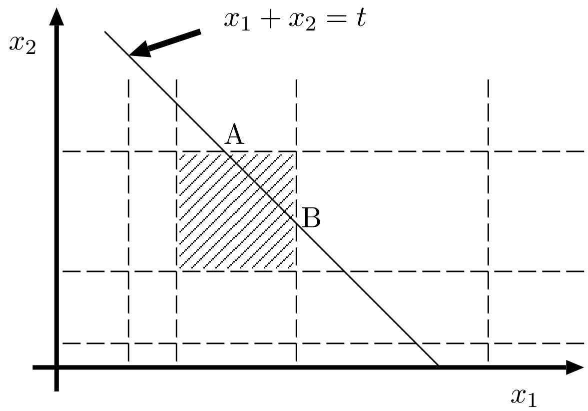


Figure 3.2: Breakpoints of $\phi_1(x_1)$ and $\phi_2(x_2)$ on the plane of x_1 and x_2

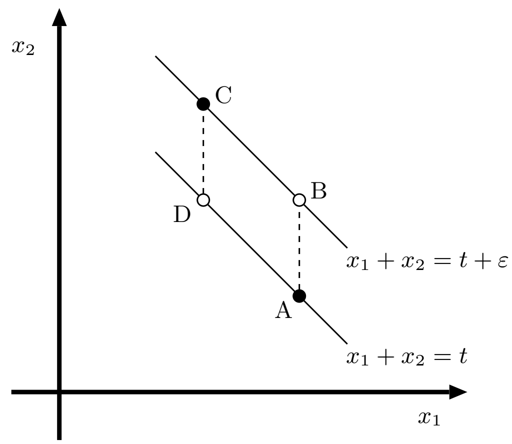


Figure 3.3: The points that achieves $\phi(t)$ and $\phi(t + \varepsilon)$

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 ε 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 ϕ_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 $t = 0 \to +\infty$ 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.

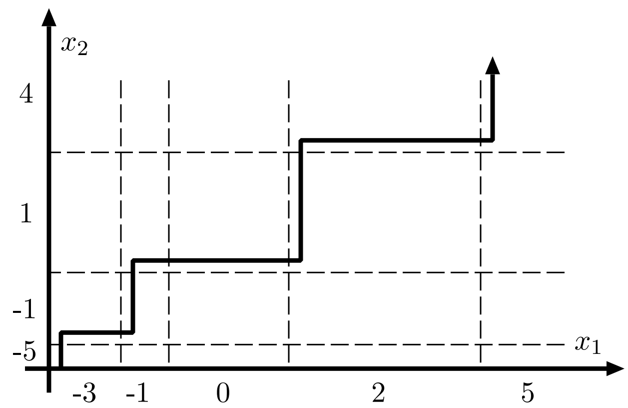


Figure 3.4: An example of the trajectory of $(x_1^*(t), x_2^*(t))$ which achieves $\phi(t)$

In order to compute ϕ, 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 ϕ_1 and ϕ_2. Whenever we determine the direction at each intersection, we compute the data of ϕ for the corresponding interval, and add it to the linked list that represents $\phi(t)$. Note that the gradient of ϕ is the same as that of the selected piece of ϕ_i. As it is not difficult to show that the gradients of linear pieces of ϕ added to the list are nondecreasing, the computed ϕ 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. $\Box$

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 $(x_1^*(t), x_2^*(t))$ 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 $t_{h-1,h}$. 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 $t_l^* = \arg\min_{0\le t'\le t}\{F_l(t') + q_{h-1,h}(t-t')\}$, i.e., $e_l(t) = F_l(t_l^*)+q_{h-1,h}(t-t_l^*)$, and $t_{l'}^* = \arg\min_{0\le t'\le t}\{F_{l'}(t')+q_{h-1,h}(t-t')\}$. Then $t - t_l^* \ge t - t_{l'}^*$ holds, since $t_l^* \in \mathrm{Cl}(S_l)$ and $t_{l'}^* \in \mathrm{Cl}(S_{l'})$. If $t_l^* = t_{l'}^*$ the lemma

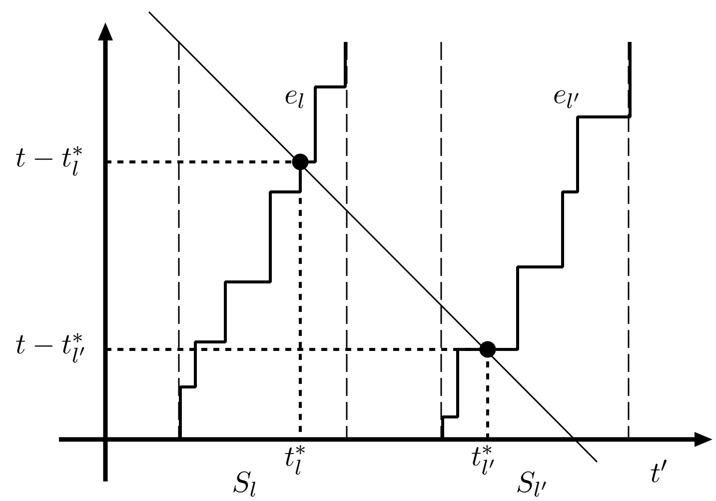


Figure 3.5: e_l and $e_{l'}$

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 $q_{h-1,h}(t - t_{l'}^*)$ ($= q_{h-1,h}(t-t_l^*)$). 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 ε 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* ($l \le L$), 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 illustrates 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

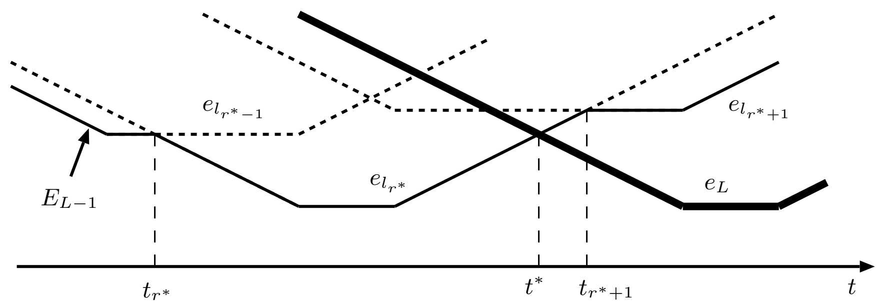


Figure 3.6: An example of the lower envelope

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 and $e_{l_{r^*}}$ cross 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 $e_{l_r}(t)$ 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 $e_{l_{r^*}}$ cross, a linear piece of $e_{l_{r^*}}$ 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_h^k from f_{h-1}^k by recursion (3.3.15), we execute the following three steps:

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

2. Compute their lower envelope $E_{\widehat{\delta}(f_{h-1}^k)}(t)$, which gives $\min_{t'}\{f_{h-1}^k(t') + q_{h-1,h}(t-t')\}$ in recursion (3.3.15).

3. Compute $p_h(t) + E_{\widehat{\delta}(f_{h-1}^k)}(t)$, i.e., $f_h^k(t)$.

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 $O(\sum_{l=1}^{\widehat{\delta}(f_{h-1}^k)} \delta(e_l))$ time to compute all e_1, e_2, $\ldots$, $e_{\widehat{\delta}(f_{h-1}^k)}$.

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

3. We can add p_h to the lower envelope $E_{\widehat{\delta}(f_{h-1}^k)}$ in $O(\delta(p_h) + \sum_{l=1}^{\widehat{\delta}(f_{h-1}^k)} \delta(e_l))$ time, since the lower envelope $E_{\widehat{\delta}(f_{h-1}^k)}$ has at most $\sum_{l=1}^{\widehat{\delta}(f_{h-1}^k)} \delta(e_l)$ linear pieces.

Hence, we can compute f_h^k from f_{h-1}^k in $O(\delta(p_h) + \sum_{l=1}^{\widehat{\delta}(f_{h-1}^k)} \delta(e_l))$ time. For convenience, we introduce the following notations,

$$\Delta_p^k = \textstyle\sum_{h=1}^{n_k+1} \delta(p_h), \quad \widehat{\Delta}_p^k = \sum_{h=1}^{n_k+1} \widehat{\delta}(p_h) \quad \text{and} \quad \Delta_q^k = \sum_{h=1}^{n_k+1} \delta(q_{h-1,h}).$$

For the number of convex intervals,

$$\widehat{\delta}(f_h^k) \le \widehat{\delta}(f_{h-1}^k) + \widehat{\delta}(p_h) - 1 \tag{3.3.25}$$

holds, because the number of convex intervals of the lower envelope $E_{\widehat{\delta}(f_{h-1}^k)}$ is less than or equal to $\widehat{\delta}(f_{h-1}^k)$. For the number of linear pieces of f_h^k,

$$\begin{aligned} \delta(f_h^k) &\le \delta(p_h) + \sum_{l=1}^{\widehat{\delta}(f_{h-1}^k)} \delta(e_l) \\ &\le \delta(p_h) + \sum_{l=1}^{\widehat{\delta}(f_{h-1}^k)} \{\delta(F_l) + \delta(q_{h-1,h})\} \\ &\le \delta(p_h) + \delta(f_{h-1}^k) + \widehat{\delta}(f_{h-1}^k)\delta(q_{h-1,h}) \end{aligned} \tag{3.3.26}$$

holds. Note that the first inequality holds because $\delta(E_{\widehat{\delta}(f_{h-1}^k)}) \le \sum_{l=1}^{\widehat{\delta}(f_{h-1}^k)} \delta(e_l)$, and the second inequality is from Theorem 3.3.3. By applying (3.3.25) recursively, we have $\widehat{\delta}(f_h^k) = O(\widehat{\Delta}_p^k)$. Similarly, from (3.3.26), we have $\delta(f_h^k) = O(\Delta_p^k + \widehat{\Delta}_p^k \Delta_q^k)$. Consequently, the time to compute f_h^k from f_{h-1}^k is

$$\begin{aligned} O\left(\delta(p_h) + \sum_{l=1}^{\widehat{\delta}(f_{h-1}^k)} \delta(e_l)\right) &= O\left(\delta(p_h) + \delta(f_{h-1}^k) + \widehat{\delta}(f_{h-1}^k)\delta(q_{h-1,h})\right) \\ &= O(\Delta_p^k + \widehat{\Delta}_p^k \Delta_q^k). \end{aligned}$$

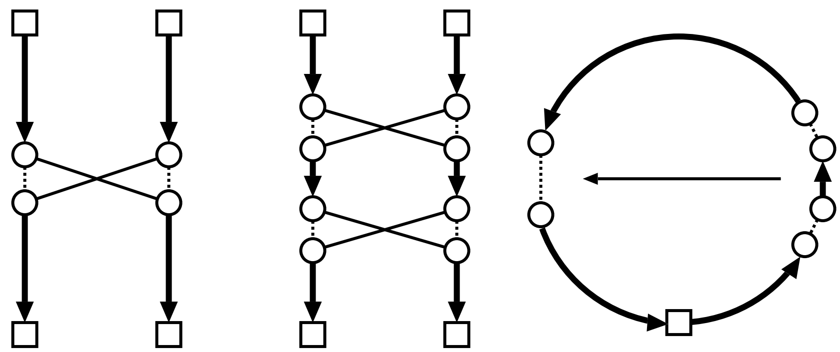
Then the time complexity of computing all $f_1^k, f_2^k, \ldots, f_{n_k+1}^k$ is $O\left(n_k(\Delta_p^k + \widehat{\Delta}_p^k \Delta_q^k)\right)$. Since all linear pieces of input functions are explicitly given as linked lists, the input size is $\Delta_p^k + \Delta_q^k$. Thus the above time complexity is polynomial in the input size.

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

$$\Delta(\sigma_k) = \sum_{h=1}^{n_k+1} \delta(p_{\sigma_k(h)}) + \left(\sum_{h=1}^{n_k+1} \widehat{\delta}(p_{\sigma_k(h)})\right)\left(\sum_{h=1}^{n_k+1} \delta(q_{\sigma_k(h-1),\sigma_k(h)})\right). \tag{3.3.27}$$

3.4 Local search for finding visiting orders $\boldsymbol{\sigma}$

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



(a) 2-opt* (b) Cross exchange (c) Or-opt

Figure 3.7: Neighborhoods in our local search

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^{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 $L^{\text{intra}}_{\text{path}}$ (a parameter) and inserts it into another position of the same route, where the position is limited within length $L^{\text{intra}}_{\text{ins}}$ (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^{cross} (i.e., $L^{\text{cross}} =$

n). ILS is summarized as follows:

Algorithm: Iterated Local Search (ILS)

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

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

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

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

3.5 Efficient implementation of local search

A solution $\boldsymbol{\sigma}$ is evaluated by $d_{\text{sum}}(\boldsymbol{\sigma}) + a_{\text{sum}}(\boldsymbol{\sigma}) + (p+q)^*_{\text{sum}}(\boldsymbol{\sigma})$, where $(p+q)^*_{\text{sum}}(\boldsymbol{\sigma})$ denotes the minimum time window and traveling time cost for $\boldsymbol{\sigma}$. For this, it is important to see that dynamic programming computation of $(p+q)^*_{\text{sum}}(\boldsymbol{\sigma})$ 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 $\sigma_k(0) \to \sigma_k(1) \to \cdots \to \sigma_k(h)$ and latter part $\sigma_k(h+1) \to \sigma_k(h+2) \to \cdots \to \sigma_k(n_k+1)$ for some h.

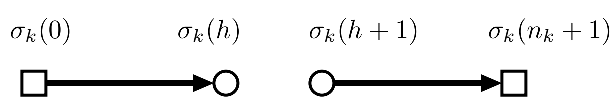


Figure 3.8: The former and latter parts of a route σ_k

In this scheme, the distance of route σ_k is computed in $O(1)$ time, from distances of its former and latter parts. The amount of capacity excess on route σ_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

$\sum_{i=1}^{h} d_{\sigma_k(i-1),\sigma_k(i)}$, $\sum_{i=h+1}^{n_k+1} d_{\sigma_k(i-1),\sigma_k(i)}$, $\sum_{i=1}^{h} a_{\sigma_k(i)}$ and $\sum_{i=h}^{n_k} a_{\sigma_k(i)}$ for each customer $\sigma_k(h)$ and vehicle k whenever the current route is updated.

Now we concentrate on the computation of the minimum cost $(p+q)^*_{\mathrm{sum}}(\sigma_k)$, which is the sum of time window and traveling time costs on route σ_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., $\min_{s_{\sigma_k(h)}=t} \sum_{i=h}^{n_k+1} p_{\sigma_k(i)}(s_{\sigma_k(i)}) + \sum_{i=h+1}^{n_k+1} q_{\sigma_k(i-1),\sigma_k(i)}(s_{\sigma_k(i)} - s_{\sigma_k(i-1)})$). We call this a *backward minimum cost function*. Let $f^k_h(t)$ be the forward minimum cost function at the hth customer in route σ_k, which was discussed in Section 3.3. Then, $b^k_h(t)$ can be computed as follows in a symmetric manner:

$$
\begin{aligned}
b^k_{n_k+1}(t) &= p_0(t) \\
b^k_h(t) &= p^k_h(t) + \min_{t'} \left(b^k_{h+1}(t') + q_{h,h+1}(t'-t) \right), \qquad 1 \le h \le n_k.
\end{aligned}
\tag{3.5.28}
$$

We can then obtain the optimal cost of route σ_k by

$$
(p+q)^*_{\mathrm{sum}}(\sigma_k) = \min_{t} \left(f^k_h(t) + \min_{t'} (b^k_{h+1}(t') + q_{h,h+1}(t'-t)) \right) \tag{3.5.29}
$$

for any h ($0 \le h \le n_k$). 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 σ_k in $O(\Delta(\sigma_k))$ time, if we keep the data

$$
\sum_{i=1}^{h} a_{\sigma_k(i)} \quad \text{and} \quad \sum_{i=h}^{n_k} a_{\sigma_k(i)}, \tag{3.5.30}
$$

$$
\sum_{i=1}^{h} d_{\sigma_k(i-1),\sigma_k(i)} \quad \text{and} \quad \sum_{i=h+1}^{n_k+1} d_{\sigma_k(i-1),\sigma_k(i)}, \tag{3.5.31}
$$

$$
f^k_h(t) \quad \text{and} \quad b^k_h(t) \tag{3.5.32}
$$

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.

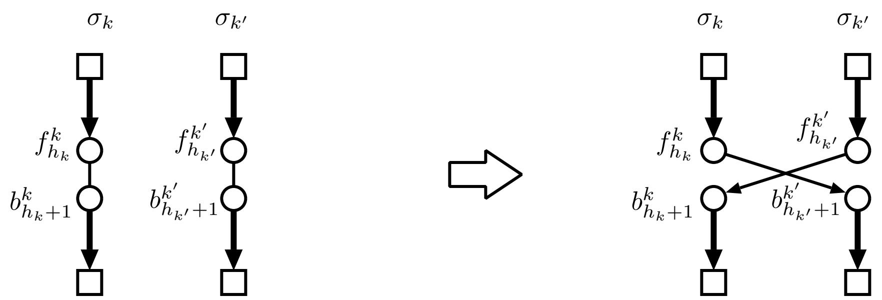


Figure 3.9: An example of a 2-opt* operation

In Figure 3.9, an example of a 2-opt* operation on routes σ_k and $\sigma_{k'}$ is shown. The sum of time window and traveling time costs for σ_k, after a 2-opt* operation is applied, can be computed by

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

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 σ_k and $\sigma_{k'}$.

