MATHEMATICAL ENGINEERING
TECHNICAL REPORTS

An Iterated Local Search Algorithm for the Vehicle Routing Problem with Convex Time Penalty Functions

Toshihide Ibaraki, Shinji Imahori, Koji Nonobe,
Kensuke Sobue, Takeaki Uno, Mutsunori Yagiura

METR 2006–36 June 2006

DEPARTMENT OF MATHEMATICAL INFORMATICS
GRADUATE SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY
THE UNIVERSITY OF TOKYO
BUNKYO-KU, TOKYO 113-8656, JAPAN

WWW page: http://www.i.u-tokyo.ac.jp/mi/mi-e.htm

An Iterated Local Search Algorithm for the Vehicle Routing Problem with Convex Time Penalty Functions

Toshihide Ibaraki [1] Shinji Imahori [2] Koji Nonobe [3]
Kensuke Sobue [4] Takeaki Uno [5] Mutsunori Yagiura [6]

[1] Department of Informatics, School of Science and Technology,
Kwansei Gakuin University, Sanda 669-1337, Japan.
E-mail: ibaraki@ksc.kwansei.ac.jp

[2] Department of Mathematical Informatics, Graduate School of Information Science and Technology,
University of Tokyo, Tokyo 113-8656, Japan.
E-mail: imahori@simplex.t.u-tokyo.ac.jp

[3] Department of Art and Technology, Faculty of Engineering,
Hosei University, Koganei 184-8584, Japan.
E-mail: nonobe@k.hosei.ac.jp

[4] Toyota Motor Corporation, Toyota 471-8571, Japan.

[5] National Institute of Informatics, Tokyo 101-8430, Japan.
E-mail: uno@nii.jp

[6] Department of Computer Science and Mathematical Informatics,
Graduate School of Information Science, Nagoya University, Nagoya 464-8603, Japan.
E-mail: yagiura@nagoya-u.jp

Abstract: We propose an iterated local search algorithm for the vehicle routing problem with time window constraints. We treat the time window constraint for each customer as a penalty function, and assume that it is convex and piecewise linear. Given an order of customers each vehicle to visit, dynamic programming (DP) is used to determine the optimal start time to serve the customers so that the total time penalty is minimized. This DP algorithm is then incorporated in the iterated local search algorithm to efficiently evaluate solutions in various neighborhoods. The amortized time complexity of evaluating a solution in the neighborhoods is a logarithmic order of the input size (i.e., the total number of linear pieces that define the penalty functions). Computational comparisons on benchmark instances with up to 1000 customers show that the proposed method is quite effective, especially for large instances.

Keywords: Vehicle routing problem with time windows, metaheuristics, dynamic programming

1 Introduction

The *vehicle routing problem* (VRP) is the problem of minimizing the total distance traveled by a number of vehicles, under various constraints, where each customer must be visited exactly once by a vehicle. This is one of the representative combinatorial optimization problems and is known to be NP-hard. Among variants of VRP, the VRP with capacity and time window constraints, called the *vehicle routing problem with time windows* (VRPTW), has been widely studied in the last decade. The capacity constraint signifies that the total load on a route cannot

exceed the capacity of the vehicle serving the route. The time window constraint signifies that each vehicle must start the service at each customer in the period specified by the customer. The VRPTW has a wide range of applications such as bank deliveries, postal deliveries, school bus routing and so on. For an extensive survey on heuristic and metaheuristic approaches for the VRPTW, see references [5, 6] by Bräysy and Gendreau.

If the capacity and time window constraints must be satisfied strictly, such problem is called the VRPHTW (H stands for hard). For this problem, even just finding a feasible solution with a given number of vehicles is known to be NP-complete, because it includes the (one-dimensional) bin packing problem [10] as a special case. Thus, it is inefficient to search only within the feasible region of the VRPHTW, especially when the constraints are tight. Moreover, in many real-world situations, these constraints can be violated to some extent. Considering these, we treat these two types of constraints as soft (i.e., can be violated) in this paper. This problem is called the VRPSTW (S stands for soft), and the amount of violation of soft constraints is penalized by using penalty functions and added to the objective function. In this case, it is not trivial to determine the optimal start time of services at all customers so that the total penalty is minimized after fixing the order of customers for each vehicle to visit.

The time penalty function is the function that penalizes the amount of violation of time window constraints for customers. In most of the previous work for the VRPTW [23, 28, 29, 31], only one time window is allowed for each customer and in this case the time penalty function is convex. In the literature [8, 11, 23, 31], algorithms for particular convex time penalty functions were proposed in order to treat the time window constraint as a soft constraint. In [31], the time penalty for each customer is $+\infty$ for earliness and linear for tardiness, and an $\mathbf{O}(1)$ time algorithm to compute approximately the optimal time penalty for a solution in neighborhoods was proposed. In [8, 23], the time penalty is linear for both of earliness and tardiness, and $\mathbf{O}(n_k^2)$ time algorithms to compute the penalty of a given route of vehicle k were proposed, where n_k is the number of customers assigned to vehicle k. If the time penalty function for each customer is the absolute deviation from a specified time, the problem to determine the optimal start time of services at all customers is called the *isotonic median regression problem*, which has been extensively studied. To our knowledge, the best time complexity for this problem (for a vehicle k) is $\mathbf{O}(n_k \log n_k)$ [1, 11, 17].

The problem with multiple time windows [9, 20] and other variants of VRPTW [16, 30] have also been considered. In our previous paper [20], the time penalty function can be non-convex and discontinuous as long as piecewise linear. Let δ_k be the total number of linear pieces of the time penalty functions for the depot and all customers assigned to vehicle k, and let $\delta_{\max}$ be the maximum of δ_k among all vehicles. We proposed a dynamic programming (DP) algorithm that runs in $\mathbf{O}(n_k \delta_k)$ time when the problem of minimizing the time penalty of vehicle k is solved from scratch. This DP algorithm was incorporated into metaheuristic algorithms based on local search. We also designed a sophisticated data structure with which the optimal time penalty of a solution in neighborhoods is computed in $\mathbf{O}(\delta_{\max})$ amortized time. However, this computation time is still expensive when the number of customers becomes large, since δ_k usually depends linearly on the number of customers assigned to vehicle k. Moreover, it is observed in many practical situations that each customer has only one time window. In this paper, hence, the time penalty function is assumed to be convex and piecewise linear.

One of the main contribution of this paper is to propose an efficient algorithm to deal with convex time penalty functions. We use the DP technique in order to compute the optimal time penalty of a vehicle that serves customers in a specified order. The time complexity of our DP algorithm for convex time penalty functions is $\mathbf{O}(\delta_k \log \delta_k)$ for each vehicle k, while the time to evaluate a solution in various neighborhoods is $\mathbf{O}(\log \delta_{\max})$ amortized time.

The essential part of the VRP, i.e., assigning customers to vehicles and determining the visiting order of each vehicle, is determined by a local search (LS) algorithm. We use basic neighborhoods such as the cross exchange and 2-opt*, limiting the neighborhood sizes by using parameters. The LS based on these neighborhoods is then incorporated in the framework of metaheuristics. Among many possible metaheuristics based on the LS, we utilize the iterated local search (ILS) [21]. As it is not easy to specify appropriate weights of constraints a priori, we introduce a mechanism of adaptively controlling penalty weights into the ILS, which turns out to be very effective.

To see the performance of our algorithm, we conduct computational experiments on representative benchmark instances of the VRPTW: (1) Solomon's benchmark instances [29] and (2) Gehring and Homberger's benchmark instances [13]. For Solomon's instances, the solution quality of our algorithm is competitive with those of recent algorithms developed for the VRPTW. For Gehring and Homberger's instances, our results are the best among the tested algorithms. This tendency becomes clearer for larger scale instances such as ones with 800 or 1000 customers. It should be pointed out that these benchmark instances are special cases of the instances that our algorithm can treat.

The remainder of this paper is organized as follows. In Section 2, we define the problem that we consider in this paper. In Section 3, we explain the problem to determine the optimal start time of services for a given route, and propose a DP algorithm. In Section 4, we propose local search and metaheuristic algorithms for finding a good set of routes. In Section 5, we propose some ideas to accelerate the speed of our local search algorithm. In Section 6, we show our computational results on representative benchmark instances with up to 1000 customers. We give concluding remarks and discuss some extensions of our algorithm in Section 7.

2 Problem

In this section, we formulate the vehicle routing problem with convex time penalty functions. Let $G = (V, E)$ be a complete directed graph with a vertex set $V = \{0, 1, \ldots, n\}$ and an edge set $E = \{(i, j) \mid i, j \in V, i \neq j\}$, and $M = \{1, 2, \ldots, m\}$ be a set of vehicles. Vertex 0 is the depot and other vertices are customers. The following parameters are associated with each customer $i \in V \setminus \{0\}$, the depot, each edge $(i, j) \in E$, and each vehicle $k \in M$:

- an amount q_i (≥ 0) of the resource to be delivered from the depot to customer i,
- a service time u_i (≥ 0) at customer i,
- a time penalty function $p_i(t)$ (≥ 0) of the start time t of the service for customer i,
- a time penalty function $p_0^{\mathrm{d}}(t)$ (≥ 0) of the departure time t from the depot,
- a time penalty function $p_0^{\mathrm{a}}(t)$ (≥ 0) of the arrival time t at the depot,
- a distance d_{ij} (≥ 0) for edge (i, j),
- a travel time t_{ij} (≥ 0) for edge (i, j),
- a capacity Q_k (≥ 0) for vehicle k.

Distances d_{ij} and travel times t_{ij} are asymmetric in general; i.e., $d_{ij} \neq d_{ji}$ and $t_{ij} \neq t_{ji}$ may hold. Each time penalty function $p_i(t)$, $p_0^{\mathrm{d}}(t)$ or $p_0^{\mathrm{a}}(t)$ is nonnegative, convex and piecewise linear.

For convenience, we assume $\min_t p_i(t) = 0$, $\min_t p_0^{\mathrm{d}}(t) = 0$ and $\min_t p_0^{\mathrm{a}}(t) = 0$ without loss of generality. It is also assumed that each function is represented by a linked list of linear pieces.

We define a *route* by a sequence of the customers served by one vehicle. Let σ_k denote the route traveled by vehicle k, and $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_m)$. We denote by $\sigma_k(h)$ the hth customer in σ_k, and we define $\sigma_k(0) = \sigma_k(n_k + 1) = 0$, where n_k is the number of customers for vehicle $k \in M$ (i.e., each vehicle k starts from the depot, visits n_k customers, and comes back to the depot). We define 0-1 variables $y_{ik}(\boldsymbol{\sigma}) \in \{0, 1\}$ for $i \in V \setminus \{0\}$ and $k \in M$ by

$$y_{ik}(\boldsymbol{\sigma}) = 1 \iff \exists h \in \{1, 2, \ldots, n_k\},\ i = \sigma_k(h).$$

That is, $y_{ik}(\boldsymbol{\sigma}) = 1$ if and only if vehicle k visits customer i. Moreover, let s_i be the start time of service at customer i, s_k^{d} be the departure time of vehicle k from the depot and s_k^{a} be the arrival time of vehicle k at the depot, and let $\boldsymbol{s} = (s_1, s_2, \ldots, s_n, s_1^{\mathrm{d}}, s_2^{\mathrm{d}}, \ldots, s_m^{\mathrm{d}}, s_1^{\mathrm{a}}, s_2^{\mathrm{a}}, \ldots, s_m^{\mathrm{a}})$. Note that each vehicle is allowed to wait at customers before starting services. The total traveling cost $d_{\mathrm{sum}}(\boldsymbol{\sigma})$ of all vehicles, the total penalty $p_{\mathrm{sum}}(\boldsymbol{s})$ for time window constraints, and the total amount $q_{\mathrm{sum}}(\boldsymbol{\sigma})$ of capacity excess are expressed as

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

The vehicle routing problem with convex time penalty functions is now formulated as follows:

$$\begin{aligned}
&\text{minimize} && cost(\boldsymbol{\sigma}, \boldsymbol{s}) = d_{\mathrm{sum}}(\boldsymbol{\sigma}) + p_{\mathrm{sum}}(\boldsymbol{s}) + q_{\mathrm{sum}}(\boldsymbol{\sigma}) && && (1) \\
&\text{subject to} && \sum_{k \in M} y_{ik}(\boldsymbol{\sigma}) = 1, && i \in V \setminus \{0\}, && (2) \\
& && s_k^{\mathrm{d}} + t_{\sigma_k(0)\sigma_k(1)} \le s_{\sigma_k(1)}, && k \in M, && (3) \\
& && s_{\sigma_k(h)} + u_{\sigma_k(h)} + t_{\sigma_k(h)\sigma_k(h+1)} \le s_{\sigma_k(h+1)}, && k \in M,\ h = 1, 2, \ldots, n_k - 1, && (4) \\
& && s_{\sigma_k(n_k)} + u_{\sigma_k(n_k)} + t_{\sigma_k(n_k)\sigma_k(n_k+1)} \le s_k^{\mathrm{a}}, && k \in M. && (5)
\end{aligned}$$

