Available online at www.sciencedirect.com

ScienceDirect

Discrete Applied Mathematics 154 (2006) 2271–2290

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

The vehicle routing problem with flexible time windows and traveling times

Hideki Hashimoto[a,*], Toshihide Ibaraki[b], Shinji Imahori[c], Mutsunori Yagiura[a]

[a]*Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan*
[b]*Department of Informatics, School of Science and Technology, Kwansei Gakuin University, Japan*
[c]*Department of Mathematical Informatics, Graduate School of Information Science and Technology, University of Tokyo, Japan*

Received 2 July 2004; received in revised form 2 December 2004; accepted 19 May 2005
Available online 6 June 2006

Abstract

We generalize the standard vehicle routing problem by allowing soft time window and soft traveling time constraints, where both constraints are treated as cost functions. With the proposed generalization, the problem becomes very general. In our algorithm, we use local search to determine the routes of vehicles. After fixing the route of each vehicle, we must determine the optimal start times of services at visited customers. 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. Finally we report computational results on benchmark instances, and confirm the benefits of the proposed generalization.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Dynamic programming; General time windows; Flexible traveling time; Local search; Vehicle routing problem

1. Introduction

The *vehicle routing problem* (VRP) is the problem of minimizing the total travel distance of a number of vehicles, under various constraints, where every customer must be visited exactly once by a vehicle [10,11,24]. 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 [15,19,22,25]. The capacity constraint signifies that the total load on a route cannot exceed the capacity of the assigned vehicle. 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, and it has been a subject of intensive research focused mainly on heuristic and metaheuristic approaches [6,7]. Among recent approaches are a multi-start local search [8], a two-stage hybrid local search [2], a hybrid genetic algorithm [3], a reactive variable neighborhood search [5], a two-phase hybrid metaheuristic algorithm [13], a variable neighborhood decent with constraint-based operators [21] and so on. See extensive surveys by Bräysy and Gendreau [6,7] for heuristic and metaheuristic approaches.

* Corresponding author. Tel./fax: +81 75 753 5514.
E-mail address: hasimoto@amp.i.kyoto-u.ac.jp (H. Hashimoto).

0166-218X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2006.04.009

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). For VRPHTW, even just finding a feasible schedule with a given number of vehicles is known to be NP-complete, because it includes the one-dimensional bin packing problem as a special case [12]. It may not be reasonable to restrict the search only within the feasible region of VRPHTW, 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 treated as soft in this paper. In our previous paper [14], we have already considered soft time window constraints, where their cost functions can be non-convex and/or discontinuous as long as it is piecewise linear. This formulation is quite general; e.g., one or more time slots can be assigned to each customer.

In this paper, 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 non-trivial 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 paper.

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.

2. Problem

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

(1) a fixed quantity a_i ($\geqslant 0$) 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 ($\geqslant 0$) of *k*;
(4) a distance d_{ij} ($\geqslant 0$) 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 non-negative, piecewise linear and lower semicontinuous (i.e., $p_i(t)\leqslant \lim_{\varepsilon\to 0}\min\{p_i(t+\varepsilon), p_i(t-\varepsilon)\}$ 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 non-negative. Similarly, we assume that each traveling time cost function $q_{ij}(t)$ is non-negative, piecewise linear and lower semicontinuous. We also assume $q_{ij}(t)=+\infty$ for $t<0$ so that the traveling time *t* between customers is non-negative. 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

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

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 $k \in M$ 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 $i \in V\backslash\{0\}$ 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

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

$$p_{\mathrm{sum}}(\boldsymbol{s}) = \sum_{i\in V\backslash\{0\}} p_i(s_i) + \sum_{k\in M} p_0(s_k^{\mathrm{a}}), \tag{2}$$

$$q_{\mathrm{sum}}(\boldsymbol{\sigma}, \boldsymbol{s}) = \sum_{k\in M} \sum_{h=0}^{n_k-1} q_{\sigma_k(h),\sigma_k(h+1)}(s_{\sigma_k(h+1)} - s_{\sigma_k(h)}) + \sum_{k\in M} q_{\sigma_k(n_k),0}(s_k^{\mathrm{a}} - s_{\sigma_k(n_k)}), \tag{3}$$

$$a_{\mathrm{sum}}(\boldsymbol{\sigma}) = \sum_{k\in M} \max\left\{ \sum_{i\in V\backslash\{0\}} a_i y_{ik}(\boldsymbol{\sigma}) - u_k, 0 \right\}. \tag{4}$$

Then, the problem can be formulated as follows:

$$\text{minimize} \quad d_{\mathrm{sum}}(\boldsymbol{\sigma}) + p_{\mathrm{sum}}(\boldsymbol{s}) + a_{\mathrm{sum}}(\boldsymbol{\sigma}) + q_{\mathrm{sum}}(\boldsymbol{\sigma}, \boldsymbol{s}) \tag{5}$$

$$\text{subject to} \quad \sum_{k\in M} y_{ik}(\boldsymbol{\sigma}) = 1, \quad i \in V\backslash\{0\}. \tag{6}$$

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 (5)–(6) by using the following $p_i(t)$ and $q_{ij}(t)$:

