Effective Local Search Algorithms for Routing and Scheduling Problems with General Time Window Constraints

T. Ibaraki,[†] S. Imahori,[†] M. Kubo,[‡] T. Masuda,[♭] T. Uno[♮] and M. Yagiura[†*]

[†]Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan, Phone/Fax: +81-75-753-5494, E-mail: {ibaraki,yagiura}@i.kyoto-u.ac.jp, imahori@amp.i.kyoto-u.ac.jp

[‡]Logistics and Information Engineering, Tokyo University of Mercantile Marine, Tokyo 135-8533, Japan, E-mail: kubo@ipc.tosho-u.ac.jp

[♭]Communication & High Technology, Accenture, Nihon Seimei Akasaka Daini Bldg., 7-1-16 Akasaka Minato-ku, Tokyo 107-8672, Japan, E-mail: tomoyasu.masuda@accenture.com

[♮]National Institute of Informatics, Algorithm Foundation Research, National Center of Sciences, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan, E-mail: uno@nii.ac.jp

[*]corresponding author

Abstract: We propose local search algorithms for the vehicle routing problem with soft time window constraints. The time window constraint for each customer is treated as a penalty function, which is very general in the sense that it can be non-convex and discontinuous as long as it is piecewise linear. In our algorithm, we use local search to assign customers to vehicles and to find orders of customers for vehicles to visit. It employs an advanced neighborhood, called the cyclic exchange neighborhood, in addition to standard neighborhoods for the vehicle routing problem. After fixing the order of customers for a vehicle to visit, we must determine the optimal start times of processing at customers so that the total penalty is minimized. We show that this problem can be efficiently solved by using dynamic programming, which is then incorporated in our algorithm. We then report computational results for various benchmark instances of the vehicle routing problem. The generality of time window constraints allows us to handle a wide variety of scheduling problems. As such an example, we mention in this paper an application to a production scheduling problem with inventory cost, and report computational results for real world instances.

Keywords: adaptive multi-start local search, dynamic programming, general time window constraints, local search, metaheuristics, vehicle routing problem, very large scale neighborhood.

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 (Desrochers et al. 1988, Desrosiers et al. 1995, Solomon and Desrosiers 1988). 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 (Koskosidis, Powell and Solomon 1992, Potvin et al. 1996, Savelsbergh 1992, Taillard et al. 1997). 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, and it has been a subject of intensive research focused mainly on heuristic and metaheuristic approaches (Bräysy and Gendreau 2003a, 2003b). Among recent approaches are a two-stage hybrid local search by Bent and Van Hentenryck (2001), a hybrid genetic algorithm by Berger, Barkaoui and Bräysy (2003), a reactive variable neighborhood search by Bräysy (2003), a two-phase hybrid metaheuristic algorithm by Homberger and Gehring (2003), a variable neighborhood decent with constraint-based operators by Rousseau, Gendreau and Pesant (2002) and so on. See an extensive survey by Bräysy and Gendreau (2003a, 2003b) for heuristic and metaheuristic approaches. To the best of our knowledge, only a convex time window constraint is allowed for each customer in most of the previous work of VRPTW (Potvin et al. 1996, Solomon 1987, Taillard et al. 1997). The case with multiple time windows (De Jong, Kant and Van Vliet 1996) and other variants of VRPTW (Solomon and Desrosiers 1988) are also considered to be important.

A constraint is called *hard* if it must be satisfied and is called *soft* if it can be violated. The amount of 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 (Garey and Johnson 1979). Therefore, searching within the feasible region of VRPHTW may be inefficient, especially when the constraints are tight. Moreover, in many real-world situations, constraints of time window and capacity can be violated to some extent. Considering these, we treat these two constraints as soft in this paper.

The main contribution of this paper is to propose an efficient algorithm to deal with general time window constraints, which is then incorporated in metaheuristic algorithms to develop general problem solvers. The time window constraints we consider in this paper are general in the sense that one or more time slots can be assigned to each customer. That is, the corresponding penalty function can be non-convex and discontinuous as long as it is piecewise linear. We call the resulting problem as the *vehicle routing problem with general time windows* (VRPGTW). In this case, after fixing the order of customers for a vehicle to visit, we must determine the optimal start times of services at all customers so that the total time penalty of the vehicle is minimized. We show that this problem can be efficiently solved by using dynamic programming.

Let n_k be the number of customers assigned to vehicle k, and δ_k be the total number of linear pieces of the penalty functions for those customers. The information of each linear piece is usually explicitly given, and in this case, δ_k is considered as the input size of the penalty functions of the customers. Moreover, $\delta_k = O(n_k)$ holds in many cases, since the number of linear pieces of the penalty function for each customer is usually $O(1)$. For example, if the penalty for a customer is the weighted sum of earliness and tardiness, then the number of linear pieces of the penalty function is at most 3. The time complexity of our dynamic programming is $O(n_k\delta_k)$ if the problem for vehicle k is solved from scratch. We also show that the optimal time penalty of each solution in the neighborhood of the current solution can be evaluated in $O(\sum_{k\in M'}\delta_k)$ time from the information of the current solution, where M' is the set of indices of vehicles which the neighborhood operation involves.

Special cases of convex penalty functions were considered in the literature of VRPSTW and scheduling problems, e.g., Davis and Kanet (1993), Garey, Tarjan and Wilfong (1988), Koskosidis, Powell and Solomon (1992), Taillard et al. (1997), Tamaki, Komori and Abe (1999), and Tamaki, Sugimoto and Araki (1999). In Taillard et al. (1997), the time penalty for each customer is $+\infty$ for earliness and linear for tardiness, and an $O(1)$ time algorithm to approximately compute the optimal time penalty of a solution in the neighborhood was proposed. In Davis and Kanet (1993), Koskosidis, Powell and Solomon (1992), Tamaki, Komori and Abe (1999), Tamaki, Sugimoto and Araki (1999), the time penalty is linear for both of earliness and tardiness, and an $O(n_k^2)$ time algorithm for a given route of vehicle k was proposed in Davis and Kanet (1993) and Tamaki, Sugimoto and Araki (1999). If the penalty function for each customer is the absolute deviation from a specified time, this problem becomes the *isotonic median regression* problem, which has been extensively studied. To our knowledge, the best time complexity for this problem (for a vehicle k) is $O(n_k \log n_k)$ (Ahuja and Orlin 2001, Garey, Tarjan and Wilfong 1988, Hochbaum and Queyranne 2003). Very fast algorithms for general convex functions are also known (Ahuja and Orlin 2001, Hochbaum and Queyranne 2003).

The essential part of VRPGTW, i.e., assigning customers to vehicles and determining the visiting order of each vehicle, is solved by local search (LS) algorithms. In the literature, three types of neighborhoods, called the cross exchange, 2-opt* and Or-opt neighborhoods, have been widely used (Or 1976, Potvin et al. 1996, Reiter and Sherman 1965, Taillard et al. 1997). We refer to these neighborhoods as *standard neighborhoods*. In our local search, in addition to these standard neighborhoods, we use an advanced neighborhood called the cyclic exchange neighborhood (Ahuja et al. 2002, Ahuja, Orlin and Sharma 2000). This is defined to be the set of solutions obtainable by cyclically exchanging two or more paths of length at most L^{cyclic} (a parameter), which stems from the cyclic transfer algorithm of Thompson and Psaraftis (1993) and the ejection chains of Glover (1996). As the size of this neighborhood grows exponentially with the input size, we search an improving solution by using the *improvement graph*, whose concept is proposed, e.g., in Ahuja et al. (2002) and Ahuja, Orlin and Sharma (2000), and is applicable to wide range of problems. We also propose time-oriented neighbor-lists to make the search in the cross exchange and 2-opt* neighborhoods more efficient.

Among many possible metaheuristics based on local search, we use the multi-start local search (MLS) (Lin 1965, Lin and Kernighan 1973, Reiter and Sherman 1965), the iterated local search (ILS) (Johnson 1990, Martin, Otto and Felten 1991, Martin, Otto and Felten 1992) and the adaptive multi-start local search (AMLS) (Boese, Kahng and Muddu 1994, Taillard et al. 1997). MLS repeatedly applies LS to a number of initial solutions which are generated randomly or by greedy methods, and the best solution obtained in the entire search is output. ILS is a variant of MLS, in which the initial solutions for LS are generated by perturbing good solutions obtained in the previous search. AMLS is also a variant of MLS, which keeps a set P of good solutions found in the previous search, and generates initial solutions by combining the parts of the solutions in P.

We conduct computational experiments on three different types of instances: (1) Solomon's VRPHTW benchmark instances (Solomon 1987), (2) artificially generated instances of the parallel machine scheduling problem with various types of time windows, and (3) real world instances of a production scheduling problem with inventory cost. The computational results exhibit a good prospect of ILS and AMLS. For Solomon's benchmark instances, the solution quality of our algorithms is competitive with those of recent algorithms developed for VRPHTW, though our algorithms require slightly more computational time. It should be pointed out that Solomon's instances have only convex penalty functions, while our algorithms can handle more general penalty functions. For the parallel machine scheduling problem, we generate instances with various types of time penalty functions (including non-convex ones), whose optimal solutions are known. Our algorithms find optimal or near optimal solutions with high probability. It is also observed that the cyclic exchange neighborhood is quite effective in dealing with non-convex time penalty functions. For the production scheduling problem with inventory cost, we use real world data provided by a company, and observe that the proposed algorithms find better schedules than those currently used in the company.

1 Problem

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

- a quantity q_i (≥ 0) of goods to be delivered,
- a penalty function $p_i(t)$ (≥ 0) of the start time t of the service,
- a service time u_i (≥ 0),
- a capacity Q_k (≥ 0),
- a distance d_{ij} (≥ 0), and
- a travel time t_{ij} (≥ 0).

We assume $q_0 = 0$ and $u_0 = 0$ for the depot 0. Each vehicle can start from the depot after time 0. Each penalty function $p_i(t)$ is nonnegative, piecewise linear and satisfies $p_i(t) \leq \lim_{\varepsilon \to 0} \min\{p_i(t+\varepsilon), p_i(t-\varepsilon)\}$ at every discontinuous point t (see Fig. 1 for example). This (Figure 1) assumption is necessary to ensure the existence of an optimal solution. Note that $p_i(t)$ can be non-convex and discontinuous as long as it satisfies the above conditions. Distances d_{ij} and travel times t_{ij} are in general asymmetric; i.e., $d_{ij} \neq d_{ji}$ and $t_{ij} \neq t_{ji}$ may hold.

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

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

Note that each customer i is included in exactly one σ_k, and is visited by the vehicle exactly once. We denote by n_k the number of customers in σ_k for $k \in M$. 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). We also use a set of directed edges $\{(\sigma_k(h), \sigma_k(h+1)) \in E \mid h = 0, 1, \ldots, n_k,\ k \in M\}$ to represent a set of routes $\boldsymbol{\sigma}$. Moreover, let s_i be the start time of the service at customer i and s_k^{a} be the arrival time of vehicle k at the depot, and let

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

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

For convenience, we define 0-1 variables $y_{ik}(\boldsymbol{\sigma}) \in \{0, 1\}$ for $i \in V$ and $k \in M$ by

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

That is, $y_{ik}(\boldsymbol{\sigma}) = 1$ if and only if vehicle k visits customer i. Then the total distance $d_{\mathrm{sum}}(\boldsymbol{\sigma})$ traveled by all vehicles, the total penalty $p_{\mathrm{sum}}(\boldsymbol{s})$ for start times of services, and the total amount $q_{\mathrm{sum}}(\boldsymbol{\sigma})$ of capacity excess are expressed as

$$\begin{aligned} d_{\mathrm{sum}}(\boldsymbol{\sigma}) &= \sum_{k \in M} \sum_{h=0}^{n_k} d_{\sigma_k(h), \sigma_k(h+1)} \\ p_{\mathrm{sum}}(\boldsymbol{s}) &= \sum_{i \in V \setminus \{0\}} p_i(s_i) + \sum_{k \in M} p_0(s_k^{\mathrm{a}}) \end{aligned}$$

$$q_{\text{sum}}(\boldsymbol{\sigma}) \quad = \quad \sum_{k \in M} \max\left\{\sum_{i \in V} q_i y_{ik}(\boldsymbol{\sigma}) - Q_k,\ 0\right\}.$$

The VRPGTW is now formulated as follows:

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

Constraint (2) means that every customer $i \in V \setminus \{0\}$ must be served only once by exactly one vehicle. Constraints (3), (4) and (5) require that the start time s_i of the service at customer i must be after the arrival time at customer i. The time window and capacity constraints are treated as soft, and their violation is evaluated as the penalties $p_{\text{sum}}(\boldsymbol{s})$ and $q_{\text{sum}}(\boldsymbol{\sigma})$ in the objective function.

As for the objective function (1), the weighted sum $d_{\text{sum}}(\boldsymbol{\sigma}) + \alpha p_{\text{sum}}(\boldsymbol{s}) + \beta q_{\text{sum}}(\boldsymbol{\sigma})$ with constants α (≥ 0) and β (≥ 0) might seem more natural; however, such weights can be treated in the above formulation by regarding $\alpha p_i(t)$, βq_i and βQ_k $(i \in V,\ k \in M)$ as the given data, and hence the weights are omitted for simplicity.

We can also consider a penalty function $\tilde{p}_k(t)$ of the departure time t of vehicle k from the depot, though in the above formulation we assume that all vehicles have the same departure time 0. For this, we introduce m dummy customers $i = n+1, n+2, \ldots, n+m$ with penalty functions $p_i(t) = \tilde{p}_{i-n}(t)$ to represent the penalty of the departure time of vehicle k from the depot. Finally, the distances d_{ij} and the travel times t_{ij} are defined so that each vehicle k must visit customer $n+k$ first (i.e., very large cost is incurred if vehicle k visits other customers first).

The time window constraint of customer i is often regarded as hard and defined by an interval $[w_i^{\text{r}}, w_i^{\text{d}}]$ (instead of a penalty function $p_i(t)$), within which the service of i must be started. Such an instance can be treated as a special case of our formulation by setting penalty functions as either

$$p_i(t) = \alpha \cdot \max\left\{w_i^{\text{r}} - t,\ 0,\ t - w_i^{\text{d}}\right\} \tag{6}$$

or

$$p_i(t) = \begin{cases} 0, & t \in [w_i^{\text{r}}, w_i^{\text{d}}], \\ \alpha, & \text{otherwise}, \end{cases} \tag{7}$$

where α is a large positive value.

To avoid confusion, we note throughout the paper (unless otherwise stated) that a feasible solution denotes a solution that satisfies constraints (2)–(5); i.e., a feasible solution (of our formulation) does not necessarily satisfy the time window and capacity constraints. Cases with hard time window and/or capacity constraints can be treated by setting the penalties $p_{\text{sum}}(\boldsymbol{s})$ and $q_{\text{sum}}(\boldsymbol{\sigma})$ sufficiently large. See Section 3.2 for more discussion about this matter.

Remark: The number of vehicles m is a given constant in this paper, although it is sometimes treated as a decision variable in the literature. We adopt this assumption for the following two reasons. (1) In many applications, the number of vehicles m (or the number of machines in scheduling problems) is fixed. (2) Algorithms become simpler if m is treated as a constant, since route elimination operators are not 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.

2 Optimal start time of services

In this section, we consider the problem of determining the time to start services of customers in a given route σ_k so that the total time penalty is minimized. How to determine σ_k will be discussed in Section 3. Let $\delta^{(i)}$ be the number of pieces in the piecewise linear function $p_i(t)$, and let the total number of pieces in the penalty functions for all the customers in σ_k (including the depot) be $\delta_k = \sum_{h=0}^{n_k} \delta^{(\sigma_k(h))}$. Furthermore, let $\delta = \sum_{k \in M} \delta_k = \sum_{i \in V \setminus \{0\}} \delta^{(i)} + m\delta^{(0)}$ be the total number of pieces in the penalty functions of all customers (including the depot), where $\delta^{(0)}$ is multiplied by m.

We propose an $O(n_k \delta_k)$ time algorithm based on the dynamic programming (DP) to solve this problem. (It is pointed out by Hochbaum (2002b) that this time complexity can also be achieved by using a more general algorithm in Hochbaum (1997), Hochbaum (2002a) and Hochbaum and Naor (1994); however, our DP algorithm is simpler and is used to achieve a more efficient algorithm in Section 4.1.2.) If the penalty functions are convex, this problem can be formulated as a convex programming problem, which is efficiently solvable by using existing methods (Bertsekas 1995, Chvátal 1983) even without using efficient algorithms for this problem (Ahuja and Orlin 2001, Hochbaum and Queyranne 2003). Here, it is emphasized that the proposed DP algorithm is applicable even if $p_i(t)$ are non-convex and discontinuous.

2.1 The dynamic programming algorithm

We define $f_h^k(t)$ 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 before time t. Throughout this paper, we call this a *forward minimum penalty function*. For convenience, we also define

$$\begin{aligned} p_h^k(t) &= p_{\sigma_k(h)}(t), \\ \tau_h^k &= u_{\sigma_k(h)} + t_{\sigma_k(h),\sigma_k(h+1)}, \end{aligned}$$

where $p_h^k(t)$ is the penalty function for the hth customer of vehicle k, and τ_h^k is the sum of the service time at the hth customer and the travel time from the hth customer to the $(h+1)$st customer. Then $f_h^k(t)$ can be computed by

$$
\begin{aligned}
f_0^k(t) &= \begin{cases} +\infty, & t \in (-\infty, 0) \\ 0, & t \in [0, +\infty) \end{cases} \\
f_h^k(t) &= \min_{t' \le t} \left(f_{h-1}^k(t' - \tau_{h-1}^k) + p_h^k(t') \right), \qquad 1 \le h \le n_k + 1,\ -\infty < t < +\infty.
\end{aligned}
\tag{8}
$$

Then the minimum penalty value for the route σ_k can be obtained by

$$\min_t f_{n_k+1}^k(t). \tag{9}$$

The optimal start time $s_{\sigma_k(h)}$ of the service for each $h = 1, 2, \ldots, n_k$ and the time s_k^{a} of vehicle k to return to the depot can be computed by

$$
\begin{aligned}
s_k^{\mathrm{a}} &= \min \arg \min_t f_{n_k+1}^k(t) \\
s_{\sigma_k(h)} &= \min \arg \min_{t \le s_{\sigma_k(h\,+\,1)} - \tau_h^k} f_h^k(t), \quad 1 \le h \le n_k,
\end{aligned}
\tag{10}
$$

where $s_0 = s_k^{\mathrm{a}}$ is assumed for convenience. The first min in the right hand side of (10) signifies the leftmost t if $\min f_{n_k+1}^k(t)$ (or $\min f_h^k(t)$) is achieved by multiple t.