Constraint (2) means that every customer $i \in V \setminus \{0\}$ must be served exactly once by a vehicle. Constraints (3) and (4) require that the start time s_i of service for customer i must not be before the arrival time at customer i, and constraint (5) means that each vehicle k can return to the depot only after serving all the customers assigned to vehicle k. As for the objective function (1), the time window and capacity constraints are treated as soft constraints, and their violations are evaluated as the penalties $p_{\mathrm{sum}}(\boldsymbol{s})$ and $q_{\mathrm{sum}}(\boldsymbol{\sigma})$, respectively, in the objective function. Note that the weighted sum $d_{\mathrm{sum}}(\boldsymbol{\sigma}) + \mu^{\mathrm{p}} p_{\mathrm{sum}}(\boldsymbol{s}) + \mu^{\mathrm{q}} q_{\mathrm{sum}}(\boldsymbol{\sigma})$ with constants μ^{p} (≥ 0) and μ^{q} (≥ 0) might seem more general; however, such a weighted sum can be represented in the above formulation by regarding $\mu^{\mathrm{p}} p_i(t)$, $\mu^{\mathrm{p}} p_0^{\mathrm{d}}(t)$, $\mu^{\mathrm{p}} p_0^{\mathrm{a}}(t)$, $\mu^{\mathrm{q}} q_i$ and $\mu^{\mathrm{q}} Q_k$ ($i \in V \setminus \{0\}$, $k \in M$) as the given input, and hence the weights are omitted for simplicity unless otherwise stated.

Note that we call a solution that satisfies constraints (2) to (5) a feasible solution throughout the paper; i.e., a feasible solution does not necessarily satisfy the time window and capacity constraints. If time window and/or capacity constraints are to be satisfied, the penalties $p_{\mathrm{sum}}(\boldsymbol{s})$ and/or $q_{\mathrm{sum}}(\boldsymbol{\sigma})$ should be set sufficiently large.

Remark: Although the number m of vehicles is sometimes treated as a decision variable in the literature, we consider it as a given constant in this paper for the following two reasons. (1) In many applications, the number of vehicles m is fixed. (2) Algorithms become simpler if m is treated as a constant, since route elimination operators will not be necessary. For problems where m is a decision variable, we need to try various values of m to find a small feasible m; however, in many practical situations, an appropriate range of m is known in advance.

3 Optimal start time of services

In this section, we consider the problem of determining the start time to serve the customers in a given route σ_k so that the total time penalty is minimized. This problem can be solved independently for each route. (How to determine a route σ_k will be discussed in Section 4.) We call this problem the optimal start time problem, which is described as follows:

$$\begin{aligned}
&\text{minimize} && p_0^{\mathrm{d}}(s_k^{\mathrm{d}}) + \sum_{i=1}^{n_k} p_{\sigma_k(i)}(s_{\sigma_k(i)}) + p_0^{\mathrm{a}}(s_k^{\mathrm{a}}), \\
&\text{subject to} && s_k^{\mathrm{d}} + t_{\sigma_k(0)\sigma_k(1)} \le s_{\sigma_k(1)}, \\
& && s_{\sigma_k(h)} + u_{\sigma_k(h)} + t_{\sigma_k(h)\sigma_k(h+1)} \le s_{\sigma_k(h+1)}, \qquad h = 1, 2, \ldots, n_k - 1, \\
& && s_{\sigma_k(n_k)} + u_{\sigma_k(n_k)} + t_{\sigma_k(n_k)\sigma_k(n_k+1)} \le s_k^{\mathrm{a}}.
\end{aligned}$$

Let $\delta(p)$ be the number of pieces in a piecewise linear function $p(t)$, and let the total number of pieces in the penalty functions for all customers in a route σ_k (including the depot) be $\delta_k = \sum_{h=1}^{n_k} \delta(p_{\sigma_k(h)}) + \delta(p_0^{\mathrm{d}}) + \delta(p_0^{\mathrm{a}})$. We propose an $\mathbf{O}(\delta_k \log \delta_k)$ time algorithm to solve this problem by dynamic programming (DP). For this problem, there are several efficient algorithms that have the same time complexity as the algorithm in this section [1, 17]. However, our DP algorithm is simpler and utilized to design a more efficient algorithm for evaluating solutions in various neighborhoods in Section 5.1.2.

3.1 Dynamic programming

We define functions $f_h^k(t)$ for $h = 0, 1, \ldots, n_k + 1$ to be the minimum sum of the penalty values for customers $\sigma_k(0), \sigma_k(1), \ldots, \sigma_k(h)$ under the condition that all of them are served in this order and the service for $\sigma_k(h)$ starts by time t. We call this the *forward minimum penalty function*. For convenience, we also define values

$$\tau_h^k = u_{\sigma_k(h)} + t_{\sigma_k(h)\sigma_k(h+1)}$$

for customers $h = 1, 2, \ldots, n_k$; i.e., τ_h^k is the sum of the service time at the hth customer and the travel time from this to the next customer. Based on the idea of DP, $f_h^k(t)$ can be computed by

$$\begin{aligned}
f_0^k(t) &= \min_{t' \le t} p_0^{\mathrm{d}}(t'), \\
f_h^k(t) &= \min_{t' \le t} \left(f_{h-1}^k(t' - \tau_{h-1}^k) + p_{\sigma_k(h)}(t') \right), \qquad h = 1, 2, \ldots, n_k, \\
f_{n_k+1}^k(t) &= \min_{t' \le t} \left(f_{n_k}^k(t' - \tau_{n_k}^k) + p_0^{\mathrm{a}}(t') \right),
\end{aligned} \tag{6}$$

where $\tau_0^k = t_{\sigma_k(0)\sigma_k(1)}$. The minimum time penalty value for the entire route σ_k (denoted by $p_{\mathrm{sum}}^*(\sigma_k)$) can be obtained by

$$p_{\mathrm{sum}}^*(\sigma_k) = \min_t f_{n_k+1}^k(t). \tag{7}$$

Moreover, the optimal start time $s_{\sigma_k(h)}$ of the service for each customer $\sigma_k(1), \sigma_k(2), \ldots, \sigma_k(n_k)$, the departure time s_k^{d} from the depot and the arrival time s_k^{a} at the depot can be computed backward by

$$\begin{aligned} s_k^{\mathrm{a}} &= \arg\min_t f_{n_k+1}^k(t), \\ s_{\sigma_k(h)} &= \arg \min_{t \le s_{\sigma_k(h+1)} - \tau_h^k} f_h^k(t), \qquad h = n_k, n_k - 1, \ldots, 1, \\ s_k^{\mathrm{d}} &= \arg \min_{t \le s_{\sigma_k(1)} - \tau_0^k} f_0^k(t), \end{aligned} \tag{8}$$

where $s_{\sigma_k(n_k+1)} = s_k^{\mathrm{a}}$.

3.2 Algorithm and time complexity

In this section, we consider the data structure and algorithm for computing forward minimum penalty functions f_h^k in the recurrence formula (6).

We first consider some characteristics of f_h^k. Since functions p_i, p_0^{d} and p_0^{a} are convex and piecewise linear, each f_h^k is also convex and piecewise linear. Moreover, each f_h^k is nonincreasing by definition. The number of linear pieces for f_h^k is $\mathbf{O}(\delta_k)$.

In order to compute f_h^k in the recurrence formula (6), we need the following three operations:

1. Computing $f(t) := f(t - \tau)$ (called shift operation).
2. Computing $f(t) := f(t) + g(t)$ (called add operation).
3. Computing $f(t) := \min_{t' \le t} f(t')$ (called minimize operation).

Recall that functions f and g are convex and piecewise linear, and τ is a constant. Note also that we represent function g by a linked list of linear pieces. To support the above operations efficiently, we use the following data structure to represent function f:

- A balanced binary search tree with $\delta(f)$ leaves, whose height is denoted by h^{tree} and has a value c (called tree value). Each leaf v has an interval $[t_v^{\mathrm{l}}, t_v^{\mathrm{r}}]$, and each node v (including internal nodes and leaves) of this tree has values a_v, b_v and ζ_v.
- Each leaf of the tree represents a linear piece of function f, where the αth leaf in the inorder corresponds to the αth linear piece of f counted from left.

We denote the root of the tree by v_0, and the parent, the left child and the right child of a node v by $H(v)$, $L(v)$ and $R(v)$, respectively. We also denote the rightmost leaf in the descendants of an internal node v by $\tilde{R}(v)$, and we define $\tilde{R}(v) = v$ for each leaf v. Each node v has pointers to $H(v)$, $L(v)$, $R(v)$ and $\tilde{R}(v)$, if they exist. We represent the set of nodes in the path from a node v to a node v' as $\wp(v, v')$, where v' is a descendant of v. Consider a linear piece of function f that has a gradient a, an intercept b and an interval $[t_l, t_r]$, which is assigned to leaf v. The interval of leaf v satisfies $[t_v^{\mathrm{l}}, t_v^{\mathrm{r}}] = [t_l - c, t_r - c]$, where c is the tree value. The values $a_{v'}$ and $b_{v'}$ kept in nodes $v' \in \wp(v_0, v)$ satisfy the following equations:

$$\sum_{v' \in \wp(v_0, v)} a_{v'} = a, \qquad \sum_{v' \in \wp(v_0, v)} b_{v'} = b.$$

The tree value c, intervals $[t_v^{\mathrm{l}}, t_v^{\mathrm{r}}]$ of leaves v, and values a_v and b_v of nodes v are appropriately updated during the algorithm as explained later.

It is possible to shift a function f in $\mathbf{O}(1)$ time by updating the tree value c. For a given t, we can find the linear piece whose interval $[t_l, t_r]$ satisfies $t_l \leq t < t_r$ in $\mathbf{O}(h^{\text{tree}})$ time: We first set $v := v_0$. For the leaf node $\tilde{R}(L(v))$, if $t < t_v^{\text{r}} + c$ holds, then we set $v := L(v)$. Otherwise, we set $v := R(v)$. We repeat this operation until v becomes a leaf node of the tree, and then output the linear piece corresponding to the leaf v.

As for the value ζ_v of each node v, we set

$$\zeta_v = \sum_{v' \in \wp(v, \tilde{R}(v))} a_{v'}.$$

It is also possible to find the rightmost linear piece whose gradient is less than a given constant α in $\mathbf{O}(h^{\text{tree}})$ time: We first set $v := v_0$. The gradient of the linear piece corresponding to the leaf node $\tilde{R}(L(v))$ can be computed by

$$\sum_{v' \in \wp(v_0, v)} a_{v'} + \zeta_{L(v)}. \tag{9}$$

If it is larger than or equal to α, we set $v := L(v)$. Otherwise, we set $v := R(v)$. We repeat this operation until v becomes a leaf, and output the linear piece corresponding to the leaf v. The computation of (9) can be done in constant time for each v, because we can use the value of $\sum_{v' \in \wp(v_0, H(v))} a_{v'}$ already computed for the parent.

We now explain how to calculate and store the function f_h^k by using this data structure. First, we show the procedure for the first equation of (6). We construct a balanced binary tree with leaves corresponding to the linear pieces of p_0^{d} whose gradients are less than 0. For each leaf v, we set an interval $[t_v^{\text{l}}, t_v^{\text{r}}]$ and values a_v, b_v for the corresponding linear piece. We add another leaf v to the rightmost position of this tree; a_v is 0, t_v^{r} is $+\infty$, and we set b_v and t_v^{l} so that the resulting function becomes continuous. We set $c = 0$ for this tree, $a_v = 0$ and $b_v = 0$ for all internal nodes v, and $\zeta_v = a_{\tilde{R}(v)}$ for all nodes v. The time complexity of this computation is $\mathbf{O}(\delta(p_0^{\text{d}}))$.