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

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

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 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 3.1. Next, in Section 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, 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:

$$\begin{aligned} &\text{minimize} \quad f_{\mathrm{OSTP}}(\boldsymbol{s}) = \sum_{i=1}^{n_k+1} p_i(s_i) + \sum_{i=1}^{n_k+1} q_{i-1,i}(s_i - s_{i-1}) \\ &\text{subject to} \quad s_0 = 0. \end{aligned} \tag{7}$$

3.1. NP-hardness

Theorem 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, to OSTP. KP with n' items is defined by

$$\begin{aligned} &\text{maximize} \quad \sum_{i=1}^{n'} c_i x_i \\ &\text{subject to} \quad \sum_{i=1}^{n'} w_i x_i \leqslant b, \\ &\qquad\qquad\quad x_i \in \{0, 1\}, \quad i = 1, 2, \ldots, n'. \end{aligned} \tag{8}$$

Note that objective function (8) is equivalent to

$$\text{minimize} \sum_{i=1}^{n'} c_i(1 - x_i) = \sum_{i=1}^{n'} c_i - \sum_{i=1}^{n'} c_i x_i. \tag{9}$$

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

$$p_i(t) = \begin{cases} 0, & t \in [0, b], \\ +\infty, & t \in (b, +\infty), \end{cases} \tag{10}$$

$$q_{i-1,i}(t) = \begin{cases} c_i, & t \in [0, w_i), \\ 0, & t \in [w_i, +\infty). \end{cases} \tag{11}$$

We will show that this OSTP instance has the same objective value as KP with objective function (9).

Let us define a vector $\tilde{\boldsymbol{s}} = (\tilde{s}_0, \tilde{s}_1, \ldots, \tilde{s}_{n_k+1})$ 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 $i \geqslant 1$:

$$\tilde{s}_i = \begin{cases} \tilde{s}_{i-1} & \text{if } \tilde{x}_i = 0, \\ \tilde{s}_{i-1} + w_i & \text{if } \tilde{x}_i = 1. \end{cases} \tag{12}$$

Then

$$\tilde{s}_i = \sum_{j \leqslant i} w_j \tilde{x}_j \leqslant b, \quad i = 1, 2, \ldots, n_k + 1 \tag{13}$$

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 (10) and (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 (11) and (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:

$$\hat{x}_i = \begin{cases} 1 & \text{if } \hat{s}_i - \hat{s}_{i-1} \geqslant w_i, \\ 0 & \text{if } \hat{s}_i - \hat{s}_{i-1} < w_i. \end{cases} \tag{14}$$

Then we have

$$\sum_{i=1}^{n_k+1} w_i \hat{x}_i \leqslant \sum_{i=1}^{n_k+1} (\hat{s}_i - \hat{s}_{i-1})\hat{x}_i \leqslant \sum_{i=1}^{n_k+1} (\hat{s}_i - \hat{s}_{i-1}) = \hat{s}_{n_k+1} \leqslant b. \tag{15}$$

Note that the last inequality is derived from (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

$$\sum_{i=1}^{n_k+1} c_i(1 - \hat{x}_i) = \sum_{i=1}^{n_k+1} p_i(\hat{s}_i) + \sum_{i=1}^{n_k+1} q_{i-1,i}(\hat{s}_i - \hat{s}_{i-1}) \tag{16}$$

holds from definitions (11) and (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. □

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 $0, 1, 2, \ldots, h$ served by vehicle k in this order under the condition that customers $0, 1, \ldots, h-1$ are served before time t and customer h is served exactly at time t (i.e., $\min_{s_0=0, s_h=t} \sum_{i=1}^{h} p_i(s_i) + \sum_{i=1}^{h} q_{i-1,i}(s_i - s_{i-1})$).

Throughout this paper, we call this a *forward minimum cost function*. Then $f_h^k(t)$ can be computed by the following recursive formula:

$$f_0^k(t) = \begin{cases} +\infty, & t \neq 0, \\ 0, & t = 0, \end{cases}$$
$$f_h^k(t) = p_h(t) + \min_{0 \leqslant t' \leqslant t} \{f_{h-1}^k(t') + q_{h-1,h}(t - t')\}, \quad 1 \leqslant h \leqslant n_k + 1, \ -\infty < t < +\infty. \tag{17}$$

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 (see Appendix for the proof).

Lemma 3.1. *An OSTP instance with integer input has an optimal solution whose start times are integers.*

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

$$f_h^k(t) = p_h(t) + \min_{t' \in \{0,1,\ldots,t\}} \{f_{h-1}^k(t') + q_{h-1,h}(t - t')\}, \quad t = 0, 1, 2, \ldots, \tag{18}$$

instead of Eq. (17). In order to compute Eq. (18), 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 $\mathrm{O}(T)$ 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 $\mathrm{O}(n_k T^2)$ 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 non-negative because of the non-negativity of these functions, and hence the gradient of the last piece of $f_h^k(t)$ is also non-negative. Then,

$$T_h = \begin{cases} 0, & h = 0, \\ \max\{T_{h-1} + Q_{h-1,h}, P_h\}, & h \geqslant 1 \end{cases} \tag{19}$$

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 satisfies $T_{h-1} + Q_{h-1,h} < P_h$ (if $T_{h-1} + Q_{h-1,h} \geqslant P_h$ 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 $T = T_{n_k+1} = P_{h^*} + \sum_{h=h^*+1}^{n_k+1} Q_{h-1,h} \leqslant \max_{h\in\{1,2,\ldots,n_k+1\}} P_h + \sum_{h=1}^{n_k+1} Q_{h-1,h}$. Recall that these functions are all given explicitly as the input. This indicates that the time complexity $\mathrm{O}(n_k T^2)$ is pseudo-polynomial order.