An example of the computation of recursion (8) is shown in Fig. 2. This figure represents the computation of $f_h^k(t)$ from $f_{h-1}^k(t)$ and $p_h^k(t)$, where $\tau_{h-1}^k = 2$, and $f_{h-1}^k(t)$ and $p_h^k(t)$ are respectively defined by (Figure 2)

$$
f_{h-1}^k(t) = \begin{cases} -t + 3, & t < 1 \\ 2, & 1 \le t < 4 \\ -t + 6, & 4 \le t < 5 \\ 1, & 5 < t \end{cases}
$$

and

$$
p_h^k(t) = \begin{cases} -0.5t + 3, & t < 4 \\ t - 3, & 4 \le t < 8 \\ -4t + 37, & 8 \le t < 9 \\ t - 8, & 9 < t. \end{cases}
$$

2.2 Implementation and time complexity of the algorithm

Let us consider the data structure for computing recursion (8). Since penalty function $p_h^k(t)$ is piecewise linear, $f_h^k(t)$ is also piecewise linear. Therefore we can keep the functions that appear in (8) in linked lists, whose components store the intervals and the associated linear functions (i.e., linear pieces) of the piecewise linear functions. In computing $f_h^k(t)$, the intervals of $f_{h-1}^k(t)$ are first shifted by τ_{h-1}^k to the right to obtain $f_{h-1}^k(t - \tau_{h-1}^k)$. Then, $f_{h-1}^k(t - \tau_{h-1}^k) + p_h^k(t)$ is computed by merging the intervals of $f_{h-1}^k(t-\tau_{h-1}^k)$ and $p_h^k(t)$ while adding the linear functions of the corresponding pieces, and by storing $f_{h-1}^k(t-\tau_{h-1}^k)+p_h^k(t)$ in a new linked list. Finally, $f_h^k(t)$ is obtained by taking the minimum of $f_{h-1}^k(t'-\tau_{h-1}^k)+p_h^k(t')$ over all $t' \leq t$, which can be achieved by scanning the new list from the left. Fig. 3 shows the linked lists for the functions in Fig. 2. (Figure 3)

The computation of $f_{h-1}^k(t-\tau_{h-1})+p_h^k(t)$ and $f_h^k(t)$ from $f_{h-1}^k(t)$ and $p_h^k(t)$ can be achieved in $O(\delta_k)$ time, since the total number of pieces in $f_{h-1}^k(t)$ and $p_h^k(t)$ is $O(\delta_k)$. The computation of total time penalty (9) and optimal start time (10) for each h can also be achieved in $O(\delta_k)$ time, since both computations only require to scan the list of $f_h^k(t)$. Therefore, determining the minimum time penalty for a given route and the optimal start times of services for all the customers in the route can be done in $O(n_k\delta_k)$ time. (For evaluating the optimal time penalty of a solution in the neighborhood, a more efficient algorithm will be explained in Section 4.1.2.)

3 Local search

In this section, we describe the framework of our local search (LS). The search space of LS is the set of all visiting orders $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ satisfying condition (2). Then a solution $\boldsymbol{\sigma}$ is evaluated by

$$cost(\boldsymbol{\sigma}) = d_{\text{sum}}(\boldsymbol{\sigma}) + p^*_{\text{sum}}(\boldsymbol{\sigma}) + q_{\text{sum}}(\boldsymbol{\sigma}), \tag{11}$$

where $p^*_{\text{sum}}(\boldsymbol{\sigma})$ is the minimum value of $p_{\text{sum}}(\boldsymbol{s})$ among those $\boldsymbol{s}$ satisfying conditions (3)–(5) for the given $\boldsymbol{\sigma}$. Such an $\boldsymbol{s}$ is computed by the dynamic programming of Section 2 (a more efficient method will be explained in Section 4.1.2). For convenience, let $d_{\text{sum}}(\sigma_k)$ be the total distance of route σ_k, $p^*_{\text{sum}}(\sigma_k)$ be the optimal time penalty of route σ_k, and $q_{\text{sum}}(\sigma_k)$ be the total amount of capacity excess of route σ_k, and define $cost(\sigma_k) = d_{\text{sum}}(\sigma_k) + p^*_{\text{sum}}(\sigma_k) + q_{\text{sum}}(\sigma_k)$. Then $cost(\boldsymbol{\sigma}) = \sum_{k \in M} cost(\sigma_k)$ holds.

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 (to be described later). The LS starts from an initial solution $\boldsymbol{\sigma}$ and repeats replacing $\boldsymbol{\sigma}$ with a better solution $\boldsymbol{\sigma}'$ (i.e., $cost(\boldsymbol{\sigma}') < cost(\boldsymbol{\sigma})$) in its neighborhood $N(\boldsymbol{\sigma})$ until no better solution is found in $N(\boldsymbol{\sigma})$. We will use more than one neighborhood in our LS.

In the subsequent sections, we will explain the details of LS. In Section 3.1, we explain the neighborhoods used in our algorithm. In Section 3.2, we explain how we define the incumbent solution. In Section 3.3, we describe the search order in the neighborhoods, and summarize the framework of LS.

3.1 Neighborhoods

In our algorithm, we use the standard neighborhoods (i.e., the cross exchange, 2-opt* and Or-opt neighborhoods) with slight modifications. We also use the cyclic exchange neighborhood, whose basic idea was proposed in Ahuja et al. (2002) and Ahuja, Orlin and Sharma (2000).

3.1.1 Standard neighborhoods

The cross exchange neighborhood was proposed in Taillard et al. (1997). A cross exchange operation removes two paths from two different routes (one from each), whose length (i.e., the number of customers in the path) is at most L^{cross} (a parameter), and exchanges them. Let $N^{\mathrm{cross}}(\boldsymbol{\sigma}, k, k')$ be the set of all solutions obtainable by cross exchange operations on two routes σ_k and $\sigma_{k'}$ in the current solution $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_m)$, and let $N^{\mathrm{cross}}(\boldsymbol{\sigma}) = \bigcup_{k<k'} N^{\mathrm{cross}}(\boldsymbol{\sigma}, k, k')$. The size of the cross exchange neighborhood is $O(n^2(L^{\mathrm{cross}})^2)$.

Fig. 4 illustrates a cross exchange operation. In this figure, squares represent the depot (which is duplicated at each end) and circles represent customers in the route. A thin line represents a route edge and a thick line represents a path (i.e., more than two customers may be included). First, two edges $(\sigma_k(h_1^k - 1), \sigma_k(h_1^k))$ and $(\sigma_k(h_2^k - 1), \sigma_k(h_2^k))$ are removed from route σ_k and two edges $(\sigma_{k'}(h_1^{k'} - 1), \sigma_{k'}(h_1^{k'}))$ and $(\sigma_{k'}(h_2^{k'} - 1), \sigma_{k'}(h_2^{k'}))$ are also removed from route $\sigma_{k'}$. Then, four new edges $(\sigma_k(h_1^k - 1), \sigma_{k'}(h_1^{k'}))$, $(\sigma_{k'}(h_2^{k'} - 1), \sigma_k(h_2^k))$, $(\sigma_{k'}(h_1^{k'} - 1), \sigma_k(h_1^k))$ and $(\sigma_k(h_2^k - 1), \sigma_{k'}(h_2^{k'}))$ are added to exchange two paths $\sigma_k(h_1^k) \to \sigma_k(h_2^k - 1)$ and $\sigma_{k'}(h_1^{k'}) \to \sigma_{k'}(h_2^{k'} - 1)$. (Figure 4)

The 2-opt* neighborhood was proposed in Potvin et al. (1996), which is a variant of the 2-opt neighborhood (Lin 1965) 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 (see Fig. 5). Let $N^{\mathrm{2opt}^*}(\boldsymbol{\sigma}, k, k')$ be the set of all solutions obtainable by 2-opt* operations on two routes σ_k and $\sigma_{k'}$ of the current solution $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_m)$, and let $N^{\mathrm{2opt}^*}(\boldsymbol{\sigma}) = \bigcup_{k<k'} N^{\mathrm{2opt}^*}(\boldsymbol{\sigma}, k, k')$. The size of the 2-opt* neighborhood is $O(n^2)$. (Figure 5)

Note that $N^{\mathrm{2opt}^*}(\boldsymbol{\sigma}) \subseteq N^{\mathrm{cross}}(\boldsymbol{\sigma})$ holds if $L^{\mathrm{cross}} = n$, since the cross exchange operation generates solutions in $N^{\mathrm{2opt}^*}(\boldsymbol{\sigma})$ when the last customers of the two paths to be exchanged ($\sigma_k(h_2^k - 1)$ and $\sigma_{k'}(h_2^{k'} - 1)$ in Fig. 4) are the last customers of the routes. However, parameter L^{cross} is usually set small to keep $|N^{\mathrm{cross}}(\boldsymbol{\sigma})|$ small and there are many solutions in $N^{\mathrm{2opt}^*}(\boldsymbol{\sigma}) \setminus N^{\mathrm{cross}}(\boldsymbol{\sigma})$. Hence searching in $N^{\mathrm{2opt}^*}(\boldsymbol{\sigma})$ separately from $N^{\mathrm{cross}}(\boldsymbol{\sigma})$ is meaningful.

The cross exchange and 2-opt* operations always change the assignment of customers to the routes. We therefore use the intra-route neighborhood to improve a solution within a route, which is a variant of Or-opt neighborhood used in TSP (Or 1976, Reiter and Sherman 1965).

An intra-route operation removes a path of length at most $L^{\rm intra}_{\rm path}$ (a parameter) and inserts it into another position of the same route, where the position is limited within distance $L^{\rm intra}_{\rm ins}$ (a parameter) from the original position. This is illustrated in Fig. 6. If the removed path is $\sigma_k(h_1) \to \sigma_k(h_1 + l - 1)$ $(1 \le l \le L^{\rm intra}_{\rm path})$, it is inserted either between $\sigma_k(h_1 - l' - 1)$ and $\sigma_k(h_1 - l')$, or between $\sigma_k(h_1 + l + l' - 1)$ and $\sigma_k(h_1 + l + l')$, for $1 \le l' \le L^{\rm intra}_{\rm ins}$. If the removed path is inserted between $\sigma_k(h_1 + l + l' - 1)$ and $\sigma_k(h_1 + l + l')$ (resp., between $\sigma_k(h_1 - l' - 1)$ and $\sigma_k(h_1 - l')$), it is called a forward (resp., backward) insertion. In Fig. 6, we show a forward insertion, where it is denoted $h_2 = h_1 + l + l' - 1$ for simplicity. For each insertion, we consider two cases: (1) with its visiting order preserved (denoted a normal insertion), and (2) with its visiting order reversed (denoted a reverse insertion). Note that, even if we only consider $l' \ge 1$, the operation of just inverting the order of the removed path with its position unchanged is an intra-route operation using a forward reverse insertion with $l' = 1$ (i.e., the case with $h_2 = h_1 + l$ in Fig. 6 (b)), where the path $\sigma_k(h_1) \to \sigma_k(h_1 + l)$ is regarded as the removed path. (Figure 6)

Let $N^{\rm intra}(\boldsymbol{\sigma}, k)$ be the set of all solutions obtainable by intra-route operations on route σ_k of the current solution $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_m)$, and let $N^{\rm intra}(\boldsymbol{\sigma}) = \bigcup_{k \in M} N^{\rm intra}(\boldsymbol{\sigma}, k)$. The size of the intra-route neighborhood is $O(n\ L^{\rm intra}_{\rm path}\ L^{\rm intra}_{\rm ins})$.

3.1.2 The cyclic exchange neighborhood

In this section, we define the cyclic exchange neighborhood. Let ψ_{il} denote the path in a solution $\boldsymbol{\sigma}$, whose initial customer is i and whose length is l. That is, ψ_{il} denotes the sequence of l consecutive customers $i = \sigma_k(h), \sigma_k(h+1), \ldots, \sigma_k(h + l - 1)$. Let $k[i]$ denote the vehicle that visits customer i. For a sequence of paths $\psi_{i_1 l_1}, \psi_{i_2 l_2}, \ldots, \psi_{i_r l_r}$ satisfying $k[i_j] \ne k[i_{j'}]$ for all $j \ne j'$, the cyclic exchange operation is defined as follows: for each $j = 1, 2, \ldots, r$, $\psi_{i_{j-1}, l_{j-1}}$ is removed from route $\sigma_{k[i_{j-1}]}$ and inserted into route $\sigma_{k[i_j]}$ between the two customers to which $\psi_{i_j l_j}$ was connected ($i_0 = i_r$ and $l_0 = l_r$ are assumed for convenience). Note that there are two directions in inserting a path: (1) with its visiting order unchanged (denoted normal insertion) and (2) with its visiting order reversed (denoted reverse insertion). As a result, there are 2^r possible cyclic exchange operations on a sequence of paths $\psi_{i_1 l_1}, \psi_{i_2 l_2}, \ldots, \psi_{i_r l_r}$. Fig. 7 represents an example of such cyclic exchange operations. In this figure, route $\sigma_{k[i_1]}$ is duplicated at the right end for simplicity. (Figure 7)

As a special case of the cyclic exchange, we also consider the acyclic exchange operation. Let ψ_{i0} $(i = 1, 2, \ldots, n)$ and ψ^k_{00} $(k = 1, 2, \ldots, m)$ denote empty paths (i.e., paths of length 0 starting from customer i and the depot, respectively) in a solution $\boldsymbol{\sigma}$. An empty path means that no customer is removed from the route. Let $\psi_{i'l'}$ be the path to be inserted into the route from which ψ_{i0} or ψ^k_{00} is removed. Then ψ_{i0} signifies that $\psi_{i'l'}$ is inserted between customer i and its predecessor, while ψ^k_{00} signifies that $\psi_{i'l'}$ is inserted between customer $\sigma_k(n_k)$ and the depot. Then the cyclic exchange operation on a sequence of paths $\psi_{i_1 l_1}, \psi_{i_2 l_2}, \ldots, \psi_{i_r l_r}$ is called the acyclic exchange, if an empty path is removed from route $k[i_r]$. Fig. 8 shows an acyclic exchange operation. (Figure 8)

The cyclic exchange neighborhood $N^{\text{cyclic}}(\boldsymbol{\sigma})$ is defined to be the set of solutions obtainable by applying cyclic exchange operations (including acyclic exchange operations) on every possible combination of paths ψ_{il} satisfying the following three conditions:

1. all paths belong to different vehicles,
2. the length of each path is at most L^{cyclic} (a parameter),
3. the number of participating paths is at most ν^{cyclic} (a parameter).

If $L^{\text{cyclic}} \geq L^{\text{cross}}$ and $\nu^{\text{cyclic}} \geq 2$, then $N^{\text{cross}}(\boldsymbol{\sigma}) \subseteq N^{\text{cyclic}}(\boldsymbol{\sigma})$ holds.

The size of $N^{\text{cyclic}}(\boldsymbol{\sigma})$ is usually very large and grows exponentially with ν^{cyclic}. Therefore, enumerating all solutions in $N^{\text{cyclic}}(\boldsymbol{\sigma})$ is computationally infeasible. However, the concept of the improvement graph (Ahuja et al. 2002, Ahuja, Orlin and Sharma 2000) can be utilized to implicitly search the neighborhood. We will describe the improvement graph and how we search an improved solution in Section 4.2.

3.2 The incumbent solution

The *incumbent solution* in our algorithm is defined as follows. In our formulation of VRPGTW, the capacity and time window constraints are treated as soft. That is, they are not included in the constraints (2)–(5) of VRPGTW. Therefore, a locally optimal solution output by LS may not satisfy the two constraints (i.e., it may not be feasible to VRPHTW even though it is always feasible to VRPGTW). This situation may happen even if we set α in (6) or (7) very large, or multiply the values of q_i and Q_k by a very large number, to make $p^*_{\text{sum}}(\boldsymbol{\sigma})$ and $q_{\text{sum}}(\boldsymbol{\sigma})$ relatively large. On the other hand, the solutions found during the search of LS often satisfy the two constraints even if $p^*_{\text{sum}}(\boldsymbol{\sigma})$ and $q_{\text{sum}}(\boldsymbol{\sigma})$ are not set very large. Some applications prefer such solutions that satisfy the two constraints (i.e., VRPHTW). Thus, in addition to the original function $cost(\boldsymbol{\sigma})$ of (11), we allow the algorithm to have another criterion, $besteval(\boldsymbol{\sigma})$, and to keep as the incumbent the feasible solution to VRPGTW (i.e., constraints (2)–(5) are satisfied), which has the smallest $besteval(\boldsymbol{\sigma})$ among those found during the search by then. For example, *besteval* may be $cost(\boldsymbol{\sigma})$ or

$$besteval(\boldsymbol{\sigma}) = \begin{cases} d_{\text{sum}}(\boldsymbol{\sigma}), & \text{if } p^*_{\text{sum}}(\boldsymbol{\sigma}) + q_{\text{sum}}(\boldsymbol{\sigma}) = 0 \\ cost(\boldsymbol{\sigma}) + \beta, & \text{otherwise,} \end{cases}$$

where β is an appropriate constant that satisfies $\beta > d_{\text{sum}}(\boldsymbol{\sigma})$ for all $\boldsymbol{\sigma}$. Here we emphasize that *besteval* does not affect the search process of our algorithm, i.e., the search is conducted entirely on the basis of *cost*.