We then explain the procedure for the second equation of (6). Since the procedure for the third equation is similar, we omit its explanation. For the shift operation $f_{h-1}^k(t' - \tau)$, we update the tree value by $c := c + \tau$ so that the intervals of the leaves are shifted. The add operation $f_{h-1}^k(t' - \tau) + p_{\sigma_k(h)}(t')$ is realized as follows. Our basic strategy for this operation is adding each linear piece of $g(t)$ $(= p_{\sigma_k(h)}(t'))$ to function $f(t)$ $(= f_{h-1}^k(t' - \tau))$ one by one. Consider a situation of adding a linear piece having gradient a, intercept b and interval $[t_l, t_r]$ to the binary tree corresponding to the function $f(t)$ with tree value c. We find the leaf v_α (resp., v_β) satisfying $t_{v_\alpha}^{\text{l}} \leq t_l - c < t_{v_\alpha}^{\text{r}}$ (resp., $t_{v_\beta}^{\text{l}} < t_r - c \leq t_{v_\beta}^{\text{r}}$). If $t_{v_\alpha}^{\text{l}} \neq t_l - c$ holds, we divide leaf v_α into two leaves which have intervals $[t_{v_\alpha}^{\text{l}}, t_l - c]$ and $[t_l - c, t_{v_\alpha}^{\text{r}}]$, and call the new leaf with interval $[t_l - c, t_{v_\alpha}^{\text{r}}]$ as v_α. We divide leaf v_β into two leaves and define v_β similarly (if necessary). In order to add the gradient a and intercept b to leaves whose intervals are $[t_l - c, t_{v_\alpha}^{\text{r}}], [t_{v_\alpha}^{\text{r}}, t''], \ldots, [t_{v_\beta}^{\text{l}}, t_r - c]$, we add a and b to the values a_v and b_v of nodes v that satisfy one of the following three conditions:

1. $R(H(v)) = v$, $H(v)$ is not an ancestor of v_β but that of v_α, v is not an ancestor of v_α,
2. $L(H(v)) = v$, $H(v)$ is not an ancestor of v_α but that of v_β, v is not an ancestor of v_β,
3. $v = v_\alpha$ or $v = v_\beta$.

It is easy to see that the number of nodes whose values are changed is $\mathbf{O}(h^{\text{tree}})$ and we can change those values in $\mathbf{O}(h^{\text{tree}})$ time. The resulting tree may not satisfy the conditions of a

balanced tree; we apply a balancing procedure that runs in $\mathbf{O}(h^{\mathrm{tree}})$ time if necessary. Such an insertion is conducted for all linear pieces of $g(t)$. Thus, the time complexity of computing $f(t)+g(t)$ is $\mathbf{O}(\delta(g)h^{\mathrm{tree}})$ time.

For the minimize operation $\min_{t'\le t}\left(f^k_{h-1}(t'-\tau)+p_{\sigma_k(h)}(t')\right)$, we find the rightmost linear piece whose gradient is less than 0 in $\mathbf{O}(h^{\mathrm{tree}})$ time. Recall that function $f^k_{h-1}(t-\tau)+p_{\sigma_k(h)}(t)$ is convex. We then remove the unnecessary portion of the tree (i.e., those with nonnegative gradient), and add a new leaf v to the rightmost position of this tree; a_v is 0, t^{r}_v is $+\infty$, and we set b_v and t^{l}_v so that the resulting function becomes continuous. We apply a balancing procedure for this tree (if necessary). The computational complexity for these operations is $\mathbf{O}(h^{\mathrm{tree}})$.

Now, we estimate the time complexity of our DP algorithm. We can compute $f^k_{n_k+1}$ in $\mathbf{O}(\delta_k \log \delta_k)$ time using the above procedures and operations, since $h^{\mathrm{tree}} = \mathbf{O}(\log \delta_k)$ holds throughout the algorithm.

For the computation of (8) and algorithms in Section 5, we must store f^k_h for all $h = 0, 1, \ldots, n_k+1$. If we realize this naively, just by keeping the binary search trees of all functions independently, we need $\mathbf{O}(n_k\delta_k)$ time and space. To avoid this, we share common parts among the trees to save the computation time and memory. Consider the case of computing f^k_h. We first generate a tree with only the root node v_0 and set $L(v_0) = L(v'_0)$, $R(v_0) = R(v'_0)$ and $\tilde{R}(v_0) = \tilde{R}(v'_0)$, where v'_0 is the root of f^k_{h-1}. Then, whenever we need to update some information of a node in the process of computing f^k_h, we duplicate the node and apply the updates only on the copy, while sharing the other nodes with the tree of f^k_{h-1}. As the number of new nodes is proportional to the number of updates, we can calculate and store functions f^k_h for all $h = 0, 1, \ldots, n_k+1$ in $\mathbf{O}(\delta_k \log \delta_k)$ time and space.

After computing the functions f^k_h by (6), we can compute the minimum time penalty value for route σ_k in $\mathbf{O}(\log \delta_k)$ time by (7), and the optimal start time $s_{\sigma_k(h)}$ of services for all customers in this route in $\mathbf{O}(n_k \log \delta_k)$ time by (8).

4 Local search for finding a good set of routes

In this section, we describe a local search (LS) algorithm to find a good set of routes. The following ingredients must be specified in designing the LS: Search space, how to generate an initial solution, a function to evaluate solutions, neighborhoods and move strategy. The search space of our LS is the set of all visiting orders $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ satisfying condition (2). Note that a set of visiting orders $\boldsymbol{\sigma}$ is called a "solution" in Sections 4 and 5, while a solution means $(\boldsymbol{\sigma}, \boldsymbol{s})$ in other parts of this paper. We generate an initial solution (i.e., an initial set of visiting orders) randomly. The objective function is often used in the literature as the evaluation function; however, we do not use the objective function directly in this paper. The details of our evaluation function will be explained in Section 4.1. The neighborhood $N(\boldsymbol{\sigma})$ of a feasible solution $\boldsymbol{\sigma}$ is a set of solutions obtainable from $\boldsymbol{\sigma}$ by applying some specified operations. In Section 4.2, we explain the neighborhoods utilized in our LS. As the move strategy, we adopt (a slightly simplified version of) the aspiration plus strategy [14], which will be explained in Section 5.1. We describe a metaheuristic algorithm based on the LS in Section 4.3.

4.1 Evaluation function

Let $p^*_{\mathrm{sum}}(\boldsymbol{\sigma})$ be the minimum value of $p_{\mathrm{sum}}(\boldsymbol{s})$ among those $\boldsymbol{s}$ satisfying conditions (3) to (5) for a given set of routes $\boldsymbol{\sigma}$. Such an $\boldsymbol{s}$ can be computed by solving the optimal start time problem for each route with the DP algorithm of Section 3. (A more efficient method will be presented in Section 5.1.2.)

Then the objective function (1) becomes

$$cost(\boldsymbol{\sigma}) = d_{\mathrm{sum}}(\boldsymbol{\sigma}) + p^*_{\mathrm{sum}}(\boldsymbol{\sigma}) + q_{\mathrm{sum}}(\boldsymbol{\sigma}). \tag{10}$$

Using this objective function as an evaluation function of LS may not work well for problem instances with hard time window and/or capacity constraints. In such cases, a large amount of penalty should be imposed on the violation of the time window and/or capacity constraints to satisfy them strictly, but it prevents the search from visiting the infeasible region of VRPHTW. To avoid such phenomenon, we therefore adopt the following function $eval(\boldsymbol{\sigma})$ instead of $cost(\boldsymbol{\sigma})$ to evaluate solutions in LS:

$$eval(\boldsymbol{\sigma}) = d_{\mathrm{sum}}(\boldsymbol{\sigma}) + \kappa^{\mathrm{p}} p^*_{\mathrm{sum}}(\boldsymbol{\sigma}) + \kappa^{\mathrm{q}} q_{\mathrm{sum}}(\boldsymbol{\sigma}), \tag{11}$$

where κ^{p} (resp., κ^{q}) is a weight of $p^*_{\mathrm{sum}}(\boldsymbol{\sigma})$ (resp., $q_{\mathrm{sum}}(\boldsymbol{\sigma})$). We change these weights whenever a local search stops at a locally optimal solution. As for the control mechanism of parameters κ^{p} and κ^{q} in our evaluation function *eval*, we control them as follows. If we find a solution whose time penalty (resp., capacity penalty) is equal to 0 in LS, the time penalty weight κ^{p} (resp., the capacity penalty weight κ^{q}) is decreased to $0.9\kappa^{\mathrm{p}}$ (resp., $0.9\kappa^{\mathrm{q}}$) after completion of LS in order to emphasize the other part $d_{\mathrm{sum}}(\boldsymbol{\sigma}) + q_{\mathrm{sum}}(\boldsymbol{\sigma})$ (resp., $d_{\mathrm{sum}}(\boldsymbol{\sigma}) + p^*_{\mathrm{sum}}(\boldsymbol{\sigma})$). Otherwise, κ^{p} (resp., κ^{q}) is increased to $\min\{1.1\kappa^{\mathrm{p}}, 1\}$ (resp., $\min\{1.1\kappa^{\mathrm{q}}, 1\}$) after completion of LS so that the influence of $p^*_{\mathrm{sum}}(\boldsymbol{\sigma})$ (resp., $q_{\mathrm{sum}}(\boldsymbol{\sigma})$) is reduced.

4.2 Neighborhoods

The neighborhood is a very important factor that determines the effectiveness of LS. In our algorithm, we utilize some standard neighborhoods for the VRP such as the cross exchange and 2-opt* neighborhoods, limiting their sizes by using parameters.

Iopt neighborhood The iopt neighborhood [4] is a variant of Or-opt neighborhood [27] used for the traveling salesman problem (TSP), which is a special case of the VRP in which the number of vehicles is one. We define a *path* as a subroute; i.e., a sequence of some consecutive customers served by one vehicle. An iopt operation removes a path of length at most $L^{\mathrm{iopt}}_{\mathrm{path}}$ (a parameter) and inserts it into another position of the same route, where the position is limited within the length $L^{\mathrm{iopt}}_{\mathrm{ins}}$ (a parameter) from the current position. For each insertion, we consider the following two cases: (1) visiting order preserved (called a normal insertion), and (2) visiting order reversed (called a reverse insertion). Note that the operation of just inverting the order of a path at its current position is also an iopt operation. Let $N^{\mathrm{iopt}}(\boldsymbol{\sigma}, k)$ be the set of all solutions obtainable by applying an iopt operation to route σ_k of the current solution $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_m)$, and let $N^{\mathrm{iopt}}(\boldsymbol{\sigma}) = \bigcup_{k \in M} N^{\mathrm{iopt}}(\boldsymbol{\sigma}, k)$. The size of the iopt neighborhood is $\mathbf{O}(nL^{\mathrm{iopt}}_{\mathrm{path}} L^{\mathrm{iopt}}_{\mathrm{ins}})$.

2-opt neighborhood The 2-opt neighborhood is one of the well-known neighborhoods for the TSP. A 2-opt operation removes a path whose length is at most L^{2opt} (a parameter), and inserts it into its current position with the reversed order. In other words, we remove two edges from a route, and reconstruct a route with two other edges. Let $N^{\mathrm{2opt}}(\boldsymbol{\sigma}, k)$ be the set of all solutions obtainable by applying a 2-opt operation to route σ_k, and let $N^{\mathrm{2opt}}(\boldsymbol{\sigma}) = \bigcup_{k \in M} N^{\mathrm{2opt}}(\boldsymbol{\sigma}, k)$. The size of this neighborhood is $\mathbf{O}(nL^{\mathrm{2opt}})$. Note that if $L^{\mathrm{iopt}}_{\mathrm{path}} \geq L^{\mathrm{2opt}} - 1$, then $N^{\mathrm{2opt}}(\boldsymbol{\sigma}) \subseteq N^{\mathrm{iopt}}(\boldsymbol{\sigma})$ holds. However, parameter $L^{\mathrm{iopt}}_{\mathrm{path}}$ is usually set small in order to keep $|N^{\mathrm{iopt}}(\boldsymbol{\sigma})|$ small, and we use $N^{\mathrm{2opt}}(\boldsymbol{\sigma})$ with a large L^{2opt} independently from $N^{\mathrm{iopt}}(\boldsymbol{\sigma})$.

2-opt* neighborhood The 2-opt* neighborhood, which was proposed in [28], 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. Let $N^{2\text{opt}^*}(\boldsymbol{\sigma}, k, k')$ be the set of all solutions obtainable by applying a 2-opt* operation to two routes σ_k and $\sigma_{k'}$ of the current solution $\boldsymbol{\sigma}$, and let $N^{2\text{opt}^*}(\boldsymbol{\sigma}) = \bigcup_{k \neq k'} N^{2\text{opt}^*}(\boldsymbol{\sigma}, k, k')$. The size of the 2-opt* neighborhood is $\mathbf{O}(n^2)$.