Theorem 3.2. *Problem OSTP with integer input can be solved in pseudo-polynomial time.*

3.3. Polynomial time algorithm for convex traveling time cost functions

In this section, we propose a polynomial time algorithm for OSTP (7) in which each traveling time cost function $q_{h-1,h}$ is convex. Here we do not assume integer input as in Section 3.2. 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 [4,9]. 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 [1].

3.3.1. 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 (17) 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, Fig. 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 Fig. 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 $\mathrm{O}(\delta(g) + \delta(g'))$ time.

A non-trivial part in recursion (17) is the computation of $\min_{0\leqslant t'\leqslant t}\{f_{h-1}^k(t') + q_{h-1,h}(t - t')\}$. Even if $q_{h-1,h}$ is convex, f_{h-1}^k 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 oder of their contents. For example, the function g in Fig. 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_{h-1}^k into convex intervals $S_1, S_2, \ldots, S_K$, where $K = \widehat{\delta}(f_{h-1}^k)$, and define the following convex functions F_l for $l = 1, 2, \ldots, K$, which are extended from $f_{h-1}^k(t)$ on S_l. Let $\mathrm{Cl}(S_l) = [c_l^{\mathrm{L}}, c_l^{\mathrm{R}}]$ denote the closure of S_l.

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

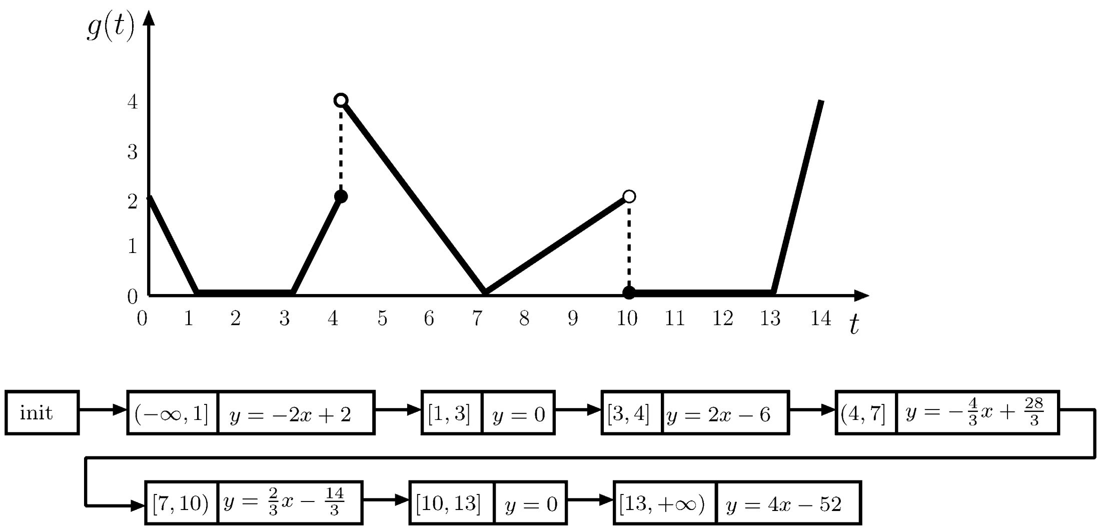
$g(t)$
0
1
2
3
4
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 t
init
$(-\infty, 1]$ $y = -2x + 2$
$[1, 3]$ $y = 0$
$[3, 4]$ $y = 2x - 6$
$(4, 7]$ $y = -\frac{4}{3}x + \frac{28}{3}$
$[7, 10)$ $y = \frac{2}{3}x - \frac{14}{3}$
$[10, 13]$ $y = 0$
$[13, +\infty)$ $y = 4x - 52$

Fig. 1. A function g and the linked list that represents g.

Let

$$e_l(t) = \min_{0 \leqslant t' \leqslant t} \{F_l(t') + q_{h-1,h}(t - t')\}. \tag{20}$$

Then we have

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

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

3.3.2. Computation of e_l

To compute $e_l(t)$ of (20), we use the next theorem since both of F_l and $q_{h-1,h}$ are convex piecewise linear.

Theorem 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*

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

in $\mathrm{O}(\delta(\phi))$ *time, where* $\delta(\phi) \leqslant \delta(\phi_1) + \delta(\phi_2)$ *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 Fig. 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 Fig. 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.