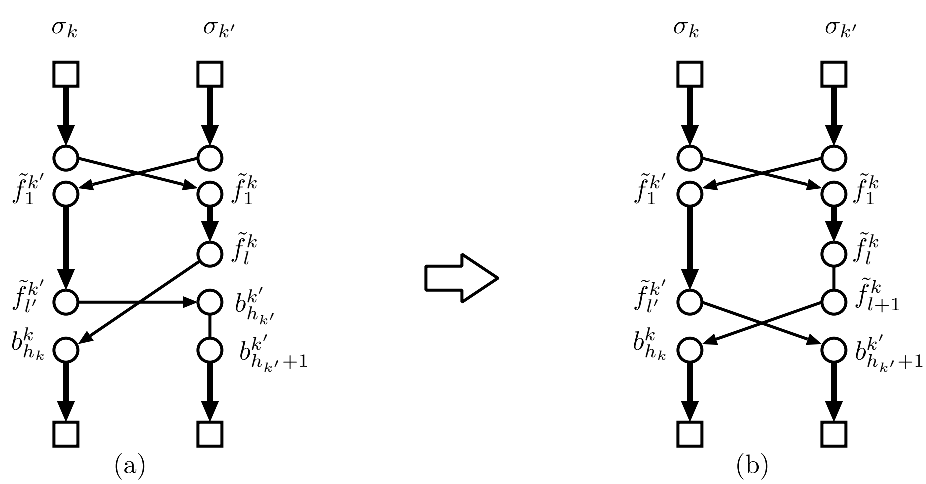


Figure 3.10: An example of the search order in the cross exchange neighborhood

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 σ_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 3.10 explains the situation. In Figure 3.10 (a), backward minimum cost functions $b^k_{h_k}$, $b^k_{h_{k'}}$ and $b^k_{h_{k'}+1}$of the current solution are available, and we have already computed the forward minimum cost functions $\tilde{f}^k_1, \tilde{f}^k_2, \ldots, \tilde{f}^k_l$ and $\tilde{f}^{k'}_1, \tilde{f}^{k'}_2, \ldots, \tilde{f}^{k'}_{l'}$, which we have temporarily computed to evaluate $(p+q)^*_{\mathrm{sum}}(\sigma_k) + (p+q)^*_{\mathrm{sum}}(\sigma_{k'})$ of Figure 3.10 (a). (We can obtain $(p+q)^*_{\mathrm{sum}}(\sigma_k)$ (resp., $(p+q)^*_{\mathrm{sum}}(\sigma_{k'})$) from $\tilde{f}^k_l$ and $b^k_{h_k}$ (resp., from $\tilde{f}^{k'}_{l'}$ and $b^{k'}_{h_{k'}}$)) We then extend the length of the right path by one (Figure 3.10 (b)). For this, we can compute $\tilde{f}^k_{l+1}$ from $\tilde{f}^k_l$ by recursion of the dynamic programming in $O(\Delta(\sigma_k))$ time, and evaluate $(p+q)^*_{\mathrm{sum}}(\sigma_k)+(p+q)^*_{\mathrm{sum}}(\sigma_{k'})$ in $O\left(\Delta(\sigma_k) + \Delta(\sigma_{k'})\right)$ time. Thus, the change in the cost after a cross exchange operation (from the current solution to the solution in Figure 3.10 (b)) is obtained in $O\left(\Delta(\sigma_k) + \Delta(\sigma_{k'})\right)$ time.

Similarly, the change in the cost for an intra-route operation of route σ_k can be computed in $O\left(\Delta(\sigma_k)\right)$ time, by searching solutions in a specific order. Actually, this case is slightly more complicated than the case of cross exchange neighborhood. For details, see Section 2.6.

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 $[t^{\mathrm{r}}_i, t^{\mathrm{d}}_i]$, an amount of requirement a_i and a service time τ_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 $(m, d_{\mathrm{sum}}(\boldsymbol{\sigma}))$ 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

June 2, 2004 from http://www.sintef.no/static/am/opti/projects/top/vrp/bknown.html).

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:

$$
\begin{aligned}
p_i(t) &= \begin{cases} \alpha_1(t_i^{\mathrm{r}} - t), & t < t_i^{\mathrm{r}} \\ 0, & t_i^{\mathrm{r}} \le t \le t_i^{\mathrm{d}} \\ \alpha_1(t - t_i^{\mathrm{d}}), & t_i^{\mathrm{d}} < t, \end{cases} \\
q_{ij}(t) &= \begin{cases} +\infty, & t < 0.9(\tau_i + t_{ij}) \\ \alpha_2(\tau_i + t_{ij} - t), & 0.9(\tau_i + t_{ij}) \le t < \tau_i + t_{ij} \\ 0, & \tau_i + t_{ij} \le t, \end{cases}
\end{aligned} \tag{3.6.33}
$$

where α_1 and α_2 are positive parameters. For other parameters, we used $L^{\mathrm{cross}} = 3$, $L^{\mathrm{intra}}_{\mathrm{path}} = 3$ and $L^{\mathrm{intra}}_{\mathrm{ins}} = 20$, 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., $P = p_{\mathrm{sum}}(\boldsymbol{s})/\alpha_1$), and column "$Q$" denotes the total amount of shortened traveling time (i.e., $Q = q_{\mathrm{sum}}(\boldsymbol{\sigma}, \boldsymbol{s})/\alpha_2$). 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_{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 α_1 and α_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., $p_{\mathrm{sum}}(\boldsymbol{s}) = q_{\mathrm{sum}}(\boldsymbol{\sigma}, \boldsymbol{s}) = a_{\mathrm{sum}}(\boldsymbol{\sigma}) = 0$), 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 "$P_{\max}$" denotes the maximum deviation of start time of services from time windows (i.e., $P_{\max} = \max\{p_i(s_i)/\alpha_1 \mid i \in V\}$).

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_{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

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

Table 3.2: Computational results on Solomon's instances

instance	d_{sum}	P	Q	feasible	instance	d_{sum}	P	Q	feasible
R101	*1616.54	0.57(2)	0.12(1)	1695.80	R201	*1252.37	0	0	*1252.37
R102	*1422.78	1.73(2)	0.54(2)	–	R202	*1183.53	1.11(1)	0	1195.31
R103	*1174.57	3.23(2)	1.42(1)	–	R203	949.80	0.33(1)	0.01(1)	953.89
R104	1018.67	0	0.08(1)	1019.55	R204	847.87	0	0	847.87
R105	*1372.18	0	0.10(1)	*1377.11	R205	1009.83	0	0	1009.83
R106	1257.96	0	0	1257.96	R206	935.90	0	0	935.90
R107	1122.82	0	0.14(1)	1125.62	R207	915.60	0	0	915.60
R108	967.05	0	0.34(1)	989.05	R208	749.56	0	0	749.56
R109	1197.42	0	0	1197.42	R209	945.70	0	0	945.70
R110	1142.81	0	0.58(1)	1150.28	R210	961.10	0	0	961.10
R111	1096.73	0	0	1096.73	R211	934.27	0	0	934.27
R112	986.41	0	0	986.41					
C101	*828.94	0	0	*828.94	C201	*591.56	0	0	*591.56
C102	*828.94	0	0	*828.94	C202	*591.56	0	0	*591.56
C103	*828.06	0	0	*828.06	C203	*591.17	0	0	*591.17
C104	*824.78	0	0	*824.78	C204	601.18	0	0	601.18
C105	*828.94	0	0	*828.94	C205	*588.88	0	0	*588.88
C106	*828.94	0	0	*828.94	C206	*588.49	0	0	*588.49
C107	*828.94	0	0	*828.94	C207	*588.29	0	0	*588.29
C108	*828.94	0	0	*828.94	C208	*588.32	0	0	*588.32
C109	*828.94	0	0	*828.94					
RC101	*1629.99	0.25(1)	5.70(4)	–	RC201	1414.59	0.06(1)	0.77(1)	1424.65
RC102	*1442.53	8.39(1)	4.93(6)	–	RC202	*1321.07	0.92(2)	0	1397.45
RC103	*1261.67	0	0	*1261.67	RC203	1058.80	0.01(1)	0	1061.98
RC104	1160.60	0	0	1160.60	RC204	825.24	0	0	825.24
RC105	*1506.65	0	4.21(3)	–	RC205	1297.65	0	0	1297.65
RC106	*1382.03	0	1.87(2)	–	RC206	*1146.30	0.10(1)	0	1155.33
RC107	*1212.48	0	0.76(1)	1232.20	RC207	1065.74	0	0.47(1)	1071.43
RC108	*1133.81	0	0.42(2)	*1139.82	RC208	862.46	0	0	862.46

Table 3.3: Computational results with smaller number of vehicles than the best known solutions

instance	$d_{\rm sum}$	P	Q	$a_{\rm sum}$	$P_{\max}$
R101	*1636.28	0.30(1)	0	0	0.30
R102	*1473.77	0	1.65(2)	0	0
R103	*1268.77	2.82(1)	1.42(1)	0	2.82
R104	*988.16	38.05(14)	145.27(85)	1	11.23
R105	1495.92	0	5.90(4)	0	0
R106	1360.94	4.39(1)	0	0	4.39
R107	*1098.82	10.23(4)	41.42(30)	0	7.03
R108	*937.96	8.74(6)	99.97(61)	0	4.66
R109	1285.81	0	39.02(25)	0	0
R110	*1105.73	5.85(4)	68.30(42)	0	1.86
R111	1134.38	12.11(7)	77.00(50)	0	6.17
R112	*948.94	11.91(7)	118.87(70)	3	7.65
C101	1037.42	822.23(46)	685.48(83)	70	103.66
C102	1146.93	0	0	10	0
C103	967.44	0	0	10	0
C104	912.23	0	0	10	0
C105	1019.59	130.44(16)	459.37(58)	80	22.23
C106	1150.63	62.28(6)	366.03(45)	40	20.44
C107	968.71	21.02(3)	125.09(23)	30	9.20
C108	1112.67	2.40(1)	70.12(15)	30	2.40
C109	954.78	0	0	10	0
RC101	1682.87	3.50(3)	15.64(10)	0	2.68
RC102	*1497.87	8.39(1)	11.80(9)	0	8.39
RC103	1347.96	0	2.21(1)	0	0
RC104	1150.75	0.06(1)	16.18(12)	12	0.06
RC105	*1625.64	0	7.80(5)	0	0
RC106	*1350.07	21.29(7)	57.44(32)	1	11.44
RC107	1330.42	0	3.28(4)	0	0
RC108	1153.31	0	29.66(22)	19	0

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 (≥ 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 (≥ 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

$t < 0$ 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

$$\begin{aligned} & t + \lambda_{ij}(t) = t' + \lambda_{ij}(t') \\ & \Rightarrow t + \lambda_{ij}(t) = \alpha t + (1-\alpha)t' + \lambda_{ij}(\alpha t + (1-\alpha)t'), \quad 0 \le \alpha \le 1 \end{aligned} \tag{4.2.1}$$

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

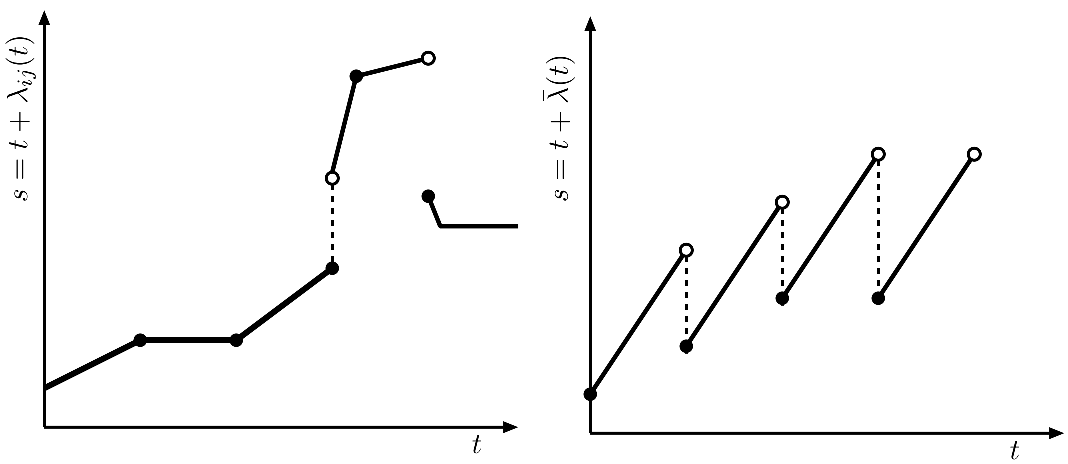


Figure 4.1: An example of λ_{ij} which satisfies condition (4.2.1), and a function $\bar{\lambda}$ which does not satisfy condition (4.2.1)

in [87] (i.e., $t \le t' \Rightarrow t + \lambda_{ij}(t) \le t' + \lambda_{ij}(t')$) 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 σ_k denote the route traveled by vehicle k, where $\sigma_k(h)$ denotes the hth customer in σ_k, and let

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

Note that each customer i is included in exactly one route σ_k, and is visited by vehicle k exactly once. We denote by n_k the number of customers in σ_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

$$\boldsymbol{s} = (s_1, s_2, \ldots, s_n, s_1^{\mathrm{a}}, s_2^{\mathrm{a}}, \ldots, s_m^{\mathrm{a}}).$$

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^{l} be the departure time of vehicle k from the depot, and let

$$\boldsymbol{t} = (t_1, t_2, \ldots, t_n, t_1^{\mathrm{l}}, t_2^{\mathrm{l}}, \ldots, t_m^{\mathrm{l}}).$$

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

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

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

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

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

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

$$\begin{array}{lllr}
\text{minimize} & cost(\boldsymbol{\sigma}, \boldsymbol{s}, \boldsymbol{t}) = p_{\mathrm{sum}}(\boldsymbol{s}) + q_{\mathrm{sum}}(\boldsymbol{\sigma}, \boldsymbol{t}) + a_{\mathrm{sum}}(\boldsymbol{\sigma}) & & (4.2.2) \\
\text{subject to} & \displaystyle\sum_{k \in M} y_{ik}(\boldsymbol{\sigma}) = 1, & i \in V \setminus \{0\} & (4.2.3) \\
& s_i \le t_i, & i \in V \setminus \{0\} & (4.2.4) \\
& t_k^{\mathrm{l}} + \lambda_{0,\sigma_k(1)}(t_k^{\mathrm{l}}) \le s_{\sigma_k(1)}, & k \in M & (4.2.5) \\
& t_{\sigma_k(i)} + \lambda_{\sigma_k(i),\sigma_k(i+1)}(t_{\sigma_k(i)}) \le s_{\sigma_k(i+1)}, & & \\
& & 1 \le i \le n_k - 1,\ k \in M & (4.2.6) \\
& t_{\sigma_k(n_k)} + \lambda_{\sigma_k(n_k),0}(t_{\sigma_k(n_k)}) \le s_k^{\mathrm{a}}, & k \in M & (4.2.7) \\
& y_{ik}(\boldsymbol{\sigma}) \in \{0, 1\}, & i \in V \setminus \{0\},\ k \in M. & (4.2.8)
\end{array}$$

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 $p_{\mathrm{sum}}(\boldsymbol{s})$ and $a_{\mathrm{sum}}(\boldsymbol{\sigma})$ in the objective function. Note that, for any solution with a finite cost, $t_k^{\mathrm{l}} \geq 0$ holds because of the assumptions, and hence $\boldsymbol{s}, \boldsymbol{t} \geq 0$ 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 VRPSTW 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 $\boldsymbol{\sigma}$ are fixed. Hence our algorithm searches $\boldsymbol{\sigma}$ by local search and solve the corresponding m scheduling problems for each $\boldsymbol{\sigma}$ generated during the search. In Section 4.3, we discuss this scheduling problem. How to search σ_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 σ_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., $t_0 = t_k^{\mathrm{l}}$ and $q_{0,1}(t_0) = q_{0,1}(t_k^{\mathrm{l}})$), and let customer $n_k + 1$ represent the arrival at the depot (i.e., $s_{n_k+1} = s_k^{\mathrm{a}}$ and $p_{n_k+1}(s_{n_k+1}) = p_0(s_k^{\mathrm{a}})$).

Then, the TOSTP is described as follows:

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{h=1}^{n_k+1} p_h(s_h) + \sum_{h=0}^{n_k} q_{h,h+1}(t_h) & \\ \text{subject to} & s_h \leq t_h, & 1 \leq h \leq n_k \\ & t_h + \lambda_{h,h+1}(t_h) \leq s_{h+1}, & 0 \leq h \leq n_k. \end{array}$$

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, ..., 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:

$$
\begin{aligned}
f_0(t) &= \begin{cases} +\infty, & t < 0 \\ 0, & t \geq 0 \end{cases} \\
f_h(t) &= \min_{s_h \leq t} \left\{ p_h(s_h) + \min_{t_{h-1}:\ t_{h-1}+\lambda_{h-1,h}(t_{h-1}) \leq s_h} \{ f_{h-1}(t_{h-1}) + q_{h-1,h}(t_{h-1}) \} \right\}, \\
&\qquad\qquad 1 \leq h \leq n_k + 1, -\infty < t < +\infty. \qquad (4.3.9)
\end{aligned}
$$

The optimal cost of the TOSTP for a route σ_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 ϕ 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 $\delta(g^{-1}) = \delta(g)$.