Path insertion neighborhood A path insertion operation removes a path of length at most L^{pins} (a parameter) from a route σ_k, and insert it into a different route $\sigma_{k'}$. Let $N^{\text{pins}}(\boldsymbol{\sigma}, k, k')$ be the set of all solutions obtainable by applying a path insertion operation to two routes σ_k and $\sigma_{k'}$, and let $N^{\text{pins}}(\boldsymbol{\sigma}) = \bigcup_{k \neq k'} N^{\text{pins}}(\boldsymbol{\sigma}, k, k')$. The size of the path insertion neighborhood is $\mathbf{O}(n^2 L^{\text{pins}})$. If the length of the removed path is at most $L^{\text{pins}}_{\text{rev}}$ (a parameter), we also consider the reverse insertion, where $L^{\text{pins}}_{\text{rev}} \leq L^{\text{pins}}$.

Cross exchange neighborhood The cross exchange neighborhood was proposed in [31]. A cross exchange operation removes two paths from two different routes (one from each), whose length is at most L^{cross} (a parameter), and exchanges them. If the length of removed paths is at most $L^{\text{cross}}_{\text{rev}}$ (a parameter), we also consider the (both or either) reverse insertion. Let $N^{\text{cross}}(\boldsymbol{\sigma}, k, k')$ be the set of all solutions obtainable by applying a cross exchange operation to two routes σ_k and $\sigma_{k'}$, and let $N^{\text{cross}}(\boldsymbol{\sigma}) = \bigcup_{k \neq k'} N^{\text{cross}}(\boldsymbol{\sigma}, k, k')$. The size of this neighborhood is $\mathbf{O}(n^2 (L^{\text{cross}})^2)$.

Combination of the neighborhoods It is often effective to combine some neighborhoods in an LS algorithm. Our LS searches the above five types of neighborhoods in the order described as follows: iopt, 2-opt, 2-opt*, path insertion and cross exchange. Once we find a better solution in a neighborhood, we return to the iopt neighborhood. When no improvement is achieved in the five consecutive neighborhoods, this procedure outputs a locally optimal solution for the neighborhoods and terminates.

4.3 Iterated local search

If the LS is applied only once, many solutions of better quality may remain unvisited in the search space. To overcome this, we use the *iterated local search* (ILS) [21], which is one of the basic frameworks of metaheuristics. In the ILS, the LS is executed iteratively and an initial solution of each LS is generated by slightly perturbing a good solution found so far. In our ILS, a random cross exchange operation, which randomly chooses two paths from two routes and exchanges them each other, is utilized to generate initial solutions. In order to generate a new initial solution, we choose r randomly from $\{1, 2, 3\}$, and apply random cross exchange operations r times consecutively to the incumbent solution (i.e., the best solution found by then).

5 Efficient implementation of local search

In this section, we explain various useful ideas to search the neighborhoods efficiently. In Section 5.1, we propose ideas to speed up the evaluation of solutions in neighborhoods. In Section 5.2, we show two ideas to prune the neighborhood. In Section 5.3, we propose other ideas to accelerate the speed of our local search procedure.

5.1 Evaluation of solutions in neighborhoods

Let Δd_{sum} (resp., Δp_{sum} and Δq_{sum}) be the difference in the traveling cost $d_{\mathrm{sum}}(\boldsymbol{\sigma})$ (resp., the time penalty $p^*_{\mathrm{sum}}(\boldsymbol{\sigma})$ and the capacity excess $q_{\mathrm{sum}}(\boldsymbol{\sigma})$) between the current solution and a solution in its neighborhood. Then, we define $\Delta eval$ as

$$\Delta eval = \Delta d_{\mathrm{sum}} + \kappa^{\mathrm{p}} \Delta p_{\mathrm{sum}} + \kappa^{\mathrm{q}} \Delta q_{\mathrm{sum}}, \tag{12}$$

and move to a solution whose $\Delta eval$ is negative. In the subsequent subsections, we explain how to evaluate Δd_{sum}, Δp_{sum} and Δq_{sum} effectively, based on the following two facts: (1) A neighborhood operation generates at most two different routes from the current solution, and (2) each new route is generated by reconnecting a constant number of paths (more precisely, at most four paths) in the current solution.

We now explain our basic strategy. At the beginning of a neighborhood search, we compute some values and functions that will be used during the neighborhood search (called preprocessing). We then evaluate each solution in the neighborhood quickly with those values and functions. Thus, our computation consists of the preprocessing part and the evaluation part. To avoid calling the preprocessing part too frequently, we adopt the aspiration plus strategy [14], which is explained as follows. We evaluate solutions in the neighborhood until we evaluate all the solutions in the neighborhood or we use (approximately) the same computation time that was spent for the previous preprocessing. If we can find improved solutions during the search, we move to the best solution among them. Otherwise, we continue the search for the remaining neighbors, based on the first admissible move strategy.

5.1.1 Evaluation of Δq_{sum} and Δd_{sum}

At the beginning of a neighborhood search, we compute

$$\begin{aligned}
&\gamma^k_0 = 0,\\
&\gamma^k_h = \gamma^k_{h-1} + q_{\sigma_k(h)}, \qquad h = 1, 2, \ldots, n_k,\\
&\phi^k_0 = 0,\\
&\phi^k_h = \phi^k_{h-1} + d_{\sigma_k(h-1)\sigma_k(h)}, \qquad h = 1, 2, \ldots, n_k + 1,\\
&\hat{\phi}^k_{n_k+1} = 0,\\
&\hat{\phi}^k_h = \hat{\phi}^k_{h+1} + d_{\sigma_k(h+1)\sigma_k(h)}, \qquad h = n_k, n_k - 1, \ldots, 1, 0,
\end{aligned}$$

for all $k \in M$. It is possible to compute all of them in $\mathbf{O}(n)$ time. We evaluate Δq_{sum} and Δd_{sum} for each solution in the neighborhood using these γ^k_h, ϕ^k_h and $\hat{\phi}^k_h$.

As for Δq_{sum}, we observe that Δq_{sum} is equal to 0 for all solutions in N^{iopt} and N^{2opt}, and thus we only consider solutions in N^{2opt^*}, N^{pins} and N^{cross}. The sum $\sum_{l=h}^{h'} q_{\sigma_k(l)}$ of the amount of resources for the hth through the h'th customers can be computed in $\mathbf{O}(1)$ time by $\gamma^k_{h'} - \gamma^k_{h-1}$. Since there are only two different routes from the current solution and each new route is generated by reconnecting at most three paths, the time complexity to evaluate Δq_{sum} is $\mathbf{O}(1)$.

As for Δd_{sum}, the total distance $\sum_{l=h}^{h'-1} d_{\sigma_k(l)\sigma_k(l+1)}$ of the hth through the h'th customers (in the case of the normal insertion) can be computed in $\mathbf{O}(1)$ time by $\phi^k_{h'} - \phi^k_h$. The total distance $\sum_{l=h}^{h'-1} d_{\sigma_k(l+1)\sigma_k(l)}$ of the h'th through the hth customers (in the case of the reverse insertion) can be computed in $\mathbf{O}(1)$ time by $\hat{\phi}^k_h - \hat{\phi}^k_{h'}$. Thus, Δd_{sum} is also computed in $\mathbf{O}(1)$ time for each solution in the neighborhoods.

Since the size of each neighborhood $|N^{\mathrm{iopt}}|$, $|N^{\mathrm{2opt}}|$, $|N^{\mathrm{2opt}^*}|$, $|N^{\mathrm{pins}}|$ or $|N^{\mathrm{cross}}|$ is larger than $\mathbf{O}(n)$ (time for preprocessing), the amortized time complexity to evaluate Δq_{sum} and Δd_{sum} for a solution is $\mathbf{O}(1)$.

5.1.2 Evaluation of Δp_{sum}

If Δp_{sum} is computed according to (6) from scratch, it takes $\mathbf{O}(\sum_{k\in M'} \delta_k \log \delta_k)$ time, where M' is the set of indices of the vehicles related to a neighborhood operation. (Note that $|M'| \leq 2$ holds for any neighborhood considered in this paper.) Instead of this, we propose an $\mathbf{O}(\sum_{k\in M'} \log \delta_k) = \mathbf{O}(\log \delta_{\max})$ time algorithm that computes Δp_{sum} for a solution $\boldsymbol{\sigma'}$ in the neighborhoods, where $\delta_{\max} = \max_{k\in M} \delta_k$. We only explain the algorithm for cases where the number of paths contained in a new route is two or three. However, this idea can be extended to a new route obtainable by reconnecting a constant number of paths (e.g., by reconnecting four paths).

In addition to the forward minimum penalty function $f_h^k(t)$ of (6), we define $b_h^k(t)$ as the minimum sum of the time penalty values for customers $\sigma_k(h), \sigma_k(h+1), \ldots, \sigma_k(n_k), \sigma_k(n_k+1)$ under the condition that all of them are served in this order and the service for $\sigma_k(h)$ starts at time t or later. We call this the *backward minimum penalty function*. In a symmetric manner to the computation of $f_h^k(t)$, $b_h^k(t)$ can be computed by:

$$\begin{aligned} b_{n_k+1}^k(t) &= \min_{t'\geq t} p_0^{\mathrm{a}}(t'), \\ b_h^k(t) &= \min_{t'\geq t} \left(p_{\sigma_k(h)}(t') + b_{h+1}^k(t' + \tau_h^k) \right), \quad h = n_k, n_k-1, \ldots, 1, \\ b_0^k(t) &= \min_{t'\geq t} \left(p_0^{\mathrm{d}}(t') + b_1^k(t' + \tau_0^k) \right). \end{aligned} \tag{13}$$

Each function $b_h^k(t)$ is convex, piecewise linear and nondecreasing. For a vehicle k, we can calculate and store functions b_h^k for all $h = 0, 1, \ldots, n_k + 1$ in $\mathbf{O}(\delta_k \log \delta_k)$ time.

Reconnecting two paths Let us consider the computation of the minimum time penalty on a new route

$$\sigma'_k = \langle 0, \sigma_{k_1}(h_1) \rangle \text{–} \langle \sigma_{k_2}(h_2), 0 \rangle$$

generated by reconnecting two paths $\langle 0, \sigma_{k_1}(h_1) \rangle$ and $\langle \sigma_{k_2}(h_2), 0 \rangle$, where $\langle \sigma_k(h), \sigma_k(h') \rangle$ represents the path from $\sigma_k(h)$ to $\sigma_k(h')$ in route σ_k. Using the forward and backward minimum penalty functions, $p_{\mathrm{sum}}^*(\sigma'_k)$ can be computed by

$$\min_t \left(f_{h_1}^{k_1}(t) + b_{h_2}^{k_2}\left(t + \tilde{\tau}\left(k_1, k_2, h_1, h_2\right)\right) \right), \tag{14}$$

where

$$\tilde{\tau}\left(k, k', h, h'\right) = u_{\sigma_k(h)} + t_{\sigma_k(h)\sigma_{k'}(h')}.$$

If we compute $\min_t \left(f(t) + b(t)\right)$ for two piecewise linear convex functions f and b naively, it takes $\mathbf{O}(\delta(f) + \delta(b))$ time. However, using the fact that the new function $f(t) + b(t)$ is also convex, we can compute the minimum value and the corresponding time t (denoted by s^*) in $\mathbf{O}(\log \delta(f) + \log \delta(b))$ time under the assumption that functions f and b are represented by the balanced search trees defined in Section 3.2. (If the t that achieves the minimum value of $f(t) + b(t)$ is not unique, we define $s^* = \min \arg\min_t (f(t) + b(t))$.) We note that our procedure is easily extended for computing $\min_t \sum_{i=1}^r g_i(t)$ in $\mathbf{O}(\sum_{i=1}^r \log \delta(g_i))$ time.