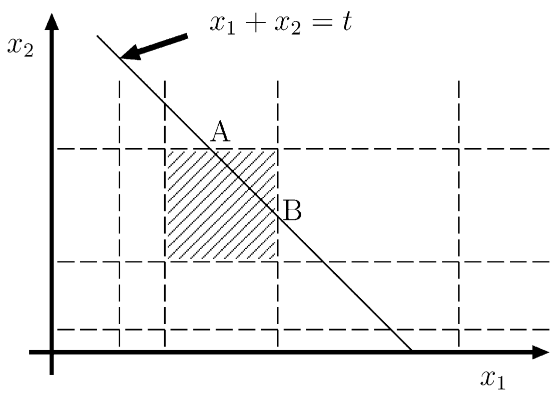

Fig. 2. Breakpoints of $\phi_1(x_1)$ and $\phi_2(x_2)$ on the plane of x_1 and x_2.

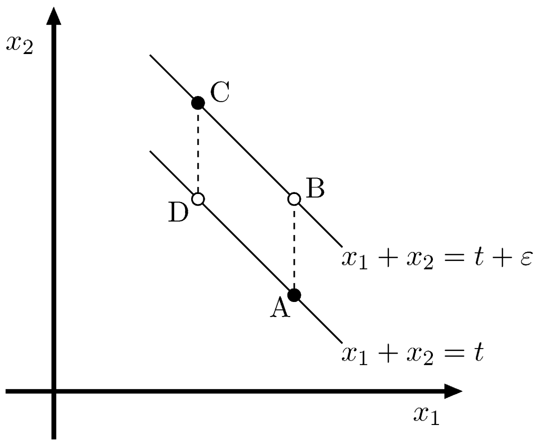

Fig. 3. The points that achieves $\phi(t)$ and $\phi(t + \varepsilon)$.

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 $x_1^*(t + \varepsilon) < x_1^*(t)$. Fig. 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 $x_1^*(t) \leqslant x_1^*(t + \varepsilon)$. Similarly we have $x_2^*(t) \leqslant x_2^*(t + \varepsilon)$ (i.e., $t - x_1^*(t) \leqslant t + \varepsilon - x_1^*(t + \varepsilon)$). Then we have

$$x_1^*(t) \leqslant x_1^*(t + \varepsilon) \leqslant x_1^*(t) + \varepsilon. \tag{21}$$

Thus, trajectory $(x_1^*(t), x_2^*(t))$ for $t = 0 \to +\infty$ is continuous and lies on the lattice of broken lines (see Fig. 2), i.e., it is a non-decreasing staircase curve from (0, 0) to its top right as shown in Fig. 4.

In oder 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. Fig. 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 non-decreasing, the computed ϕ is also convex.

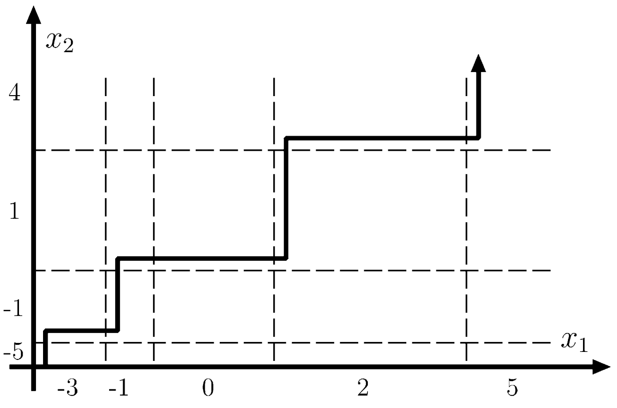

Fig. 4. An example of the trajectory of $(x_1^*(t), x_2^*(t))$ which achieves $\phi(t)$.

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

3.3.3. Lower envelope of all e_l

After obtaining convex functions $e_l(t), l = 1, 2, \ldots, K$ by the algorithm described in Section 3.3.2, 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 $E_L(t) = \min_{1 \leqslant l \leqslant L} e_l(t)$. 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 3.2 and Theorem 3.4 to obtain these data.

Lemma 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., Fig. 4), where in this case the horizontal axis denotes start time s_{h-1} and the vertical axis denotes traveling time $t_{h-1,h}$. Fig. 5 illustrates the situation in which there are two trajectories corresponding to $e_l(t)$ and $e_{l'}(t)$. For a given t, let $t_l^* = \arg\min_{0 \leqslant t' \leqslant t} \{F_l(t') + q_{h-1,h}(t - t')\}$, i.e., $e_l(t) = F_l(t_l^*) + q_{h-1,h}(t - t_l^*)$, and $t_{l'}^* = \arg\min_{0 \leqslant t' \leqslant t} \{F_{l'}(t') + q_{h-1,h}(t - t')\}$. Then $t - t_l^* \geqslant t - t_{l'}^*$ holds, since $t_l^* \in \mathrm{Cl}(S_l)$ and $t_{l'}^* \in \mathrm{Cl}(S_{l'})$. If $t_l^* = t_{l'}^*$ the lemma 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 3.4. *Each e_l appears in the envelope E_L $(l \leqslant L)$, 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 3.2 implies that $e_{l'}(t) - e_l(t)$ is non-increasing 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'. □