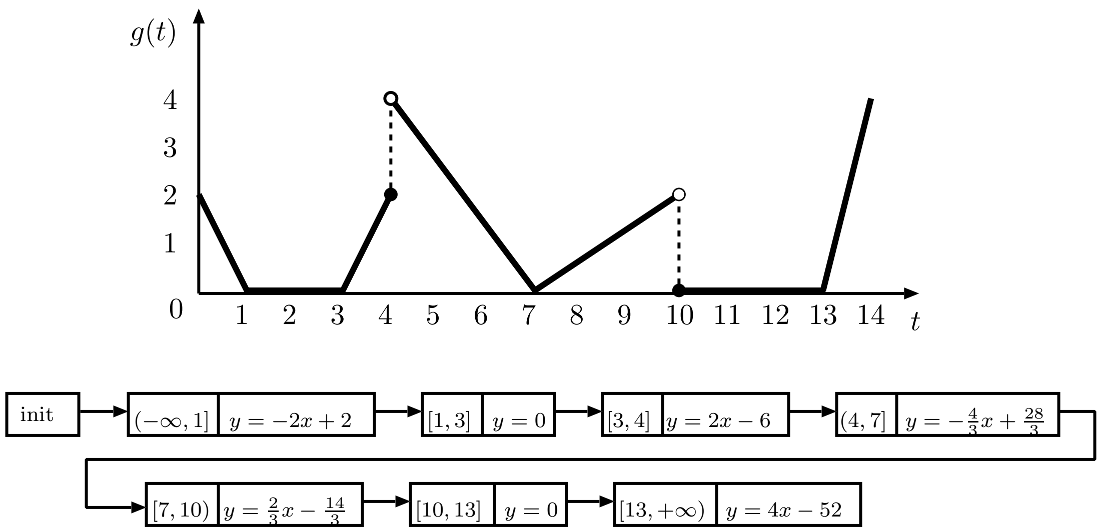


Figure 4.2: A function g and the linked list that represents g

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

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

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

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

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

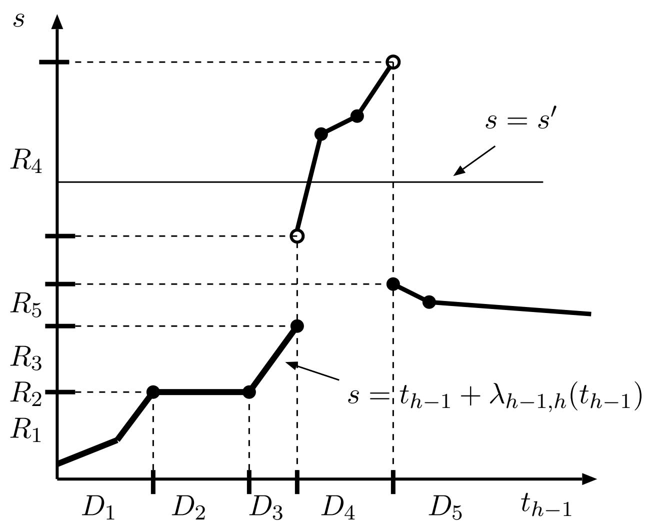


Figure 4.3: The relationship between the departure time t_{h-1} and the arriving time $s = t_{h-1} + \lambda_{h-1,h}(t_{h-1})$

$\gamma_h(s)$ partially for each domain D_l except for $s \in \cup_{l \neq l'} R_l \cap R_{l'}$. Then $\gamma_h(s)$ is completed by merging them and taking the minimums for $s \in \cup_{l \neq l'} R_l \cap R_{l'}$. Note that if $s \notin \cup_{l=1}^{L} R_l$, 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 λ_{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 λ_{ij} is given as a part of input.

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

i. Compute $\gamma_h|_{R_l}$ 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 $\gamma_h|_{R_l}$ for $l = 1, 2, \ldots, L$ and add linear pieces for the intervals with $s \notin \cup_{l=1}^{L} R_l$.

For computing each $\gamma_h|_{R_l}$ 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 γ_h by merging $\gamma_h|_{R_l}$ 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

$$\begin{aligned}
&O\left(\delta(f_{h-1}) + \delta(q_{h-1,h}) + \delta(\lambda_{h-1,h})\right) + O\left(\delta(p_h) + \delta(\gamma_h)\right)) \\
&= O\left(\delta(f_{h-1}) + \delta(p_h) + \delta(q_{h-1,h}) + \delta(\lambda_{h-1,h})\right)
\end{aligned}$$

time and we have

$$\begin{aligned}
\delta(f_h) &\leq \delta(p_h) + \delta(\gamma_h) \\
&\leq \delta(f_{h-1}) + \delta(p_h) + \delta(q_{h-1,h}) + \delta(\lambda_{h-1,h}).
\end{aligned}$$

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 σ_k, we can compute the forward minimum cost function of a customer from that of the previous customer in $O\left(\Delta(\sigma_k)\right)$ 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 σ_k in $O(n_k\Delta(\sigma_k))$ time by computing the forward minimum cost functions of n_k customers in σ_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)

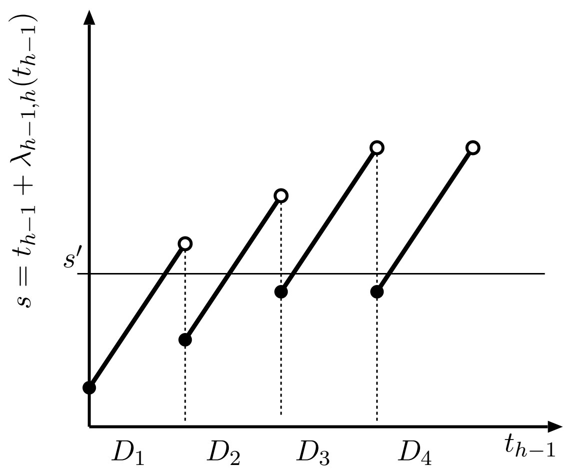


Figure 4.4: An example of λ_{ij} which does not satisfy condition (4.2.1)

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

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

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 γ_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 α 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 γ_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

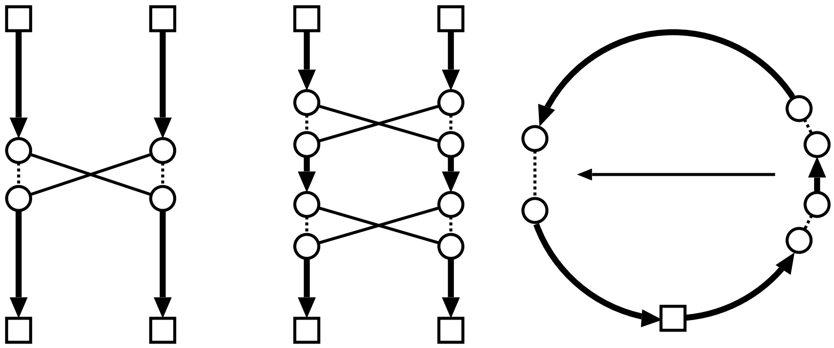
The TOSTP without time-dependency (i.e., $q_{h,h+1}(t)$ and $\lambda_{h,h+1}(t)$ are constant functions) has been intensively studied in the literature especially when the time window cost functions $p_h(s)$ are convex. Below is a brief summary of such results.

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 neighborhood was proposed. In Desrosiers et al. [42], the time window cost for each customer is linear in the time window and $+\infty$ otherwise, and an $O(n_k)$ time algorithm was presented. In Davis and Kanet [37], Koskosidis, Powell and Solomon [97], Tamaki, Komori and Abe [144], Tamaki, Sugimoto and Araki [145], the time window cost is linear for both of earliness and tardiness, and an $O(n_k^2)$ time algorithm was proposed in Davis and Kanet [37] and Tamaki, Sugimoto and Araki [145]. If the time window cost function for each customer is the absolute deviation from a specified time, this problem becomes the *isotonic median regression* problem, which has been extensively studied. To our knowledge, the best time complexity for this problem is $O(n_k \log n_k)$ (Ahuja and Orlin [7], Garey, Tarjan and Wilfong [54], Hochbaum and Queyranne [77]). In Ibaraki et al. [86], the time window cost function for each customer is a piecewise linear convex function. They proposed an $O\left(\Delta(\sigma_k)\log(\Delta(\sigma_k))\right)$ time algorithm to solve the problem from scratch, and an $O\left(\log(\max_k \Delta(\sigma_k))\right)$ amortized time algorithm to compute the optimal cost of a solution in the neighborhood. In Dumas, Soumis and Desrosiers [44], general convex time window cost functions were considered for the VRPHTW, and they proposed an algorithm whose time complexity is of the order of n_k basic operations on the functions called unidimensional minimizations. Very fast algorithms for general convex functions are also known (Ahuja and Orlin [7], Hochbaum and Queyranne [77]).

For the case without time-dependency and with non-convex time window costs, Ibaraki et al. [85] proposed an $O(n_k\Delta(\sigma_k))$ time algorithm to solve the problem from scratch, and an $O(\max_k \Delta(\sigma_k))$ amortized time algorithm to compute the optimal cost of a solution in the neighborhood. Sexton and Bodin [136] considered an TOSTP for the pickup and delivery problem and proposed a linear-time algorithm. In their formulation of TOSTP, a linear cost function on the duration between the pickup and delivery of each request is also considered, while the cost for each customer is linear for earliness and $+\infty$ for tardiness, and time-dependency is not considered.

4.4 Local search for finding visiting orders $\boldsymbol{\sigma}$

In this section, we describe the framework of our local search (LS) for finding good visiting orders $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ that satisfy condition (4.2.3). It starts from an initial solution $\boldsymbol{\sigma}$ and repeats replacing $\boldsymbol{\sigma}$ with a better solution in its neighborhood $N(\boldsymbol{\sigma})$ until no better solution is found in $N(\boldsymbol{\sigma})$. As $N(\boldsymbol{\sigma})$ 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* (b) Cross exchange (c) Or-opt

Figure 4.5: Neighborhoods in our local search

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^{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 $L^{\text{intra}}_{\text{path}}$ (a parameter) and inserts it into another position of the same route, where the position is limited within length $L^{\text{intra}}_{\text{ins}}$ (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^{cross} (i.e., $L^{\text{cross}} = n$). ILS is summarized as follows:

Algorithm: Iterated Local Search (ILS)

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

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

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

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

4.5 Efficient implementation of local search

A solution $\boldsymbol{\sigma}$ is evaluated by $(p+q)^*_{\text{sum}}(\boldsymbol{\sigma}) + a_{\text{sum}}(\boldsymbol{\sigma})$, where $(p+q)^*_{\text{sum}}(\boldsymbol{\sigma})$ denotes the minimum time window and traveling time cost for computed $\boldsymbol{\sigma}$ 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 $(p+q)^*_{\text{sum}}(\boldsymbol{\sigma})$ 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 $\langle \sigma_k(h_1) \to \sigma_k(h_2)\rangle$ the path from the h_1-th customer to the h_2-th customer in route σ_k, and by $\langle \sigma_k(h_1) \to \sigma_k(h_2)\rangle$–$\langle \sigma_{k'}(h_3) \to \sigma_{k'}(h_4)\rangle$ the path constructed by connecting two paths $\langle \sigma_k(h_1) \to \sigma_k(h_2)\rangle$ and $\langle \sigma_{k'}(h_3) \to \sigma_{k'}(h_4)\rangle$ from routes σ_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 $\langle 0 \to \sigma_k(h)\rangle$ and the latter part $\langle \sigma_k(h+1) \to 0\rangle$ for some h as illustrated in Figure 4.6. The amount of capacity

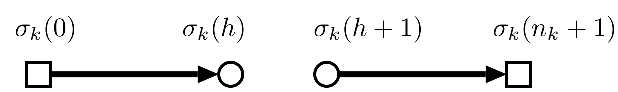


Figure 4.6: The former and latter parts of a route σ_k

excess for route σ_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 $\sum_{i=h}^{n_k} a_{\sigma_k(i)}$ for each customer $\sigma_k(h)$ and vehicle k whenever the current route is updated.

Now we concentrate on the computation of $(p+q)^*_{\text{sum}}(\sigma_k)$, which is the minimum sum of time window and traveling time costs on route σ_k.

Let $b_h^k(t)$ be the minimum sum of the time window costs for customers $\sigma_k(h), \sigma_k(h+$

1), …, $\sigma_k(n_k+1)$ and the traveling costs between them provided that all of them are served after time t.

We call this a *backward minimum cost function*. Then, $b_h^k(t)$ can be computed as follows, which is symmetric to the forward minimum cost computation discussed in Section 4.3:

$$\begin{aligned} b_{n_k+1}^k(t) &= \min\nolimits_{s\geq t} p_0(s), \\ b_h^k(t) &= \min\nolimits_{s\geq t}\{p_{\sigma_k(h)}(s) \\ &\quad + \min\nolimits_{t'\geq s}\{b_{h+1}^k(t'+\lambda_{\sigma_k(h),\sigma_k(h+1)}(t')) + q_{\sigma_k(h),\sigma_k(h+1)}(t')\}\}, \\ &\qquad\qquad 1\leq h\leq n_k. \end{aligned} \tag{4.5.12}$$

Let $f_h^k(t)$ be the forward minimum cost function at the hth customer in route σ_k. We can then obtain the optimal cost $(p+q)^*_{\text{sum}}(\sigma_k)$ by

$$\min_t \left\{ b_h^k(t) + \min_{t'+\lambda_{\sigma_k(h-1),\sigma_k(h)}(t')\leq t} f_{h-1}^k(t') + q_{\sigma_k(h-1),\sigma_k(h)}(t') \right\} \tag{4.5.13}$$

for any h ($1\leq h\leq n_k+1$). If $f_{h-1}^k(t)$ and $b_h^k(t)$ 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 $f_h^k(t)$ and $b_h^k(t)$ 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 σ_k in $O(\Delta(\sigma_k))$ time, if we keep the data $\sum_{i=1}^h a_{\sigma_k(i)}$, $\sum_{i=h}^{n_k} a_{\sigma_k(i)}$, $f_h^k(t)$ and $b_h^k(t)$ 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 $f_h^k(t)$ (resp., $b_h^k(t)$) is nonincreasing (resp., nondecreasing), there are usually many pieces with considerably large values in $f_h^k(t)$ (resp., in $b_h^k(t)$) 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.

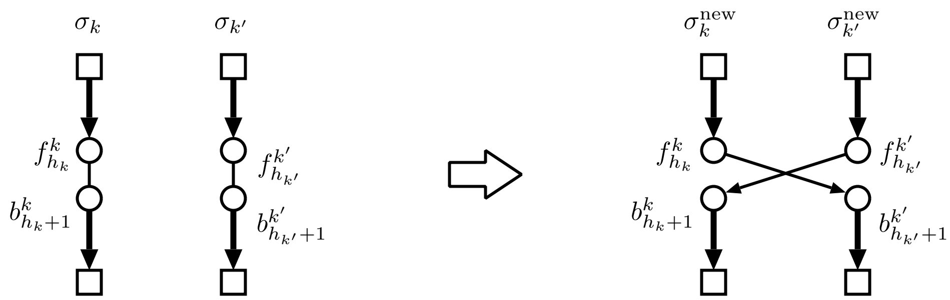


Figure 4.7: An example of a 2-opt* operation

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

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

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

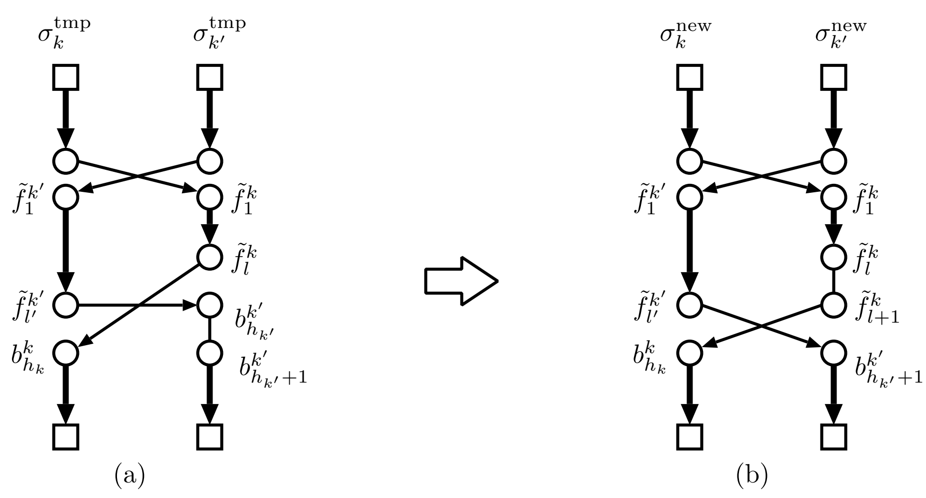


Figure 4.8: An example of the search order in the cross exchange neighborhood

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 σ_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 $\sigma_k^{\rm tmp}$ and $\sigma_{k'}^{\rm tmp}$ the routes obtained by applying a cross exchange operation on the current routes σ_k and $\sigma_{k'}$ (see Figure 4.8 (a)) and by $\sigma_k^{\rm new}$ and $\sigma_{k'}^{\rm new}$ the routes generated next (see Figure 4.8 (b)). In Figure 4.8 (a), backward minimum cost functions $b_{h_k}^k$, $b_{h_{k'}}^{k'}$ and $b_{h_{k'}+1}^{k'}$ of the current routes σ_k and $\sigma_{k'}$ are available, and we have already computed the forward minimum cost functions $\tilde{f}_1^k, \tilde{f}_2^k, \ldots, \tilde{f}_l^k$ (resp., $\tilde{f}_1^{k'}, \tilde{f}_2^{k'}, \ldots, \tilde{f}_{l'}^{k'}$) on the partial paths in $\sigma_k^{\rm tmp}$ (resp., $\sigma_{k'}^{\rm tmp}$) in the process of computing $(p{+}q)^*_{\rm sum}(\sigma_k^{\rm tmp})$ (resp., $(p{+}q)^*_{\rm sum}(\sigma_{k'}^{\rm tmp})$). We then compute $\tilde{f}_{l+1}^k$ from $\tilde{f}_l^k$ by recursion of the dynamic programming in $O(\Delta(\sigma_k^{\rm new}))$ time, and evaluate $(p{+}q)^*_{\rm sum}(\sigma_k^{\rm new}){+}(p{+}q)^*_{\rm sum}(\sigma_{k'}^{\rm new})$ in $O\left(\Delta(\sigma_k^{\rm new}) + \Delta(\sigma_{k'}^{\rm new})\right)$ time (Figure 4.8 (b)). Thus, we can compute the change in the cost after a cross exchange operation in $O\left(\Delta(\sigma_k^{\rm new}) + \Delta(\sigma_{k'}^{\rm new})\right)$ time.