3.3 The whole framework of the local search

The LS in our algorithm searches the neighborhoods in the order described as follows. We first search the three neighborhoods,

1. the intra-route neighborhood,
2. the 2-opt* neighborhood, and
3. the cross exchange neighborhood,

in this order. The search of each neighborhood is executed until no improvement is found in the neighborhood. Then, after the cross exchange neighborhood, it returns to the intra-route neighborhood. This procedure continues until no improvement is found in three consecutive neighborhoods. The cyclic exchange neighborhood is then searched after this procedure. If an improved solution is found in the cyclic exchange neighborhood, the procedure immediately returns to the intra-route neighborhood; otherwise the local search outputs the locally optimal solution and stops. Our local search, LS($\boldsymbol{\sigma}^0$), which starts from an initial solution $\boldsymbol{\sigma}^0$, is summarized as follows. (Note that we also keep the solution that minimizes $besteval(\boldsymbol{\sigma})$ of Section 3.2 among those solutions generated during the search process, though it is not explicitly described due to space limitation.)

Algorithm LS($\boldsymbol{\sigma}^0$)

Step 1 Let $\boldsymbol{\sigma} := \boldsymbol{\sigma}_0$.

Step 2 Execute the following neighborhood search cyclically until no improvement is achieved in three consecutive neighborhoods.

- **2a** (intra-route neighborhood) If there is a feasible solution $\boldsymbol{\sigma}' \in N^{\text{intra}}(\boldsymbol{\sigma})$ such that $cost(\boldsymbol{\sigma}') < cost(\boldsymbol{\sigma})$, let $\boldsymbol{\sigma} := \boldsymbol{\sigma}'$, and return to Step 2a. Otherwise go to Step 2b.
- **2b** (2-opt* neighborhood) If there is a feasible solution $\boldsymbol{\sigma}' \in N^{\text{2opt}^*}(\boldsymbol{\sigma})$ such that $cost(\boldsymbol{\sigma}') < cost(\boldsymbol{\sigma})$, let $\boldsymbol{\sigma} := \boldsymbol{\sigma}'$, and return to Step 2b. Otherwise go to Step 2c.
- **2c** (cross exchange neighborhood) If there is a feasible solution $\boldsymbol{\sigma}' \in N^{\text{cross}}(\boldsymbol{\sigma})$ such that $cost(\boldsymbol{\sigma}') < cost(\boldsymbol{\sigma})$, let $\boldsymbol{\sigma} := \boldsymbol{\sigma}'$, and return to Step 2c. Otherwise return to Step 2a.

Step 3 (cyclic exchange neighborhood) If there is a feasible solution $\boldsymbol{\sigma}' \in N^{\text{cyclic}}(\boldsymbol{\sigma})$ such that $cost(\boldsymbol{\sigma}') < cost(\boldsymbol{\sigma})$, let $\boldsymbol{\sigma} := \boldsymbol{\sigma}'$ and return to Step 2. Otherwise output the current solution $\boldsymbol{\sigma}$ and halt.

4 Efficient implementation of local search

In this section, we explain various ideas useful to search the neighborhood efficiently. In Section 4.1, we propose ideas to speed up the evaluation of the objective values of solutions in the standard neighborhoods. In Section 4.2, we explain the improvement graph and an algorithm to search in the cyclic exchange neighborhood. In Section 4.3, we propose time-oriented neighbor-lists to prune the search in the cross exchange and 2-opt* neighborhoods. Finally, in Section 4.4, we propose two more ideas to speed up the local search procedure.

4.1 Evaluation of solutions in the standard neighborhoods

Let $\Delta d_{\rm sum}$, $\Delta p_{\rm sum}$ and $\Delta q_{\rm sum}$ be the differences in the distance $d_{\rm sum}(\boldsymbol{\sigma})$, the time penalty $p^*_{\rm sum}(\boldsymbol{\sigma})$, and the amount $q_{\rm sum}(\boldsymbol{\sigma})$ of capacity excess, respectively, between the current solution and a solution in the neighborhood. Then let

$$\Delta cost = \Delta d_{\rm sum} + \Delta p_{\rm sum} + \Delta q_{\rm sum},$$

which is negative if the new solution is better than the current solution. For a neighborhood N, let *one-round time for* N be the time either (1) to find an improved solution in N and update the data stored to evaluate solutions, or (2) to conclude that there is no improved solution in N. We also define *one-round time for* N *with respect to* $d_{\rm sum}$ to be the time required to evaluate $\Delta d_{\rm sum}$ during the whole one-round time. One-round time for N with respect to $p_{\rm sum}$ and that with respect to $q_{\rm sum}$ are similarly defined. In the subsequent sections, we explain separately how we evaluate $\Delta d_{\rm sum}$, $\Delta p_{\rm sum}$ and $\Delta q_{\rm sum}$ during the search of cross exchange, 2-opt* and intra-route neighborhoods.

4.1.1 Evaluation of $\Delta d_{\rm sum}$

We can compute $\Delta d_{\rm sum}$ by the difference between the sum of distances of the inserted edges and that of the removed edges. For a solution in the cross exchange or 2-opt* neighborhood, $\Delta d_{\rm sum}$ can be computed in $O(1)$ time, since the number of removed edges and inserted edges is constant. Therefore, one-round times for $N^{\rm cross}$ and $N^{\rm 2opt^*}$ with respect to $d_{\rm sum}$ are $O(|N^{\rm cross}|)$ and $O(|N^{\rm 2opt^*}|)$, respectively.

For the intra-route neighborhood, the evaluation of $\Delta d_{\rm sum}$ requires $O(L^{\rm intra}_{\rm path})$ time if we evaluate a solution in $N^{\rm intra}(\boldsymbol{\sigma})$ from scratch, since we need to compute the sum of the distances of all edges in the removed path when it is inserted reversely. However, we shall explain below that this computational time can be reduced to $O(1)$ and, consequently, one-round time for $N^{\rm intra}$ with respect to $d_{\rm sum}$ can be reduced from $O(|N^{\rm intra}| \cdot L^{\rm intra}_{\rm path})$ to $O(|N^{\rm intra}|)$, if we evaluate the solutions in $N^{\rm intra}$ in a specified order and use the information from the previous search.

As explained in Section 3.1.1, intra-route operation is categorized into the following four groups by the types of insertion: (1) normal forward insertion, (2) reverse forward insertion, (3) normal backward insertion, and (4) reverse backward insertion. Since $\Delta d_{\rm sum}$ is easily computed in $O(1)$ time for normal insertions, and operations (2) and (4) are similar, we will only treat the case of (2). See Fig. 6 (b) for a help to understand the following discussion.

The computation of reverse forward insertions on route σ_k is done as shown in Fig. 9. The three nested loops in Fig. 9 generate all possible reverse forward insertions. The new routes σ_k^{new} tested in this loop are $\langle$the depot $\to \sigma_k(h_1-1), \sigma_k(h_1+l) \to \sigma_k(h_2), \sigma_k(h_1+l-1) \to \sigma_k(h_1), \sigma_k(h_2+1) \to$ the depot$\rangle$. To evaluate Δd_{sum} in $O(1)$ time for each σ_k^{new}, we store two values $dist^-$ and $dist^+$ in the above nested loops. For simplicity, let $d^k_{hh'}$ denote the distance from the hth customer to the h'th customer in route σ_k. If $l=1$, we initialize $dist^-$ and $dist^+$ by $dist^- := 0$ and $dist^+ := 0$; otherwise (i.e., $l>1$) we update them by (Figure 9)

$$\begin{aligned} dist^- &:= dist^- + d^k_{h_1+l-2,h_1+l-1}, \\ dist^+ &:= dist^+ + d^k_{h_1+l-1,h_1+l-2}. \end{aligned}$$

The above initialization and update of $dist^-$ and $dist^+$ can be executed in $O(1)$ time. Then, Δd_{sum} of σ_k^{new}can be computed in $O(1)$ time by

$$\begin{aligned} \Delta d_{\mathrm{sum}} = {} & (dist^+ + d^k_{h_1-1,h_1+l} + d^k_{h_2,h_1+l-1} + d^k_{h_1,h_2+1}) \\ & -(dist^- + d^k_{h_1-1,h_1} + d^k_{h_1+l-1,h_1+l} + d^k_{h_2,h_2+1}). \end{aligned}$$

Space complexity for this computation is $O(1)$.

4.1.2 Evaluation of Δp_{sum}

If Δp_{sum} is computed by obtaining p^*_{sum} of the new routes by (8) from scratch, it takes $O(\sum_{k\in M'} n_k\delta_k)$ time, where M' is the set of indices of the routes related to the neighborhood operation ($|M'|\le 2$ holds for standard neighborhoods). Instead of this, we propose an $O(\sum_{k\in M'}\delta_k) = O(\delta)$ time algorithm that computes Δp_{sum}. We also propose a simple idea to further reduce the computational time, though the worst case time complexity does not change.

Let us consider the computation of the minimum time penalty on a route σ_k. Define $b^k_h(t)$ to be the minimum sum of the penalty values 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. We call this the *backward minimum penalty function*. Then, $b^k_h(t)$ can be computed as follows in a symmetric manner to the computation of $f^k_h(t)$,

$$\begin{aligned} b^k_{n_k+1}(t) &= \min_{t'\ge t} p_0(t') \\ b^k_h(t) &= \min_{t'\ge t}\left(b^k_{h+1}(t'+\tau^k_h) + p^k_h(t')\right), \quad 1\le h\le n_k. \end{aligned} \tag{12}$$

Then,

$$p^*_{\mathrm{sum}}(\sigma_k) = \min_t \left(f^k_h(t) + b^k_{h+1}(t+\tau^k_h)\right) \tag{13}$$

holds for any h ($1\le h\le n_k$). That is, if $f^k_h(t)$ and $b^k_{h+1}(t)$ are known for some h, the minimum penalty $p^*_{\mathrm{sum}}(\sigma_k)$ can be computed by (13). This is possible in $O(\delta_k)$ time, because $f^k_h(t)$ and $b^k_{h+1}(t)$ consist of $O(\delta_k)$ linear pieces as explained in Section 2.2 (for the case of $f^k_h(t)$).

To utilize this idea, we keep $f_h^k(t)$ and $b_h^k(t)$ in memory for all $h = 1, 2, \ldots, n_k$ and $k \in M$. In the case of neighborhood N^{2opt^*}, the computation of (13) can be directly carried out by using the forward and backward minimum penalty functions stored in memory. In case of N^{cross} and N^{intra}, however, the solutions must be treated in a specified order. We will describe the algorithm only for N^{intra}, since the case of N^{cross} is simpler.

As in Section 4.1.1, we only consider the reverse forward insertions on a route σ_k as other cases are similar. Recall that solutions are searched in the order as used in Fig. 9. Let $f_{h_1,h_2,l}(t)$ be the forward minimum penalty function for the subroute $\langle$the depot $\to \sigma_k(h_1-1), \sigma_k(h_1+l) \to \sigma_k(h_2)\rangle$, and $b_{h_1,h_2,l}(t)$ be the backward minimum penalty function for the subroute $\langle \sigma_k(h_1 + l - 1) \to \sigma_k(h_1), \sigma_k(h_2+1) \to$ the depot$\rangle$ (see Fig. 6 (b)). Then we evaluate the total time penalty of the new route $\sigma_k^{\mathrm{new}} = \langle$the depot $\to \sigma_k(h_1-1), \sigma_k(h_1+l) \to \sigma_k(h_2), \sigma_k(h_1+l-1) \to \sigma_k(h_1), \sigma_k(h_2+1) \to$ the depot$\rangle$ by

$$\min_t \left(f_{h_1,h_2,l}(t) + b_{h_1,h_2,l}(t + u_{\sigma_k(h_2)} + t_{\sigma_k(h_2),\sigma_k(h_1+l-1)}) \right).$$

As stated above, this computation is possible in $O(\delta_k)$ time, if $f_{h_1,h_2,l}(t)$ and $b_{h_1,h_2,l}(t)$ are known.

Now we describe how we compute functions $f_{h_1,h_2,l}(t)$ and $b_{h_1,h_2,l}(t)$. Function $f_{h_1,h_2,l}(t)$ is computed by (8) from $f_{h_1-1}^k(t)$ if $h_2 = h_1 + l$, and from $f_{h_1,h_2-1,l}(t)$ if $h_2 > h_1 + l$. Function $b_{h_1,h_2,l}(t)$ is computed by (12) from $b_{h_2+1}^k(t)$ if $l = 1$, and from $b_{h_1,h_2,l-1}(t)$ if $l > 1$. These functions $f_{h_1-1}^k(t)$ and $b_{h_2+1}^k(t)$ are stored in memory as already mentioned. As we generate solutions in N^{intra} in the order of Fig. 9, functions $f_{h_1,h_2-1,l}(t)$ and $b_{h_1,h_2,l-1}(t)$ are already available when they are needed for computing $f_{h_1,h_2,l}(t)$ and $b_{h_1,h_2,l}(t)$, respectively. From these arguments, we can conclude that $f_{h_1,h_2,l}(t)$ and $b_{h_1,h_2,l}(t)$ are computed in $O(\delta_k)$ time (i.e., the time to execute (8) and (12)).

Finally, we consider the space complexity of the above computation. We need $O(n_k\delta_k)$ space to store functions $f_h^k(t)$ and $b_h^k(t)$ for all customers in route σ_k. To compute $b_{h_1,h_2,l}(t)$, we only need to keep $b_{h_1,h_2,l-1}(t)$ in memory, which requires $O(\delta_k)$ space. To compute $f_{h_1,h_2,l}(t)$, we need to keep $f_{h_1,h_2-1,l}(t)$ for $l = 1, 2, \ldots, \min\{L_{\mathrm{path}}^{\mathrm{intra}}, n_k\}$, which requires $O(\min\{L_{\mathrm{path}}^{\mathrm{intra}}, n_k\} \cdot \delta_k)$ space. Therefore, the memory space required to store the necessary functions during the search of $N^{\mathrm{intra}}(\boldsymbol{\sigma}, k)$ is $O(n_k\delta_k)$.

In conclusion, Δp_{sum} of a solution in N^{cross}, N^{2opt^*} or N^{intra} can be evaluated in $O(\sum_{k\in M'} \delta_k)$ $= O(\delta)$ time provided that functions $f_h^k(t)$ and $b_h^k(t)$ for $h = 1, 2, \ldots, n_k$ and $k \in M$ are available, where $M' \subseteq M$ ($|M'| \leq 2$) is the set of indices of the relevant routes. The space complexity required for this computation is $O(\sum_{k\in M} n_k\delta_k) = O(n\delta)$. At the beginning of LS, we need to compute functions $f_h^k(t)$ and $b_h^k(t)$ for all $h = 1, 2, \ldots, n_k$ and $k \in M$, which is possible in $O(\sum_{k\in M} n_k\delta_k) = O(n\delta)$ time. We also need to recompute functions $f_h^k(t)$ and $b_h^k(t)$ for all $h = 1, 2, \ldots, n_k$ and $k \in M'$, whenever the current solution is updated (i.e., the current round is over). This computation takes $O(\sum_{k\in M'} n_k\delta_k) = O(n\delta)$ time. As $|N^{\rm cross}|$, $|N^{\rm 2opt^*}|$ and $|N^{\rm intra}|$ are larger than n, the one-round times for $N^{\rm cross}$, $N^{\rm 2opt^*}$ and $N^{\rm intra}$ with respect to $p_{\rm sum}$ are $O(|N^{\rm cross}|\delta)$, $O(|N^{\rm 2opt^*}|\delta)$ and $O(|N^{\rm intra}|\delta)$, respectively, and the space complexity is $O(n\delta)$ for such neighborhoods.

We can further reduce the number of pieces in forward and backward minimum penalty functions. Since $f_h^k(t)$ (resp., $b_h^k(t)$) is nonincreasing (resp., nondecreasing), there are usually many pieces with considerably large value in $f_h^k(t)$ (resp., in $b_h^k(t)$) for small (resp., large) t. Such pieces will not be used in evaluating improved solutions. Therefore, we set a value INF and delete those pieces whose values over their intervals are not less than INF. We then add one piece whose interval is the union of the deleted intervals and function value is always INF. Even after this modification, we can compute the exact optimal time penalty $p^*_{\rm sum}(\sigma_k)$ if $p^*_{\rm sum}(\sigma_k) < {\rm INF}$ holds; otherwise we can conclude that $p^*_{\rm sum}(\sigma_k) \geq {\rm INF}$ holds. In our algorithm, INF is set to a sufficiently large value at the beginning of the local search. Then, whenever the current solution is improved during search, we set INF $:= cost(\boldsymbol{\sigma})$ for the current solution $\boldsymbol{\sigma}$, which ensures that the algorithm never misses improved solutions. Though the worst case time complexity is not improved by this modification, actual computation usually becomes much faster.

4.1.3 Evaluation of $\Delta q_{\rm sum}$

Let us consider the total amount of goods $\sum_{h=1}^{n_k} q_{\sigma_k(h)}$ to be delivered in a route σ_k. We define

$$\begin{aligned}
&\gamma_0^k = 0\\
&\gamma_h^k = \textstyle\sum_{l=1}^{h} q_{\sigma_k(l)} = \gamma_{h-1}^k + q_{\sigma_k(h)}, \quad h = 1, 2, \ldots, n_k
\end{aligned}$$

for all $k \in M$, and store these values in memory during the search. Then, the sum $\sum_{l=h}^{h'} q_{\sigma_k(l)}$ of the amount of goods for the hth through the h'th customers can be computed in $O(1)$ time by $\gamma_{h'}^k - \gamma_{h-1}^k$. Hence $\Delta q_{\rm sum}$ is computed in $O(1)$ time for a solution in $N^{\rm cross}$ or $N^{\rm 2opt^*}$ ($\Delta q_{\rm sum}$ is always 0 for solutions in $N^{\rm intra}$).

At the beginning of LS, we need to initialize all γ_h^k, which is possible in $O(\sum_{k\in M} n_k) = O(n)$ time. Whenever the current solution is updated, we also need to recompute γ_h^k for all $h = 0, 1, \ldots, n_k$ and $k \in M'$, where $M' \subseteq M$ ($|M'| \leq 2$) is the set of indices of relevant routes. The time required for this is $O(\sum_{k\in M'} n_k) = O(n)$. As sizes $|N^{\rm cross}|$, $|N^{\rm 2opt^*}|$ and $|N^{\rm intra}|$ are larger than n, one-round times for $N^{\rm cross}$, $N^{\rm 2opt^*}$ and $N^{\rm intra}$ with respect to $q_{\rm sum}$ are $O(|N^{\rm cross}|)$, $O(|N^{\rm 2opt^*}|)$ and $O(|N^{\rm intra}|)$, respectively. The space complexity of the above computation is $O(n)$.