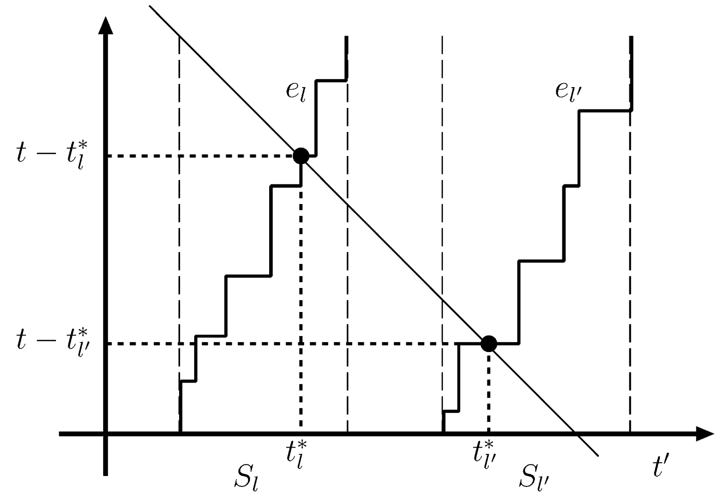

Fig. 5. e_l and $e_{l'}$.

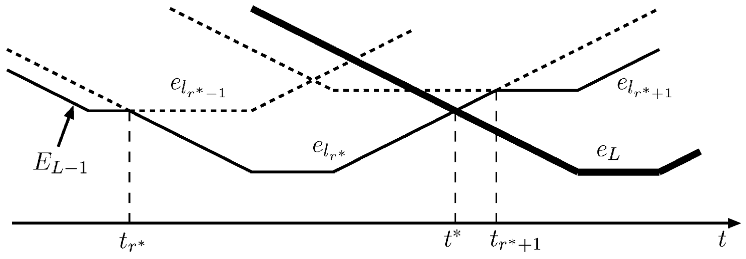

Fig. 6. An example of the lower envelope.

The computation of the lower envelope E_K proceeds as follows:
Compute-Lower-Envelope

Input: Functions $e_1, e_2, \ldots, e_K$.
Output: Their lower envelope E_K.

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

In Step 2, all we have to do is to check how e_L affects $E_{L-1}(t)$. We illustrate how to compute $E_L(t)$ from $E_{L-1}(t)$ with an example of Fig. 6. Assume that $E_{L-1}(t)$ consists of v ($\leqslant L - 1$) 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 $e_{l_r}(t_r) \leqslant e_L(t_r)$ 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 $t \leqslant t^*$ and $e_L(t)$ for $t \geqslant t^*$. 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 $t \geqslant t_r$ 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

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

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 $\mathrm{O}(\sum_{l=1}^{K-1}\delta(e_l))$. Hence the total time complexity of algorithm Compute-Lower-Envelope is $\mathrm{O}(\sum_{l=1}^{K}\delta(e_l))$.

3.3.4. *Time complexity for the dynamic programming*

In order to compute f_h^k from f_{h-1}^k by recursion (17), 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 (17).
(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 $\mathrm{O}(\delta(e_l))$ time for each l, it takes $\mathrm{O}\left(\sum_{l=1}^{\widehat{\delta}(f_{h-1}^k)}\delta(e_l)\right)$ 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 $\mathrm{O}\left(\sum_{l=1}^{\widehat{\delta}(f_{h-1}^k)}\delta(e_l)\right)$ time by algorithm Compute-Lower-Envelope.
(3) We can add p_h to the lower envelope $E_{\widehat{\delta}(f_{h-1}^k)}$ in $\mathrm{O}(\delta(p_h)+\sum_{l=1}^{\widehat{\delta}(f_{h-1}^k)}\delta(e_l))$ 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 $\mathrm{O}(\delta(p_h)+\sum_{l=1}^{\widehat{\delta}(f_{h-1}^k)}\delta(e_l))$ time. For convenience, we introduce the following notations:

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

For the number of convex intervals,

$$\widehat{\delta}(f_h^k)\leqslant\widehat{\delta}(f_{h-1}^k)+\widehat{\delta}(p_h)-1 \tag{22}$$

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,

$$\begin{aligned}\delta(f_h^k)&\leqslant\delta(p_h)+\sum_{l=1}^{\widehat{\delta}(f_{h-1}^k)}\delta(e_l)\\&\leqslant\delta(p_h)+\sum_{l=1}^{\widehat{\delta}(f_{h-1}^k)}\{\delta(F_l)+\delta(q_{h-1,h})\}\\&\leqslant\delta(p_h)+\delta(f_{h-1}^k)+\widehat{\delta}(f_{h-1}^k)\delta(q_{h-1,h})\end{aligned}\tag{23}$$

holds. Note that the first inequality holds because $\delta(E_{\widehat{\delta}(f^k_{h-1})}) \leqslant \sum_{l=1}^{\widehat{\delta}(f^k_{h-1})} \delta(e_l)$, and the second inequality is from Theorem 3.3. By applying (22) recursively, we have $\widehat{\delta}(f^k_h) = \mathrm{O}(\widehat{\Delta}^k_p)$. Similarly, from (23), we have $\delta(f^k_h) = \mathrm{O}(\Delta^k_p + \widehat{\Delta}^k_p \Delta^k_q)$. Consequently, the time to compute f^k_h from f^k_{h-1} is

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

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

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

$$\Delta(\sigma_k) = \sum_{h=1}^{n_k+1} \delta(p_{\sigma_k(h)}) + \left(\sum_{h=1}^{n_k+1} \widehat{\delta}(p_{\sigma_k(h)})\right)\left(\sum_{h=1}^{n_k+1} \delta(q_{\sigma_k(h-1),\sigma_k(h)})\right). \tag{24}$$

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 (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 Fig. 7).