Similarly, the change in the cost for an intra-route operation of route σ_k can be computed in $O\left(\Delta(\sigma_k^{\rm new})\right)$ time, by searching solutions in a specific order, where $\sigma_k^{\rm new}$ 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

$$F_h^k(U) = \begin{cases} \min\{t \mid f_h^k(t) \le U\}, & \text{if } \min_t f_h^k(t) \le U \\ +\infty, & \text{otherwise.} \end{cases}$$

This $F_h^k(U)$ 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

$$t \ge F_h^k(U) \iff f_h^k(t) \le U$$

holds. As mentioned in Section 4.3.2, we store $f_h^k(t)$ in a linked list; however, we can also store $f_h^k(t)$ in an array without sacrificing the time complexity. Using this the array data structure, we can compute $F_h^k(U)$ for a given U in $O(\log(\delta(f_h^k)))$ time because f_h^k is a nonincreasing function. (In our program, however, we did not implement the array structure, and use $O(\delta(f_h^k))$ time to compute $F_h^k(U)$, because this does not seem to be a bottle neck of computation.)

Similarly let

$$B_h^k(U) = \begin{cases} \max\{t \mid b_h^k(t) \le U\}, & \text{if } \min_t b_h^k(t) \le U \\ -\infty, & \text{otherwise.} \end{cases}$$

$B_h^k(U)$ 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

$$t \le B_h^k(U) \iff b_h^k(t) \le U$$

holds, because b_h^k is a nondecreasing function, and we can compute $B_h^k(U)$ in $O(\log(\delta(b_h^k)))$ time. Then, if

$$F_{h-1}^k(U) + \lambda_{k,h}^{\min} > B_h^k(U)$$

holds for $\lambda_{k,h}^{\min} = \min_t \lambda_{\sigma_k(h-1),\sigma_k(h)}(t)$, the cost of route σ_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 $t \ge \hat{t} \Rightarrow b(t) \ge f(\hat{t})$ and $t \le \hat{t} \Rightarrow f(t) \ge b(\hat{t})$, and we call this the *switch point* of f and b. Then the switch point $\hat{t}$ satisfies

$$\max\{f(\hat{t}), b(\hat{t})\} \le \min_t\{f(t)+b(t)\} \le f(\hat{t}) + b(\hat{t}) \le 2\max\{f(\hat{t}), b(\hat{t})\}.$$

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

$$f(\tilde{t}) \le \min_t\{f(t)+b(t)\} \le f(\tilde{t}) + b(\tilde{t}) \le 2f(\tilde{t})$$

holds, where $\tilde{t} = \arg\min_t f(t)$. 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 $f_{h-1}^k(t)$ and $b_h^k(t + \lambda_{\sigma_k(h-1),\sigma_k(h)}(t))$, and the schedule induced by the switch point becomes a 2-approximate schedule for σ_k (i.e., the cost of the schedule is at most $2(p+q)^*_{\text{sum}}(\sigma_k)$), where cost can be computed in $O\left(\log(\delta(f_{h-1}^k)) + \log(\delta(b_h^k))\right) = O(\log(\Delta(\sigma_k)))$ 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 $f_{h-1}^k(t)$ and $b_h^k(t+\lambda_{k,h}^{\min})$ 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 $\boldsymbol{\sigma}$.

[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 σ_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 σ_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 $B^{k'}_{h_{k'}}(U)$ 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 $(\sigma_k(h^k_1), \sigma_{k'}(h^{k'}_1)) \in P^{k,k'}_{\text{valid}}(U)$, 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 $L^{\text{cross}} = 3$, $L^{\text{intra}}_{\text{path}} = 3$ and $L^{\text{intra}}_{\text{ins}} = 15$ 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.

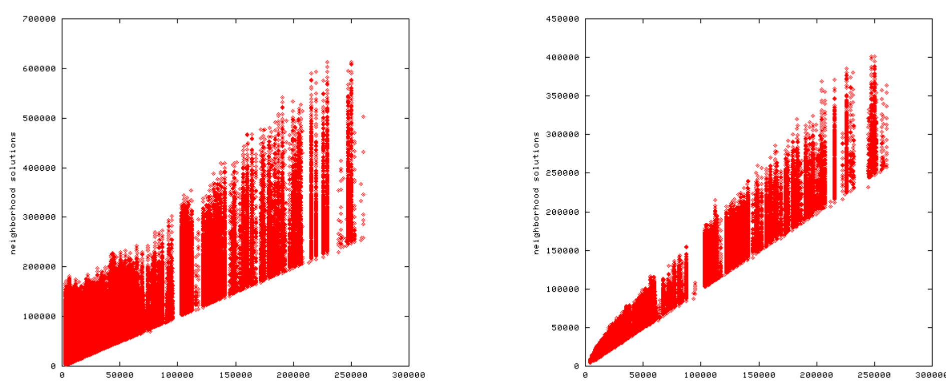


Figure 4.9: The distribution of two route cost in the 2-opt* neighborhood without (left) and with (right) restriction

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.

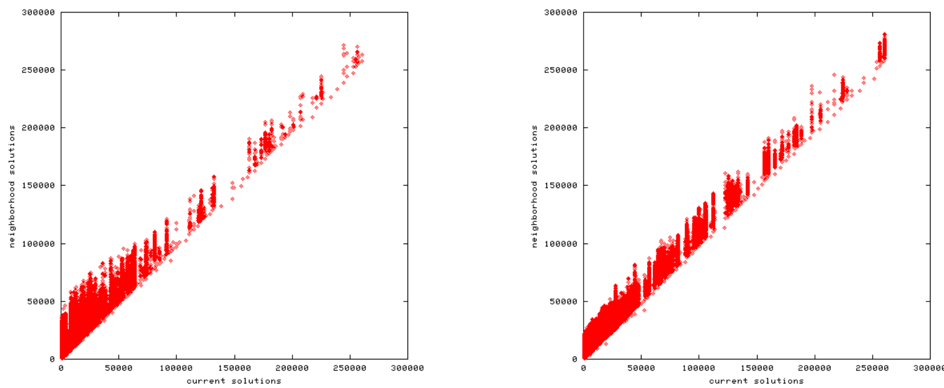


Figure 4.10: The distribution of two route cost in the cross exchange neighborhood without (left) and with (right) restriction

Table 4.1: Number of evaluations with and without restriction of neighborhood

neighborhood	2-opt*		cross exchange	
initial solution	random	locally optimal	random	locally optimal
without restriction	227746(11016.59)	4035(1253.23)	333321 (1607.82)	24875(1253.23)
with restriction	146440(17906.10)	164(1253.23)	139771 (1518.22)	111(1253.23)

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 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 λ_{ij} as follows:

$$
\begin{aligned}
p_i(t) &= \begin{cases} \alpha(r_i - t), & t < r_i \\ 0, & r_i \le t \le d_i \\ \alpha(t - d_i), & d_i < t, \end{cases} \\
q_{ij}(t) &= c_{ij}, \\
\lambda_{ij}(t) &= b_i + c_{ij},
\end{aligned}
$$

where α 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 α for each instance. We run the algorithm with some values of α from $\{1, 5, 10, 50, 100, 500, \ldots\}$ in the manner of binary search, where the time limit for each α was within 10% of the time limit reported in the tables in this section. Then we use the best α 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 α), run the algorithm by using the three values of α 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 α 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 α 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

	IIKMUY	HG99	GH02	BBB	B	BVH	HG03	IINSUY	ILS
C1	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00
	828.38*	828.38*	828.63	828.48	828.38*	828.38*	828.38*	828.38*	828.38*
C2	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00
	589.86*	589.86*	590.33	589.93	589.86*	589.86*	589.86*	589.86*	589.86*
R1	11.92	11.92	12.00	11.92	11.92	11.92	11.92	12.00	11.92
	1217.40	1228.06	1217.57	1221.10	1222.12	1213.25	1212.73*	1217.99	1213.18
R2	2.73	2.73	2.73	2.73	2.73	2.73	2.73	2.73	2.73
	959.11	969.95	961.29	975.43	975.12	966.37	955.03*	967.97	955.61
RC1	11.50	11.63	11.50	11.50	11.50	11.50	11.50	11.63	11.50
	1391.03	1392.57	1395.13	1389.89	1389.58	1384.22*	1386.44	1384.67	1384.25
RC2	3.25	3.25	3.25	3.25	3.25	3.25	3.25	3.25	3.25
	1122.79	1144.43	1139.37	1159.37	1128.38	1141.24	1123.17	1128.77	1120.50*
CNV	405*	406	406	405*	405*	405*	405*	407	405*
CTD	57444	57876	57641	57952	57710	57567	57309	57545	57282*
Computer	P 1GHz	P 200	P400	P 400	P 200	SU 10	unknown	P 2.8GHz	P2.8GHz
CPU (min)	250.0	13.8	4×20.9	30.0	87.0	120.0	n/a	16.7	16.7
Runs	1	10	5	3	1	5	n/a	1	3
Estimated time	108.7	6.0	43.6	9.4	3.8	104.3	n/a	16.7	16.7

Table 4.3: The results for 200-customer benchmark instances

	GH99	GH01	BVH	LL	LC	BHD	MB	IINSUY	ILS
C1	18.9	18.9	18.9	19.1	18.9	18.9	18.8*	18.9	18.9
	2782	2842.08	2726.63	2728.6	2743.66	2749.83	2717.21	2732.03	2721.94
C2	6	6	6	6	6	6	6	6	6
	1846	1856.99	1860.17	1854.9	1836.1	1842.65	1833.57*	1834.83	1835.96
R1	18.2	18.2	18.2	18.3	18.2	18.2	18.2	18.2	18.2
	3705	3855.03	3677.96	3736.2	3676.95	3718.3	3618.68*	3665.77	3690.34
R2	4	4	4.1	4.1	4	4	4	4	4
	3055	3032.49	3023.62	3023	2986.01	3014.28	2942.92*	2965.64	2943.88
RC1	18	18.1	18	18.3	18	18	18	18	18
	3555	3674.91	3279.99	3385.8	3449.71	3329.62	3221.34*	3287.61	3345.01
RC2	4.3	4.4	4.5	4.9	4.3	4.4	4.4	4.3	4.3
	2675	2671.34	2603.08	2518.7	2613.75	2585.89	2519.79	2562.56*	2564.68
CNV	694*	696	697	707	694*	695	694*	694*	694*
CTD	176180	179328	171715	172472	173061	172406	168573*	170484	171018
Computer	P 200	P 400	SU 10	P 545	P 933	AMD 700	P 2GHz	P 2.8GHz	P 2.8GHz
CPU (min)	4×10	4×2.1	n/a	182.1	5×10	2.4	8	33.3	33.3
Runs	1	3	n/a	3	1	3	1	1	3
Estimated time	2.4	3.8	n/a	112.4	21.7	2.1	5.9	33.0	33.0

Table 4.4: The results for 400-customer benchmark instances

	GH99	GH01	BVH	LL	LC	BHD	MB	IINSUY	ILS
C1	38	38	38	38.7	37.9	37.9	37.9	37.7	37.6*
	7584	7855.82	7220.96	7181.4	7447.09	7230.48	7148.27	7282.15	7444.06
C2	12	12	12	12.1	12	12	12	12	11.8*
	3935	3940.19	4154.4	4017.1	3940.87	3894.48	3840.85	3851.96	3982.50
R1	36.4	36.4	36.4	36.6	36.5	36.4	36.3*	36.4	36.4
	8925	9478.22	8713.37	8912.4	8839.28	8692.17	8530.03	8746.94	8998.63
R2	8	8	8	8	8	8	8	8	8
	6502	6650.28	6959.75	6610.6	6437.68	6382.63	6209.94*	6269.9	6258.00
RC1	36.1	36.1	36.1	36.5	36	36	36	36	36
	8763	9294.99	8330.98	8377.9	8652.01	8305.55	8066.44*	8405.32	8572.11
RC2	8.6	8.8	8.9	9.5	8.6	8.9	8.8	8.6	8.5*
	5518	5629.43	5631.7	5466.2	5511.22	5407.87	5243.06	5337.5	5355.59
CNV	1390	1392	1393	1414	1390	1391	1389	1387	1383*
CTD	412270	428489	410112	405656	408281	399132	390386	398938	406109
Computer	P 200	P 400	SU 10	P 545	P 933	AMD 700	P 2GHz	P 2.8GHz	P 2.8GHz
CPU (min)	4×20	4×7.1	n/a	359.8	5×20	7.9	17	66.6	66.6
Runs	1	3	n/a	3	1	3	1	1	3
Estimated time	4.8	13.0	n/a	221.8	43.3	6.8	12.5	66.6	66.6

Table 4.5: The results for 600-customer benchmark instances

	GH99	GH01	BVH	LL	LC	BHD	MB	IINSUY	ILS
C1	57.9	57.7	57.8	58.2	57.9	57.8	57.8	57.5	57.5
	14792	14817.25	14357.11	14267.30	14205.58	14165.90	14003.09	14116.97*	14296.96
C2	17.9	17.8	17.8	18.2	17.9	18	17.8	17.4	17.4
	7787	7889.96	8259.04	8202.60	7743.92	7528.73	7455.83	7945.56*	7960.138
R1	54.5	54.5	55	55.2	54.8	54.5	54.5	54.5	54.5
	20854	21864.47	19308.62	19744.80	19869.82	19081.18	18358.68*	19844.39	20363.15
R2	11	11	11	11.1	11.2	11	11	11	11
	13335	13656.15	14855.43	13592.40	13093.97	13054.83	12703.52	12539.78*	13047.18
RC1	55.1	55	55.1	55.5	55.2	55	55	55	55
	18411	19114.02	17035.91	17320.00	17678.13	16994.22	16418.63*	17278.81	17764.33
RC2	11.8	11.9	12.4	13	11.8	12.1	12.1	11.6	11.5*
	11522	11670.29	11987.89	11204.90	11034.71	11212.36	10677.46	10791.70	11315.28
CNV	2082	2079	2091	2112	2088	2084	2082	2070	2069*
CTD	867010	890121	858040	843320	836261	820372	796172	825172	847470
Computer	P 200	P 400	SU 10	P 545	P 933	AMD 700	P 2GHz	P 2.8GHz	P 2.8GHz
CPU (min)	4×30	4 × 12.9	n/a	399.8	5× 30	16.2	40	100	100
Runs	1	3	n/a	3	1	3	1	1	3
Estimated time	7.1	23.5	n/a	246.4	65.0	13.9	29.4	100.0	100.0

Table 4.6: The results for 800-customer benchmark instances

	GH99	GH01	BVH	LL	LC	BHD	MB	IINSUY	ILS
C1	76.7	76.1	76.1	77.4	76.3	76.3	76.2	75.7	75.6*
	26528	26936.68	25391.67	25337.02	25668.82	25170.88	25132.27	25487.55	25915.59
C2	24	23.7	24.4	24.4	24.1	24.2	23.7	23.4	23.4
	12451	11847.92	14253.83	11956.60	11985.11	11648.92	11352.29	11860.90*	11942.54
R1	72.8	72.8	72.7*	73	73.1	72.8	72.8	72.8	72.8
	34586	34653.88	33337.91	33806.34	33552.40	32748.06	31918.47	33275.72	34095.04
R2	15	15	15	15.1	15	15	15	15	15
	21697	21672.85	24554.63	21709.39	21157.56	21170.15	20295.28	20209.92*	20810.51
RC1	72.4	72.3	73	73.2	72.3	73	73	72.4	72.3
	38509	40532.35	30500.15	31282.54	37722.62	30005.95	30731.07	34621.63	34358.45*
RC2	16.1	16.1	16.6	17.1	15.8	16.3	15.8	15.7	15.6*
	17741	17941.23	18940.84	17561.22	17441.60	17686.65	16729.18	16666.76	17173.59
CNV	2770	2760	2778	2802	2766	2776	2765	2750	2747*
CTD	1515120	1535849	1469790	1416531	1475281	1384306	1361586	1421225	1442957
Computer	P 200	P 400	SU 10	P 545	P 933	AMD 700	P 2GHz	P 2.8GHz	P 2.8GHz
CPU (min)	4×40	4×23.2	n/a	512.9	5×40	26.2	145	133.3	133.3
Runs	1	3	n/a	3	1	3	1	1	3
Estimated time	9.5	42.3	n/a	316.1	86.6	22.5	106.5	133.3	133.3