We compute $\min_t (f(t) + b(t))$ for two convex functions f and b by a variant of binary search based on the following fact: Suppose that we have a linear piece of $f(t)$ that has gradient a_f and interval $[l_f, r_f]$, and another linear piece of $b(t)$ with gradient a_b and interval $[l_b, r_b]$. If $a_f + a_b$ is less than 0, $s^* \geq \min\{r_f, r_b\}$ holds; otherwise, $s^* \leq \max\{l_f, l_b\}$ holds. For a node v in the balanced search tree representing a function f or b, let $a(v)$ and $[l(v), r(v)]$ be the gradient and interval of the linear piece that corresponds to the leaf $\tilde{R}(L(v))$ (resp., v) if v is an internal node (resp., a leaf). We first set v_f and v_b to be the root nodes of the trees representing f and b, respectively. Using the above fact, we repeat one of the following operations until both of v_f and v_b become leaves: If $a(v_f) + a(v_b) < 0$ holds, then we set $v_f := R(v_f)$ (i.e., v_f moves to its right child) if $r(v_f) < r(v_b)$ holds, or set $v_b := R(v_b)$ otherwise (i.e., $r(v_f) \geq r(v_b)$). On the other hand, if $a(v_f) + a(v_b) \geq 0$ holds, then we set $v_f := L(v_f)$ if $l(v_f) > l(v_b)$, or set $v_b := L(v_b)$ otherwise (i.e., $l(v_f) \leq l(v_b)$). The number these operations is bounded by the sum of the heights of the trees representing f and b, since we replace v_f or v_b to its child node in each iteration. The heights of the trees are $\mathbf{O}(\log \delta(f))$ and $\mathbf{O}(\log \delta(b))$, respectively, and each iteration takes $\mathbf{O}(1)$ time. Hence the total computation time to compute $\min_t (f(t) + b(t))$ and s^* is $\mathbf{O}(\log \delta(f) + \log \delta(b))$.

Reconnecting three paths Let us consider the computation of the minimum time penalty of a new route

$$\sigma'_k = \langle 0, \sigma_{k_1}(h_1)\rangle\text{–}\langle \sigma_{k_2}(h_2), \sigma_{k_2}(h_3)\rangle\text{–}\langle \sigma_{k_3}(h_4), 0\rangle \tag{15}$$

generated by reconnecting three paths. For simplicity, we consider only the case where a new route is constructed with a normal insertion. For this computation, we use the forward and backward minimum penalty functions, τ^k_h and $\tilde{\tau}\,(k, k', h, h')$. Moreover, for all candidates of the intermediate path (e.g., $\langle \sigma_{k_2}(h_2), \sigma_{k_2}(h_3)\rangle$), we compute two types of new functions $\tilde{f}^k_{h,h'}(t)$ and $\tilde{b}^k_{h,h'}(t)$, and values $\chi(k, h, h')$. The function $\tilde{f}^k_{h,h'}(t)$ (resp., $\tilde{b}^k_{h,h'}(t)$) is the minimum sum of the time penalty values for customers $\sigma_k(h), \sigma_k(h+1), \ldots, \sigma_k(h')$ if these customers are served in this order and the service for customer $\sigma_k(h')$ (resp., $\sigma_k(h)$) starts by (resp., starts on or after) time t. Based on the idea of DP, $\tilde{f}^k_{h,h'}(t)$ and $\tilde{b}^k_{h,h'}(t)$ can be computed by

$$\begin{aligned}
\tilde{f}^k_{h,h}(t) &= \min_{t' \leq t} \; p_{\sigma_k(h)}(t'), \qquad h = 1, 2, \ldots, n_k, \\
\tilde{f}^k_{h,h'}(t) &= \min_{t' \leq t} \Big(\tilde{f}^k_{h,h'-1}(t' - \tau^k_{h'-1}) + p_{\sigma_k(h')}(t') \Big), \qquad (16) \\
&\qquad h = 1, 2, \ldots, n_k - 1, \;\; h' = h+1, h+2, \ldots, \min\{h + L^{\max} - 1, n_k\}, \\
\tilde{b}^k_{h,h}(t) &= \min_{t' \geq t} \; p_{\sigma_k(h)}(t'), \qquad h = 1, 2, \ldots, n_k, \\
\tilde{b}^k_{h,h'}(t) &= \min_{t' \geq t} \Big(p_{\sigma_k(h)}(t') + \tilde{b}^k_{h+1,h'}(t' + \tau^k_h) \Big), \qquad (17) \\
&\qquad h' = 2, 3, \ldots, n_k, \;\; h = h'-1, h'-2, \ldots, \max\{h' - L^{\max} + 1, 1\},
\end{aligned}$$

for all $k \in M$, where $L^{\max} = \max\{L^{\text{cross}}, L^{\text{pins}}\}$. Since $p_i(t)$, $p^{\text{d}}_0(t)$ and $p^{\text{a}}_0(t)$ are convex functions, $\tilde{f}^k_{h,h'}(t)$ (resp., $\tilde{b}^k_{h,h'}(t)$) is also convex and is nonincreasing (resp., nondecreasing). We calculate and store these functions as well as $f^k_h(t)$ and $b^k_h(t)$. This computation is possible in $\mathbf{O}(L^{\max}\delta_k \log \delta_k)$ time for each vehicle k. The value $\chi(k, h, h')$ is the minimum time needed to serve customers $\sigma_k(h), \sigma_k(h+1), \ldots, \sigma_k(h')$ in this order; that is,

$$\chi(k, h, h') = \sum_{h''=h}^{h'-1} \tau^k_{h''}. \tag{18}$$

Note that we can also define $\tilde{f}^k_{h,h'}(t)$, $\tilde{b}^k_{h,h'}(t)$ and $\chi(k,h,h')$ for the paths of reverse direction in a symmetric manner, and we use them to evaluate new routes constructed with reverse insertions.

We now explain how to compute the minimum time penalty value of the new route σ'_k of (15). First, we consider the following two paths

$$\langle 0, \sigma_{k_1}(h_1)\rangle\text{–}\langle \sigma_{k_2}(h_2), \sigma_{k_2}(h_3)\rangle$$

and

$$\langle \sigma_{k_2}(h_2), \sigma_{k_2}(h_3)\rangle\text{–}\langle \sigma_{k_3}(h_4), 0\rangle.$$

For these two paths, we independently calculate the minimum time penalty values and the corresponding times $s^*_{h_2}$ and $s^*_{h_3}$, which are respectively defined as follows:

$$\begin{aligned} s^*_{h_2} &= \min \arg\min_t \left(f^{k_1}_{h_1}(t - \tilde{\tau}\,(k_1,k_2,h_1,h_2)) + \tilde{b}^{k_2}_{h_2,h_3}(t) \right), \\ s^*_{h_3} &= \max \arg\min_t \left(\tilde{f}^{k_2}_{h_2,h_3}(t) + b^{k_3}_{h_4}\,(t + \tilde{\tau}\,(k_2,k_3,h_3,h_4)) \right). \end{aligned} \tag{19}$$

Now we compute the minimum time penalty of σ'_k. If

$$s^*_{h_3} - s^*_{h_2} \geq \chi(k_2,h_2,h_3) \tag{20}$$

holds, it is given by

$$\begin{aligned} &f^{k_1}_{h_1}\left(s^*_{h_2} - \tilde{\tau}\,(k_1,k_2,h_1,h_2)\right) + \tilde{b}^{k_2}_{h_2,h_3}\left(s^*_{h_2}\right) \\ &+ \tilde{f}^{k_2}_{h_2,h_3}(s^*_{h_3}) + b^{k_3}_{h_4}(s^*_{h_3} + \tilde{\tau}\,(k_2,k_3,h_3,h_4)) - \min_t \tilde{f}^{k_2}_{h_2,h_3}(t). \end{aligned} \tag{21}$$

On the other hand, if

$$s^*_{h_3} - s^*_{h_2} < \chi(k_2,h_2,h_3) \tag{22}$$

holds, it can be computed by

$$\begin{aligned} \min_t \big\{ & f^{k_1}_{h_1}(t - \tilde{\tau}\,(k_1,k_2,h_1,h_2)) + \tilde{b}^{k_2}_{h_2,h_3}(t) \\ &+ \tilde{f}^{k_2}_{h_2,h_3}(t + \chi(k_2,h_2,h_3)) \\ &+ b^{k_3}_{h_4}(t + \chi(k_2,h_2,h_3) + \tilde{\tau}\,(k_2,k_3,h_3,h_4)) \big\} \\ - \min_{t'} & \tilde{f}^{k_2}_{h_2,h_3}(t'). \end{aligned} \tag{23}$$

We now show the correctness of (21) and (23). The following lemma is crucial for this purpose, whose proof will be given later in this section.

Lemma 5.1 *Suppose $t_{h_3} - t_{h_2} \geq \chi(k_2,h_2,h_3)$. Then $\tilde{b}^{k_2}_{h_2,h_3}(t_{h_2}) + \tilde{f}^{k_2}_{h_2,h_3}(t_{h_3}) - \min_t \tilde{f}^{k_2}_{h_2,h_3}(t)$ gives the minimum total time penalty of all customers in the path $\langle \sigma_{k_2}(h_2), \sigma_{k_2}(h_3)\rangle$, provided that $\sigma_{k_2}(h_2)$ is served after time t_{h_2} and $\sigma_{k_2}(h_3)$ is served by time t_{h_3}.*

From this lemma, it is clear that

$$\begin{aligned} \min_{t_{h_3} - t_{h_2} \geq \chi(k_2,h_2,h_3)} & \left\{ f^{k_1}_{h_1}\left(t_{h_2} - \tilde{\tau}\,(k_1,k_2,h_1,h_2)\right) + \tilde{b}^{k_2}_{h_2,h_3}\left(t_{h_2}\right) \right. \\ & \left. + \tilde{f}^{k_2}_{h_2,h_3}(t_{h_3}) + b^{k_3}_{h_4}(t_{h_3} + \tilde{\tau}\,(k_2,k_3,h_3,h_4)) \right\} - \min_t \tilde{f}^{k_2}_{h_2,h_3}(t) \end{aligned} \tag{24}$$

gives the minimum total time penalty for all customers in the route

$$\sigma'_k = \langle 0, \sigma_{k_1}(h_1)\rangle\text{–}\langle \sigma_{k_2}(h_2), \sigma_{k_2}(h_3)\rangle\text{–}\langle \sigma_{k_3}(h_4), 0\rangle.$$

As for the start time s_{h_2} and s_{h_3} of the service for customers $\sigma_{k_2}(h_2)$ and $\sigma_{k_2}(h_3)$,

$$s_{h_3} - s_{h_2} \geq \chi(k_2, h_2, h_3) \tag{25}$$

is necessary and sufficient for a feasible time schedule to exist. In the above procedure, we consider two cases in which $s^*_{h_2}$ and $s^*_{h_3}$ satisfy condition (25) or not. When $s^*_{h_2}$ and $s^*_{h_3}$ satisfy (25), we will have (20). As $s^*_{h_2}$ and $s^*_{h_3}$ defined by (19) minimize disjoint terms in (24) independently without considering condition (25), they give the minimum value to (24) in this case. On the other hand, when $s^*_{h_2}$ and $s^*_{h_3}$ do not satisfy (25), the optimal start time $s^{\rm opt}_{h_2}$ and $s^{\rm opt}_{h_3}$ of the service for customers $\sigma_{k_2}(h_2)$ and $\sigma_{k_2}(h_3)$ must satisfy $s^{\rm opt}_{h_3} - s^{\rm opt}_{h_2} = \chi(k_2, h_2, h_3)$. Then (23) clearly gives the minimum value in this case.

We now give a proof to Lemma 5.1.