The 2-opt* neighborhood was proposed in [19], which is a variant of the 2-opt neighborhood [16] for the traveling salesman problem (TSP, a special case of VRP in which the number of vehicles is one). 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. The cross exchange neighborhood was proposed in [25]. 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. 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, which is a variant of Or-opt neighborhood used for TSP [18,20]. 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 $L^{\text{intra}}_{\text{ins}}$

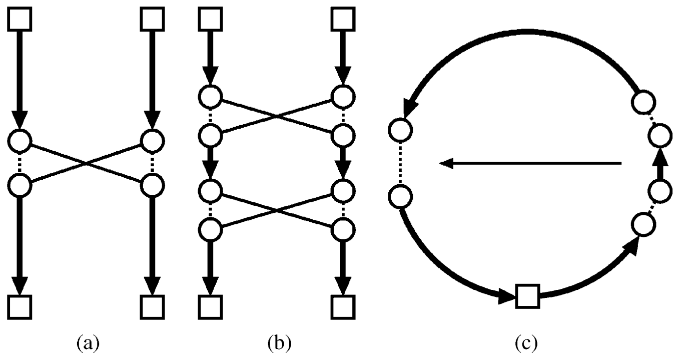

Fig. 7. Neighborhoods in our local search. (a) 2-opt*; (b) cross exchange; (c) or-opt.

(a parameter) from the original position. 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) [17], 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:

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.

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 idea was originally proposed in our previous paper [14]. In this section, for convenience, we discuss the case in which we use the polynomial time algorithm for OSTP in Section 3.3. But the idea is also applicable to the case of the pseudo-polynomial time algorithm in Section 3.2.

5.1. The 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 (Fig. 8).

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 [14].

Now we concentrate on the computation of the minimum cost $(p+q)^*_{\text{sum}}(\sigma_k)$, which is the sum of time window and traveling time costs on route σ_k. We define $b_h^k(t)$ 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 $f_h^k(t)$ be the forward minimum cost function at the hth customer in route σ_k, which was discussed in Section 3. Then, $b_h^k(t)$ can be computed as follows in a symmetric manner:

$$
\begin{aligned}
b_{n_k+1}^k(t) &= p_0(t),\\
b_h^k(t) &= p_h^k(t) + \min_{t'} (b_{h+1}^k(t') + q_{h,h+1}(t'-t)), \quad 1 \leqslant h \leqslant n_k.
\end{aligned} \tag{25}
$$

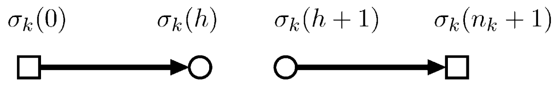

Fig. 8. The former and latter parts of a route σ_k.

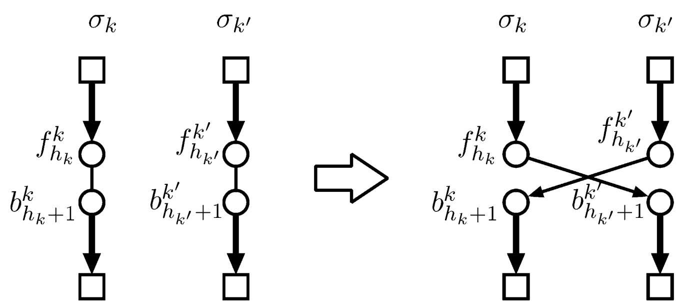

Fig. 9. An example of a 2-opt* operation.

We can then obtain the optimal cost of route σ_k by

$$(p+q)^*_{\text{sum}}(\sigma_k) = \min_t \left(f^k_h(t) + \min_{t'}(b^k_{h+1}(t') + q_{h,h+1}(t'-t)) \right) \tag{26}$$

for any h $(0 \leqslant h \leqslant n_k)$. 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 3.3.4 (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

$$\sum_{i=1}^{h} a_{\sigma_k(i)} \quad \text{and} \quad \sum_{i=h}^{n_k} a_{\sigma_k(i)}, \tag{27}$$

$$\sum_{i=1}^{h} d_{\sigma_k(i-1),\sigma_k(i)} \quad \text{and} \quad \sum_{i=h+1}^{n_k+1} d_{\sigma_k(i-1),\sigma_k(i)}, \tag{28}$$

$$f^k_h(t) \quad \text{and} \quad b^k_h(t) \tag{29}$$

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

5.2. How to apply the basic idea to the solutions in neighborhoods

Now we explain how to apply the above idea to the solutions in neighborhoods. We only discuss the sum of time window and traveling costs since other costs can be similarly treated.