Table 4.7: The results for 1000-customer benchmark instances

	GH99	GH01	BVH	LL	LC	BHD	MB	IINSUY	ILS
C1	96	95.4	95.1	96.3	95.3	95.8	95.1	94.5	94.4*
	43273	43392.59	42505.35	42428.50	43283.92	42086.77	41569.67	42459.35	43066.89
C2	30.2	29.7	30.3	30.8	29.9	30.6	29.7	29.4	29.4
	17570	17574.72	18546.13	17294.90	17443.50	17035.88	16639.54	16986.46	16822.82*
R1	91.9	91.9	92.8	92.7	92.2	92.1	92.1	91.9	91.9
	57186	58069.61	51193.47	50990.80	55176.95	50025.64	49281.48	53366.10*	54149.50
R2	19	19	19	19	19.2	19	19	19	19
	31930	31873.62	36736.97	31990.90	30919.77	31458.23	29860.32	29546.19*	30626.04
RC1	90	90.1	90.2	90.4	90	90	90	90	90
	50668	50950.14	48634.15	48892.40	49711.36	46736.92	45396.41*	48275.20	49378.71
RC2	19	18.5	19.4	19.8	18.5	19	18.7	18.3	18.3
	27012	27175.98	29079.78	26042.30	26001.11	25994.12	25063.51	24904.08*	26428.81
CNV	3461	3446	3468	3490	3451	3465	3446	3431	3430*
CTD	2276390	2290367	2266959	2176398	2225366	2133376	2078110	2155374	2204728
Computer	P 200	P 400	SU 10	P 545	P 933	AMD 700	P 2GHz	P 2.8GHz	P 2.8GHz
CPU (min)	4×50	4×30.1	n/a	606.3	5×50	39.6	600	166.7	166.7
Runs	1	3	n/a	3	1	3	1	1	3
Estimated time	11.9	54.9	n/a	373.5	108.3	34.0	440.8	166.7	166.7

Table 4.8: The detailed results for 100-customer benchmark instances

Inst	m	α	d_{sum}	LS	bknown m	bknown d_{sum}	Inst	m	α	d_{sum}	LS	bknown m	bknown d_{sum}
c101	10	5	828.94	26577	10	828.94	c201	3	5	591.56	5715	3	591.56
c102	10	5	828.94	30171	10	828.94	c202	3	5	591.56	5046	3	591.56
c103	10	5	828.06	33544	10	828.06	c203	3	5	591.17	5244	3	591.17
c104	10	5	824.78	36223	10	824.78	c204	3	5	590.60	7976	3	590.60
c105	10	5	828.94	24613	10	828.94	c205	3	5	588.88	7101	3	588.88
c106	10	5	828.94	27728	10	828.94	c206	3	5	588.49	7098	3	588.49
c107	10	5	828.94	28134	10	828.94	c207	3	5	588.29	7554	3	588.29
c108	10	5	828.94	31400	10	828.94	c208	3	5	588.32	8937	3	588.32
c109	10	5	828.94	36113	10	828.94							
r101	19	100	1650.80	35156	19	1645.79	r201	4	5	1253.23	6720	4	1252.37
r102	17	100	1486.12	30962	17	1486.12	r202	3	10	1191.80	3401	3	1191.70
r103	13	50	1292.68	28086	13	1292.68	r203	3	1	943.27	5566	3	939.54
r104	9	50000	1007.31	24758	9	1007.24	r204	2	1	832.76	2579	2	825.52
r105	14	5	1377.11	29396	14	1377.11	r205	3	10	994.43	5965	3	994.42
r106	12	5	1252.03	25695	12	1251.98	r206	3	1	906.14	6214	3	906.14
r107	10	50	1104.66	22944	10	1104.66	r207	2	5	898.16	2509	2	893.33
r108	9	10	963.99	25944	9	960.88	r208	2	1	730.54	5620	2	726.75
r109	11	10	1205.36	25423	11	1194.73	r209	3	1	915.06	5279	3	909.16
r110	10	10	1129.47	23971	10	1118.59	r210	3	5	939.37	5818	3	939.34
r111	10	50	1096.73	23646	10	1096.72	r211	2	5	906.96	2538	2	892.71
r112	9	100	991.85	28919	9	982.14							
rc101	14	100	1696.95	32400	14	1696.94	rc201	4	5	1406.94	7387	4	1406.91
rc102	12	100	1554.75	28893	12	1554.75	rc202	3	500	1367.00	3604	3	1365.64
rc103	11	10	1262.02	31070	11	1261.67	rc203	3	10	1058.33	6260	3	1049.62
rc104	10	5	1135.83	27181	10	1135.48	rc204	3	5	798.46	11137	3	798.41
rc105	13	100	1629.44	31288	13	1629.44	rc205	4	1	1297.65	6838	4	1297.19
rc106	11	100	1424.73	31523	11	1424.73	rc206	3	5	1146.32	5099	3	1146.32
rc107	11	50	1230.48	36141	11	1230.48	rc207	3	1	1061.14	4658	3	1061.14
rc108	10	50	1139.82	34620	10	1139.82	rc208	3	1	828.14	7816	3	828.14

Table 4.9: The detailed results for 200-customer benchmark instances

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

Table 4.10: The detailed results for 400-customer benchmark instances

					bknown							bknown	
Inst	m	α	d_{sum}	LS	m	d_{sum}	Inst	m	α	d_{sum}	LS	m	d_{sum}
c101	40	1	7152.06	11893	40	7152.02	c201	12	5	4116.14	3296	12	4116.05
c102	36	5000	7856.66	6817	37	7357.45	c202	12	1	3930.45	4605	12	3930.29
c103	36	5	7363.31	6264	36	7151.17	c203	12	1	3779.77	4978	12	3739.72
c104	36	1	6869.50	6508	36	6822.18	c204	11	1000000	4350.20	3120	12	3535.99
c105	40	5	7152.06	9751	40	7152.02	c205	12	1	3938.69	5564	12	3939.42
c106	40	1	7153.45	10622	40	7153.41	c206	12	1	3875.94	6480	12	3875.94
c107	39	5000	7505.24	8914	39	8043.18	c207	12	10	3897.70	6325	12	3894.13
c108	37	10000	7882.36	8587	38	7113.40	c208	12	1	3798.66	6732	12	3787.08
c109	36	1000	8086.45	7491	36	7524.32	c209	12	10	3879.83	6728	12	3876.10
c110	36	5	7419.52	6683	36	6907.26	c210	11	5000	4257.64	4662	12	3684.89
r101	40	5000	10547.11	9961	38	11084.00	r201	8	50	9319.21	1860	8	9257.92
r102	36	500	9610.16	5937	36	9161.26	r202	8	1000	7662.25	2408	8	7674.90
r103	36	500	8513.14	5744	36	7941.53	r203	8	1000	6044.85	2811	8	5988.02
r104	36	10	7649.41	4952	36	7332.93	r204	8	5	4348.34	4269	8	4331.07
r105	36	50	10270.00	5412	36	9512.25	r205	8	10	7191.03	2733	8	7143.55
r106	36	100	9197.03	5940	36	8534.05	r206	8	10	6246.39	2917	8	6163.81
r107	36	50	8089.12	5630	36	7710.41	r207	8	5	5140.19	3631	8	5082.10
r108	36	5	7701.29	4659	36	7398.68	r208	8	5	4124.64	5357	8	4068.97
r109	36	10	9660.98	5075	36	8878.19	r209	8	5	6486.50	2795	8	6493.13
r110	36	50	8748.10	5799	36	8227.49	r210	8	10	6016.55	3598	8	5895.93
rc101	36	1000	9769.63	6505	36	8960.82	rc201	11	50	6770.44	3784	11	7019.89
rc102	36	50	8820.22	6916	36	8174.27	rc202	9	10000	6419.16	1990	10	5924.84
rc103	36	50	7973.35	7037	36	7737.99	rc203	8	5	5048.38	2590	8	5114.76
rc104	36	10	7551.31	5666	36	7411.02	rc204	8	10	3700.01	5338	8	3648.64
rc105	36	100	8948.55	7427	36	8499.15	rc205	9	50	6047.21	2712	9	6063.46
rc106	36	10	8873.07	6521	36	8304.99	rc206	8	50	5998.19	2102	8	6054.21
rc107	36	10	8898.66	6386	36	8051.71	rc207	8	100	5570.20	3172	8	5519.25
rc108	36	5	8493.26	5002	36	7917.68	rc208	8	1	4916.86	2885	8	4854.16
rc109	36	5	8282.77	5030	36	7890.45	rc209	8	50	4657.48	4810	8	4628.26
rc110	36	50	8110.30	7046	36	7716.32	rc210	8	5	4427.98	4326	8	4316.36

Table 4.11: The detailed results for 600-customer benchmark instances

					bknown							bknown	
Inst	m	α	d_{sum}	LS	m	d_{sum}	Inst	m	α	d_{sum}	LS	m	d_{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	50	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	50	14089.66	6616	60	14089.70	c206	18	1	7479.48	5428	18	7478.63
c107	59	10000	14580.31	6403	59	14659.74	c207	18	10	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	4332	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	100	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	100	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	100	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	100	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	100	18583.52	4331	55	17248.87	rc206	11	1000	13396.83	1202	12	11411.08
rc107	55	100	18167.54	4495	55	16454.79	rc207	11	100	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

					bknown							bknown	
Inst	m	α	d_{sum}	LS	m	d_{sum}	Inst	m	α	d_{sum}	LS	m	d_{sum}
c101	80	5	25184.38	5074	80	25030.36	c201	24	1	11662.08	3248	24	11654.72
c102	74	1000000	26114.66	4387	75	25518.17	c202	23	10000	12773.63	2326	24	11422.34
c103	72	5	26213.54	2472	72	25438.60	c203	23	100	12503.37	2195	23	11554.18
c104	72	1	24719.93	2225	72	24040.47	c204	23	1	11342.56	2615	23	10963.49
c105	80	100	25166.28	4669	80	25166.30	c205	24	1	11434.03	4187	24	11432.92
c106	80	100	25160.85	4728	80	25160.90	c206	24	1	11348.43	4624	24	11357.86
c107	79	500	25538.54	3758	79	25518.85	c207	24	50	11468.03	4474	24	11397.54
c108	75	10000	26243.46	4762	76	25379.85	c208	23	10000	12195.91	4542	24	11206.32
c109	72	1000	27827.13	3729	73	24713.38	c209	23	10000	13069.53	3364	24	11249.00
c110	72	10	26987.10	2845	72	29536.81	c210	23	5	11627.82	3785	23	11284.46
r101	80	1000000	38056.29	4884	79	39612.20	r201	15	100	29206.74	1784	15	28440.28
r102	72	5000	35999.87	3367	72	33548.54	r202	15	10	24088.01	1517	15	23335.67
r103	72	100	32529.49	2361	72	30151.90	r203	15	10	18286.82	1935	15	17992.25
r104	72	10	30303.52	1990	72	26838.04	r204	15	5	13929.80	2422	15	13625.25
r105	72	50	38055.58	2496	72	34741.53	r205	15	50	25349.89	2135	15	24611.39
r106	72	10	34546.53	2321	72	31737.47	r206	15	10	21397.18	1868	15	20697.06
r107	72	10	31537.02	2196	72	29538.40	r207	15	10	17249.67	2134	15	17058.30
r108	72	5	29662.64	1892	72	28342.64	r208	15	10	13396.06	2769	15	13053.31
r109	72	100	35986.98	2505	72	34231.38	r209	15	10	23252.47	2147	15	22588.02
r110	72	10	34272.51	2299	72	31730.45	r210	15	5	21948.49	2080	15	21551.26
rc101	73	100	33711.89	3387	73	31590.23	rc201	19	10000	20716.21	3238	20	19989.12
rc102	72	50	35112.82	3455	72	39696.20	rc202	16	10000	19129.08	1660	17	18099.68
rc103	72	100	33015.08	3261	72	35577.87	rc203	15	100	15346.72	1823	15	15116.26
rc104	72	10	30085.34	2717	72	32654.10	rc204	15	5	11604.53	2374	15	11392.25
rc105	73	100	32344.49	3481	73	30454.15	rc205	16	100	19321.34	2195	16	19105.75
rc106	73	50	32782.06	3286	73	29674.68	rc206	15	10	19945.66	1617	15	18882.30
rc107	72	500	39643.15	3244	72	43829.43	rc207	15	5	17547.91	1713	15	17461.44
rc108	72	100	36512.04	3059	72	43694.60	rc208	15	5	16752.84	1998	15	16529.24
rc109	72	50	35660.83	3038	72	41816.70	rc209	15	5	16329.70	2180	15	15823.50
rc110	72	10	34716.75	2504	72	41182.44	rc210	15	10	15041.86	2804	15	14892.29

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

					bknown							bknown	
Inst	m	α	d_{sum}	LS	m	d_{sum}	Inst	m	α	d_{sum}	LS	m	d_{sum}
c101	100	5	42478.95	5050	100	42478.95	c201	30	5	16879.24	2797	30	16879.24
c102	90	1000	45854.82	2666	92	42920.70	c202	29	50	17473.93	2307	29	17228.82
c103	90	5	42218.92	2401	90	40934.87	c203	29	50	17043.13	2626	29	16367.59
c104	90	1	41575.26	2112	90	40410.58	c204	29	5	16896.99	2624	29	17153.19
c105	100	100	42469.18	4984	100	42469.20	c205	30	5	16568.73	3692	30	16586.46
c106	100	100	42471.28	5135	100	42471.30	c206	30	5	16348.20	4508	30	16371.65
c107	99	1000	42821.17	3984	99	42711.39	c207	30	10000	16827.81	4416	31	16578.42
c108	94	10000	43555.1	4238	96	42170.31	c208	29	1	16532.88	3500	29	18662.10
c109	91	1000	42755.59	3254	91	45386.93	c209	29	10000	17462.68	4022	30	16651.96
c110	90	100	44468.65	3344	90	40894.38	c210	29	1	16194.56	3545	29	16178.26
r101	100	1000	55922.77	3756	100	54145.31	r201	19	500	43554.40	1779	19	42922.56
r102	91	1000	56975.89	2929	91	56367.45	r202	19	50	35416.79	1562	19	34918.49
r103	91	100	51259.61	2645	91	46621.19	r203	19	50	26396.47	1787	19	25689.62
r104	91	50	47116.49	2306	91	43461.84	r204	19	5	19026.43	2214	19	18858.24
r105	91	50	61437.30	2672	91	70838.01	r205	19	100	38162.84	2040	19	37265.32
r106	91	50	55707.67	2783	91	49059.80	r206	19	100	31990.34	1745	19	30725.20
r107	91	100	50834.66	2640	91	45847.84	r207	19	50	24603.46	1942	19	24363.83
r108	91	50	46612.45	2290	91	42767.77	r208	19	5	18950.37	2220	19	18185.38
r109	91	500	59344.93	2652	91	51391.80	r209	19	10	35737.18	1939	19	33777.76
r110	91	50	56283.20	2731	91	49348.36	r210	19	5	32422.07	2046	19	31599.84
rc101	90	100	52084.03	2572	90	47143.90	rc201	21	100	30585.71	2440	22	30320.41
rc102	90	500	49503.47	2435	90	44906.58	rc202	18	1000000	29525.90	746	19	26592.40
rc103	90	100	46038.46	2488	90	43782.57	rc203	18	500	22185.99	1217	18	20588.38
rc104	90	5	43998.71	1883	90	41917.14	rc204	18	10	17645.94	1478	18	16480.17
rc105	90	50	51822.47	2569	90	47632.31	rc205	18	100	30231.94	1116	18	29383.27
rc106	90	500	51377.57	2518	90	46391.60	rc206	18	50	29668.22	1225	18	27003.30
rc107	90	500	50657.17	2489	90	46391.60	rc207	18	50	27928.74	1341	18	26161.91
rc108	90	10	50099.58	2175	90	45585.08	rc208	18	5	26631.77	1113	18	24995.00
rc109	90	5	50320.45	2004	90	45405.54	rc209	18	50	25381.34	1716	18	23582.89
rc110	90	10	47885.23	2208	90	45041.64	rc210	18	5	24502.58	1319	18	22481.03

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

	scenario1			scenario2			scenario3		
	morning	daytime	evening	morning	daytime	evening	morning	daytime	evening
category 1	0.54	0.81	0.54	0.33	0.67	0.33	0.12	0.46	0.12
category 2	0.81	1.22	0.81	0.67	1.33	0.67	0.46	1.92	0.46
category 3	1.22	1.82	1.22	1.33	2.67	1.33	0.96	3.84	0.96

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_{sum} and q_{sum} with respect to all instances of the same type. We omitted a_{sum}, since a_{sum} were always 0.