4.2 Search in the cyclic exchange neighborhood

In this section, we introduce the improvement graph (Ahuja et al. 2002, Ahuja, Orlin and Sharma 2000), which is used to find an improved solution in N^{cyclic}. Then we propose a heuristic on how to search the improvement graph.

4.2.1 Improvement graph

For simplicity, we first describe the improvement graph in which acyclic exchange operations are not considered, and then later describe how acyclic exchange operations are treated. The improvement graph $G(\boldsymbol{\sigma}) = (V(\boldsymbol{\sigma}), E(\boldsymbol{\sigma}))$ is defined for a feasible solution $\boldsymbol{\sigma}$. The set $V(\boldsymbol{\sigma})$ consists of nodes v_{il} $(i = 1, 2, \ldots, n,\ l = 1, 2, \ldots, L^{\text{cyclic}})$, where a node $v_{il} \in V(\boldsymbol{\sigma})$ represents path ψ_{il} (i.e., the path of length l from i in $\boldsymbol{\sigma}$). A directed arc $e^{\text{n}}_{ili'l'}$ (resp., $e^{\text{r}}_{ili'l'}$) $\in E(\boldsymbol{\sigma})$ from node v_{il} to $v_{i'l'}$ indicates that paths ψ_{il} and $\psi_{i'l'}$ are removed from routes $\sigma_{k[i]}$ and $\sigma_{k[i']}$, respectively, and ψ_{il} is inserted in place of $\psi_{i'l'}$ in $\sigma_{k[i']}$ with its visiting order unchanged (resp., reversed). Here $k[i]$ denotes the vehicle that visits customer i. That is, $e^{\text{n}}_{ili'l'}$ and $e^{\text{r}}_{ili'l'}$ represent normal and reverse insertions, respectively. Two arcs $e^{\text{n}}_{ili'l'}$ and $e^{\text{r}}_{ili'l'}$ are in $E(\boldsymbol{\sigma})$ if and only if $k[i] \neq k[i']$ holds. Let h_i be the index satisfying $\sigma_{k[i]}(h_i) = i$ for customer i, and define the following routes:

$$\sigma^{\text{n}}_{ili'l'} = \langle \text{the depot} \to \sigma_{k[i']}(h_{i'} - 1), \sigma_{k[i]}(h_i) \to \sigma_{k[i]}(h_i + l - 1), \sigma_{k[i']}(h_{i'} + l') \to \text{the depot} \rangle,$$
$$\sigma^{\text{r}}_{ili'l'} = \langle \text{the depot} \to \sigma_{k[i']}(h_{i'} - 1), \sigma_{k[i]}(h_i + l - 1) \to \sigma_{k[i]}(h_i), \sigma_{k[i']}(h_{i'} + l') \to \text{the depot} \rangle.$$

Then, the costs of arcs $e^{\text{n}}_{ili'l'}$ and $e^{\text{r}}_{ili'l'}$ are defined as $\mathit{cost}(\sigma^{\text{n}}_{ili'l'}) - \mathit{cost}(\sigma_{k[i']})$ and $\mathit{cost}(\sigma^{\text{r}}_{ili'l'}) - \mathit{cost}(\sigma_{k[i']})$ (i.e., the cost increases on route $\sigma_{k[i']}$ by the insertions), respectively.

We call a directed cycle C in the improvement graph $G(\boldsymbol{\sigma})$ *subset-disjoint* if $k[i] \neq k[i']$ holds for any pair of nodes v_{il} and $v_{i'l'}$ in C (i.e., all the participating paths belong to different routes). We then call a subset-disjoint cycle a *valid cycle* if the sum of their arc costs is negative. As easily understood, there is one-to-one correspondence between solutions of cyclic exchange operations in $N^{\text{cyclic}}(\boldsymbol{\sigma})$ and subset-disjoint cycles in $G(\boldsymbol{\sigma})$, and the cost of a cycle represents the cost increase of the corresponding solution from that of the current solution $\boldsymbol{\sigma}$. Hence an improved solution in $N^{\text{cyclic}}(\boldsymbol{\sigma})$ can be found by a valid cycle in $G(\boldsymbol{\sigma})$.

We need only one of $e^{\text{n}}_{ili'l'}$ and $e^{\text{r}}_{ili'l'}$ with lower cost to find a valid cycle. Therefore, instead of having two arcs $e^{\text{n}}_{ili'l'}$ and $e^{\text{r}}_{ili'l'}$, only one arc $e_{ili'l'} = (v_{il}, v_{i'l'})$, whose cost is

$$c_{ili'l'} = \min \left\{ \mathit{cost}(\sigma^{\text{n}}_{ili'l'}) - \mathit{cost}(\sigma_{k[i']}), \mathit{cost}(\sigma^{\text{r}}_{ili'l'}) - \mathit{cost}(\sigma_{k[i']}) \right\},$$

is included in $E(\boldsymbol{\sigma})$ for each ordered pair of v_{il} and $v_{i'l'}$.

We now describe how to deal with acyclic exchange operations in the improvement graph. We create nodes v_{i0} for $i = 1, 2, \ldots, n$ and v^k_{00} for $k = 1, 2, \ldots, m$, where v_{i0} and v^k_{00} correspond to empty paths ψ_{i0} and ψ^k_{00}, respectively. We also create a node $\hat{v}$. These nodes are connected by arcs $(v_{i0}, \hat{v})$, $(v^k_{00}, \hat{v})$, $(\hat{v}, v_{i'l'})$, $(v_{i'l'}, v_{i0})$ and $(v_{i'l'}, v^k_{00})$ for $i, i' = 1, 2, \ldots, n$, $l' = 1, 2, \ldots, L^{\text{cyclic}}$ and $k = 1, 2, \ldots, m$. Let us define three routes $\sigma_{i'l'}$, $\sigma^{\text{n}}_{i'l'i0}$ and $\sigma^{\text{r}}_{i'l'i0}$ by

$$\begin{aligned}
\sigma_{i'l'} &= \langle \text{the depot} \rightarrow \sigma_{k[i']}(h_{i'} - 1), \sigma_{k[i']}(h_{i'} + l') \rightarrow \text{the depot} \rangle, \\
\sigma^{\text{n}}_{i'l'i0} &= \langle \text{the depot} \rightarrow \sigma_{k[i]}(h_i - 1), \sigma_{k[i']}(h_{i'}) \rightarrow \sigma_{k[i']}(h_{i'} + l' - 1), \sigma_{k[i]}(h_i) \rightarrow \text{the depot} \rangle, \\
\sigma^{\text{r}}_{i'l'i0} &= \langle \text{the depot} \rightarrow \sigma_{k[i]}(h_i - 1), \sigma_{k[i']}(h_{i'} + l' - 1) \rightarrow \sigma_{k[i']}(h_{i'}), \sigma_{k[i]}(h_i) \rightarrow \text{the depot} \rangle.
\end{aligned}$$

Then the cost of arc $(v_{i0}, \hat{v})$ is defined as 0, that of arc $(\hat{v}, v_{i'l'})$ is defined as $cost(\sigma_{i'l'}) - cost(\sigma_{k[i']})$, and that of arc $(v_{i'l'}, v_{i0})$ is defined as

$$\min \left\{ cost(\sigma^{\text{n}}_{i'l'i0}) - cost(\sigma_{k[i]}), cost(\sigma^{\text{r}}_{i'l'i0}) - cost(\sigma_{k[i]}) \right\}.$$

The costs of the arcs $(v^k_{00}, \hat{v})$ and $(v_{i'l'}, v^k_{00})$ are similarly defined. Now a cycle C in the modified improvement graph represents acyclic exchange operations if and only if C contains $\hat{v}$. Thus the graph contains a valid cycle if and only if an improved solution (either cyclic or acyclic exchange operations) exists in $N^{\text{cyclic}}(\boldsymbol{\sigma})$.

Let us now consider the size of the improvement graph and time complexity to construct it. Since there are $O(L^{\text{cyclic}} n)$ paths to be considered for the cyclic exchange operation, $|V(\boldsymbol{\sigma})| = O(L^{\text{cyclic}} n)$ and $|E(\boldsymbol{\sigma})| = O((L^{\text{cyclic}} n)^2)$ hold. From the discussion in Section 4.1, it is not difficult to see that the cost of an arc in $E(\boldsymbol{\sigma})$ can be computed in $O(\delta_{k[i]} + \delta_{k[i']}) = O(\delta)$ time, provided that $f^k_h(t)$, $b^k_h(t)$ and γ^k_h are known for $h = 1, 2, \ldots, n_k$ and $k \in M$, and the order of computing the arc costs is carefully designed. As all $f^k_h(t)$ and $b^k_h(t)$ (resp., γ^k_h) are computed in $O(n\delta)$ (resp., $O(n)$) time, the construction of $G(\boldsymbol{\sigma})$ is possible in $O((L^{\text{cyclic}} n)^2 \delta)$ time. The space complexity to compute arc costs is $O(n\delta)$ (needed to store functions $f^k_h(t)$ and $b^k_h(t)$), and the space to store graph $G(\boldsymbol{\sigma})$ is $O((L^{\text{cyclic}} n)^2)$.

4.2.2 A heuristic algorithm to find a valid cycle

As the problem of finding a valid cycle in a general improvement graph of this type is known to be NP-hard (Thompson and Orlin 1989), we develop a heuristic algorithm, which is not guaranteed to find a valid cycle even if it exists. Our algorithm is based on the labeling algorithm for the shortest path problem.

We first briefly summarize the labeling algorithm in $G(\boldsymbol{\sigma})$. Let c_{uv} denote the cost of an arc $(u, v) \in E(\boldsymbol{\sigma})$. The cost of a path in $G(\boldsymbol{\sigma})$ is the sum of c_{uv} of arcs (u, v) in the path, and the length of a path is the number of vertices in the path. The shortest path problem asks to find a path of minimum cost from a specified node $v_s \in V(\boldsymbol{\sigma})$ to every other node in $V(\boldsymbol{\sigma})$. The algorithm computes labels $label(v, l)$ for all $v \in V(\boldsymbol{\sigma})$, from $l = 1$ to $|V(\boldsymbol{\sigma})|$, where $label(v, l)$ stores the minimum cost among all directed paths from v_s to v of length at most l. If

$label(v,l)=\infty$, no directed path of length at most l from v_s to v is found yet. We call the phase of computing $label(v,l)$ for all $v \in V(\boldsymbol{\sigma})$ the lth phase. The algorithm also maintains $prev(v)$ for all $v \in V(\boldsymbol{\sigma})$, which represents the previous node of v in the directed path to v of cost $label(v,l)$. The algorithm starts by setting $label(v_s,1) := 0$ and $label(v,1) := \infty$ for all $v \in V(\boldsymbol{\sigma}) \setminus \{v_s\}$. At the beginning of the lth phase with $l \geq 2$, labels are set $label(v,l) := label(v,l-1)$ for all $v \in V(\boldsymbol{\sigma})$. Then, for every arc (u,v), if $label(u,l-1)+c_{uv} < label(v,l)$ holds, $label(v,l)$ is updated to $label(u,l-1)+c_{uv}$ and $prev(v)$ is updated to u. This step is called the label update step. The lth phase ends if all arcs are checked. The algorithm stops when $label(v,l-1) = label(v,l)$ holds for all $v \in V(\boldsymbol{\sigma})$ at the end of the lth phase, or it reaches the end of the $|V(\boldsymbol{\sigma})|$th phase.

The algorithm is usually implemented on a node set $\hat{V}$, called the active nodes, which is set to $\{v_s\}$ in the second phase, and is the set of those v satisfying $label(v,l-1) < label(v,l-2)$ in the lth phase with $l \geq 3$. Let E_v be the set of arcs whose tail vertex is v. In the lth phase ($l \geq 2$), only the arcs in $\hat{E} = \bigcup_{v \in \hat{V}} E_v$ are scanned for the label update step, since $label(u,l-1)+c_{uv} \geq label(v,l)$ holds for all arcs (u,v) not in $\hat{E}$.

We make some modifications on the above labeling algorithm. The set of active nodes is further reduced as follows. We only consider those arcs (u,v) with $label(u,l-1) < 0$ for the label update steps in the lth phase. That is, we delete those nodes u with $label(u,l-1) \geq 0$ from $\hat{V}$. Then the set of the reduced active nodes is denoted $\tilde{V}$.

We repeat the algorithm by regarding every node as the source. Recall that we only consider valid cycles of length at most ν^{cyclic} (Section 3.1.2). This parameter is motivated by the fact that longer cycles are less likely to be subset-disjoint. As a cycle of length more than m will not be subset-disjoint, we assume that $\nu^{\text{cyclic}} \leq m$ holds. Our algorithm finishes the search with the current source node and exits to a further search with another source node, if either of the following two conditions holds: (1) the algorithm reaches the $(\nu^{\text{cyclic}}+1)$st phase, (2) the set of active nodes $\tilde{V}$ becomes empty. Finally, the entire algorithm halts either (1) when a valid cycle is identified, or (2) when the search is completed with all source nodes in $V(\boldsymbol{\sigma})$.

Further modification is that the label update step is executed only if the resulting path is subset-disjoint. In order to check whether a path considered in each label update step is subset-disjoint or not in $O(1)$ time, we maintain $set(v,k)$ for all $v \in V(\boldsymbol{\sigma})$ and $k \in M$. Let $P[v]$ denote the path from v_s to v obtained by tracing $prev(v)$ from node v back to v_s, and let k_v be the index of the route that contains the path (of customers) corresponding to node $v \in V(\boldsymbol{\sigma})$. The value of $set(v,k)$ is 1, if $P[v]$ contains a vertex u satisfying $k_u = k$, and 0 otherwise. Then the label update step for an arc $(u,v) \in \tilde{E}$ is executed only if $set(u,k_v) = 0$ holds (i.e., the new path $P[v]$ is assured to be subset-disjoint after the label update step). At the beginning of the lth phase with $l \geq 3$, $set(v,k)$ are updated for those $v \in \tilde{V}$ by first resetting $set(v,k) := 0$ for all k and then setting $set(v,k_u) := 1$ for all u in $P[v]$.

The final modification is that the algorithm checks if $label(v, l) + c_{vv_s} < 0$ holds for some $v \in \tilde{V}$, at the end of the lth phase. If such a v is found, the resulting cycle $(P[v], v_s)$ is a valid cycle.

The modification of active nodes from $\hat{V}$ to $\tilde{V}$ is motivated by the following simple well-known lemma (see, e.g., Lin and Kernighan 1973).

Lemma 4.1 If $\sum_{i=1}^{l} c_i < 0$ holds, then there exists a j such that $\sum_{i=j}^{k} c_i < 0$ holds for all $k = j, j+1, \ldots, l$, and $\sum_{i=1}^{k} c_i + \sum_{i=j}^{l} c_i < 0$ holds for all $k = 1, 2, \ldots, j-1$.

Proof: Let k^* be the largest index among those k that maximizes $\sum_{i=1}^{k} c_i$. Then $\sum_{i=1}^{k} c_i + \sum_{i=k^*+1}^{l} c_i \leq \sum_{i=1}^{k^*} c_i + \sum_{i=k^*+1}^{l} c_i < 0$ holds if $1 \leq k \leq k^*$, and $\sum_{i=k^*+1}^{k} c_i = \sum_{i=1}^{k} c_i - \sum_{i=1}^{k^*} c_i < 0$ holds if $k^* + 1 \leq k \leq l$. Hence $j = k^* + 1$ satisfies the conditions in the lemma. □

This lemma indicates that, in a valid cycle C, there exists a vertex $v \in C$ such that the costs of all the directed paths in C starting from v are negative. Thus, although only maintaining labels with negative cost might miss some valid cycles in the search with the current source node, the algorithm will eventually find such valid cycles starting from other source node.

The algorithm, called FVC, to find a valid cycle in a given directed graph $G(\boldsymbol{\sigma})$ is summarized as follows. In the algorithm, $\hat{V}$ is the sets of vertices whose labels have been improved in the label update step of the current phase, $\tilde{V}$ is the subset of $\hat{V}$ whose labels are negative, and $\tilde{E}$ is $\bigcup_{v \in \tilde{V}} E_v$.

Algorithm FVC$(G(\boldsymbol{\sigma}) = (V(\boldsymbol{\sigma}), E(\boldsymbol{\sigma})))$

Step 1 Let $U := V(\boldsymbol{\sigma})$.

Step 2 Choose a node $v_s \in U$ and let $U := U \setminus \{v_s\}$, $label(u, 1) := \infty$ for all $u \in V(\boldsymbol{\sigma}) \setminus \{v_s\}$, $label(v_s, 1) := 0$, $\tilde{V} := \{v_s\}$, $\hat{V} := \emptyset$, $\tilde{E} := E_{v_s}$, $set(v, k) := 0$ for all $v \in V(\boldsymbol{\sigma}) \setminus \{v_s\}$ and $k \in M$, $set(v_s, k_{v_s}) := 1$ and $l := 2$.

Step 3 If $l > \nu^{\text{cyclic}}$, go to Step 8. Otherwise, let $label(v, l) := label(v, l-1)$ for all $v \in V(\boldsymbol{\sigma})$.

Step 4 Choose an arc $e_{uv} = (u, v) \in \tilde{E}$ and let $\tilde{E} := \tilde{E} \setminus \{e_{uv}\}$. If both $set(u, k_v) = 0$ and $label(u, l-1) + c_{uv} < label(v, l)$ hold, let $label(v, l) := label(u, l-1) + c_{uv}$, $prev(v) := u$ and $\hat{V} := \hat{V} \cup \{v\}$.

Step 5 If $\tilde{E} \neq \emptyset$, return to Step 4.

Step 6 Let $set(v, k) := 0$ for all $v \in \tilde{V}$ and $k \in M$. Then let $\tilde{V} := \emptyset$.

Step 7 Choose a node $v \in \hat{V}$ and let $\hat{V} := \hat{V} \setminus \{v\}$. If $label(v, l) \geq 0$ holds, proceed to Step 8. If $label(v, l) + c_{vv_s} < 0$ holds, output the corresponding valid cycle and halt. Otherwise, let $\tilde{V} := \tilde{V} \cup \{v\}$, $\tilde{E} := \tilde{E} \cup E_v$ and $set(v, k_u) := 1$ for all u in $P[v]$.