In Fig. 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 $O(\Delta(\sigma_{k'}))$ 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 $\sigma_{k'}$.

To evaluate solutions in the cross exchange neighborhood efficiently (see Fig. 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. Fig. 10 explains the situation. In Fig. 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 $(p+q)^*_{\text{sum}}(\sigma_k) + (p+q)^*_{\text{sum}}(\sigma_{k'})$ of Fig. 10(a). (We can obtain $(p+q)^*_{\text{sum}}(\sigma_k)$ (resp., $(p+q)^*_{\text{sum}}(\sigma_{k'})$) from $\tilde{f}^k_l$ and $b^k_{h_k}$ (resp., from $\tilde{f}^{k'}_{l'}$ and $b^{k'}_{h_{k'}}$).) We then extend

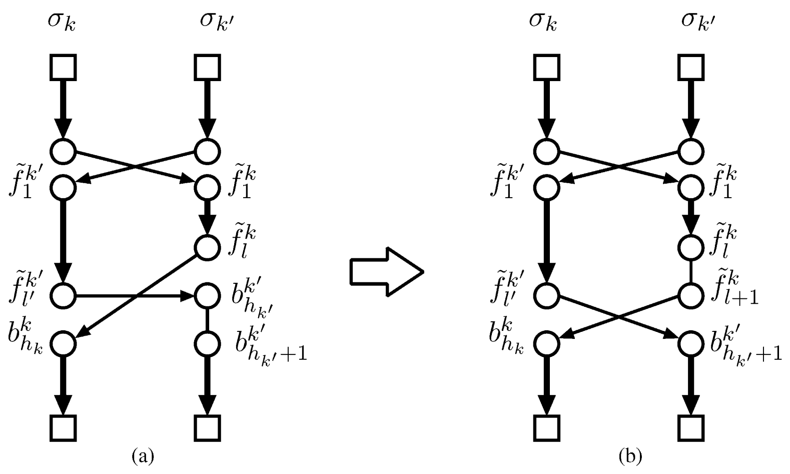

Fig. 10. An example of the search order in the cross exchange neighborhood.

the length of the right path by one (Fig. 10(b)). For this, we can compute $\tilde{f}_{l+1}^k$ from $\tilde{f}_l^k$ by recursion of the dynamic programming in $\mathrm{O}(\Delta(\sigma_k))$ time, and evaluate $(p+q)^*_{\mathrm{sum}}(\sigma_k)+(p+q)^*_{\mathrm{sum}}(\sigma_{k'})$ in $\mathrm{O}(\Delta(\sigma_k)+\Delta(\sigma_{k'}))$ time. Thus, the change in the cost after a cross exchange operation (from the current solution to the solution in Fig. 10(b)) is obtained in $\mathrm{O}(\Delta(\sigma_k)+\Delta(\sigma_{k'}))$ time.

Similarly, the change in the cost for an intra-route operation of route σ_k can be computed in $\mathrm{O}(\Delta(\sigma_k))$ 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 [14].

6. Computational results

We conducted computational experiments to evaluate the proposed algorithm ILS (see Section 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 [23] 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 $[t_i^{\mathrm{r}}, t_i^{\mathrm{d}}]$, 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 1 is the best-known solutions for these instances (the data were 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:

$$p_i(t) = \begin{cases} \alpha_1(t_i^{\mathrm{r}} - t), & t < t_i^{\mathrm{r}}, \\ 0, & t_i^{\mathrm{r}} \leqslant t \leqslant t_i^{\mathrm{d}}, \\ \alpha_1(t - t_i^{\mathrm{d}}), & t_i^{\mathrm{d}} < t, \end{cases}$$

$$q_{ij}(t) = \begin{cases} +\infty, & t < 0.9(\tau_i + t_{ij}), \\ \alpha_2(\tau_i + t_{ij} - t), & 0.9(\tau_i + t_{ij}) \leqslant t < \tau_i + t_{ij}, \\ 0, & \tau_i + t_{ij} \leqslant t, \end{cases} \tag{30}$$

Table 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

where α_1 and α_2 are positive parameters. For other parameters, we used $L^{\text{cross}} = 3$, $L^{\text{intra}}_{\text{path}} = 3$ and $L^{\text{intra}}_{\text{ins}} = 20$, and set the time limit of computation to 2000 s (in conformity with the values in [14]). 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 2 and 3. In each table, column "P" denotes the total deviation of start time of services from the boundaries of time windows (i.e., $P = p_{\text{sum}}(\boldsymbol{s})/\alpha_1$), and column "$Q$" denotes the total amount of shortened traveling time (i.e., $Q = q_{\text{sum}}(\boldsymbol{\sigma}, \boldsymbol{s})/\alpha_2$). 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 2, we set the number of vehicles to be the same as the best-known solutions in Table 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., $p_{\text{sum}}(\boldsymbol{s}) = q_{\text{sum}}(\boldsymbol{\sigma}, \boldsymbol{s}) = a_{\text{sum}}(\boldsymbol{\sigma}) = 0$), otherwise "–" is written, which means that our algorithm encountered no feasible solution. In Table 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 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 non-zero. 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 practical applications. For those instances with $P > 0$ or $Q > 0$, the traveling distance of the

Table 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

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, 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.