In Table 4.15, we can observe that both p_{sum} and q_{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

	scenario1				scenario2				scenario3			
	time-dependent		const		time-dependent		const		time-dependent		const	
	p_{sum}	q_{sum}	p_{sum}	q_{sum}	p_{sum}	q_{sum}	p_{sum}	q_{sum}	p_{sum}	q_{sum}	p_{sum}	q_{sum}
C1	20.35	855.27	35.47	984.26	90.80	885.66	183.30	1301.37	342.45	1137.45	1312.33	2356.09
C2	2.31	669.94	0.00	712.52	7.43	763.61	54.25	919.15	325.91	1038.11	1554.59	1680.48
R1	32.47	1061.93	50.60	1188.67	43.41	921.41	94.92	1260.24	109.90	946.06	462.54	1378.72
R2	5.04	904.40	19.62	1021.33	3.00	809.01	77.37	1139.08	17.70	873.04	954.12	1562.06
RC1	50.10	1103.62	59.50	1298.58	55.23	967.13	96.89	1295.63	90.90	1023.89	486.02	1568.13
RC2	19.69	976.53	25.09	1140.38	12.40	877.58	106.79	1263.43	49.99	948.67	803.88	1640.06

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 feasible. 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 (≥ 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 σ_k denote the route traveled by vehicle k, where $\sigma_k(h)$ denotes the hth customer in σ_k, and let

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

Note that each customer i is included in exactly one route σ_k, and is visited by vehicle k exactly once. We denote by n_k the number of customers in σ_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. Note that each vehicle is allowed to wait at customers before starting services.

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

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

That is, $y_{ik}(\boldsymbol{\sigma}) = 1$ 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:

$$\begin{array}{llll}
\text{minimize} & \displaystyle\sum_{k \in M} d(\sigma_k) & & (5.2.1) \\
\text{subject to} & \displaystyle\sum_{k \in M} y_{ik}(\boldsymbol{\sigma}) = 1, & i \in V \setminus \{0\} & (5.2.2) \\
& \displaystyle\sum_{i \in V \setminus \{0\}} a_i y_{ik}(\boldsymbol{\sigma}) \le u, & k \in M & (5.2.3) \\
& t_{0,\sigma_k(1)} \le s_{\sigma_k(1)}, & k \in M & (5.2.4) \\
& s_{\sigma_k(i)} + t_{\sigma_k(i),\sigma_k(i+1)} \le s_{\sigma_k(i+1)}, & 1 \le i \le n_k - 1,\ k \in M & (5.2.5) \\
& s_{\sigma_k(n_k)} + t_{\sigma_k(n_k),0} \le s_k^{\mathrm{a}} \le l_0, & k \in M & (5.2.6) \\
& e_i \le s_i \le l_i, & i \in V \setminus \{0\} & (5.2.7) \\
& y_{ik}(\boldsymbol{\sigma}) \in \{0,1\}, & i \in V \setminus \{0\},\ k \in M. & (5.2.8)
\end{array}$$

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 σ_k, since the values of $y_{ik}(\boldsymbol{\sigma})$ are automatically determined from $\boldsymbol{\sigma}$, and finding appropriate values for s_i and s_k^{a}, if any, is easy when $\boldsymbol{\sigma}$ is fixed.

5.3 Local search

In this section, we describe our local search (LS). Our LS searches a visiting order $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_m)$, which can be infeasible with respect to the capacity and time window constraints. The algorithm evaluates each route σ_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 σ_k is infeasible, and it evaluates a solution $\boldsymbol{\sigma}$ 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 $\boldsymbol{\sigma}$ and repeats replacing

$\boldsymbol{\sigma}$ with a better solution (with respect to $\sum_{k \in M} p(\sigma_k)$) in its neighborhood $N(\boldsymbol{\sigma})$ until no better solution is found in $N(\boldsymbol{\sigma})$. To define the neighborhood $N(\boldsymbol{\sigma})$, 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_{nlist} (a parameter) customers as a neighbor list of i. We set $N_{\text{nlist}} = 20$ in the experiments.

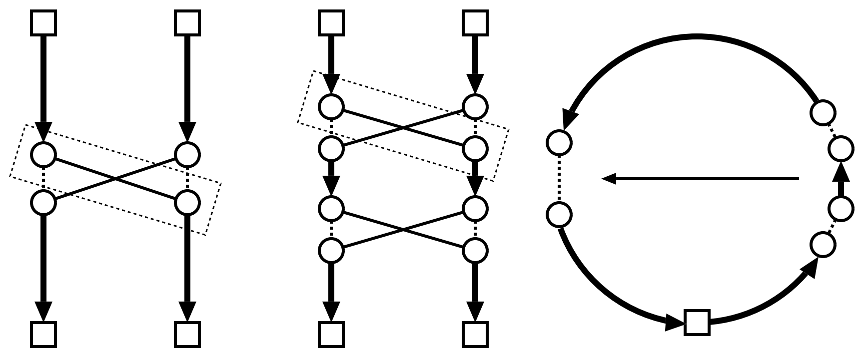
5.3.2 Neighborhoods

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 $O(N_{\text{nlist}} n)$.

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^{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 $O((L^{\text{cross}})^2 N_{\text{nlist}} n)$. 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.



(a) 2-opt* (b) Cross exchange (c) Intra-route

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 σ_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., $s_{n_k+1} = s_k^{\text{a}}$). The function we adopt is

$$p(\sigma_k) = \begin{cases} d(\sigma_k), & \text{if } \sigma_k \text{ is feasible} \\ d(\sigma_k) + \alpha p_{\text{c}}(\sigma_k) + \beta p_{\text{t}}(\sigma_k) + \sum_{h=1}^{n_k} \gamma_h, & \text{otherwise,} \end{cases} \tag{5.4.9}$$

where $p_{\mathrm{c}}(\sigma_k)$ is the amount of capacity excess (i.e., $p_{\mathrm{c}}(\sigma_k) = \max\{0, \sum_{i=1}^{n_k} a_i - u\}$) and $p_{\mathrm{t}}(\sigma_k)$ is the minimum total amount of traveling times to be shortened to satisfy the constraints; i.e.,

$$p_{\mathrm{t}}(\sigma_k) = \min \left\{ \sum_{h=1}^{n_k+1} \tau_h \;\middle|\; \begin{array}{l} s_0 \geq 0,\ s_{h-1} + t_{h-1,h} - \tau_h \leq s_h, \\ \tau_h \geq 0,\ e_h \leq s_h \leq l_h,\ h = 1, \ldots, n_k + 1 \end{array} \right\}.$$

In function p, α, β and γ_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 $p_{\mathrm{t}}(\sigma_k)$. 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 $p_{\mathrm{t}}(\sigma_k)$, 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 σ_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 σ_k, let $\mathcal{F}_h$ (resp., $\mathcal{B}_h$) be some data structure that contains the information of the path (of σ_k) from the depot to h (resp., from h to the depot). Note that $\mathcal{F}_h$ and $\mathcal{B}_h$ signify the information of the paths of the new route σ_k. For example, if σ_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 $p(\sigma_k)$. 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 σ_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 σ_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 $p_{\mathrm{t}}(\sigma_k)$ 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. 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

$p_{\mathrm{t}}(\sigma_k)$ from $\mathcal{F}_h$ and $\mathcal{B}_{h+1}$ can also be done in $O(1)$ time. Hence the algorithm can evaluate each neighborhood solution in $O(1)$ time.

Let f_h be the minimum total amount of traveling times to be shortened to satisfy the time window constraints for customers $1, 2, \ldots, h$ when vehicle k visits them along the route. Let s_h^{f} be the start time of service at h that attains f_h together with $s_1^{\mathrm{f}}, \ldots, s_{h-1}^{\mathrm{f}}$, and let $\mathcal{F}_h = (f_h, s_h^{\mathrm{f}})$. Then the forward computation can be done by:

$$s_{h+1}^{\mathrm{f}} = \min\left\{l_{h+1}, \max\{s_h^{\mathrm{f}} + t_{h,h+1}, e_{h+1}\}\right\} \tag{5.4.10}$$

$$f_{h+1} = f_h + \max\{s_h^{\mathrm{f}} + t_{h,h+1}, e_{h+1}\} - s_{h+1}^{\mathrm{f}}. \tag{5.4.11}$$

In (5.4.10), if $l_{h+1} < \max\{s_h^{\mathrm{f}} + t_{h,h+1}, e_{h+1}\}$ holds, the traveling time is shortened to satisfy the time window constraint and this amount is added to f_{h+1} in (5.4.11).

The backward computation can be done similarly. Let b_h be the minimum total amount of traveling times to be shortened to satisfy the time window constraints for customers $h, h+1, \ldots, n_k+1$ when vehicle k starts from h and returns to the depot along the route. Let s_h^{b} be the start time of service at h that attains b_h together with $s_{h+1}^{\mathrm{b}}, \ldots, s_{n_k+1}^{\mathrm{b}}$, and let $\mathcal{B}_h = (b_h, s_h^{\mathrm{b}})$. Then the backward computation can be done by:

$$s_h^{\mathrm{b}} = \max\left\{\min\{l_h, s_{h+1}^{\mathrm{b}} - t_{h,h+1}\}, e_h\right\} \tag{5.4.12}$$

$$b_h = b_{h+1} + s_h^{\mathrm{b}} - \min\{l_h, s_{h+1}^{\mathrm{b}} - t_{h,h+1}\}. \tag{5.4.13}$$

We can compute $p_{\mathrm{t}}(\sigma_k)$ from $\mathcal{F}_h = (f_h, s_h^{\mathrm{f}})$ and $\mathcal{B}_{h+1} = (b_{h+1}, s_{h+1}^{\mathrm{b}})$ by

$$s_{h+1}^{\mathrm{f}} = \min\left\{l_{h+1}, \max\{s_h^{\mathrm{f}} + t_{h,h+1}, e_{h+1}\}\right\} \tag{5.4.14}$$

$$p_{\mathrm{t}}(\sigma_k) = f_h + b_{h+1} + \max\{0, s_{h+1}^{\mathrm{f}} - s_{h+1}^{\mathrm{b}}\}. \tag{5.4.15}$$

5.5 Adaptive mechanism to control parameters

In this section, we describe an adaptive mechanism to control the parameters α, β and γ_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 α and β

Let $p_{\mathrm{c}}^{\mathrm{sum}}(\boldsymbol{\sigma}) = \sum_{k \in M} p_{\mathrm{c}}(\sigma_k)$ and $p_{\mathrm{t}}^{\mathrm{sum}}(\boldsymbol{\sigma}) = \sum_{k \in M} p_{\mathrm{t}}(\sigma_k)$, and let $p_{\mathrm{c}}^{\min}$ (resp., $p_{\mathrm{t}}^{\min}$) be the minimum $p_{\mathrm{c}}^{\mathrm{sum}}(\boldsymbol{\sigma})$ (resp., $p_{\mathrm{t}}^{\mathrm{sum}}(\boldsymbol{\sigma})$) of the solutions in the current reference set R of good solutions, where rules for maintaining R are described in Section 5.6. Let P_{c} (resp., P_{t}) be the number of moves, during the last call to the LS, to a solution $\boldsymbol{\sigma}$ whose $p_{\mathrm{c}}^{\mathrm{sum}}(\boldsymbol{\sigma})$ (resp.,

$p_{\mathrm{t}}^{\mathrm{sum}}(\boldsymbol{\sigma})$) is less than $p_{\mathrm{c}}^{\min}$ (resp., $p_{\mathrm{t}}^{\min}$) or equals to 0. Let N_{total} be the total number of moves during the last call to LS, and let $N_{\mathrm{c}} = N_{\mathrm{total}} - P_{\mathrm{c}}$ and $N_{\mathrm{t}} = N_{\mathrm{total}} - P_{\mathrm{t}}$. We use parameters δ_{inc}, δ_{dec}, $\delta_{\mathrm{inc}}^{\mathrm{cust}}$ and $\delta_{\mathrm{dec}}^{\mathrm{cust}}$, and in the experiments, we set $\delta_{\mathrm{inc}} = 0.05$, $\delta_{\mathrm{dec}} = 0.1$, $\delta_{\mathrm{inc}}^{\mathrm{cust}} = 0.1$ and $\delta_{\mathrm{dec}}^{\mathrm{cust}} = 0.01$. If the LS found, during last call, a solution $\boldsymbol{\sigma}$ that satisfied $p_{\mathrm{c}}^{\mathrm{sum}}(\boldsymbol{\sigma}) < p_{\mathrm{c}}^{\min}$ and $p_{\mathrm{t}}^{\mathrm{sum}}(\boldsymbol{\sigma}) < p_{\mathrm{t}}^{\min}$, the parameters α and β are decreased by

$$\alpha := \left(1 - \frac{P_{\mathrm{c}}}{\max\{P_{\mathrm{c}}, P_{\mathrm{t}}\}}\delta_{\mathrm{dec}}\right)\alpha, \qquad \beta := \left(1 - \frac{P_{\mathrm{t}}}{\max\{P_{\mathrm{c}}, P_{\mathrm{t}}\}}\delta_{\mathrm{dec}}\right)\beta.$$

Even if the LS did not find such a solution, if $N_{\mathrm{c}} = 0$ (resp, $N_{\mathrm{t}} = 0$) holds, α (resp., β) is decreased by the same equation. Otherwise they are increased by

$$\alpha := \left(1 + \frac{N_{\mathrm{c}}}{\max\{N_{\mathrm{c}}, N_{\mathrm{t}}\}}\delta_{\mathrm{inc}}\right)\alpha, \qquad \beta := \left(1 + \frac{N_{\mathrm{t}}}{\max\{N_{\mathrm{c}}, N_{\mathrm{t}}\}}\delta_{\mathrm{inc}}\right)\beta.$$

5.5.2 Update of the parameters γ_i for each customer i

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

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 ρ (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 $p_{\mathrm{c}}^{\min}$ and the other which achieves $p_{\mathrm{t}}^{\min}$. After a feasible solution is found (i.e., $p_{\mathrm{c}}^{\min} = 0$ and $p_{\mathrm{t}}^{\min} = 0$), 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 $\boldsymbol{\sigma}_{\mathrm{lopt}}$ is exchanged with the worst (with respect to p) solution $\boldsymbol{\sigma}_{\mathrm{worst}}$ in R (excluding the above solutions), provided that $\boldsymbol{\sigma}_{\mathrm{lopt}}$ is not worse than $\boldsymbol{\sigma}_{\mathrm{worst}}$ and is different from all solution in R.

5.6.2 Path relinking operation

A path relinking operation is applied to two solutions $\boldsymbol{\sigma}_{\mathrm{A}}$ (initiating solution) and $\boldsymbol{\sigma}_{\mathrm{B}}$ (guiding solution) randomly chosen from R, where a random perturbation is applied to $\boldsymbol{\sigma}_{\mathrm{B}}$ 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 $\boldsymbol{\sigma}_{\mathrm{B}}$. We use a cyclic operation, which exchanges partial paths between different routes cyclically, as a random perturbation. Note that a cyclic operation with more than two routes is different from any neighborhood operation we use for the LS, and hence the local search does not get the solution back by one move. In the path relinking operation, we focus on route edges which are used in vehicle routes of a solution. Let $dist(\boldsymbol{\sigma}, \boldsymbol{\sigma}')$ be the number of different route edges between two solutions $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}'$. It is not difficult to see that the distance $dist(\boldsymbol{\sigma}, \boldsymbol{\sigma}')$ between two different solutions $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}'$ can be shortened at least one by applying an appropriate 2-opt* operation or intra-route operation to $\boldsymbol{\sigma}$. The path relinking operation generates a sequence of solutions $(\boldsymbol{\sigma}_{\mathrm{A}} = \boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \ldots, \boldsymbol{\sigma}_q, \ldots, \boldsymbol{\sigma}_{\mathrm{B}})$ by repeating the following procedure starting from $q = 1$ until $\boldsymbol{\sigma}_q = \boldsymbol{\sigma}_{\mathrm{B}}$ holds: Let $\boldsymbol{\sigma}_{q+1}$ be the best solution with respect to p among those that satisfy $dist(\boldsymbol{\sigma}_{q+1}, \boldsymbol{\sigma}_{\mathrm{B}}) < dist(\boldsymbol{\sigma}_q, \boldsymbol{\sigma}_{\mathrm{B}})$ and obtainable from $\boldsymbol{\sigma}_q$ by a 2-opt* or intra-route operation, and then let $q := q + 1$.

We call a solution $\boldsymbol{\sigma}_q$ locally minimal in the sequence if $p(\boldsymbol{\sigma}_q) < \min\{p(\boldsymbol{\sigma}_{q-1}), p(\boldsymbol{\sigma}_{q+1})\}$ holds. Let S be the best π (a parameter) solutions among the locally minimal solutions in the sequence. Every solution in S is used as an initial solution of the LS. We set $\pi = 20$ in the experiments. The next path relinking is initiated whenever all solutions in S are exhausted as the starting solutions for the local search.

The proposed algorithm is summarized in Algorithm 16. The algorithm stops when it reaches a given time limit. In Algorithm 16, a call to the local search starting from a solution $\boldsymbol{\sigma}$ is denoted by $LS(\boldsymbol{\sigma})$, whose output is the obtained locally optimal solution.