Step 8 If $\hat{V} \neq \emptyset$, return to Step 7. Otherwise, if $\tilde{V} \neq \emptyset$, let $l := l + 1$ and return to Step 3.

Step 9 If $U = \emptyset$, halt. Otherwise, return to Step 2.

Let us consider the time complexity of algorithm FVC. The number of repetitions of the loop from Step 2 through Step 9 is $O(|V(\boldsymbol{\sigma})|)$. The initialization in Step 2 requires $O(|V(\boldsymbol{\sigma})|m)$ time for letting $set(v,k) := 0$ for all $u \in V(\boldsymbol{\sigma})$ and $k \in M$. The loop of Steps 3 to 8 is repeated $O(\nu^{\text{cyclic}})$ times. In Step 3, it takes $O(|V(\boldsymbol{\sigma})|)$ time to initialize labels $label(v,l)$ for all $v \in V(\boldsymbol{\sigma})$. Steps 4 and 5 are repeated $O(|\tilde{E}|)$ times, and take $O(1)$ time for each execution. Then, Step 6 requires $O(|\tilde{V}|l)$ time, since resetting $set(v,k) := 0$ for all $k \in M$ can be done by tracing the vertices in the path $P[v]$ for all $v \in \tilde{V}$. For the same reason, Step 7 is carried out in $O(l)$ time to update $set(v,k)$. Step 8 is executed in $O(1)$ time. The loop of Steps 7 and 8 is repeated $O(|\hat{V}|)$ times. Consequently the total time required for FVC is $O\left(|V(\boldsymbol{\sigma})|^2 m + \nu^{\text{cyclic}}|V(\boldsymbol{\sigma})||E(\boldsymbol{\sigma})| + (\nu^{\text{cyclic}})^2|V(\boldsymbol{\sigma})|^2\right) = O(\nu^{\text{cyclic}}(L^{\text{cyclic}}n)^3)$, since $m \le n$, $|E(\boldsymbol{\sigma})| = O((L^{\text{cyclic}}n)^2)$ and $|V(\boldsymbol{\sigma})| = O(L^{\text{cyclic}}n)$ hold. This complexity seems rather large; however, its actual computational time is usually much smaller, since the sizes of sets $\tilde{E}$, $\tilde{V}$ and $\hat{V}$ tend to decrease rapidly as l increases.

4.3 Time-oriented neighbor-lists

In this section, we propose the time-oriented neighbor-lists to prune the neighborhood search heuristically. A similar technique, called the neighbor-lists, was successfully applied to TSP (Johnson and McGeoch 1997) and VRP (e.g., Park, Okano and Imai 2000), in which the neighbor-list of customer i is determined on the basis of the distance from i. However, this is not appropriate for VRPGTW, since we should also take into account the start time of service at i.

Recall that all vehicles can leave the depot after time 0. Here we also assume that it is desirable to return to the depot before time w_0^{d}. We partition the period of length w_0^{d} into ν^{nlist} (a parameter) intervals, and let $\rho = w_0^{\text{d}}/\nu^{\text{nlist}}$. For each customer i, we construct ν^{nlist} time-oriented neighbor-lists, $\text{nlist}_i^{\phi} = (\text{nlist}_i^{\phi}[1], \text{nlist}_i^{\phi}[2], \ldots, \text{nlist}_i^{\phi}[n])$, for $\phi = 1, 2, \ldots, \nu^{\text{nlist}}$, where customer $\text{nlist}_i^{\phi}[l]$ with a smaller l is more desirable to be served immediately after i, provided that ϕ is the largest index satisfying $(\phi - 1)\rho \le s_i$. The desirability of a customer j to be scheduled immediately after i is determined in the increasing order of

$$\text{key}_i^{\phi}[j] = \begin{cases} d_{ij} + \min\limits_{0 \le t \le \mu^{\text{nlist}}W} p_j((\phi-1)\rho + t), & \text{if } u_i + t_{ij} \le \mu^{\text{nlist}}W \\ +\infty, & \text{otherwise,} \end{cases} \tag{14}$$

where $W = m w_0^{\text{d}}/n$ and μ^{nlist} is a parameter. In other words, $\text{nlist}_i^{\phi}[l]$ is the customer j with the lth smallest value of $\text{key}_i^{\phi}[j]$. The structure of the time-oriented neighbor-lists is illustrated in Fig. 10. (Figure 10)

The meaning of (14) is explained as follows. If time $u_i + t_{ij}$ is large, serving j immediately after i is not desirable, since it will delay the service of subsequent customers. Therefore we set $\text{key}_i^\phi[j]$ to $+\infty$ if $u_i + t_{ij} > \mu^{\text{nlist}} W$, where W is a rough estimate of the average time spent for one customer (i.e., $s_{\sigma_k(h+1)} - s_{\sigma_k(h)}$), and μ^{nlist} is set to 3 in our experiment. For those j with $u_i + t_{ij} \le \mu^{\text{nlist}} W$, the first term in (14) is the distance from customer i to customer j, while the second term represents the minimum time penalty of customer j if it is served in the time interval of $[(\phi-1)\rho, (\phi-1)\rho + \mu^{\text{nlist}} W]$.

Before proceeding further, we comment on parameter ν^{nlist}. It might be natural to set ν^{nlist} proportional to n/m based on the same consideration as W; however, large memory space is necessary if ν^{nlist} is set large. We therefore set ν^{nlist} small to keep the memory requirement reasonable.

Let us now consider the time complexity of constructing the time-oriented neighbor-lists. The computation of $\text{key}_i^\phi[j]$ requires $O(\delta^{(j)})$ time ($\delta^{(j)}$ was defined in Section 2). Therefore the computation of $\text{key}_i^\phi[j]$ for all $j = 1, 2, \ldots, n,\ j \ne i$ requires $\sum_{j=1}^n O(\delta^{(j)}) = O(\delta)$ time. It follows that the construction of nlist_i^ϕ for each of $i = 1, 2, \ldots, n$ and $\phi = 1, 2, \ldots, \nu^{\text{nlist}}$ requires $O(\delta + n \log n)$ time; hence $O(n\nu^{\text{nlist}}(\delta + n \log n))$ in total.

We now describe how we use the time-oriented neighbor-lists for the search in N^{2opt^*} and N^{cross}. We first explain the case of N^{2opt^*}. For a customer $i = \sigma_k(h_1^k - 1)$ (see Fig. 5), the customers selected for $\sigma_{k'}(h_1^{k'})$ are $\text{nlist}_i^\phi[l]$, $l = 1, 2, \ldots, L_{\text{2opt}^*}^{\text{nlist}}$ ($L_{\text{2opt}^*}^{\text{nlist}}$ is a parameter), where ϕ is fixed to $\phi = \lfloor s_i/\rho \rfloor + 1$. Similarly, in the case of N^{cross}, for a customer $i = \sigma_k(h_1^k - 1)$ (see Fig. 4), customers selected for $\sigma_{k'}(h_1^{k'})$ are $\text{nlist}_i^\phi[l]$, $l = 1, 2, \ldots, L_{\text{cross}}^{\text{nlist}}$ ($L_{\text{cross}}^{\text{nlist}}$ is a parameter), where ϕ is fixed to $\phi = \lfloor s_i/\rho \rfloor + 1$. Then, all possible values for h_2^k and $h_2^{k'}$ (i.e., $h_2^k = h_1^k, h_1^k + 1, \ldots, h_1^k + L^{\text{cross}}$, $h_2^{k'} = h_1^{k'}, h_1^{k'} + 1, \ldots, h_1^{k'} + L^{\text{cross}}$) are considered for each pair of h_1^k and $h_1^{k'}$. We do not use the time-oriented neighbor-lists in the search of N^{intra}, since it seems impossible to keep the time complexity of computing the minimum time penalty to $O(\delta)$ if this modification is added.

By using the time-oriented neighbor-lists, the neighborhood size of N^{cross} is reduced from $O((L^{\text{cross}})^2\, n^2)$ to $O((L^{\text{cross}})^2 n L_{\text{cross}}^{\text{nlist}})$, and that of N^{2opt^*} is reduced from $O(n^2)$ to $O(n L_{\text{2opt}^*}^{\text{nlist}})$. We set $L_{\text{2opt}^*}^{\text{nlist}} := \min\{n,\, (L^{\text{cross}})^2 L_{\text{cross}}^{\text{nlist}}\}$ in our algorithm so that the size of N^{2opt^*} becomes $O(\min\{n^2,\, (L^{\text{cross}})^2 n L_{\text{cross}}^{\text{nlist}}\})$. This parameter setting does not affect the overall neighborhood size, since the size of N^{cross} is already $O((L^{\text{cross}})^2 n L_{\text{cross}}^{\text{nlist}})$.

4.4 Other speedup techniques

We explain two other techniques to speed up the local search. First, we maintain tables $I^{\text{cross}}(k, k')$, $I^{\text{2opt}^*}(k, k')$ and $I^{\text{intra}}(k)$ for $k,\ k' \in M$ so that we do not search the regions where no improvement is expected. We set $I^{\text{cross}}(k, k') := 0$ whenever the algorithm finds no improvement in $N^{\text{cross}}(\boldsymbol{\sigma}, k, k')$, and set $I^{\text{cross}}(k, k') := 1$ whenever an improvement occurs by any neighborhood operation on either σ_k or $\sigma_{k'}$. Tables $I^{\text{2opt}^*}(k, k')$ and $I^{\text{intra}}(k)$ are defined similarly. Then, we search $N^{\text{cross}}(\boldsymbol{\sigma}, k, k')$ only if $I^{\text{cross}}(k, k') = 1$ holds. Similarly, we search $N^{\text{2opt}^*}(\boldsymbol{\sigma}, k, k')$ (resp., $N^{\text{intra}}(\boldsymbol{\sigma}, k)$) only if $I^{\text{2opt}^*}(k, k') = 1$ (resp., $I^{\text{intra}}(k) = 1$) holds.

The second idea is to improve the search in the cyclic exchange neighborhood. The construction of the improvement graph is the most expensive part. (Though the worst case time complexity of algorithm FVC of Section 4.2.2 seems to become more expensive if $\delta = o(\nu^{\text{cyclic}} L^{\text{cross}} n)$ hold, its actual computational time is usually much smaller than that of the construction of the improvement graph.) To alleviate this, we do not construct the whole improvement graph from scratch. This is motivated by the fact that there may be some routes not modified during the search after the previous construction. In such a case, the costs of arcs between such routes do not change. For this, we maintain a table $I^{\text{cyclic}}(k)$ for $k \in M$. The algorithm sets $I^{\text{cyclic}}(k) := 0$ for all $k \in M$ whenever the improvement graph is constructed. Then, if route k is involved in the update of solutions (by any neighborhood operation), the algorithm sets $I^{\text{cyclic}}(k) := 1$. In the reconstruction, the cost of an arc (u, v) is updated if and only if $I^{\text{cyclic}}(k_u) = 1$ or $I^{\text{cyclic}}(k_v) = 1$ holds.

Finally, we find more than one valid cycle in the improvement graph by applying algorithm FVC repeatedly. This is motivated by the fact that the costs on many arcs remain unchanged even after a cyclic exchange operation is applied. It is executed as follows. If a solution is improved by a cyclic exchange operation, we remove from the improvement graph those arcs whose head or tail node is in those routes involved in the update of the solution. Then, algorithm FVC is again applied to the reduced improvement graph. This procedure is repeated until either (1) FVC is unable to find a valid cycle, or (2) the number of the routes not involved in these updates of solutions becomes less than two. Feasible cyclic exchange operations always correspond to the valid cycles found during the above repeated calls to FVC, since $k_u \neq k_v$ holds for any two vertices u and v in these valid cycles.

5 Metaheuristic algorithms

In this section, we describe three metaheuristic algorithms, the *multi-start local search* (MLS) (Lin 1965, Lin and Kernighan 1973, Reiter and Sherman 1965), the *iterated local search* (ILS) (Johnson 1990, Martin, Otto and Felten 1991, Martin, Otto and Felten 1992), and the *adaptive multi-start local search* (AMLS) (Boese, Kahng and Muddu 1994, Taillard et al. 1997). All of these algorithms are based on the LS described so far. We describe how to generate initial solutions within their frameworks.

5.1 The multi-start local search and iterated local search

In the multi-start local search (MLS), LS is repeatedly applied from a number of initial solutions and the best solution found during the entire search is output. Algorithm MLS is summarized as follows.

Algorithm MLS

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

Step 2 Improve $\boldsymbol{\sigma}^0$ by LS; i.e., $\boldsymbol{\sigma} := \text{LS}(\boldsymbol{\sigma}^0)$.

Step 3 If a solution $\boldsymbol{\sigma}'$ satisfying $\mathit{besteval}(\boldsymbol{\sigma}') < \mathit{besteval}(\boldsymbol{\sigma}^{\text{best}})$ is found during the LS in Step 2, let $\boldsymbol{\sigma}^{\text{best}} := \boldsymbol{\sigma}'$.

Step 4 If some stopping criterion is satisfied, output $\boldsymbol{\sigma}^{\text{best}}$ and halt; otherwise generate a solution $\boldsymbol{\sigma}^0$ and return to Step 2.

In the computational experiment in Section 6, the initial solutions of MLS are generated randomly.

The iterated local search (ILS) is a variant of MLS, in which initial solutions are generated by slightly perturbing a solution $\boldsymbol{\sigma}^{\text{seed}}$, which is a good (not necessarily the best) solution found during the search. The algorithm of ILS is summarized as follows.

Algorithm ILS

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

Step 2 Improve $\boldsymbol{\sigma}^0$ by LS; i.e., let $\boldsymbol{\sigma} :=\text{LS}(\boldsymbol{\sigma}^0)$.

Step 3 If a solution $\boldsymbol{\sigma}'$ satisfying $\mathit{besteval}(\boldsymbol{\sigma}') < \mathit{besteval}(\boldsymbol{\sigma}^{\text{best}})$ is found during the LS in Step 2, let $\boldsymbol{\sigma}^{\text{best}} := \boldsymbol{\sigma}'$.

Step 4 If $\mathit{cost}(\boldsymbol{\sigma}) \leq \mathit{cost}(\boldsymbol{\sigma}^{\text{seed}})$, let $\boldsymbol{\sigma}^{\text{seed}} := \boldsymbol{\sigma}$.

Step 5 If some stopping criterion is satisfied, output $\boldsymbol{\sigma}^{\text{best}}$ and halt; otherwise generate a solution $\boldsymbol{\sigma}^0$ by slightly perturbing $\boldsymbol{\sigma}^{\text{seed}}$ and return to Step 2.

In our ILS, the random cross exchange operation is used to perturb $\boldsymbol{\sigma}^{\text{seed}}$, which randomly chooses two paths from two routes with no restriction on the length of the paths and exchanges them. The algorithm executes the random cross exchange operation $\nu_{\text{ptb}}^{\text{ILS}}$ (a parameter) times on $\boldsymbol{\sigma}^{\text{seed}}$ to generate a new initial solution $\boldsymbol{\sigma}^0$ in Step 5.

5.2 The adaptive multi-start local search

AMLS maintains a set of solutions $P = \{\boldsymbol{\sigma}^1, \boldsymbol{\sigma}^2, \ldots, \boldsymbol{\sigma}^{|P|}\}$ during the search, where $|P| \leq \nu_{\text{pop}}^{\text{AMLS}}$ holds ($\nu_{\text{pop}}^{\text{AMLS}}$ is a parameter). Solutions in P are selected from the locally optimal solutions found in the previous search, and are used to generate the initial solution for the next local search. We first describe how we generate the initial solution for the next LS, then describe how we initialize P and update it during the search, and finally summarize the outline of algorithm AMLS.

Let the current P consist of $\boldsymbol{\sigma}^\lambda = (\sigma_1^\lambda, \sigma_2^\lambda, \ldots, \sigma_m^\lambda)$, $\lambda = 1, 2, \ldots, |P|$, and $R[P]$ be the set of $|P|m$ routes $\{\sigma_1^1, \sigma_2^1, \ldots, \sigma_m^1, \sigma_1^2, \sigma_2^2, \ldots, \sigma_m^2, \ldots, \sigma_1^{|P|}, \sigma_2^{|P|}, \ldots, \sigma_m^{|P|}\}$. The initial solution $\boldsymbol{\sigma}^0 = (\sigma_1^0, \sigma_2^0, \ldots, \sigma_m^0)$ is constructed by selecting routes σ from $R[P]$ one by one. Let $\boldsymbol{\sigma}^0$ be the current solution being constructed, where some routes in $\boldsymbol{\sigma}^0$ may be empty, and let σ be a route in $R[P]$. V_σ and $V_{\boldsymbol{\sigma}^0}$ denote the sets of customers in route σ and solution $\boldsymbol{\sigma}^0$, respectively. Then let

$$c_{\mathrm{gen}}(\sigma, \boldsymbol{\sigma}^0) = \begin{cases} \frac{cost(\sigma)}{|V_\sigma \setminus V_{\boldsymbol{\sigma}^0}|}, & \text{if } |V_\sigma \setminus V_{\boldsymbol{\sigma}^0}| \neq 0 \\ \infty, & \text{otherwise.} \end{cases}$$

Cost $c_{\mathrm{gen}}(\sigma, \boldsymbol{\sigma}^0)$ measures the attractiveness of route σ with respect to the current solution $\boldsymbol{\sigma}^0$ under consideration. If $|V_\sigma \setminus V_{\boldsymbol{\sigma}^0}| = 0$ holds, all customers in σ have already been assigned to $\boldsymbol{\sigma}^0$ and hence it is useless to insert σ into $\boldsymbol{\sigma}^0$, and hence c_{gen} is set ∞. Otherwise, $c_{\mathrm{gen}}(\sigma, \boldsymbol{\sigma}^0)$ indicates the average cost to visit a customer not in $\boldsymbol{\sigma}^0$.