7. Conclusion

In this paper, 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

Table 3
Computational results with smaller number of vehicles than the best-known solutions

Instance	d_{sum}	P	Q	a_{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

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.

Appendix A. Proof of Lemma 3.1

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 (7), $s_0^* = 0$ holds. Assume that s_i^* are integers for $i \leqslant h-1$ but s_h^* is not an integer, where $h \leqslant n_k$ 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

$$\frac{\partial}{\partial s_h^*} f_{\text{OSTP}}(\boldsymbol{s}^*) = \frac{\partial}{\partial s_h^*}\{q_{h-1,h}(s_h^* - s_{h-1}^*) + p_h(s_h^*) + q_{h,h+1}(s_{h+1}^* - s_h^*)\}, \tag{A.1}$$

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 (A.1) 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:

$$p_{\tilde{h}}(t) = p_h(t) + p_{h+1}(t + s^*_{h+1} - s^*_h), \tag{A.2}$$

$$q_{h-1,\tilde{h}}(t) = q_{h-1,h}(t), \tag{A.3}$$

$$q_{\tilde{h},h+2}(t) = q_{h,h+1}(s^*_{h+1} - s^*_h) + q_{h+1,h+2}(t - s^*_{h+1} + s^*_h), \tag{A.4}$$

where we define $q_{\tilde{h},h+2}(t)$ only if $h \leqslant n_k - 1$. 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 $(i \leqslant n_k)$ are integers. If $s^*_{n_k+1}$ is not an integer, the gradient of the objective function for $s^*_{n_k+1}$

$$\frac{\partial}{\partial s^*_{n_k+1}} f_{\mathrm{OSTP}}(\boldsymbol{s}^*) = \frac{\partial}{\partial s^*_{n_k+1}} \{q_{n_k,n_k+1}(s^*_{n_k+1} - s^*_{n_k}) + p_{n_k+1}(s^*_{n_k+1})\} \tag{A.5}$$

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

References

[1] R.K. Ahuja, D.S. Hochbaum, J.B. Orlin, Solving the convex cost integer dual network flow problem, Manage. Sci. 49 (2003) 950–964.

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

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

[4] D.P. Bertsekas, Nonlinear Programming, Athena Scientific, Belmont, 1995.

[5] O. Bräysy, A reactive variable neighborhood search for the vehicle routing problem with time windows, INFORMS J. Comput. 15 (2003) 347–368.

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

[7] O. Bräysy, M. Gendreau, Vehicle routing problem with time windows, Part II: metaheuristics, Transportation Sci. 39 (2005) 119–139.

[8] O. Bräysy, G. Hasle, W. Dullaert, A multi-start local search algorithm for the vehicle routing problem with time windows, European J. Oper. Res. 159 (2004) 586–605.

[9] V. Chvátal, Linear Programming, Freeman, New York, 1983.

[10] M. Desrochers, J.K. Lenstra, M.W.P. Savelsbergh, F. Soumis, Vehicle routing with time windows: optimization and approximation, in: B.L. Golden, A.A. Assad (Eds.), Vehicle Routing: Methods and Studies, North-Holland, Amsterdam, 1988, pp. 65–84.

[11] J. Desrosiers, Y. Dumas, M.M. Solomon, F. Soumis, Time constrained routing and scheduling, in: M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser (Eds.), Handbooks in Operations Research and Management Science, Network Routing, vol. 8, North-Holland, Amsterdam, 1995, pp. 35–139.

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

[13] J. Homberger, H. Gehring, A two-phase hybrid metaheuristic for the vehicle routing problem with time windows, European J. Oper. Res. 162 (2005) 220–238.

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

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

[16] S. Lin, Computer solutions of the traveling salesman problem, Bell System Tech. J. 44 (1965) 2245–2269.

[17] H.R. Lourenço, O.C. Martin, T. Stützle, Iterated local search, in: F. Glover, G.A. Kochenberger (Eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, Boston, 2003, pp. 321–353.

[18] 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.

[19] J.Y. Potvin, T. Kervahut, B.L. Garcia, J.M. Rousseau, The vehicle routing problem with time windows, Part 1: tabu search, INFORMS J. Comput. 8 (1996) 158–164.

[20] S. Reiter, G. Sherman, Discrete optimizing, J. SIAM 13 (1965) 864–889.

[21] L.-M. Rousseau, M. Gendreau, G. Pesant, Using constraint-based operators to solve the vehicle routing problem with time windows, J. Heuristics 8 (2002) 43–58.

[22] M.W.P. Savelsbergh, The vehicle routing problem with time windows: minimizing route duration, ORSA J. Comput. 4 (1992) 146–154.

[23] M.M. Solomon, The vehicle routing and scheduling problems with time window constraints, Oper. Res. 35 (1987) 254–265.

[24] M.M. Solomon, J. Desrosiers, Time window constrained routing and scheduling problems, Transportation Sci. 22 (1988) 1–13.

[25] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, J.Y. Potvin, A tabu search heuristic for the vehicle routing problem with soft time windows, Transportation Sci. 31 (1997) 170–186.