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

Construct the neighbor lists.
Let R be ρ randomly generated solutions. For each $\boldsymbol{\sigma} \in R$, let $\boldsymbol{\sigma}_{\text{lopt}} := LS(\boldsymbol{\sigma})$ and then let $R := (R \setminus \{\boldsymbol{\sigma}\}) \cup \boldsymbol{\sigma}_{\text{lopt}}$.
Let $S := \emptyset$.
while the stopping criterion is not satisfied **do**
  while $S = \emptyset$ **do**
    Randomly choose two solutions $\boldsymbol{\sigma}_{\text{A}}$ and $\boldsymbol{\sigma}_{\text{B}}$ from R ($\boldsymbol{\sigma}_{\text{A}} \neq \boldsymbol{\sigma}_{\text{B}}$).
    With probability 1/2, apply a cyclic operation to $\boldsymbol{\sigma}_{\text{B}}$.
    Apply the path relinking operation to $\boldsymbol{\sigma}_{\text{A}}$ and $\boldsymbol{\sigma}_{\text{B}}$, and then let S be the set of best π locally minimal solutions in the generated sequence.
  end while
  Randomly choose $\boldsymbol{\sigma} \in S$, and let $S := S \setminus \{\boldsymbol{\sigma}\}$ and $\boldsymbol{\sigma}_{\text{lopt}} := LS(\boldsymbol{\sigma})$.
  Update the penalty weights.
  Choose the worst $\boldsymbol{\sigma}_{\text{worst}} \in R$ among those that satisfy (1) $\boldsymbol{\sigma}_{\text{worst}}$ is not the unique feasible solution in R, (2) $\exists \boldsymbol{\sigma}_{\text{c}} \in R \setminus \{\boldsymbol{\sigma}_{\text{worst}}\}$, $p_{\text{c}}^{\text{sum}}(\boldsymbol{\sigma}_{\text{c}}) \leq p_{\text{c}}^{\text{sum}}(\boldsymbol{\sigma}_{\text{worst}})$ and (3) $\exists \boldsymbol{\sigma}_{\text{t}} \in R \setminus \{\boldsymbol{\sigma}_{\text{worst}}\}$, $p_{\text{t}}^{\text{sum}}(\boldsymbol{\sigma}_{\text{t}}) \leq p_{\text{t}}^{\text{sum}}(\boldsymbol{\sigma}_{\text{worst}})$.
  if $p(\boldsymbol{\sigma}_{\text{lopt}}) \leq p(\boldsymbol{\sigma}_{\text{worst}})$ and $\boldsymbol{\sigma}_{\text{lopt}}$ is different from all solutions in R **then**
    $R := (R \setminus \{\boldsymbol{\sigma}_{\text{worst}}\}) \cup \boldsymbol{\sigma}_{\text{lopt}}$
  end if
end while
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

References		100	200	400	600	800	1000
Hashimoto et al.	CNV	405	**692**	**1381**	2069	2746	3430
(in press) [75]	CTD	57,282	171,223	406,646	847,470	1,444,513	2,204,728
	P4 2.8GHz	17	33	67	100	133	167
Ibaraki et al.	CNV	407	694	1387	2070	2750	3431
(in press) [86]	CTD	57,545	170,484	398,938	825,172	1,421,225	2,155,374
	P4 2.8GHz	17	33	67	100	133	167
Bräysy et al.	CNV	–	695	1391	2084	2776	3465
(2004) [25]	CTD	–	172,406	399,132	820,372	1,384,306	2,133,376
	AMD 700MHz	–	3×2	3×8	3×16	3×26	3×40
Prescott-Gagnon et al.	CNV	405	694	1385	2071	2745	3432
(2007) [127]	CTD	**57,240**	168,556	389,011	800,797	1,391,344	2,096,823
	Opt 2.3GHz	5×30	5×53	5×89	5×105	5×129	5×162
Pisinger and Ropke	CNV	405	694	1385	2071	2758	3438
(2007) [124]	CTD	57,322	169,042	393,210	807,470	1,358,291	2,110,925
	P4 3GHz	10×2	10×8	5×16	5×18	5×23	5×27
Mester and Bräysy	CNV	–	694	1389	2082	2765	3446
(2005) [110]	CTD	–	168,573	390,386	796,172	1,361,586	2,078,110
	P 2GHz	–	8	17	40	145	600
Le Bouthillier et al.	CNV	405	694	1389	2086	2761	3442
(2005) [103]	CTD	57,360	169,959	396,612	809,494	1,443,400	2,133,645
	5×P 850MHz	12	10	20	30	40	50
Le Bouthillier and Crainic	CNV	407	694	1390	2088	2766	3451
(2005) [102]	CTD	57,412	173,061	408,281	836,261	1,475,281	2,225,366
	5×P 850MHz	12	10	20	30	40	50
Gehring and Homberger	CNV	406	696	1392	2079	2760	3446
(2001) [57]	CTD	57,641	179,328	428,489	890,121	1,535,849	2,290,367
	4×P 400MHz	5×14	3×2	3×7	3×13	3×23	3×30
Gehring and Homberger	CNV	415	694	1390	2082	2770	3461
(1999) [56]	CTD	56,946	176,180	412,270	867,010	1,515,120	2,276,390
	4×P 200MHz	5	10	20	30	40	50
Homberger and Gehring	CNV	408	699	1397	2088	2773	3459
(2005) [80]	CTD	57,422	180,602	431,089	890,293	1,516,648	2,288,819
	P 400MHz	5×17	3×2	3×5	3×10	3×18	3×31
Ours	CNV	405	694	1383	**2068**	**2737**	**3420**
	CTD	57,484	169,070	392,507	800,982	1,367,971	2,085,125
	Xeon 2.8GHz	17	33	67	100	133	167

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.[1] 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.

[1] 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].)

Table 5.2: The detailed results of our algorithm for the 100–400-customer instances

r1/100	r2/100	c1/100	c2/100	rc1/100	rc2/100
01/19/1650.80	01/4/1253.23	01/10/828.94	01/3/591.56	01/14/1696.95	01/4/1413.52
02/17/1486.12	02/3/1191.70	02/10/828.94	02/3/591.56	02/12/1554.75	02/3/1367.00
03/13/1292.68	03/3/949.66	03/10/828.06	03/3/591.17	03/11/1261.67	03/3/1068.60
04/9/1007.31	04/2/844.63	04/10/824.78	04/3/590.60	04/10/1135.48	04/3/816.33
05/14/1377.11	05/3/994.43	05/10/828.94	05/3/588.88	05/13/1629.44	05/4/1297.65
06/12/1252.03	06/3/929.49	06/10/828.94	06/3/588.49	06/11/1424.73	06/3/1207.75
07/10/1109.88	07/2/911.14	07/10/828.94	07/3/588.29	07/11/1230.48	07/3/1094.95
08/9/969.30	08/2/727.69	08/10/828.94	08/3/588.32	08/10/1139.82	08/3/841.18
09/11/1194.73	09/3/913.32	09/10/828.94			
10/10/1131.27	10/3/966.90				
11/10/1096.73	11/**2/891.89**				
12/9/1032.47					
r1/200	r2/200	c1/200	c2/200	rc1/200	rc2/200
01/20/4784.11	01/4/4504.88	01/20/2704.57	01/6/1931.44	01/18/3667.40	01/6/3102.30
02/18/4045.33	02/4/3655.26	02/18/2917.89	02/6/1863.16	02/18/3249.65	02/5/2827.43
03/18/3395.72	03/4/2945.24	03/18/2707.35	03/6/1777.56	03/18/3011.09	03/4/2623.02
04/18/3080.53	04/4/2025.06	04/18/2644.42	04/6/1716.20	04/18/2870.29	04/4/2164.55
05/18/4123.47	05/4/3400.49	05/20/2702.05	05/6/1878.85	05/18/3379.51	05/4/2911.46
06/18/3642.30	06/4/2954.85	06/20/2701.04	06/6/1857.35	06/18/3367.31	06/4/2880.06
07/18/3152.45	07/4/2476.50	07/20/2701.04	07/6/1849.46	07/18/3215.33	07/4/2563.62
08/18/3009.65	08/4/1887.98	08/19/2775.48	08/6/1820.53	08/18/3104.40	08/4/2325.73
09/18/3773.41	09/4/3125.55	09/18/2687.83	09/6/1830.05	09/18/3088.57	09/4/2270.31
10/18/3321.50	10/4/2694.35	10/18/2645.08	10/6/1806.58	10/18/3015.06	10/4/2057.02
r1/400	r2/400	c1/400	c2/400	rc1/400	rc2/400
01/40/10407.99	01/8/9297.61	01/40/7152.06	01/12/4116.14	01/36/8925.01	01/**11/6682.37**
02/36/9198.06	02/8/7662.52	02/36/7921.43	02/12/3930.05	02/36/8073.32	02/9/6407.92
03/36/7921.12	03/8/6190.56	03/36/7072.47	03/12/3774.30	03/36/7631.08	03/8/5054.14
04/36/7368.95	04/8/4329.59	04/**36/6803.26**	04/11/3939.40	04/36/7428.82	04/8/3648.30
05/36/9554.74	05/8/7160.08	05/40/7152.06	05/12/3943.03	05/36/8312.17	05/9/6005.94
06/36/8623.44	06/8/6215.73	06/40/7153.45	06/12/3875.94	06/36/8297.14	06/8/6045.96
07/36/7719.15	07/8/5153.95	07/39/7461.24	07/12/3894.16	07/36/8093.52	07/8/5558.01
08/36/7391.15	08/8/4113.46	08/37/7419.34	08/12/3792.76	08/36/7876.78	08/8/4946.25
09/36/8977.98	09/8/6500.77	09/36/7107.59	09/12/3870.80	09/36/7853.69	09/8/4569.14
10/36/8285.40	10/8/5999.98	10/36/6889.23	10/**11/3964.56**	10/36/7687.41	10/8/4350.64

Table 5.3: The detailed results of our algorithm for the 600–1000-customer instances

r1/600	r2/600	c1/600	c2/600	rc1/600	rc2/600
01/59/21713.75	01/11/18774.60	01/60/14095.64	01/18/7774.16	01/55/17586.75	01/14/13691.04
02/54/19493.96	02/11/15106.93	02/56/14604.33	02/**17/8347.09**	02/55/16233.10	02/12/11763.41
03/54/17471.73	03/11/11599.34	03/56/13850.66	03/17/7666.95	03/55/15413.87	03/11/9807.61
04/54/16037.12	04/11/8285.24	04/56/13628.62	04/**17/6983.26**	04/55/15000.79	04/11/7377.47
05/54/20681.01	05/11/15486.82	05/60/14085.72	05/18/7575.20	05/55/17045.97	05/12/12405.59
06/54/18616.76	06/11/12779.53	06/60/14089.66	06/18/7479.48	06/55/17136.28	06/11/12491.87
07/54/17114.55	07/11/10389.22	07/58/15069.55	07/18/7517.63	07/55/16511.30	07/**11/10928.52**
08/54/15884.43	08/11/7969.11	08/56/14797.70	08/**17/7694.69**	08/55/16237.07	08/11/10436.44
09/54/19837.64	09/11/13871.90	09/56/13735.89	09/**17/8465.73**	09/55/16325.51	09/11/10096.96
10/54/18383.25	10/11/12527.23	10/56/13677.35	10/**17/7280.70**	10/55/16034.97	10/11/9413.25
r1/800	r2/800	c1/800	c2/800	rc1/800	rc2/800
01/80/37400.64	01/15/28839.78	01/80/25184.38	01/24/11662.08	01/**72/34551.38**	01/**18/21154.34**
02/72/33573.64	02/15/23157.11	02/**72/27012.87**	02/23/12460.76	02/**72/31308.67**	02/16/18799.25
03/72/30349.76	03/15/18265.81	03/72/24558.69	03/23/11770.80	03/**72/29152.99**	03/15/14939.88
04/72/28459.74	04/15/13759.46	04/72/23959.50	04/23/11160.68	04/**72/27449.31**	04/15/11410.24
05/72/34855.19	05/15/24779.24	05/80/25166.28	05/24/11428.66	05/**72/34173.25**	05/**15/19497.33**
06/72/31798.47	06/15/20775.72	06/80/25160.85	06/**23/12673.80**	06/**72/32434.12**	06/15/18769.39
07/72/29655.26	07/15/17102.33	07/78/26003.16	07/**24/11370.84**	07/**72/32064.83**	07/15/17196.55
08/72/28349.07	08/15/13059.96	08/74/25844.26	08/**23/11363.96**	08/**72/31042.27**	08/15/16239.34
09/72/33468.43	09/15/23017.30	09/72/24793.22	09/**23/11835.04**	09/**72/30910.06**	09/**15/15556.86**
10/72/31871.25	10/15/21074.60	10/72/24522.58	10/**23/11163.36**	10/**72/30348.47**	10/15/14726.07
r1/1000	r2/1000	c1/1000	c2/1000	rc1/1000	rc2/1000
01/100/54955.74	01/19/43054.76	01/100/42478.95	01/30/16879.24	01/90/48248.59	01/**20/30912.50**
02/91/52208.39	02/19/34293.42	02/**90/43355.70**	02/29/17452.36	02/90/45344.22	02/19/26597.14
03/91/46574.96	03/19/25934.52	03/90/40548.32	03/**28/17519.99**	03/90/43261.35	03/18/20698.99
04/91/43696.02	04/19/18629.76	04/90/39908.25	04/**28/16783.84**	04/90/42625.25	04/18/16402.21
05/91/55002.93	05/19/37357.94	05/100/42469.18	05/30/16563.10	05/90/47247.58	05/**18/27715.20**
06/91/50124.16	06/19/30879.06	06/100/42471.28	06/**29/17491.11**	06/90/46651.17	06/18/28256.73
07/91/46193.38	07/19/24075.04	07/**97/43867.54**	07/**29/18727.92**	07/90/46061.01	07/18/25763.52
08/91/43264.54	08/19/18229.89	08/93/43120.86	08/**28/16839.40**	08/90/45524.59	08/18/24454.85
09/91/53848.20	09/19/34224.22	09/90/42731.02	09/**29/16680.50**	09/90/45508.55	09/18/23802.84
10/91/50639.84	10/19/31390.03	10/90/40624.36	10/**28/16584.54**	10/90/44918.74	10/**18/22365.07**

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.

Bibliography

[1] E. Aarts and M. Verhoeven. Local search. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, chapter 11, pages 163–180. John Wiley and Sons, 1997.

[2] E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley and Sons, 1997.

[3] E. H. L. Aarts, P. J. M. van Laarhoven, C. L. Liu, and P. Pan. VLSI layout synthesis. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 415–440. John Wiley and Sons, 1997.

[4] R. Agarwal, R. K. Ahuja, G. Laporte, and Z.-J. M. Shen. A composite very large-scale neighborhood search algorithm for the vehicle routing problem. In J. Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 49. Chapman and Hall/CRC, 2004.

[5] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123:75–102, 2002.

[6] R. K. Ahuja, D. S. Hochbaum, and J. B. Orlin. Solving the convex cost integer dual network flow problem. *Management Science*, 49(7):950–964, 2003.

[7] R. K. Ahuja and J. B. Orlin. A fast scaling algorithm for minimizing separable convex functions subject to chain constraints. *Operations Research*, 49(5):784–789, 2001.

[8] E. J. Anderson, C. A. Glass, and C. N. Potts. Machine scheduling. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 361–414. John Wiley and Sons, 1997.

[9] E. Angel. A survey of approximation results for local search algorithms. In E. Bampis, K. Jansen, and C. Kenyon, editors, *Efficient Approximation and Online*

Algorithms: Recent Progress on Classical Combinatorial Optimization Problems and New Applications, Lecture Notes in Computer Science, volume 3484, pages 30–73. Springer, 2006.

[10] E. M. Arkin and R. Hassin. On local search for weighted k-set packing. *Mathematics of Operations Research*, 23(3):640–648, 1998.

[11] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[12] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[13] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.

[14] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[15] R. Bent and P. Van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38:515–530, 2004.

[16] R. Bent and P. Van Hentenryck. Randomized adaptive spatial decoupling for large-scale vehicle routing with time windows. In *AAAI*, pages 173–178. AAAI Press, 2007.

[17] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[18] J. Berger, M. Barkaoui, and O. Bräysy. A route-directed hybrid genetic approach for the vehicle routing problem with time windows. *INFOR*, 41(2):179–194, 2003.

[19] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1995.

[20] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[21] J. Bramel and D. Simchi-Levi. Probabilistic analyses and practical algorithms for the vehicle routing problem with time windows. *Operations Research*, 44(3):501–509, 1996.

[22] O. Bräysy. A reactive variable neighborhood search for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 15(4):347–368, 2003.

[23] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part I: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.

[24] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part II: Metaheuristics. *Transportation Science*, 39(1):119–139, 2005.

[25] O. Bräysy, G. Hasle, and W. Dullaert. A multi-start local search algorithm for the vehicle routing problem with time windows. *European Journal of Operational Research*, 159:586–605, 2004.

[26] A. M. Campbell and M. Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, 38(3):369–378, 2004.

[27] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, page 441. Academic Press, 1976.

[28] H. K. Chung and J. P. Norback. A clustering and insertion heuristic applied to a large routeing problem in food distribution. *The Journal of the Operational Research Society*, 42(7):555–564, 1991.

[29] V. Chvátal. *Linear Programming*. W. H. Freeman, 1983.

[30] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[31] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[32] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[33] P. Crescenzi and V. Kann. A compendium of NP optimization problems. Technical report, Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza", 1995. http://www.nada.kth.se/˜viggo/problemlist/compendium.html.

[34] G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

[35] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[36] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[37] J. S. Davis and J. J. Kanet. Single-machine scheduling with early and tardy completion costs. *Naval research logistics*, 40:85–101, 1993.

[38] M. De Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 1997.

[39] G. Desaulniers, J. Desrosiers, A. Erdmann, M. M. Solomon, and F. Soumis. VRP with pickup and delivery. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 9, pages 225–242. Society for Industrial and Applied Mathematics, 2002.

[40] G. Desaulniers, J. Desrosiers, I. Ioachim, M. M. Solomon, F. Soumis, and D. Villeneuve. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. G. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, pages 57–93. Kluwer, Boston, 1998.