All the routes in $\boldsymbol{\sigma}^0$ are initially set to be empty, and then the route selection step is executed m times. In the kth route selection step, the route σ^* with the minimum $c_{\mathrm{gen}}(\sigma, \boldsymbol{\sigma}^0)$ is selected from $R[P]$ and we set $\sigma_k^0 := \sigma^*$. Here we restrict $\boldsymbol{\sigma}^0$ to contain at most $\lceil m/\nu_{\mathrm{pop}}^{\mathrm{AMLS}} \rceil$ routes from one $\boldsymbol{\sigma}^\lambda$ in P to keep its diversity. This is achieved simply by setting $c_{\mathrm{gen}}(\sigma, \boldsymbol{\sigma}^0) := \infty$ for all $\sigma \in \boldsymbol{\sigma}^\lambda$ when $\boldsymbol{\sigma}^0$ contains $\lceil m/\nu_{\mathrm{pop}}^{\mathrm{AMLS}} \rceil$ routes from $\boldsymbol{\sigma}^\lambda$. After m route selection steps, some customers may be assigned to more than one route in $\boldsymbol{\sigma}^0$. We call them duplicate customers, and apply the following customer deletion steps. Let i be a duplicate customer, which is assigned to Z routes $\sigma_{k_1}^0, \sigma_{k_2}^0, \ldots, \sigma_{k_Z}^0$. We define $\hat{\sigma}_{k_z}$ to be the route obtained by deleting customer i from $\sigma_{k_z}^0$. Then we set $\sigma_{k_z}^0 := \hat{\sigma}_{k_z}^0$ for $z = 1, 2, \ldots, Z$ except for $z = \arg\min_z\{cost(\hat{\sigma}_{k_z}^0) - cost(\sigma_{k_z}^0)\}$. This customer deletion step is applied to all duplicate customers in a random order. After this procedure, no customer is duplicate, but some customers may remain unassigned. For all of the unassigned customers, the following customer insertion steps are applied. Let i be such a customer, and j be the customer $\mathrm{nlist}_i^\phi[l]$ that has the smallest l among those already assigned to $\boldsymbol{\sigma}^0$, where ϕ is randomly chosen from $\{1, 2, \ldots, \nu^{\mathrm{nlist}}\}$. Then, customer i is inserted between j and its preceding customer in the current route. The entire procedure of generating an initial solution from P, called GENINIT(P), is summarized as follows.

Procedure GENINIT(P)

Step 1 (initialization) Let all routes in $\boldsymbol{\sigma}^0$ be empty and let $k := 1$.

Step 2 (routes selection) Let $\sigma^* := \arg\min_{\sigma \in R[P]} c_{\mathrm{gen}}(\sigma, \boldsymbol{\sigma}^0)$. Then let $\sigma_k^0 := \sigma^*$ and $k := k+1$. If $k \leq m$, return to Step 2.

Step 3 (customer deletion) Execute the customer deletion step on duplicate customers until there is no such customers.

Step 4 (customer insertion) Execute the customer insertion step on unassigned customers until there is no such customers.

Step 5 Output $\boldsymbol{\sigma}^0 = (\sigma_1^0, \sigma_2^0, \ldots, \sigma_m^0)$ as the initial solution and halt.

Initially, P is set empty. If $|P| < \nu_{\rm pop}^{\rm AMLS}$ holds, an initial solution for LS is generated randomly. Then, for the locally optimal solution $\boldsymbol{\sigma}^{\rm lopt}$ obtained by the LS, we let $P := P \cup \{\boldsymbol{\sigma}^{\rm lopt}\}$ if $cost(\boldsymbol{\sigma}^{\rm lopt})$ differs from the costs of all solutions in P (P remains unchanged otherwise). If $|P| = \nu_{\rm pop}^{\rm AMLS}$ holds, an initial solution for LS is generated by calling GENINIT(P). Then for the locally optimal solution $\boldsymbol{\sigma}^{\rm lopt}$ obtained by the LS, P is updated as follows. Let $\boldsymbol{\sigma}^{\rm worst}$ be the solution with the worst cost in P. If $cost(\boldsymbol{\sigma}^{\rm lopt}) > cost(\boldsymbol{\sigma}^{\rm worst})$ holds, P remains unchanged. Otherwise, solution $\boldsymbol{\sigma}^{\rm lopt}$ is added to P and some solutions are deleted from P by the following rules. For this, we define

$$\textsc{Rclose}(\sigma, \sigma') = \frac{|V_\sigma \cap V_{\sigma'}|}{|V_\sigma \cup V_{\sigma'}|}$$

for a pair of routes σ and σ', which measures the closeness of the assigned customers between the two routes. $\textsc{Rclose}(\sigma, \sigma') = 1$ holds if and only if the assigned customers of σ and σ' are exactly the same. For a pair of solutions $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ and $\boldsymbol{\sigma}' = (\sigma_1', \sigma_2', \ldots, \sigma_m')$, let

$$\textsc{Sclose}(\boldsymbol{\sigma}, \boldsymbol{\sigma}') = \frac{1}{m} \sum_{k=1}^{m} \max_{k'} \{\textsc{Rclose}(\sigma_k, \sigma_{k'})\},$$

which measures the closeness of the assigned customers between the two solutions. $\textsc{Sclose}(\boldsymbol{\sigma}, \boldsymbol{\sigma}') = 1$ holds if and only if the assigned customers are exactly the same. Let $\bar{m}$ (resp., $\bar{s}$) be the mean (resp., standard deviation) of $\textsc{Sclose}(\boldsymbol{\sigma}, \boldsymbol{\sigma}')$ between all pairs $(\boldsymbol{\sigma}, \boldsymbol{\sigma}')$ of solutions in P, and let $\xi(P) = \bar{m} + 2\bar{s}$ and

$$S = \{\boldsymbol{\sigma} \in P \mid \textsc{Sclose}(\boldsymbol{\sigma}^{\rm lopt}, \boldsymbol{\sigma}) \geq \xi(P)\}.$$

S is the set of solutions in P whose assigned customers are similar to that of $\boldsymbol{\sigma}^{\rm lopt}$. If $S = \emptyset$ holds, we let $P := P \cup \{\boldsymbol{\sigma}^{\rm lopt}\} \setminus \{\boldsymbol{\sigma}^{\rm worst}\}$. Otherwise (i.e., $S \neq \emptyset$), we let $S' := \{\boldsymbol{\sigma} \in S \mid cost(\boldsymbol{\sigma}) \geq cost(\boldsymbol{\sigma}^{\rm lopt})\}$ and then let $P := P \cup \{\boldsymbol{\sigma}^{\rm lopt}\} \setminus S'$ if $S' \neq \emptyset$ (P remains unchanged if $S' = \emptyset$). The above rules of updating set P for the obtained locally optimal solution $\boldsymbol{\sigma}^{\rm lopt}$ is summarized as follows.

Procedure UPDATE(P, $\boldsymbol{\sigma}^{\rm lopt}$)

Step 1 If $|P| < \nu_{\rm pop}^{\rm AMLS}$ and $cost(\boldsymbol{\sigma}^{\rm lopt})$ differs from the costs of all solutions in P, let $P := P \cup \{\boldsymbol{\sigma}^{\rm lopt}\}$ and halt.

Step 2 ($|P| = \nu_{\rm pop}^{\rm AMLS}$ holds.) For the worst solution $\boldsymbol{\sigma}^{\rm worst}$ in P, halt if $cost(\boldsymbol{\sigma}^{\rm lopt}) > cost(\boldsymbol{\sigma}^{\rm worst})$ holds.

Step 3 Let $S := \{\boldsymbol{\sigma} \in P \mid \textsc{Sclose}(\boldsymbol{\sigma}^{\rm lopt}, \boldsymbol{\sigma}) \geq \xi(P)\}$. If $S = \emptyset$, let $P := P \cup \{\boldsymbol{\sigma}^{\rm lopt}\} \setminus \{\boldsymbol{\sigma}^{\rm worst}\}$. Otherwise, compute $S' := \{\boldsymbol{\sigma} \in S \mid \mathit{cost}(\boldsymbol{\sigma}) \geq \mathit{cost}(\boldsymbol{\sigma}^{\rm lopt})\}$, and let $P := P \cup \{\boldsymbol{\sigma}^{\rm lopt}\} \setminus S'$ if $S' \neq \emptyset$.

Then algorithm AMLS is summarized as follows.

Algorithm AMLS

Step 1 Let $\mathit{best} := \infty$ and $P := \emptyset$.

Step 2 If $|P| < \nu_{\rm pop}^{\rm AMLS}$, generate an initial solution $\boldsymbol{\sigma}^0$ randomly. Otherwise, let $\boldsymbol{\sigma}^0 :=$ GENINIT(P).

Step 3 Improve $\boldsymbol{\sigma}^0$ by LS; i.e., let $\boldsymbol{\sigma}^{\rm lopt} := \mathrm{LS}(\boldsymbol{\sigma}^0)$.

Step 4 If a solution $\boldsymbol{\sigma}'$ satisfying $\mathit{besteval}(\boldsymbol{\sigma}') < \mathit{best}$ is found during the execution of LS, let $\mathit{best} := \mathit{besteval}(\boldsymbol{\sigma}')$ and $\boldsymbol{\sigma}^{\rm best} := \boldsymbol{\sigma}'$.

Step 5 Update set P by calling UPDATE$(P, \boldsymbol{\sigma}^{\rm lopt})$.

Step 6 If some stopping criterion is satisfied, output $\boldsymbol{\sigma}^{\rm best}$ and halt; otherwise return to Step 2.

6 Computational results

We conducted various computational experiments to evaluate the proposed algorithms. The algorithms were coded in C language and run on a handmade PC (Intel Pentium III 1 GHz, 1 GB memory). The program parameters were set as shown in Table 1 unless otherwise stated. The computational time was measured by using the subroutine available at Tsuchimura's web site (http://www.nn.iij4u.or.jp/˜tutimura/c/cpu_time.c). (Table 1)

6.1 The vehicle routing problem with hard time windows

6.1.1 Solomon's benchmark instances

The benchmark instances by Solomon (1987) are widely used in the literature. The number of customers in each instance is 100, and their locations are distributed in the square $[0, 100]^2$ of 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 (including the depot) has one time window $[w_i^{\rm r}, w_i^{\rm d}]$, an amount of requirement q_i and a service time u_i. All vehicles k have a fixed capacity Q. 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 $(m, d_{\rm sum}(\boldsymbol{\sigma}))$ in the lexicographical order. (For two vectors $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_l)$ and $\beta = (\beta_1, \beta_2, \ldots, \beta_l)$, α is smaller than β in the lexicographical order if and only if there is an index $i \in [1, l]$ that satisfies $\alpha_j = \beta_j$ for all $j < i$ and $\alpha_i < \beta_i$.) 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 distributed in type R and are clustered in groups in type C, and the 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.

6.1.2 Computational results

As Solomon's benchmark instances are constructed for VRPHTW, we use the lexicographical order of the vector $(q_{\mathrm{sum}}(\boldsymbol{\sigma}), p^*_{\mathrm{sum}}(\boldsymbol{\sigma}), d_{\mathrm{sum}}(\boldsymbol{\sigma}))$ as $besteval(\boldsymbol{\sigma})$ in Section 3.2. As the penalty function $p_i(t)$ for customer i, we use formula (6). How we set the number of vehicles m and parameter α in (6) is described in Appendix.

We first tested algorithm MLS without incorporating N^{cyclic} to see the effectiveness of the time-oriented neighbor-lists. Recall that the time-oriented neighbor-lists are used only for N^{2opt^*} and N^{cross}. Neighborhood N^{intra} is also used in this experiment, since the solution quality is poor without N^{intra}, and its computational time is small compared to that of N^{cross}. We tested two values $L^{\mathrm{nlist}}_{\mathrm{cross}} = 30$, 100 ($L^{\mathrm{nlist}}_{\mathrm{cross}} = 100$ means that the whole neighborhood $N^{\mathrm{cross}} \cup N^{\mathrm{2opt}^*}$ is used). The value of $L^{\mathrm{nlist}}_{\mathrm{2opt}^*}$ is set to $\min\{(L^{\mathrm{cross}})^2 L^{\mathrm{nlist}}_{\mathrm{cross}},\, n\}$ as discussed in Section 4.3. Algorithm LS was repeated from randomly generated initial solutions 50 times for each value of $L^{\mathrm{nlist}}_{\mathrm{cross}}$, and the average values of the cost and the computational time were compared. Though the results are omitted due to space limitation, it was observed that, if $L^{\mathrm{nlist}}_{\mathrm{cross}}$ is set to 30, the average computational time is reduced to 20–30% of those with $L^{\mathrm{nlist}}_{\mathrm{cross}} = 100$, while maintaining almost equivalent solution quality. Actually, LS with $L^{\mathrm{nlist}}_{\mathrm{cross}} = 30$ obtained better average cost for some instances.

We then tested the effectiveness of the cyclic exchange neighborhood. For this, we compared the quality of solutions obtained by ILS with and without N^{cyclic}. Both algorithms were run 1800 seconds for each instance, and the solution quality and the number of calls to LS were compared. Though the results are also omitted due to space limitation, not much difference in solution quality was observed. One of the conceivable reasons for this is that the number of calls to LS in ILS without N^{cyclic} is much larger than that of ILS with N^{cyclic} (i.e., many LS can compensate the power of the cyclic exchange neighborhood). However, this does not immediately mean that the cyclic exchange neighborhood is useless, as will be observed in Section 6.2.

We finally compare the best solutions obtained by algorithms ILS and AMLS with other existing methods. (The results of MLS are much worse than those of ILS and AMLS, and are omitted.) Both ILS and AMLS were run up to 15000 seconds for each instance. The results are shown in Table 2. "MNV" represents the mean number of vehicles, "MTD" represents the mean total distance, "CNV" represents the cumulative number of vehicles, and "CTD" represents the (Table 2) cumulative total distance, which are usually used in the literature to compare the results on Solomon's instances. Column "ILS 2000s" is the result of ILS, where the time limit for each instance is 2000 seconds. The meaning of columns ILS 15000s, AMLS 2000s and AMLS 15000s are similar. Column "H&G (1999)" is the result by Homberger and Gehring (1999), "G&H (2002)" is the result by algorithm HM4C in Gehring and Homberger (2002), "BBB (2003)" is the result by Berger, Barkaoui and Bräysy (2003), "Br (2003)" is the result by algorithm RVNS(2) in Bräysy (2003), "BVH (2001)" is the result by Bent and Van Hentenryck (2001), and "H&G (2003)" is the result by Homberger and Gehring (2003). According to a recent survey by Bräysy and Gendreau (2003b), not many algorithms achieved CNV 406 or less, and only those algorithms cited in Table 2 achieved CNV 405. The average computational time of algorithms H&G (1999), G&H (2002), BBB (2003), Br (2003), and BVH (2001) for each instance are roughly estimated as 800, 6000, 1300, 500, and 14000 seconds, respectively, if they were run on our computer (see Appendix for the comparison of different computers). Computational time of H&G (2003) is not clearly stated in Homberger and Gehring (2003). The solution quality of ILS and AMLS with the time limit of 2000 seconds are competitive with H&G (1999) and G&H (2002), but slightly worse than BBB (2003), Br (2003), BVH (2001), and H&G (2003). If much longer computational time, 15000 seconds, is allowed, both ILS and AMLS exhibit better quality than BBB (2003), Br (2003), and BVH (2001). Note that the computational time of our algorithms is roughly equivalent with algorithm BVH (2001).

These results are significant, since our algorithms are very general and not tailored to VRPHTW. More details of the computational results on Solomon's instances are presented in Appendix.

6.2 Parallel machine scheduling problem

6.2.1 The problem definition

For the parallel machine scheduling problem (PMP), we are given n jobs $\{1, 2, \ldots, n\}$ and m identical machines $\{1, 2, \ldots, m\}$. A machine can process only one job at a time, and processing of a job can not be stopped, once it begins, until it is completed (i.e., no preemption allowed). Each job i has:

- a processing time u_i (≥ 0), and
- a penalty function $p_i(s_i)$ (≥ 0) of the start time s_i of job i.

We are also given a penalty function $p_0(t)$ of the completion time t of all jobs in each machine. The objective is to find an assignment of jobs to machines, start times s_i of jobs i $(i = 1, 2, \ldots, n)$ and completion times s^{a}_k of machines k $(k = 1, 2, \ldots, m)$ so that the following total penalty is minimized:

$$\sum_{i=1}^{n} p_i(s_i) + \sum_{k=1}^{m} p_0(s_k^{\mathrm{a}}).$$

As can be easily understood, this is a special case of VRPGTW in which the distance and traveling time between customers, and the quantity of goods to be delivered to customers are always 0. We considered this problem in order to observe the performance of our algorithm in dealing with general time penalty functions by generating instances whose optimal values are known.

6.2.2 Generation of instances

We generate three instances of PMP, which we call LINEAR, NCONV1 and NCONV2. For each instance, 100 jobs with $u_i = 10$ $(i = 1, 2, \ldots, 100)$ are scheduled to 10 machines, and $p_0(t)$ is defined as

$$p_0(t) = \max\{-t, 0, t - 110\}.$$

We define $p_i(t)$ for $i = 1, 2, \ldots, 100$ so that $p_i(i) = 0$ holds for all i in all instances (concrete definition of $p_i(t)$ depends on the instance, and will be given later). We define the sets of jobs $I_k = \{i \mid i \equiv k - 1 \pmod{10}\}$, $k = 1, 2, \ldots, 10$. Then, an optimal schedule of cost 0 can be obtained by the following rules.

1. All jobs $i \in I_k$ are assigned to machine k for $k = 1, 2, \ldots, 10$.
2. Start time s_i of job i is set to i.
3. Completion time s_k^{a} is the completion time of the last job processed on machine k.