Proof of Lemma 5.1. Let $\hat{s}_{h_2}, \hat{s}_{h_2+1}, \ldots, \hat{s}_{h_3}$ be a sequence of start times of the service for customers in $\langle \sigma_{k_2}(h_2), \sigma_{k_2}(h_3) \rangle$ that achieves $\min_t \tilde{f}^{k_2}_{h_2,h_3}(t)$. Note that $\min_t \tilde{f}^{k_2}_{h_2,h_3}(t) = \min_t \tilde{b}^{k_2}_{h_2,h_3}(t) = \sum_{h=h_2}^{h_3} p_{\sigma_{k_2}(h)}(\hat{s}_h)$ holds by definition. For $h_2 \leq h \leq h' \leq h_3$, we call a sequence $(h, h+1, \ldots, h')$ of indices a *block* if it is maximal among those that satisfy $\hat{s}_{h''} + \tau^{k_2}_{h''} = \hat{s}_{h''+1}$ for all $h'' = h, h+1, \ldots, h'-1$. Let $a^+_h(t) = \lim_{\varepsilon \to +0}\{p_{\sigma_{k_2}(h)}(t+\varepsilon) - p_{\sigma_{k_2}(h)}(t)\}/\varepsilon$ and $a^-_h(t) = \lim_{\varepsilon \to +0}\{p_{\sigma_{k_2}(h)}(t) - p_{\sigma_{k_2}(h)}(t-\varepsilon)\}/\varepsilon$. Then, for any block $(h, h+1, \ldots, h')$ and $h'' \in [h, h']$, both $a^+_{h''}(\hat{s}_{h''}) + a^+_{h''+1}(\hat{s}_{h''+1}) + \cdots + a^+_{h'}(\hat{s}_{h'}) \geq 0$ and $a^-_h(\hat{s}_h) + a^-_{h+1}(\hat{s}_{h+1}) + \cdots + a^-_{h''}(\hat{s}_{h''}) \leq 0$ hold, since otherwise $\hat{s}_{h_2}, \hat{s}_{h_2+1}, \ldots, \hat{s}_{h_3}$ cannot be optimal. Moreover, from convexity of time penalty functions, $a^+_h(t)$ and $a^-_h(t)$ are monotonically nondecreasing with t. For a given t_{h_2}, let

$$\tilde{s}_h = \max\{\hat{s}_h,\ t_{h_2} + \chi(k_2, h_2, h)\}$$

for $h = h_2, h_2+1, \ldots, h_3$. This time schedule $\tilde{s}_{h_2}, \tilde{s}_{h_2+1}, \ldots, \tilde{s}_{h_3}$ achieves the minimum total time penalty $\tilde{b}^{k_2}_{h_2,h_3}(t_{h_2})$ of customers in $\langle \sigma_{k_2}(h_2), \sigma_{k_2}(h_3) \rangle$ under the constraint that $\sigma_{k_2}(h_2)$ must be served after time t_{h_2} for the following reasons. Consider the blocks defined for this time schedule. From the above observation, for any block $(h, h+1, \ldots, h')$ and $h'' \in [h, h']$, we have $a^+_{h''}(\tilde{s}_{h''}) + a^+_{h''+1}(\tilde{s}_{h''+1}) + \cdots + a^+_{h'}(\tilde{s}_{h'}) \geq 0$, and also have $a^-_h(\tilde{s}_h) + a^-_{h+1}(\tilde{s}_{h+1}) + \cdots + a^-_{h''}(\tilde{s}_{h''}) \leq 0$ or $\tilde{s}_{h''} = t_{h_2} + \chi(k_2, h_2, h'')$. These indicate that any sufficiently small change to this schedule that keeps its feasibility will not decrease the total time penalty. As the problem of minimizing the total time penalty is a convex programming problem (i.e., its objective function and feasible region are convex), this is sufficient to confirm its optimality. Similarly, we can obtain a time schedule $\bar{s}_{h_2}, \bar{s}_{h_2+1}, \ldots, \bar{s}_{h_3}$ that achieves the minimum total time penalty $\tilde{f}^{k_2}_{h_2,h_3}(t_{h_3})$ of customers in $\langle \sigma_{k_2}(h_2), \sigma_{k_2}(h_3) \rangle$ under the constraint that $\sigma_{k_2}(h_3)$ must be served by time t_{h_3} by

$$\bar{s}_h = \min\{\hat{s}_h,\ t_{h_3} - \chi(k_2, h, h_3)\}$$

for $h = h_2, h_2+1, \ldots, h_3$. For the same reason, noting that $t_{h_3} - t_{h_2} \geq \chi(k_2, h_2, h_3)$ implies $t_{h_2} + \chi(k_2, h_2, h) \leq t_{h_3} - \chi(k_2, h, h_3)$, a time schedule

$$\check{s}_h = \mathrm{median}\{\hat{s}_h,\ t_{h_2} + \chi(k_2, h_2, h),\ t_{h_3} - \chi(k_2, h, h_3)\}$$

for $h = h_2, h_2+1, \ldots, h_3$ is feasible and achieves the minimum total time penalty of customers in the path under the constraint that $\sigma_{k_2}(h_2)$ must be served after time t_{h_2} and $\sigma_{k_2}(h_3)$ must be served by time t_{h_3}. If $t_{h_2} + \chi(k_2, h_2, h) \leq \hat{s}_h \leq t_{h_3} - \chi(k_2, h, h_3)$ holds, $\hat{s}_h = \tilde{s}_h = \bar{s}_h = \check{s}_h$ holds. If $\hat{s}_h \leq t_{h_2} + \chi(k_2, h_2, h) \leq t_{h_3} - \chi(k_2, h, h_3)$ holds, $\hat{s}_h = \bar{s}_h$ and $\tilde{s}_h = \check{s}_h$ hold. If $t_{h_2} + \chi(k_2, h_2, h) \leq t_{h_3} - \chi(k_2, h, h_3) \leq \hat{s}_h$ holds, $\hat{s}_h = \tilde{s}_h$ and $\bar{s}_h = \check{s}_h$ hold. Thus, for all

$h = h_2, h_2 + 1, \ldots, h_3$, $p_{\sigma_{k_2}(h)}(\check{s}_h) = p_{\sigma_{k_2}(h)}(\tilde{s}_h) + p_{\sigma_{k_2}(h)}(\bar{s}_h) - p_{\sigma_{k_2}(h)}(\hat{s}_h)$ holds, which implies the lemma. ∎

Finally, we estimate the time complexity of our procedure to compute Δp_{sum}. Our procedure consists of two types of computation; computing $\tilde{f}^k_{h,h'}(t)$, $\tilde{b}^k_{h,h'}(t)$ and $\chi(k,h,h')$ for preprocessing before starting a neighborhood search, and computing (19), (21) and (23) to evaluate each solution in our neighborhoods. As we described before, we can compute the former in $\mathbf{O}(L^{\max}\delta_k \log \delta_k)$ time for each vehicle k. We can compute the latter in $\mathbf{O}(\sum_{k\in M'} \log \delta_k)$ time, where M' is the set of indices of the vehicles related to new route σ'_k. $L^{\max}\delta_k$ is usually smaller than the size of neighborhoods, and under this assumption, the amortized computation time to compute Δp_{sum} for a solution in neighborhoods becomes $\mathbf{O}(\sum_{k\in M'} \log \delta_k) = \mathbf{O}(\log \delta_{\max})$.

5.2 Pruning the neighborhoods

In this section, we propose two types of neighbor-lists, called *distance-oriented* and *time-oriented* neighbor-lists, which are used in our local search algorithm to prune the neighborhoods.

The distance-oriented neighbor-list was successfully applied to the TSP [22]. At the beginning of our algorithm, for each customer i, we construct a list of L^{dlist} nearest customers from customer i, where L^{dlist} is a parameter. It is possible to construct these lists for all customers in $\mathbf{O}(n^2 L^{\mathrm{dlist}})$ time and to store these lists in $\mathbf{O}(nL^{\mathrm{dlist}})$ space. Using this neighbor-list, our neighborhood search is limited to those operations that connect a customer i to one of the customers in its list. By using this idea, the size of neighborhood N^{2opt^*} (resp., $N^{\mathrm{pins}}, N^{\mathrm{cross}}$) is reduced from $\mathbf{O}(n^2)$ (resp., $\mathbf{O}(n^2 L^{\mathrm{pins}}), \mathbf{O}(n^2 (L^{\mathrm{cross}})^2)$) to $\mathbf{O}(nL^{\mathrm{dlist}})$ (resp., $\mathbf{O}(nL^{\mathrm{dlist}} L^{\mathrm{pins}}), \mathbf{O}(nL^{\mathrm{dlist}} (L^{\mathrm{cross}})^2)$). To reduce the size of neighborhoods further, we add another rule: We do not evaluate a solution if the distance of a new edge is ω^{dlist} times larger than the distance of the current edge, where ω^{dlist} is another parameter.

Taking into account the time window constraints, we propose another technique to prune the neighborhoods on the basis of the start time of services at customers. In our local search algorithm, three types of neighborhood operations (i.e., 2-opt*, path insertion and cross exchange) are conducted on a pair of routes. After fixing a pair of routes σ_k and $\sigma_{k'}$, we construct a list of customer pairs whose start times of service are close to each other in the following manner. For each customer $\sigma_k(h)$ in route σ_k, we find the customer $\sigma_{k'}(h')$ whose start time $s_{\sigma_{k'}(h')}$ of service is the closest to $s_{\sigma_k(h)}$ among the customers assigned to route $\sigma_{k'}$, and then, store customer pairs $(\sigma_k(h),\ \sigma_{k'}(h' - L^{\mathrm{tlist}}))$, $(\sigma_k(h),\ \sigma_{k'}(h' - L^{\mathrm{tlist}} + 1))$, $\ldots$, $(\sigma_k(h),\ \sigma_{k'}(h' + L^{\mathrm{tlist}}))$, where L^{tlist} is a parameter. Since start times of service for customers in a route are nondecreasing, we can compute all such pairs in $\mathbf{O}(L^{\mathrm{tlist}} n_k + n_{k'})$ time for route σ_k. This procedure is conducted for the customers assigned to route $\sigma_{k'}$ similarly, and we merge those lists in order to eliminate double entries. We prune the neighborhood using the above time-oriented neighbor-list. For each pair of routes σ_k and $\sigma_{k'}$, the number of solutions generated by 2-opt*, path insertion and cross exchange operations are reduced from $\mathbf{O}(n_k n_{k'})$, $\mathbf{O}(n_k n_{k'} L^{\mathrm{pins}})$, $\mathbf{O}(n_k n_{k'} (L^{\mathrm{cross}})^2)$ to $\mathbf{O}(L^{\mathrm{tlist}}(n_k + n_{k'}))$, $\mathbf{O}(L^{\mathrm{tlist}}(n_k + n_{k'}) L^{\mathrm{pins}})$, $\mathbf{O}(L^{\mathrm{tlist}}(n_k + n_{k'})(L^{\mathrm{cross}})^2)$. The time complexity of constructing the time-oriented neighbor-list for a pair of routes σ_k and $\sigma_{k'}$ is $\mathbf{O}(L^{\mathrm{tlist}}(n_k + n_{k'}))$; this computation time does not affect the total computational time. The two types of neighborhoods resulting from distance-oriented and time-oriented neighbor-lists are then searched sequentially.

5.3 Other speed up techniques

In this section, we propose two other ideas, which do not change the worst case time complexity, but accelerate the speed in practical sense. As mentioned in Section 5.1, we can compute Δq_{sum} and Δd_{sum} in constant time, while it takes $\mathbf{O}(\sum_{k\in M'} \log \delta_k)$ time to compute Δp_{sum}.

The first idea is to reduce the number of linear pieces of functions $f_h^k(t)$, $b_h^k(t)$, $\tilde{f}_{h,h'}^k(t)$ and $\tilde{b}_{h,h'}^k(t)$. Since $f_h^k(t)$ and $\tilde{f}_{h,h'}^k(t)$ (resp., $b_h^k(t)$ and $\tilde{b}_{h,h'}^k(t)$) are nonincreasing (resp., nondecreasing), $f_h^k(t)$ and $\tilde{f}_{h,h'}^k(t)$ (resp., $b_h^k(t)$ and $\tilde{b}_{h,h'}^k(t)$) usually have many pieces with extremely large value for small (resp., large) t. Without missing any improved solutions, we delete unnecessary linear pieces as follows. Let $\boldsymbol{\sigma}_0$ be the initial solution of our local search algorithm, and let $\mathrm{UB} := eval(\boldsymbol{\sigma}_0)$. We then delete those pieces whose minimum values are not less than $\mathrm{UB}/\kappa^{\mathrm{P}}$. For a vehicle k, this deletion procedure of unnecessary pieces can be done in $\mathbf{O}(\log \delta_k)$ time. Whenever the current solution $\boldsymbol{\sigma}$ is improved during the search, we update UB to $eval(\boldsymbol{\sigma})$ and call the deletion procedure.

The second idea is to compute a lower bound of $p_{\mathrm{sum}}^*(\sigma'_k)$ for each new route σ'_k in constant time so that we skip the evaluation of hopeless solutions. For each function $f_h^k(t), b_h^k(t), \tilde{f}_{h,h'}^k(t)$ and $\tilde{b}_{h,h'}^k(t)$, we compute and store $\min_t f_h^k(t), \min_t b_h^k(t), \min_t \tilde{f}_{h,h'}^k(t)$ and $\min_t \tilde{b}_{h,h'}^k(t)$ when we construct a binary search tree for each function. By using these values, we can obtain a lower bound of the minimum time penalty for each route. For example, a lower bound of the minimum time penalty for a new route $\sigma'_k = \langle 0, \sigma_{k_1}(h_1)\rangle$–$\langle \sigma_{k_2}(h_2), 0\rangle$ generated by reconnecting two paths is calculated by $\min_t f_{h_1}^{k_1}(t) + \min_t b_{h_2}^{k_2}(t)$, and a lower bound for another new route $\sigma'_k = \langle 0, \sigma_{k_1}(h_1)\rangle$–$\langle \sigma_{k_2}(h_2), \sigma_{k_2}(h_3)\rangle$–$\langle \sigma_{k_3}(h_4), 0\rangle$ generated by reconnecting three paths is calculated by $\min_t f_{h_1}^{k_1}(t) + \min_t \tilde{f}_{h_2,h_3}^{k_2}(t) + \min_t b_{h_4}^{k_3}(t)$. Moreover, for a new route generated by reconnecting three paths, we can use another lower bound

$$f_{h_1}^{k_1}(s_{h_2}^* - \tilde{\tau}\,(k_1, k_2, h_1, h_2)) + \tilde{b}_{h_2,h_3}^{k_2}(s_{h_2}^*) + \min_t b_{h_4}^{k_3}(t)$$

after computing $s_{h_2}^*$ for the first equation of (19).