[41] M. Desrochers, J. K. Lenstra, M. W. P. Savelsbergh, and F. Soumis. Vehicle routing with time windows: Optimization and approximation. In B. L. Golden and A. A. Assad, editors, *Vehicle Routing: Methods and Studies*, Studies in management science and systems, pages 65–84. North-Holland, Amsterdam, 1988.

[42] J. Desrosiers, Y. Dumas, M. M. Solomon, and F. Soumis. Time constrained routing and scheduling. In M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 2, pages 35–139. North-Holland, Amsterdam, 1995.

[43] J. Dréo, P. Siarry, A. Pétrowski, and E. Taillard. *Metaheuristics for Hard Optimization: Methods and Case Studies*. Springer, 2006.

[44] Y. Dumas, F. Soumis, and J. Desrosiers. Optimizing the schedule for a fixed vehicle path with convex inconvenience costs. *Transportation Science*, 24(2):145–152, 1990.

[45] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[46] P. G. Eibl, R. Mackenzie, and D. B. Kidner. Vehicle routeing and scheduling in the brewing industry: A case study. *International Journal of Physical Distribution and Logistics Management*, 24(6):27–37, 1994.

[47] Ö. Ergun. *New neighborhood search algorithms based on exponentially large neighborhoods*. PhD thesis, Massachusetts Institute of Technology, 2001.

[48] Ö. Ergun, J. B. Orlin, and A. Steele-Feldman. Creating very large scale neighborhoods out of smaller ones by compounding moves. *Journal of Heuristics*, 12:115–140, 2006.

[49] M. Fisher. Vehicle routing. In M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 1, pages 1–33. North-Holland, Amsterdam, 1995.

[50] M. L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11:109–124, 1981.

[51] M. L. Fredman, D. Johnson, L. A. McGeoch, and G. Ostheimer. Data structures for traveling salesmen. *Journal of Algorithms*, 18:432–479, 1995.

[52] B. Funke, T. Grünert, and S. Irnich. Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of Heuristics*, 11:267–306, 2005.

[53] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[54] M. R. Garey, R. E. Tarjan, and G. T. Wilfong. One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research*, 13(2):330–348, 1988.

[55] W. W. Garvin, H. W. Crandall, J. B. John, and R. A. Spellman. Applications of linear programming in the oil industry. *Management Science*, 3(4):407–430, 1957.

[56] H. Gehring and J. Homberger. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In *Proceedings of EUROGEN99*, pages 57–64, 1999.

[57] H. Gehring and J. Homberger. A parallel two-phase metaheuristic for routing problems with time-windows. *Asia-Pacific Journal of Operational Research*, 18:35–47, 2001.

[58] H. Gehring and J. Homberger. Parallelization of a two-phase metaheuristic for routing problems with time windows. *Journal of Heuristics*, 8:251–276, 2002.

[59] M. Gendreau, G. Laporte, and J.-Y. Potvin. Vehicle routing: modern heuristics. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 311–336. John Wiley and Sons, 1997.

[60] M. Gendreau and J.-Y. Potvin. Metaheuristics in combinatorial optimization. *Annals of Operations Research*, 140:189–213, 2005.

[61] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[62] F. Glover. Tabu search–part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[63] F. Glover. Tabu search–part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[64] F. Glover. Tabu search and adaptive memory programming – advances, applications and challenges. In R. S. Barr, R. V. Helgason, and J. L. Kennington, editors, *Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies*, pages 1–75. Kluwer Academic Publishers, 1997.

[65] F. Glover and G. A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

[66] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[67] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–684, 2000.

[68] F. Glover, M. Laguna, and R. Martí. Scatter search and path relinking: Advances and applications. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 1–35. Kluwer Academic Publishers, 2003.

[69] B. L. Golden, A. A. Assad, and E. A. Wasil. Routing vehicles in the real world: Applications in the solid waste, beverage, food, dairy, and newspaper industries. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 11, pages 245–286. Society for Industrial and Applied Mathematics, 2002.

[70] B. L. Golden and E. A. Wasil. Metaheuristics. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of Applied Optimization*, pages 123–130. Oxford University Press New York, 2002.

[71] L. Gouveia and S. Voß. A classification of formulations for the (time-dependent) traveling salesman problem. *European Journal of Operational Research*, 83:69–92, 1995.

[72] M. M. Halldórsson. Approximating k-set cover and complementary graph coloring. In *Proceedings of the Fifth International Integer Programming and Combinatorial Optimization Conference, Lecture Notes in Computer Science*, volume 1084, pages 118–131, 1996.

[73] S. Hart and M. Sharir. Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes. *Combinatorics*, 6:151–177, 1986.

[74] H. Hashimoto, T. Ibaraki, S. Imahori, and M. Yagiura. The vehicle routing problem with flexible time windows and traveling times. *Discrete Applied Mathematics*, 154:2271–2290, 2006.

[75] H. Hashimoto, M. Yagiura, and T. Ibaraki. An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. *Discrete Optimization*, in press.

[76] D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems.* PWS Publishing Company, 1997.

[77] D. S. Hochbaum and M. Queyranne. Minimizing a convex cost closure set. *SIAM Journal on Discrete Mathematics*, 16(2):192–207, 2003.

[78] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* University of Michigan Press, 1975.

[79] J. Homberger and H. Gehring. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR*, 37(3):297–318, 1999.

[80] J. Homberger and H. Gehring. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, 162:220–238, 2005.

[81] I. S. Honkala and P. R. J. Östergård. Code design. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 441–456. John Wiley and Sons, 1997.

[82] H. H. Hoos and T. Stützle. *Stochastic local search: foundations and applications.* Morgan Kaufmann, 2005.

[83] T. Ibaraki. *Enumerative Approaches to Combinatorial Optimization: part I*, volume 10 of *Annals of Operations Research.* J.C. Baltzer AG, 1987.

[84] T. Ibaraki. *Enumerative Approaches to Combinatorial Optimization: part II*, volume 11 of *Annals of Operations Research*. J.C. Baltzer AG, 1987.

[85] T. Ibaraki, S. Imahori, M. Kubo, T. Masuda, T. Uno, and M. Yagiura. Effective local search algorithms for routing and scheduling problems with general time-window constraints. *Transportation Science*, 39(2):206–232, 2005.

[86] T. Ibaraki, S. Imahori, K. Nonobe, K. Sobue, T. Uno, and M. Yagiura. An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. *Discrete Applied Mathematics*, In press.

[87] S. Ichoua, M. Gendreau, and J.-Y. Potvin. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, 144:379–396, 2003.

[88] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley and Sons, 1997.

[89] D. S. Johnson, C. H. Papadimtriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37:79–100, 1988.

[90] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[91] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[92] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49:291–307, 1970.

[93] S. Khuller, R. Bhatia, and R. Pless. On local search and placement of meters in networks. *SIAM Journal on Computing*, 32(2):470–487, 2003.

[94] G. A. P. Kindervater and M. W. P. Savelsbergh. Vehicle routing: handling edge exchanges. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 337–360. John Wiley and Sons, 1997.

[95] V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic Press, 1972.

[96] J. Könemann and R. Ravi. A matter of degree: improved approximation algorithms for degree-bounded minimum spanning trees. In *Proceedings of the Thirty-Second Annual ACM symposium on Theory of Computing*, pages 537–546, New York, NY, USA, 2000. ACM.

[97] Y. A. Koskosidis, W. B. Powell, and M. M. Solomon. An optimization-based heuristic for vehicle routing and scheduling with soft time window constraints. *Transportation Science*, 26(2):69–85, 1992.

[98] M. W. Krentel. Structure in locally optimal solutions. In *Proceedings of Thirtieth Annual IEEE Symposium on Foundations of Computer Science*, pages 216–221, 1989.

[99] M. W. Krentel. On finding and verifying locally optimal solutions. *SIAM Journal on Computing*, 19(4):742–749, 1990.

[100] G. Laporte. Vehicle routing. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, chapter 14, pages 223–240. John Wiley and Sons, 1997.

[101] G. Laporte and I. H. Osman. Routing problems: A bibliography. *Annals of Operations Research*, 61:227–262, 1995.

[102] A. Le Bouthillier and T. G. Crainic. A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Computers and Operations Research*, 32:1685–1708, 2005.

[103] A. Le Bouthillier, T. G. Crainic, and P. Kropf. A guided cooperative search for the vehicle routing problem with time windows. *IEEE Intelligent Systems*, 20(4):36–42, 2005.

[104] H. Li and A. Lim. Local search with annealing-like restarts to solve the VRPTW. *European Journal of Operational Research*, 150:115–127, 2003.

[105] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44:2245–2269, 1965.

[106] S. Lin and B. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[107] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, 2003.

[108] C. Malandraki and R. B. Dial. A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operational Research*, 90:44–55, 1996.

[109] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley and Sons, 1990.

[110] D. Mester and O. Bräysy. Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Computers and Operations Research*, 32:1593–1614, 2005.

[111] J. E. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of Applied Optimization*, pages 65–78. Oxford University Press New York, 2002.

[112] Y. Nagata. Effective memetic algorithm for the vehicle routing problem with time windows: Edge assembly crossover for the VRPTW. In *Proceedings of the Seventh Metaheuristics International Conference (MIC2007)*, 2007.

[113] G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*. John Wiley and Sons, 1988.

[114] T. A. J. Nicholson. A sequential method for discrete optimization problems and its application to the assignment, travelling salesman, and three machine scheduling problems. *Journal of the Institute of Mathematics and its Applications*, 3:362–375, 1967.

[115] J. Oppen and A. Løkketangen. Transportation of livestock to slaughterhouse. In *Proceedings of the Sixth Metaheuristics International Conference (MIC2005)*, pages 719–724, 2005.

[116] I. Or. *Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking*. PhD thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, 1976.

[117] J. B. Orlin, A. P. Punnen, and A. S. Schulz. Approximate local search in combinatorial optimization. *SIAM Journal on Computing*, 33(5):1201–1214, 2004.

[118] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

[119] I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:511–623, 1996.

[120] E. Page. On Monte Carlo methods in congestion problems: I. searching for an optimum in discrete situations. *Operations Research*, 13(2):291–299, 1965.

[121] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover, 1998.

[122] N. Park, H. Okano, and H. Imai. A path-exchange-type local search algorithm for vehicle routing and its efficient search strategy. *Journal of the Operations Research Society of Japan*, 43(1):197–208, 2000.

[123] J.-C. Picard and M. Queyranne. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations research*, 26(1):86–110, 1978.

[124] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34:2403–2435, 2007.

[125] J.-Y. Potvin, T. Kervahut, B.-L. Garcia, and J.-M. Rousseau. The vehicle routing problem with time windows part I: tabu search. *INFORMS Journal on Computing*, 8(2):158–164, 1996.

[126] J.-Y. Potvin and J.-M. Rousseau. An exchange heuristic for routeing problems with time windows. *The Journal of the Operational Research Society*, 46:1433–1446, 1995.

[127] E. Prescott-Gagnon, G. Desaulniers, and L.-M. Rousseau. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. Technical report, GERAD (Group for Research in Decision Analysis), 2007.

[128] J. Privé, J. Renaud, F. Boctor, and G. Laporte. Solving a vehicle-routing problem arising in soft-drink distribution. *Journal of the Operational Research Society*, 57:1045–1052, 2006.

[129] S. Reiter and G. Sherman. Discrete optimizing. *Journal of the Society for Industrial and Applied Mathematics*, 13(3):864–889, 1965.

[130] S. Ropke. *Heuristic and exact algorithms for vehicle routing problems*. PhD thesis, Computer science department at the University of Copenhagen (DIKU), 2005.

[131] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977.

[132] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

[133] M. W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4(2):146–154, 1992.

[134] A. A. Schäffer and M. Yannakakis. Simple local search problems that are hard to solve. *SIAM Journal on Computing*, 20(1):56–87, 1991.

[135] A. Schrijver. *Theory of linear and integer programming.* John Wiley and Sons, 1986.

[136] T. R. Sexton and L. D. Bodin. Optimizing single vehicle many-to-many operations with desired delivery times: I. scheduling. *Transportation Science*, 19(4):378–410, 1985.

[137] M. Sigurd, D. Pisinger, and M. Sig. Scheduling transportation of live animals to avoid the spread of diseases. *Transportation Science*, 38(2):197–209, 2004.

[138] SINTEF. http://www.top.sintef.no/vrp/benchmarks.html.

[139] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985.

[140] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[141] M. M. Solomon and J. Desrosiers. Time window constrained routing and scheduling problems. *Transportation Science*, 22(1):1–13, 1988.

[142] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186, 1997.

[143] É. D. Taillard, L. M. Gambardella, M. Gendreau, and J.-Y. Potvin. Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research*, 135:1–16, 2001.

[144] H. Tamaki, T. Komori, and S. Abe. A heuristic approach to parallel machine scheduling with earliness and tardiness penalties. In *Proceedings of the Seventh IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '99)*, pages 1367–1370, Barcelona, Catalonia, Spain, 1999.

[145] H. Tamaki, T. Sugimoto, and M. Araki. Parallel machine scheduling problem with a non-regular objective function. minimizing the sum of earliness and tardiness penalties (in Japanese). *Transactions of the Society of Instrument and Control Engineers*, 35(9):1176–1182, 1999.

[146] P. M. Thompson and J. B. Orlin. The theory of cyclic transfers. Technical report, Massachusetts Institute of Technology, Operations Research Center, 1989.

[147] P. M. Thompson and H. N. Psaraftis. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research*, 41(5):935–946, 1993.

[148] P. Toth and D. Vigo. An overview of vehicle routing problems. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 1, pages 1–26. Society for Industrial and Applied Mathematics, 2002.

[149] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2002.

[150] P. Toth and D. Vigo. VRP with backhauls. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 8, pages 195–224. Society for Industrial and Applied Mathematics, 2002.

[151] R. J. M. Vaessens, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by local search. *INFORMS Journal on Computing*, 8(3):302–317, 1996.

[152] P. M. Vaidya. A new algorithm for minimizing convex functions over convex sets. *Mathematical Programming*, 73(3):291–341, 1996.

[153] A. van Vliet, C. G. E. Boender, and A. H. G. Rinnooy Kan. Interactive optimization of bulk sugar deliveries. *Interfaces*, 22(3):4–14, 1992.

[154] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[155] S. Voß. Meta-heuristics: The state of the art. In A. Nareyek, editor, *Proceedings of the International Workshop on Local Search for Planning and Scheduling, Lecture Notes in Computer Science*, volume 2148, pages 1–23, 2001.

[156] D. J. A. Welsh. *Matroid Theory*. Academic Press, 1976.

[157] M. Yagiura and T. Ibaraki. On metaheuristic algorithms for combinatorial optimization problems. *Systems and Computers in Japan*, 32(3):33–55, 2001.

[158] M. Yagiura and T. Ibaraki. Local search. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of Applied Optimization*, pages 104–123. Oxford University Press New York, 2002.

[159] M. Yannakakis. The analysis of local search problems and their heuristics. In *Proceedings of the Seventh Annual Symposium on Theoretical Aspects of Computer Science*, pages 298–311, 1990.

[160] M. Yannakakis. Computational complexity. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 19–55. John Wiley and Sons, 1997.

A List of Author's Work

Journals

1. H. Hashimoto, T. Ibaraki, S. Imahori, and M. Yagiura, "The Vehicle Routing Problem with Flexible Time Windows and Traveling Times," *Discrete Applied Mathematics*, 154 (2006) 2271-2290.

2. H. Hashimoto, M. Yagiura and T. Ibaraki, "An Iterated Local Search Algorithm for the Time-Dependent Vehicle Routing Problem with Time Windows," *Discrete Optimization*, (in press).

International Conferences with Review

1. H. Hashimoto, T. Ibaraki, S. Imahori, and M. Yagiura, "A Local Search Algorithm for Routing and Scheduling Problems with Time Window and Traveling Time Constraints," *Proceedings of International Symposium on Scheduling* (2004) 143–146.

2. H. Hashimoto, M. Yagiura and T. Ibaraki, "An Iterated Local Search Algorithm for the Time-Dependent Vehicle Routing Problem with Time Windows," *Proceedings of the 6th Metaheuristics International Conference (MIC 2005)* (2005) 506-513.

3. S. Boussier, H. Hashimoto, M. Vasquez, "A Greedy Randomized Adaptive Search Procedure for Technicians and Interventions Scheduling for Telecommunications," *The 7th Metaheuristics International Conference (MIC 2007)* (2007).

4. H. Hashimoto, Y. Ezaki, M. Yagiura, K. Nonobe, T. Ibaraki, and A. Løkketangen, "A Set Covering Approach for the Pickup and Delivery Problem with General Constraints on Each Route," *Proceedings of Engineering Stochastic Local Search Algorithms: Designing, Implementing and Analyzing Effective Heuristics (SLS 2007)*, LNCS 4638 (2007) 192–196.

5. H. Hashimoto and M. Yagiura, "Path Relinking Approach with an Adaptive Mechanism to Control Parameters for the Vehicle Routing Problem with Time Windows," *8th European Conference on Evolutionary Computation in Combinatorial Optimization*, LNCS (to appear), March 26-28 2008 (Naples/Napoli, Italy)

Unpublished Manuscripts

1. H. Hashimoto, Y. Ezaki, M. Yagiura, K. Nonobe, T. Ibaraki and A. Løkketangen, "A Set Covering Approach for the Pickup and Delivery Problem with General Constraints on Each Route," submitted for publication.

2. H. Hashimoto, S. Boussier, M. Vasquez and C. Wilbaut, "A GRASP-Based Approach for Technicians and Interventions Scheduling for Telecommunications," submitted for publication.