The three instances differ only in their penalty functions $p_i(t)$, which are defined as follows (see Fig. 11). (Figure 11)

LINEAR: Penalty functions of $i = 1, 2, \ldots, 100$ are defined by

$$p_i(t) = \max\{i - t, t - i\}.$$

NCONV1: Penalty functions of $i = 1, 2, \ldots, 100$ are defined by

$$p_i(t) = \begin{cases} i - 2 - t, & t < i - 3, \\ t - i + 4, & i - 3 \le t < i - 2, \\ i - t, & -2 \le t < i, \\ t - i, & i \le t < i + 2, \\ i + 4 - t, & i + 2 \le t < i + 3, \\ t - i - 2, & i + 3 \le t. \end{cases}$$

NCONV2: There are two types, f-type and b-type, of penalty functions for NCONV2. We denote penalty functions of f-type and b-type for job i as $p_i^{\mathrm{f}}(t)$ and $p_i^{\mathrm{b}}(t)$, respectively. These functions are defined by

$$p_i^{\mathrm{f}}(t) = \begin{cases} i-7-t, & t < i-7, \\ t-i+7, & i-7 \le t < i-3.5, \\ i-t, & i-3.5 \le t < i, \\ t-i, & i \le t, \end{cases}$$

and

$$p_i^{\mathrm{b}}(t) = \begin{cases} i-t, & t < i, \\ t-i, & i \le t < i+3.5, \\ i+7-t, & i+3.5 \le t < i+7, \\ t-i-7, & i+7 \le t. \end{cases}$$

Then, we set $p_i(t) = p_i^{\mathrm{b}}(t)$ for $1 \le i \le 10$, $p_i(t) = p_i^{\mathrm{f}}(t)$ for $91 \le i \le 100$, and we randomly choose $p_i^{\mathrm{f}}(t)$ or $p_i^{\mathrm{b}}(t)$ as $p_i(t)$ for $11 \le i \le 90$.

6.2.3 Computational results

Let MLS$^-$, ILS$^-$ and AMLS$^-$ be algorithms MLS, ILS and AMLS, respectively, in which N^{cyclic} is not incorporated. We tested six algorithms, MLS, MLS$^-$, ILS, ILS$^-$ AMLS and AMLS$^-$. Each algorithm was run three times for each instance, where each run was terminated after 1800 seconds. Table 3 shows the best costs, average costs and the average number of calls to algorithm LS. It can be observed from the table that MLS, ILS and AMLS are much more powerful than (Table 3) MLS$^-$, ILS$^-$ and AMLS$^-$, indicating the effectiveness of the cyclic exchange neighborhood. It can also observed that ILS produces slightly better solutions than MLS and AMLS. In summary, incorporating the cyclic exchange neighborhood is useful for these instances.

6.3 A production scheduling problem with inventory cost

6.3.1 The problem definition

We consider the production of $\tilde{n}$ items in a given period $[0, X]$ on m identical machines. Each item $i = 1, 2, \ldots, \tilde{n}$ has

1. a demand Z_i,
2. an inventory cost coefficient c_i^{I},
3. a production rate R_i,

and each ordered pair of (i, j) for $i, j \in \{1, 2, \ldots, \tilde{n}\}$ has

1. a setup cost c_{ij}^{S},
2. a setup time t_{ij}^{S},

where Z_i is the amount of item i to produce during $[0, X]$, c_i^{I} is the cost incurred for a unit of the accumulated inventory of item i, R_i is the time required to produce one unit of item i, c_{ij}^{S} is the cost incurred whenever the production on a machine is changed from item i to item j ($c^{\mathrm{S}}_{ii} = 0$), and t^{S}_{ij} is the time required for the setup operation ($t^{\mathrm{S}}_{ii} = 0$). We assume that the amount Z_i/X of item i is consumed per unit time. The production of an item can be divided into lots of possibly different sizes. Let ζ_i be the number of lots of item i, i_r ($r = 1, 2, \ldots, \zeta_i$) be the rth lot of item i, θ_{i_r} be the amount of production of lot i_r (i.e., the lot size) and s_{i_r} be the start time of lot i_r. Then $\sum_{r=1}^{\zeta_i} \theta_{i_r} = Z_i$ must hold. Moreover, let $\sigma_k = (\sigma_k(1), \sigma_k(2), \ldots, \sigma_k(n_k))$ denote the production order on machine k, where $\sigma_k(h)$ denote the hth lot in σ_k. Then, a schedule $\boldsymbol{\pi} = (\boldsymbol{\zeta}, \boldsymbol{\theta}, \boldsymbol{\sigma}, \boldsymbol{s})$ is determined by:

1. the number of lots ζ_i of each item $i = 1, 2, \ldots, \tilde{n}$,
2. the size θ_{i_r} of each lot i_r, $r = 1, 2, \ldots, \zeta_i,\ i = 1, 2, \ldots, \tilde{n}$,
3. a production order σ_k on machines $k = 1, 2, \ldots, m$,
4. start time s_{i_r} of production of each lot i_r, $r = 1, 2, \ldots, \zeta_i,\ i = 1, 2, \ldots, \tilde{n}$.

Determining the start times s_{i_r} is not trivial, since the machines can have idle time between adjacent lots. Let $a_i^{\boldsymbol{\pi}}(x)$ denote the accumulated amount of item i produced before time x ($0 \leq x \leq X$), which is determined by the schedule $\boldsymbol{\pi}$. Then define

$$b_i^{\boldsymbol{\pi}} = \max\left\{ \max_{0 \leq x \leq X} \{(Z_i/X)x - a_i^{\boldsymbol{\pi}}(x)\},\ 0 \right\},$$

which is the amount of item i required to be stored at time 0 so that the shortage of item i will not occur during the entire scheduling period. Then $a_i^{\boldsymbol{\pi}}(X) = Z_i - b_i^{\boldsymbol{\pi}}$ must hold. If $b_i^{\boldsymbol{\pi}}$ is stored at time 0, then the accumulated inventory $I_i^{\boldsymbol{\pi}}$ of item i over the whole period $[0, X]$ is given by

$$I_i^{\boldsymbol{\pi}} = \int_0^X (a_i^{\boldsymbol{\pi}}(x) + b_i^{\boldsymbol{\pi}} - (Z_i/X)x)dx.$$

Our objective function to be minimized is the sum of the total setup costs and the total inventory cost:

$$\sum_{k=1}^{m} \sum_{h=1}^{n_k - 1} c^{\mathrm{S}}_{\sigma_k(h), \sigma_k(h+1)} + \sum_{i=1}^{\tilde{n}} c^{\mathrm{I}}_i \cdot I_i^{\boldsymbol{\pi}}.$$

6.3.2 Formulation to VRPGTW

The production scheduling problem with inventory cost can be formulated to VRPGTW as follows. The formulation includes some approximation on lot sizing and inventory cost. First, we regard the number of lots ζ_i as a parameter, and divide the demand Z_i into ζ_i lots of the same amount Z_i/ζ_i. (If ζ_i are set larger, the approximation is more precise, since consecutive production of lots of the same item can be regarded as one lot.) In the VRPGTW formulation, each customer represents the production of a lot, and each vehicle represents a machine. Then, visiting customer i_r by vehicle k signifies that the rth lot of item i is produced on machine k. The service time u_{i_r} of customer i_r then becomes $R_i Z_i/\zeta_i$. We let the distance $d_{i_r j_{r'}}$ from customer

i_r to customer $j_{r'}$ be $d_{i_r j_{r'}} = c^{\mathrm{S}}_{ij}$, and the travel time $t_{i_r j_{r'}}$ from customer i_r to customer $j_{r'}$ be $t_{i_r j_{r'}} = t^{\mathrm{S}}_{ij}$. Finally, we set the time penalty function $p_{i_r}(t)$ to represent the inventory cost of item i as follows. Since we divide demand Z_i into ζ_i lots, the desirable start time of lot i_r is

$$s^*_{i_r} = (r-1)X/\zeta_i.$$

If the production of lot i_r begins earlier than $s^*_{i_r}$ by one unit time, the accumulated inventory $I^{\boldsymbol{\pi}}_i$ increases by Z_i/ζ_i. Conversely, if the production of lot i_r begins later than $s^*_{i_r}$ by one unit time, $b^{\boldsymbol{\pi}}_i$ should be increased to avoid shortage, and we consider that the accumulated inventory $I^{\boldsymbol{\pi}}_i$ increases by Z_i. Based on these observations, we set the time penalty function $p_{i_r}(t)$ as

$$p_{i_r}(t) = \max\left\{\alpha^{\mathrm{r}}_{i_r}(s^*_{i_r} - x), \alpha^{\mathrm{d}}_{i_r}(x - s^*_{i_r})\right\},$$

where $\alpha^{\mathrm{r}}_{i_r} = c^{\mathrm{I}}_i Z_i/\zeta_i$ and $\alpha^{\mathrm{d}}_{i_r} = c^{\mathrm{I}}_i Z_i$. (The above penalty becomes larger than the actual inventory cost if more than one lot of one item is late, since the amounts of inventory to be added to $b^{\boldsymbol{\pi}}_i$ are summed up in our penalty function, whereas only the maximum of such amounts for each item contributes to the real inventory cost.)

6.3.3 Computational results

We conducted computational experiment on real data from Kokuyo Co., Ltd. In the company, the scheduling period X is one month, and one unit of operating time is eight hours. It is requested by the company that the minimum production amount of item i at a time should be the amount to be produced in eight hours. For this, we set parameter ζ_i so that Z_i/ζ_i becomes equal to that amount. The number of machines m is 3, and the total number of lots $n = \sum_{i=1}^{\tilde{n}} \zeta_i$ is summarized in Table 4. (Table 4)

For this problem, we only tested ILS, since the objective of this experiment is to see the wide applicability of our problem formulation. For each instance, ILS was run until the number of calls to LS reaches 300. Table 4 shows the costs in yen of the schedules obtained by ILS and those of real schedules currently used by the company. (Note that the costs of our algorithm are recomputed after the schedules are obtained; i.e., they are not the approximate values used in the algorithm.) The computational times of ILS in seconds and the reduction ratio of the costs in % are also shown. Significant reduction in cost can be observed from the table.

7 Conclusion

We considered the vehicle routing problem with general time window constraints (VRPGTW), and proposed local search algorithms. Problem VRPGTW is quite general in that time window constraints are represented by general penalty functions (e.g., more than one time window for each customer can be treated). We proposed a dynamic programming algorithm to efficiently compute the optimal start times of services for customers in a given route. We also incorporated an advanced neighborhood, the cyclic exchange neighborhood, and the time-oriented neighbor-lists to make the local search more effective. As for the metaheuristic frameworks of local search, we tested the multi-start local search (MLS), the iterated local search (ILS) and the adaptive multi-start local search (AMLS). The computational results on Solomon's benchmark instances indicated that the proposed algorithms are quite effective. A parallel machine scheduling problem and a production scheduling problem with inventory cost were also solved to show a wide applicability of the proposed formulation of VRPGTW.

Acknowledgments

The authors are grateful to Akira Tanaka, Kokuyo Co., Ltd., for providing us the problem instances in Section 6.3. They are also grateful to Olli Bräysy, SINTEF Applied Mathematics, Dorit S. Hochbaum, University of California, Jörg Homberger, University of Cooperative Education, and the anonymous referees for valuable comments. This research was partially supported by Scientific Grant-in-Aid, by the Ministry of Education, Culture, Sports, Science and Technology of Japan, and by the Telecommunications Advancement Foundation of Japan.

References

R. K. Ahuja, O. Ergun, J. B. Orlin, and A. P. Punnen, "A Survey of Very Large-Scale Neighborhood Search Techniques," *Discrete Applied Mathematics* 123, 75–102 (2002).

R. K. Ahuja and J. B. Orlin, "A Fast Scaling Algorithm for Minimizing Separable Convex Functions Subject to Chain Constraints," *Operations Research* 49, 784–789 (2001).

R. K. Ahuja, J. B. Orlin, and D. Sharma, "Very Large-Scale Neighborhood Search," *International Transactions in Operations Research* 7, 301–317 (2000).

R. Bent and P. Van Hentenryck, "A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows," Technical Report, CS-01-06, Department of Computer Science, Brown University, Providence, RI 02912, September 2001.

J. Berger, M. Barkaoui, and O. Bräysy, "A Route-Directed Hybrid Genetic Approach for the Vehicle Routing Problem with Time Windows," *INFOR*, to appear (2003).

D. P. Bertsekas, *Nonlinear Programming*, Athena Scientific, Belmont (1995).

K. D. Boese, A. B. Kahng, and S. Muddu, "A New Adaptive Multi-Start Technique for Combinatorial Global Optimizations," *Operations Research Letters* 16, 101–113 (1994).

O. Bräysy, "A Reactive Variable Neighborhood Search for the Vehicle Routing Problem with Time Windows," *INFORMS Journal on Computing*, to appear (2003).

O. Bräysy and M. Gendreau "Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms," *Transportation Science*, to appear (2003a).

O. Bräysy and M. Gendreau "Vehicle Routing Problem with Time Windows, Part II: Metaheuristics," *Transportation Science*, to appear (2003b).

V. Chvátal, *Linear Programming*, Freeman, New York (1983).

J. S. Davis and J. J. Kanet, "Single-Machine Scheduling with Early and Tardy Completion Costs," *Naval Research Logistics* 40, 85–101 (1993).

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).

M. Desrochers, J. K. Lenstra, M. W. P. Savelsbergh, and F. Soumis, "Vehicle Routing with Time Windows: Optimization and Approximation," in *Vehicle Routing: Methods and Studies*, B. L. Golden and A. A. Assad (eds), 65–84, North-Holland, Amsterdam, 1988.

J. Desrosiers, Y. Dumas, M. M. Solomon, and F. Soumis, "Time Constrained Routing and Scheduling," in *Handbooks in Operations Research and Management Science*, Vol. 8, *Network Routing*, M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser (eds), 35–139, North-Holland, Amsterdam, 1995.

J. J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software," Technical Report No. CS-89–85, Computer Science Department, University of Tennessee, Knoxville, TN 37996-1301, July 29, 2002 (latest version is available as http://www.netlib.org/benchmark/performance.ps).

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

M. R. Garey, R. E. Tarjan, and G. T. Wilfong, "One-Processor Scheduling with Symmetric Earliness and Tardiness Penalties," *Mathematics of Operations Research* 13, 330–348 (1988).

H. Gehring and J. Homberger, "Parallelization of a Two-Phase Metaheuristic for Routing Problems with Time Windows," *Journal of Heuristics* 8, 251–276 (2002).

F. Glover, "Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems," *Discrete Applied Mathematics* 65, 223–253 (1996).

D. S. Hochbaum, "Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems," in *Approximation Algorithms for NP-Hard Problems*, D. S. Hochbaum (ed), 94–143, PWS Publishing Company, Boston, 1997.

D. S. Hochbaum, "Solving Integer Programs Over Monotone Inequalities in Three Variables: A Framework for Half Integrality and Good Approximations," *European Journal of Operational Research* 140, 291–321 (2002a).

D. S. Hochbaum, private communication, 2002b.

D. S. Hochbaum and J. Naor, "Simple and Fast Algorithms for Linear and Integer Programs with Two Variables Per Inequality," *SIAM Journal on Computing* 23, 1179–1192 (1994).

D. S. Hochbaum and M. Queyranne, "Minimizing a Convex Cost Closure Set," *SIAM Journal on Discrete Mathematics* 16, 192–207 (2003).

J. Homberger and H. Gehring, "Two Evolutionary Metaheuristics for the Vehicle Routing Problem with Time Windows," *INFOR* 37, 297–318 (1999).

J. Homberger and H. Gehring, "A Two-Phase Hybrid Metaheuristic for the Vehicle Routing Problem with Time Windows," *European Journal of Operational Research*, to appear (2003).

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.

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.

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, 69–85 (1992).

S. Lin, "Computer Solutions of the Traveling Salesman Problem," *Bell System Technical Journal* 44, 2245–2269 (1965).

S. Lin and B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research* 21, 498–516 (1973).

O. Martin, S. W. Otto, and E. W. Felten, "Large-Step Markov Chains for the Traveling Salesman Problem," *Complex Systems* 5, 299–326 (1991).

O. Martin, S. W. Otto, and E. W. Felten, "Large-Step Markov Chains for the TSP Incorporating Local Search Heuristic," *Operations Research Letters* 11, 219–224 (1992).

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).

N. Park, H. Okano, and H. Imai, “A Path-Exchange-Type Local Search Algorithm for Vehicle Routing and Its Efficient Search Strategy,” *Journal of the Operations Research Society of Japan* 43, 197–208 (2000).

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, 158–164 (1996).

S. Reiter and G. Sherman, “Discrete Optimizing,” *J. Society for Industrial and Applied Mathematics* 13, 864–889 (1965).

L.-M. Rousseau, M. Gendreau, and G. Pesant, “Using Constraint-Based Operators to Solve the Vehicle Routing Problem with Time Windows,” *Journal of Heuristics* 8, 43–58 (2002).

M. W. P. Savelsbergh, “The Vehicle Routing Problem with Time Windows: Minimizing Route Duration,” *ORSA Journal on Computing* 4, 146–154 (1992).

M. M. Solomon, “The Vehicle Routing and Scheduling Problems with Time Window Constraints,” *Operations Research* 35, 254–265 (1987).

M. M. Solomon and J. Desrosiers, “Time Window Constrained Routing and Scheduling Problems,” *Transportation Science* 22, 1–13 (1988).

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, 170–186 (1997).

H. Tamaki, T. Komori, and S. Abe, “A Heuristic Approach to Parallel Machine Scheduling with Earliness and Tardiness Penalties,” *Proc. 7th IEEE International Conference on Emerging Technologies and Factory Automation* (ETFA’99), pp. 1367–1370, 1999.

H. Tamaki, T. Sugimoto, and M. Araki, “Parallel Machine Scheduling Problem with a Non-Regular Objective Function — Minimizing the Sum of Earliness and Tardiness Penalties (in Japanese),” *Transactions of the Society of Instrument and Control Engineers* 35, 1176–1182 (1999).

S. R. Thangiah, I. H. Osman, and T. Sun, “Hybrid Genetic Algorithm, Simulated Annealing and Tabu Search Methods for Vehicle Routing Problems with Time Windows,” Working Paper UKC/IMS/OR94/4, Institute of Mathematics and Statistics, University of Kent, Canterbury, 1994.

P. M. Thompson and J. B. Orlin, “The Theory of Cyclic Transfers,” Working Paper OR200-89, Operations Research Center, MIT, Cambridge, MA, 1989.

P. M. Thompson and H. N. Psaraftis, “Cyclic Transfer Algorithms for Multivehicle Routing and Scheduling Problems,” *Operations Research* 41, 935–946 (1993).

Appendix: Detailed results for Solomon’s instances

The details of the results on Solomon’s instances are explained. We will omit the computational results for type C, since these instances are easy and our algorithms always output the best known values (see, e.g., the column “Our best solution” in Tables II and III of Taillard et al. (1997), or Table 8 below) in short time under wide ranges of parameter settings.