6 Computational experiments

We conducted various computational experiments to evaluate the proposed algorithm. The algorithm was coded in C language and run on an IBM-compatible PC (Intel Pentium 4, 2.8 GHz, 1 GB memory). We used the following benchmark instances of the VRPHTW: (1) Solomon's benchmark instances [29], and (2) Gehring and Homberger's benchmark instances [13].

6.1 Benchmark instances

The benchmark instances by Solomon [29] are 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 proportional to the corresponding distances. Each customer i has one time window $[w_i^{\mathrm{l}}, w_i^{\mathrm{r}}]$, an amount of requirement q'_i and a service time u_i. The depot is located at the center of the square and it also has one time window $[w_0^{\mathrm{l}}, w_0^{\mathrm{r}}]$; that is, each vehicle departs from the depot at w_0^{l} and must return to the depot by w_0^{r}. All vehicles have an identical capacity Q'. Both time window and capacity constraints must be satisfied strictly. 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. Solomon's benchmark instances

consist of six different sets of problem instances named C1, C2, R1, R2, RC1 and RC2, respectively. Customers are clustered in groups in type C and are uniformly distributed in type R, and type RC has these two characteristics. 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. On the other hand, for 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 a larger number of customers have been added by Gehring and Homberger [13], where these instances are divided into five groups by the number of customers; 200, 400, 600, 800 and 1000, and each group has 60 instances (more precisely, 10 instances for each of six sets C1, C2, R1, R2, RC1 and RC2).

6.2 Parameters

In order to solve the above instances by our algorithm, we define the penalty functions as follows. For each customer i, one time window $[w_i^{\mathrm{l}}, w_i^{\mathrm{r}}]$ is given, and the service for customer i must be started in this period. Based on the given time window, we define the time penalty function $p_i(t)$ by the following equation:

$$p_i(t) = \begin{cases} 100000(-t + w_i^{\mathrm{l}}), & -\infty \leq t < w_i^{\mathrm{l}}, \\ 0, & w_i^{\mathrm{l}} \leq t \leq w_i^{\mathrm{r}}, \\ 100000(t - w_i^{\mathrm{r}}), & w_i^{\mathrm{r}} < t \leq +\infty. \end{cases}$$

The time penalty functions for the depot, $p_0^{\mathrm{d}}(t)$ and $p_0^{\mathrm{a}}(t)$, are defined as follows:

$$p_0^{\mathrm{d}}(t) = \max\{100000(-t + w_0^{\mathrm{l}}), 0\},$$
$$p_0^{\mathrm{a}}(t) = \max\{100000(t - w_0^{\mathrm{r}}), 0\},$$

where the depot has an identical time window $[w_0^{\mathrm{l}}, w_0^{\mathrm{r}}]$ for all vehicles. As for the capacity constraint, we set $q_i := 100000 q_i'$ and $Q_k := 100000 Q'$ for the given input q_i' and Q'.

We then consider how to determine the number of vehicles for each instance; it is a given input for our formulation but a decision variable for the test instances used in our experiments. We first set the number of vehicles to the previous best known values as of May 11, 2006, presented in the web site (`http://www.sintef.no/static/am/opti/projects/top/vrp/benchmarks.html`). If we cannot find a solution which satisfies the time window and capacity constraints, we increase the number of vehicles by one and conduct another experiment. Such experiments are repeated until we can find a feasible solution for VRPHTW. On the other hand, if we can find a feasible solution of VRPHTW with the best known m and the current m is larger than the trivial lower bound $\left\lceil \sum_{i \in V \setminus \{0\}} q_i / Q \right\rceil$, we decrease the number of vehicles by one and conduct another experiment. Such experiments are repeated until we cannot find a feasible solution for VRPHTW.

As initial values of κ^{p} and κ^{q}, we conducted two types of experiments: (1) $\kappa^{\mathrm{p}} = 1/100000$ and $\kappa^{\mathrm{q}} = 1/100000$ (called ILS-1), and (2) $\kappa^{\mathrm{p}} = 1$ and $\kappa^{\mathrm{q}} = 1$ (called ILS-2). The best solution found in our algorithm may not be a feasible solution of VRPHTW even if we can find feasible solutions of VRPHTW in our search process. To overcome such a situation, we also store the best feasible solution of VRPHTW found in our algorithm. We set other parameter values as follows: $L_{\mathrm{path}}^{\mathrm{iopt}} = 3$, $L_{\mathrm{ins}}^{\mathrm{iopt}} = 10$, $L^{\mathrm{2opt}} = 10$, $L^{\mathrm{pins}} = 3$, $L_{\mathrm{rev}}^{\mathrm{pins}} = 3$, $L^{\mathrm{cross}} = 3$, $L_{\mathrm{rev}}^{\mathrm{cross}} = 3$, $L^{\mathrm{dlist}} = 20$, $\omega^{\mathrm{dlist}} = 1.5$ and $L^{\mathrm{tlist}} = 1$ for all experiments. The time limit of our metaheuristic algorithm for instances with 100, 200, 400, 600, 800 and 1000 customers are 1000, 2000, 4000, 6000, 8000 and 10000 seconds, respectively.

6.3 Computational results

We compare the solutions obtained by our algorithm with existing results. The results for the Solomon's instances are shown in Table 1, and the results for the Gehring and Homberger's instances are shown in Tables 2, 3, 4, 5 and 6. Each row C1, C2, R1, R2, RC1 and RC2 denotes the problem set. The upper part of each cell represents the mean number of vehicles (called MNV) with respect to all instances in the set, and the lower part of each cell represents the mean total distance (called MTD). "CNV" stands for the cumulative number of vehicles, and "CTD" stands for the cumulative total distance, which are usually used in the literature to compare the results on these instances. "Computer" describes the spec of computers used in the experiments, where "P", "S" and "A" mean Pentium, SUN Ultra and Advanced Micro Devices, respectively, and the number in this row means the clock frequency of the CPU (e.g., S 10M means SUN Ultra 10 MHz and P 2.8G means Pentium 2.8 GHz). "CPU" and "Runs" describe the average CPU time in minutes and the number of independent runs for each instance, reported in each article. Using the spec data presented in the web page (`http://www.specbench.org/`), we estimate the total computation time used for one instance if each algorithm will be run on a PC with Pentium 2.8 GHz processor. "Estimated time" represents this estimated time in minutes. An asterisk "$*$" in the row MNV (resp., CNV) means that it is the smallest without ties in the row, and thus the algorithm marked "$*$" gives the best performance for the instance set. 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 Table 1, columns "B" is the result of algorithm RVNS(2) proposed by Bräysy [4], "BBB" is the result by Berger et al. [3], "BVH" is the result by Bent and Van Hentenryck [2], "GH02" is the result of algorithm HM4C proposed by Gehring and Homberger [12], "HG99" is the result by Homberger and Gehring [18], "HG05" is the result by Homberger and Gehring [19], "IIKTUY" is the result of algorithm ILS(15000s) by Ibaraki et al. [20], "ILS-1" and "ILS-2" are the results of our algorithm with different initial values κ^{p} and κ^{q}. Computation time of "HG05" is not clearly stated in [19]. In this table, we can see that our algorithms (ILS-1 and ILS-2) perform slightly worse than the other algorithms.

In Tables 2 to 6, columns "BHD" is the result by Bräysy et al. [7], "BVH" is the result by Bent and Van Hentenryck [2], "GH99" is the result by Gehring and Homberger [13], "GH02" is the result by Gehring and Homberger [12], "LC" is the result by Le Bouthillier and Crainic [24], "LL" is the result by Li and Lim [25], "MB" is the result by Mester and Bräysy [26], "ILS-1" and "ILS-2" are the results of our algorithm. Computation time of "BVH" is not clearly stated in [2]. The solution quality of our algorithm for 200 customer instances is competitive with the best quality attained by Mester and Bräysy [26]; however, we use more computation time. The solution quality of our algorithm for larger instances is much better than the other algorithms in Tables 3, 4, 5 and 6. For all of these instance sets, we succeeded in reducing the best-known CNV. These results indicates that our algorithm is highly efficient to solve the vehicle routing problem with hard time windows, in spite of its generality, especially for large scale instances. More details of our computational results and the best known solutions found by our algorithm on Gehring and Homberger's instances are presented in our web page (`http://www.al.cm.is.nagoya-u.ac.jp/~yagiura/papers/vrpctw_abst.html`).

7 Conclusions and discussions

In this paper, we considered the vehicle routing problem with convex time penalty functions and proposed a metaheuristic algorithm. We utilized the iterated local search to assign customers

Table 1: The results for 100-customer benchmark instances

	B	BBB	BVH	GH02	HG99	HG05	IIKTUY	ILS-1	ILS-2
C1	10	10	10	10	10	10	10	10	10
	828.38*	828.48	828.38*	828.63	828.38*	828.38*	828.38*	828.38*	828.38*
C2	3	3	3	3	3	3	3	3	3
	589.86*	589.93	589.86*	590.33	589.86*	589.86*	589.86*	589.86*	589.86*
R1	11.92	11.92	11.92	12	11.92	11.92	11.92	12	12.08
	1222.12	1221.1	1213.25	1217.57	1228.06	1212.73*	1217.4	1217.99	1212.09
R2	2.73	2.73	2.73	2.73	2.73	2.73	2.73	2.73	2.73
	975.12	975.43	966.37	961.29	969.95	955.03*	959.11	967.97	960.95
RC1	11.5	11.5	11.5	11.5	11.63	11.5	11.5	11.63	11.5
	1389.58	1389.89	1384.22*	1395.13	1392.57	1386.44	1391.03	1384.67	1391.46
RC2	3.25	3.25	3.25	3.25	3.25	3.25	3.25	3.25	3.25
	1128.38	1159.37	1141.24	1139.37	1144.43	1123.17	1122.79*	1128.77	1127
CNV	405	405	405	406	406	405	405	407	407
CTD	57710	57952	57567	57641	57876	57309*	57444	57545	57437
Computer	P 200M	P 400M	S 10M	P 400M	P 200M	unknown	P 1G	P 2.8G	P 2.8G
CPU (min)	87	30	120	4×20.9	13.8	n/a	250	16.7	16.7
Runs	1	3	5	5	10	n/a	1	1	1
Estimated time	3.8	9.4	104.3	43.6	6	n/a	108.7	16.7	16.7

Table 2: The results for 200-customer benchmark instances

	BHD	BVH	GH99	GH02	LC	LL	MB	ILS-1	ILS-2
C1	18.9	18.9	18.9	18.9	18.9	19.1	18.8*	18.9	18.9
	2749.83	2726.63	2782	2842.08	2743.66	2728.6	2717.21	2732.03	2734.42
C2	6	6	6	6	6	6	6	6	6
	1842.65	1860.17	1846	1856.99	1836.1	1854.9	1833.57	1834.83	1833.37*
R1	18.2	18.2	18.2	18.2	18.2	18.3	18.2	18.2	18.2
	3718.3	3677.96	3705	3855.03	3676.95	3736.2	3618.68*	3665.77	3655.24
R2	4	4.1	4	4	4	4.1	4	4	4
	3014.28	3023.62	3055	3032.49	2986.01	3023	2942.92*	2965.64	2958.56
RC1	18	18	18	18.1	18	18.3	18	18	18
	3329.62	3279.99	3555	3674.91	3449.71	3385.8	3221.34*	3287.61	3275.38
RC2	4.4	4.5	4.3	4.4	4.3	4.9	4.4	4.3	4.3
	2585.89	2603.08	2675	2671.34	2613.75	2518.7	2519.79	2562.56*	2576.12
CNV	695	697	694	696	694	707	694	694	694
CTD	172406	171715	176180	179328	173061	172472	168573*	170484	170331
Computer	A 700M	S 10M	P 200M	P 400M	P 933M	P 545M	P 2G	P 2.8G	P 2.8G
CPU (min)	2.4	n/a	4×10	4×2.1	5×10	182.1	8	33.3	33.3
Runs	3	n/a	1	3	1	3	1	1	1
Estimated time	2.1	n/a	2.4	3.8	21.7	112.4	5.9	33	33

Table 3: The results for 400-customer benchmark instances

	BHD	BVH	GH99	GH02	LC	LL	MB	ILS-1	ILS-2
C1	37.9	38	38	38	37.9	38.7	37.9	37.7	37.6*
	7230.48	7220.96	7584	7855.82	7447.09	7181.4	7148.27	7282.15	7302.50
C2	12	12	12	12	12	12.1	12	12	11.8*
	3894.48	4154.4	3935	3940.19	3940.87	4017.1	3840.85	3851.96	3985.21
R1	36.4	36.4	36.4	36.4	36.5	36.6	36.3*	36.4	36.4
	8692.17	8713.37	8925	9478.22	8839.28	8912.4	8530.03	8746.94	8788.54
R2	8	8	8	8	8	8	8	8	8
	6382.63	6959.75	6502	6650.28	6437.68	6610.6	6209.94*	6269.9	6251.54
RC1	36	36.1	36.1	36.1	36	36.5	36	36	36
	8305.55	8330.98	8763	9294.99	8652.01	8377.9	8066.44*	8405.32	8471.85
RC2	8.9	8.9	8.6	8.8	8.6	9.5	8.8	8.6	8.6
	5407.87	5631.7	5518	5629.43	5511.22	5466.2	5243.06	5337.5	5328.84*
CNV	1391	1393	1390	1392	1390	1414	1389	1387	1384*
CTD	399132	410112	412270	428489	408281	405656	390386	398938	401285
Computer	A 700M	S 10M	P 200M	P 400M	P 933M	P 545M	P 2G	P 2.8G	P 2.8G
CPU (min)	7.9	n/a	4×20	4×7.1	5×20	359.8	17	66.6	66.6
Runs	3	n/a	1	3	1	3	1	1	1
Estimated time	6.8	n/a	4.8	13	43.3	221.8	12.5	66.6	66.6

Table 4: The results for 600-customer benchmark instances

	BHD	BVH	GH99	GH02	LC	LL	MB	ILS-1	ILS-2
C1	57.8	57.8	57.9	57.7	57.9	58.2	57.8	57.5	57.5
	14165.9	14357.11	14792	14817.25	14205.58	14267.3	14003.09	14116.97*	14128.87
C2	18	17.8	17.9	17.8	17.9	18.2	17.8	17.4	17.4
	7528.73	8259.04	7787	7889.96	7743.92	8202.6	7455.83	7945.56*	7991.70
R1	54.5	55	54.5	54.5	54.8	55.2	54.5	54.5	54.5
	19081.18	19308.62	20854	21864.47	19869.82	19744.8	18358.68*	19844.39	19963.56
R2	11	11	11	11	11.2	11.1	11	11	11
	13054.83	14855.43	13335	13656.15	13093.97	13592.4	12703.52	12539.78	12496.54*
RC1	55	55.1	55.1	55	55.2	55.5	55	55	55
	16994.22	17035.91	18411	19114.02	17678.13	17320	16418.63*	17278.81	17395.51
RC2	12.1	12.4	11.8	11.9	11.8	13	12.1	11.6	11.6
	11212.36	11987.89	11522	11670.29	11034.71	11204.9	10677.46	10791.7	10743.03*
CNV	2084	2091	2082	2079	2088	2112	2082	2070	2070
CTD	820372	858040	867010	890121	836261	843320	796172	825172*	827192
Computer	A 700M	S 10M	P 200M	P 400M	P 933M	P 545M	P 2G	P 2.8G	P 2.8G
CPU (min)	16.2	n/a	4×30	4 × 12.9	5× 30	399.8	40	100	100
Runs	3	n/a	1	3	1	3	1	1	1
Estimated time	13.9	n/a	7.1	23.5	65	246.4	29.4	100	100.0

Table 5: The results for 800-customer benchmark instances

	BHD	BVH	GH99	GH02	LC	LL	MB	ILS-1	ILS-2
C1	76.3	76.1	76.7	76.1	76.3	77.4	76.2	75.7*	75.8
	25170.88	25391.67	26528	26936.68	25668.82	25337.02	25132.27	25487.55	25337.93
C2	24.2	24.4	24	23.7	24.1	24.4	23.7	23.4*	23.5
	11648.92	14253.83	12451	11847.92	11985.11	11956.6	11352.29	11860.9	11726.25
R1	72.8	72.7*	72.8	72.8	73.1	73	72.8	72.8	72.8
	32748.06	33337.91	34586	34653.88	33552.4	33806.34	31918.47	33275.72	33413.41
R2	15	15	15	15	15	15.1	15	15	15
	21170.15	24554.63	21697	21672.85	21157.56	21709.39	20295.28	20209.92	20174.32*
RC1	73	73	72.4	72.3	72.3	73.2	73	72.4	72.4
	30005.95	30500.15	38509	40532.35	37722.62*	31282.54	30731.07	34621.63	35296.29
RC2	16.3	16.6	16.1	16.1	15.8	17.1	15.8	15.7*	15.8
	17686.65	18940.84	17741	17941.23	17441.6	17561.22	16729.18	16666.76	16665.08
CNV	2776	2778	2770	2760	2766	2802	2765	2750*	2753
CTD	1384306	1469790	1515120	1535849	1475281	1416531	1361586	1421225	1426133
Computer	A 700M	S 10M	P 200M	P 400M	P 933M	P 545M	P 2G	P 2.8G	P 2.8G
CPU (min)	26.2	n/a	4×40	4×23.2	5×40	512.9	145	133.3	133.3
Runs	3	n/a	1	3	1	3	1	1	1
Estimated time	22.5	n/a	9.5	42.3	86.6	316.1	106.5	133.3	133.3

Table 6: The results for 1000-customer benchmark instances

	BHD	BVH	GH99	GH02	LC	LL	MB	ILS-1	ILS-2
C1	95.8	95.1	96	95.4	95.3	96.3	95.1	94.5*	94.6
	42086.77	42505.35	43273	43392.59	43283.92	42428.5	41569.67	42459.35	42329.33
C2	30.6	30.3	30.2	29.7	29.9	30.8	29.7	29.4*	29.5
	17035.88	18546.13	17570	17574.72	17443.5	17294.9	16639.54	16986.46	16679.76
R1	92.1	92.8	91.9	91.9	92.2	92.7	92.1	91.9	91.9
	50025.64	51193.47	57186	58069.61	55176.95	50990.8	49281.48	53366.10*	54909.63
R2	19	19	19	19	19.2	19	19	19	19
	31458.23	36736.97	31930	31873.62	30919.77	31990.9	29860.32	29546.19*	29589.71
RC1	90	90.2	90	90.1	90	90.4	90	90	90
	46736.92	48634.15	50668	50950.14	49711.36	48892.4	45396.41*	48275.2	48768.87
RC2	19	19.4	19	18.5	18.5	19.8	18.7	18.3*	18.4
	25994.12	29079.78	27012	27175.98	26001.11	26042.3	25063.51	24904.08	24667.92
CNV	3465	3468	3461	3446	3451	3490	3446	3431*	3434
CTD	2133376	2266959	2276390	2290367	2225366	2176398	2078110	2155374	2169452
Computer	A 700M	S 10M	P 200M	P 400M	P 933M	P 545M	P 2G	P 2.8G	P 2.8G
CPU (min)	39.6	n/a	4×50	4×30.	5×50	606.3	600	166.7	166.7
Runs	3	n/a	1	3	1	3	1	1	1
Estimated time	34	n/a	11.9	54.9	108.3	373.5	440.8	166.7	166.7

to vehicles and to determine the visiting order for each vehicle. To make our local search efficient, we introduced some ideas to prune the neighborhoods. We also proposed an algorithm of dynamic programming to compute the optimal start time of services at customers in a given route. The time complexity of our DP algorithm is $\mathbf{O}(\delta_k \log \delta_k)$ for each vehicle k, where δ_k is the total number of linear pieces in the penalty functions for all the customers in a route σ_k. We proposed another DP algorithm to evaluate solutions in our neighborhoods efficiently; the amortized time complexity of this algorithm is $\mathbf{O}(\log \delta_k)$ for each new route σ_k under appropriate assumptions.

The objective function of our formulation is the sum of the total distance and penalties caused by violations of the time window and capacity constraints. In order to treat both of the VRPHTW and the VRPSTW effectively, we incorporated some ideas in our algorithm. To evaluate solutions in neighborhoods, we utilized an evaluation function, which is slightly different from the original objective function and equipped with an adaptive mechanism to control the penalty weights.

We conducted computational experiments on representative benchmark instances of the VRP with capacity and time window constraints, and compared our algorithm with some existing algorithms for this problem. Our results for larger scale instances such as those with 400, 600, 800 or 1000 customers are the best among the recent results for these benchmark instances.

For simplicity of the paper, we did not explain some ideas to treat more general problems, which we have implemented. We have assumed that all vehicles have the same time penalty functions p_0^{d} and p_0^{a} for departure from the depot and arrival at the depot. However, we can treat a situation where each vehicle has its own time penalty function for departure and arrival, without increasing the time complexity. In this case, however, we cannot use the 2-opt* operation in the form explained in Section 4.2, since the operation connects the first part and the last part of two different routes. Therefore, we should treat a 2-opt* operation as a special case of the cross exchange operations. We have also assumed that each customer i has an amount q_i of the resource to be delivered from the depot in this paper. We can treat some general problems as for the capacity constraint with slight modifications of our framework; e.g., the VRP with multi-resource and the VRP with backhauls [15].

Acknowledgment

This research was partially supported by Scientific Grant-in-Aid, by the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

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

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

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

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

[5] O. Bräysy and M. Gendreau, "Vehicle routing problem with time windows, Part I: route construction and local search algorithms," *Transportation Science*, 39 (2005) 104–118.

[6] O. Bräysy and M. Gendreau, "Vehicle routing problem with time windows, Part II: metaheuristics," *Transportation Science*, 39 (2005) 119–139.

[7] 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 (2004) 586–605.

[8] J. S. Davis and J. J. Kanet, "Single-machine scheduling with early and tardy completion costs," *Naval Research Logistics*, 40 (1993) 85–101.

[9] C. De Jong, G. Kant and A. Van Vliet, "On finding minimal route duration in the vehicle routing problem with multiple time windows," Manuscript, Department of Computer Science, Utrecht University, Holland, 1996 (available from `http://www.cs.uu.nl/research/projects/alcom/wp4.3.html`).

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

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

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

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

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

[15] M. Goetschalckx and C. Jacobs-Blecha, "The vehicle routing problem with backhauls," *European Journal of Operational Research*, 42 (1989) 39–51.

[16] H. Hashimoto, T. Ibaraki, S. Imahori and M. Yagiura, "The vehicle routing problem with flexible time windows and traveling times," *Discrete Applied Mathematics*, (to appear).

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

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

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

[20] 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 (2005) 206–232.

[21] D. S. Johnson, "Local optimization and the traveling salesman problem," in *Automata, Languages and Programming*, Vol. 443 of *Lecture Notes in Computer Science*, M. S. Paterson (ed.), 446–461, Springer-Verlag, Berlin, 1990.

[22] D. S. Johnson and L. A. McGeoch, "The traveling salesman problem: a case study," in *Local Search in Combinatorial Optimization*, E. H. L. Aarts and J. K. Lenstra (eds.), 215–310, John Wiley & Sons, Chichester, 1997.

[23] 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 (1992) 69–85.

[24] A. Le Bouthillier and T. G. Crainic, "A hybrid genetic algorithm for the vehicle routing problem with time windows," *Working Paper, Centre for Research on Transportation* University of Montreal, Canada, 2003.

[25] H. Li and A. Lim, "Large scale time-constrained vehicle routing problems: a general metaheuristic framework with extensive experimental results", *AI Review*, submitted for publication.

[26] D. Mester and O. Bräysy, "Active guided evolution strategies for the large scale vehicle routing problem with time windows", *Computers & Operations Research*, 32 (2005) 1593–1614.

[27] I. Or, Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking, Ph.D. Thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL (1976).

[28] J. Y. Potvin, T. Kervahut, B. L. Garcia and J. M. Rousseau, "The vehicle routing problem with time windows, Part 1: tabu search," *INFORMS Journal on Computing*, 8 (1996) 158–164.

[29] M. M. Solomon, "The vehicle routing and scheduling problems with time window constraints," *Operations Research*, 35 (1987) 254–265.

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

[31] E. 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 (1997) 170–186.