The number of vehicles m and parameter α in (6) used in our experiments are listed in Table 5. (As for the instance name, the last two digits represent the instance index and the rest indicates the type. For example, twelve instances R101–R112 belong to type R1.) According to Bräysy (2003), these values of m are taken from the best published solutions except for R101, for which Thangiah, Osman and Sun (1994) reported a solution with one vehicle less. As ILS (resp., AMLS) failed to find a feasible solution with the best known m within 2000 seconds for R104 (resp., RC105), we also ran our algorithms with one vehicle more for these two instances R104 and RC105. Parameter α was chosen after testing 10, 50 and 100 for type 1 instances, and 1, 5 and 10 for type 2 instances, respectively, in preliminary experiments except for RC105 with $m = 13$. For RC105 with $m = 13$, AMLS failed to find a feasible solution with $\alpha \leq 100$, and we decided to use $\alpha = 5000$ after testing $\alpha = 500, 1000$ and 5000. (Table 5)

The best solutions obtained by ILS and AMLS are shown in Tables 6 and 7, respectively. The figures in columns “distance” denote the distances $d_{\text{sum}}(\boldsymbol{\sigma})$ of the best solutions $\boldsymbol{\sigma}$ obtained by the algorithms within the time limit shown in the first row. The total number of calls to algorithm LS within the time limit is shown in columns “#LS in total.” Columns “time to best (s)” show the CPU seconds when the best solutions are found. Feasible solutions to VRPHTW (i.e., $p^*_{\text{sum}}(\boldsymbol{\sigma}) = 0$ and $q_{\text{sum}}(\boldsymbol{\sigma}) = 0$) were obtained for all instances except for those with “—” marks. The last two columns labeled “to find feasible” are the number of calls to algorithm LS (column “#LS to feas.”) and CPU seconds (column “time to feas. (s)”) when the first feasible solution is found. We compare our results with the best known values reported in http://www.sintef.no/static/am/opti/projects/top/vrp/bknown.html (the data was taken on March 6, 2003), which are summarized in Table 8. A single asterisk “∗” indicates a tie with the best known solution and a double asterisk “∗∗” indicates that a better solution was found. ILS obtained three tie solutions, while AMLS obtained one better solution and six tie solutions, among 39 instances in the table. (Table 6) (Table 7) (Table 8)

Finally, we estimate the computational time of algorithms H&G (1999), G&H (2002), BBB (2003), Br (2003), and BVH (2001). The benchmark results of SPECint2000, SPECint95 from

SPEC web page (http://www.specbench.org/) and Dongarra's LINPACK results (Dongarra 2002) of related CPUs are summarized in Table 9. (As we could not find LINPACK results for Pentium 400 MHz, Pentium 200 MHz, and Sun Ultra 10 440 MHz in Dongarra (2002), the corresponding results (i.e., those data in parentheses) were taken from Bräysy and Gendreau (2003b).) We also include Mflops and Mips reported in Gehring and Homberger (2002). As the results of SPECint95 and LINPACK for our CPU (Pentium III 1 GHz) are not available, the results of similar CPUs, Pentium III 933 MHz and 800 MHz, are shown. (Table 9)

Based on these data, we give in Table 10 rough estimates on the computational time spent by algorithms H&G (1999), G&H (2002), BBB (2003), Br (2003), and BVH (2001) for types R and RC instances. (The average computational time of Br (2003) reported in Bräysy (2003) on types R and RC instances ranges from 129.8 to 288.1 minutes which are the estimated values on Sun Sparc 10 using Dongarra's factors. The computational time of Br (2003) on Pentium 200 MHz in the table is therefore estimated from these data.) If the results are the best of multiple runs, we multiply the number of runs. Algorithm G&H (2002) is designed for distributed computation, where 5 computers (1 for master and 4 for slaves) were used to obtain the results in Table 2. In this case, we multiply 4, the number of slaves, to estimate the time in sequential computation, though this might be an overestimate. (Table 10)

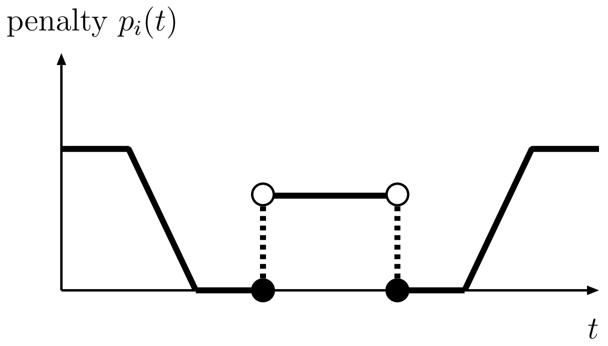


Figure 1. An example of a penalty function $p_i(t)$

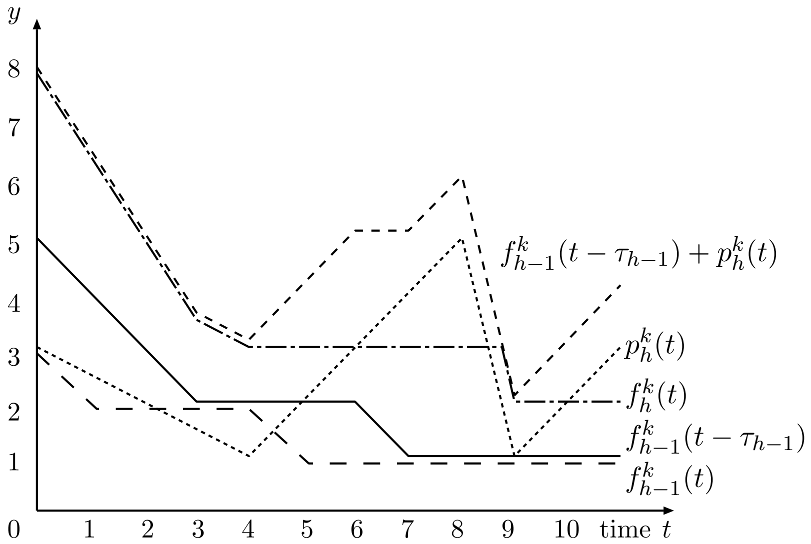


Figure 2. An example of the computation of $f^k_h(t)$ from $f^k_{h-1}(t)$ and $p^k_h(t)$

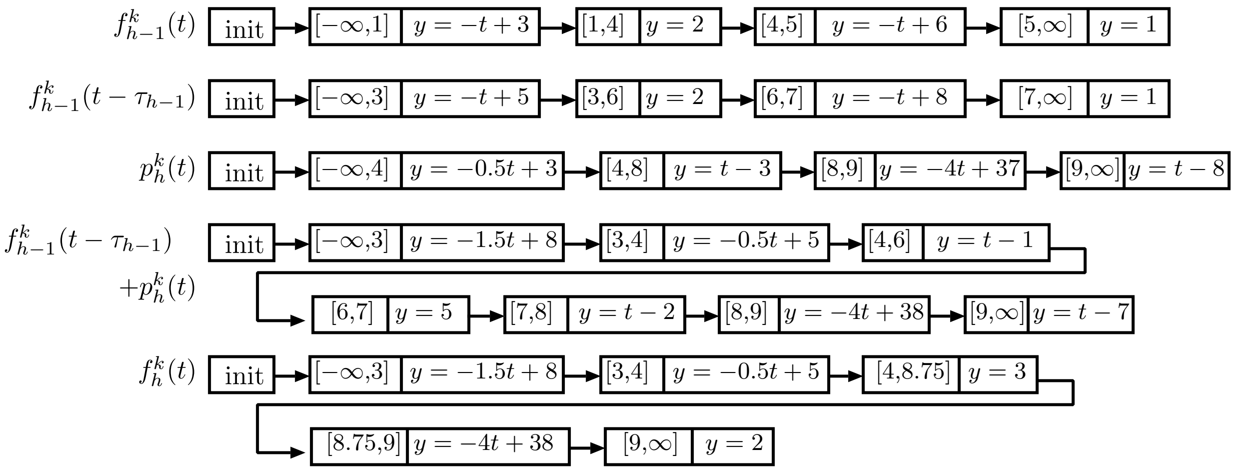


Figure 3. The linked lists representing the 4 functions in Fig. 2

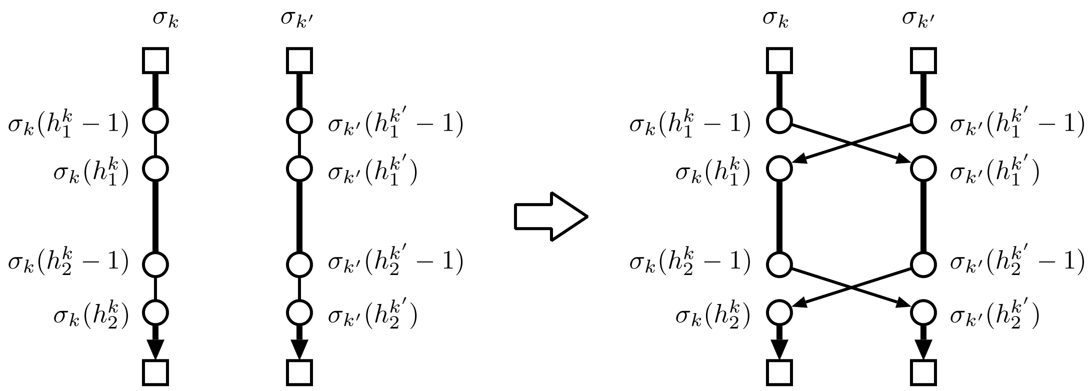


Figure 4. A cross exchange operation

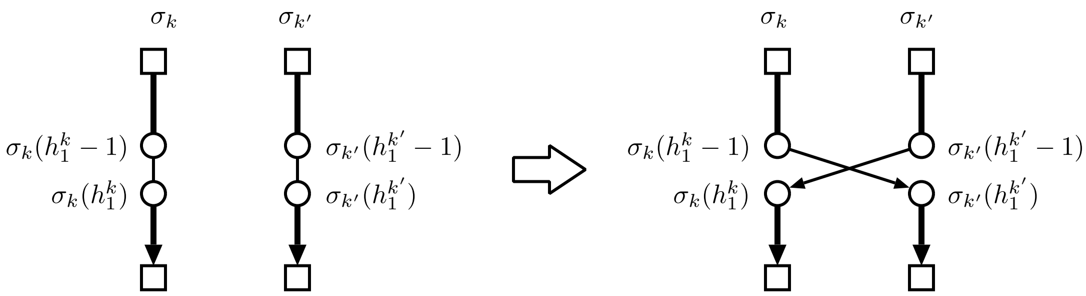


Figure 5. A 2-opt* operation

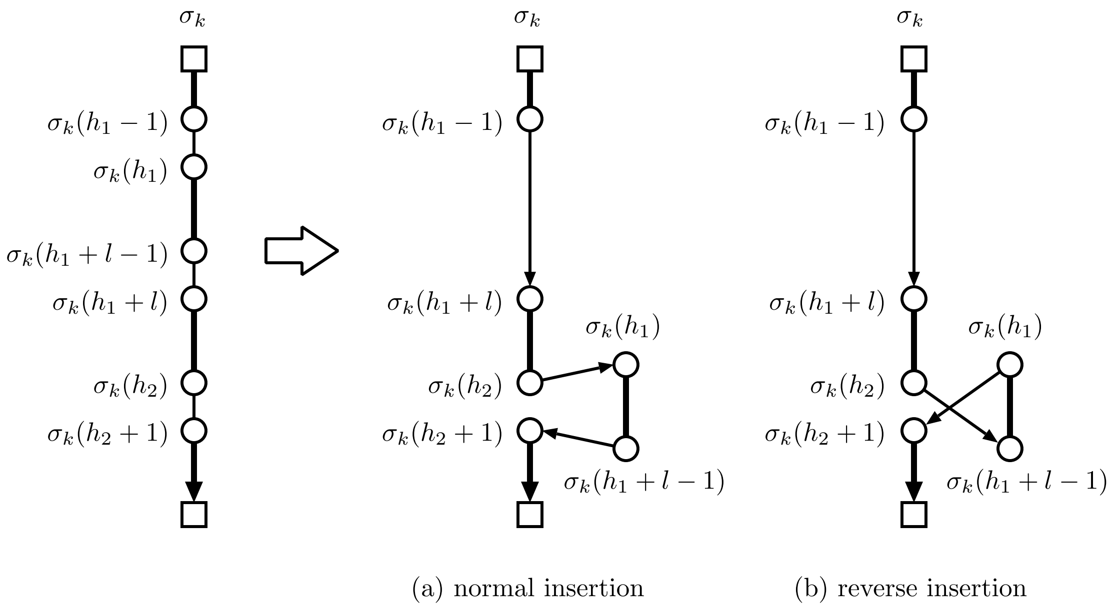


Figure 6. An intra-route operation: (a) normal and (b) reverse insertions

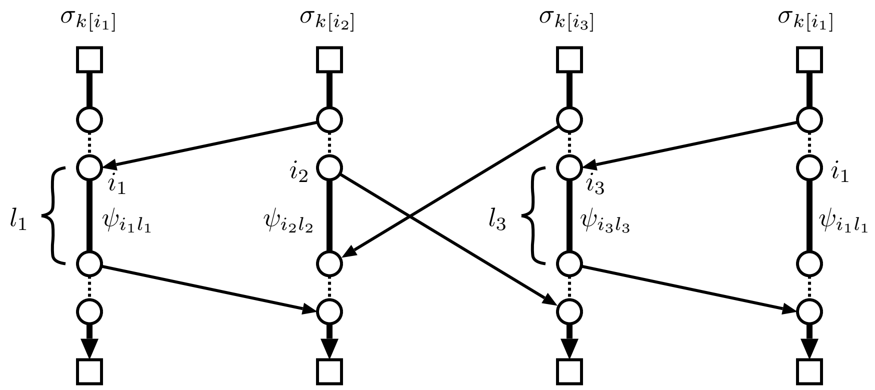
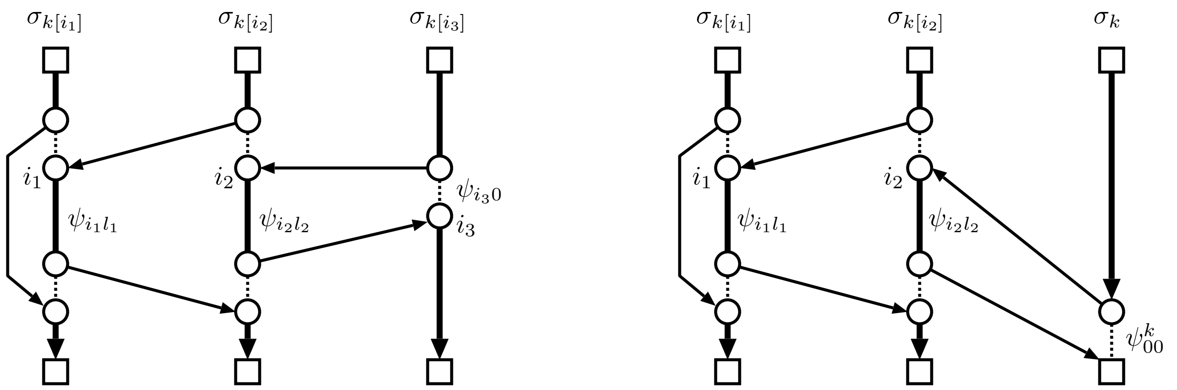


Figure 7. A cyclic exchange operation on three routes (the first and last routes represent the same route)



(a) An acyclic exchange on $\psi_{i_1 l_1}$, $\psi_{i_2 l_2}$ and $\psi_{i_3 0}$

(b) An acyclic exchange on $\psi_{i_1 l_1}$, $\psi_{i_2 l_2}$ and ψ_{00}^k

Figure 8. An acyclic exchange operation on three routes

for $h_1 = 1, 2, \ldots, n_k - 1$ **do**

  for $h_2 = h_1 + 1, h_1 + 2, \ldots, \min\{h_1 + L_{\text{ins}}^{\text{intra}}, n_k\}$ **do**

    for $l = 1, 2, \ldots, \min\{L_{\text{path}}^{\text{intra}}, h_2 - h_1\}$ **do**

      evaluate $\Delta cost$ of the new solution σ_k^{new} obtained by removing a path $\sigma_k(h_1) \rightarrow \sigma_k(h_1 + l - 1)$ and reinserting it between $\sigma_k(h_2)$ and $\sigma_k(h_2 + 1)$ with its visiting order reversed

    end for

  end for

end for.

Figure 9. Search order in the intra-route neighborhood with respect to the reverse forward insertion

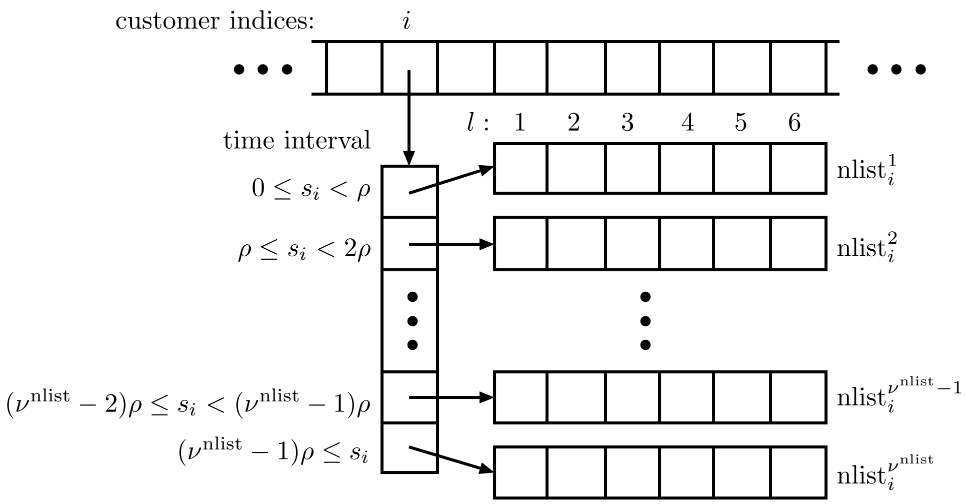


Figure 10. An illustration of the time-oriented neighbor-lists

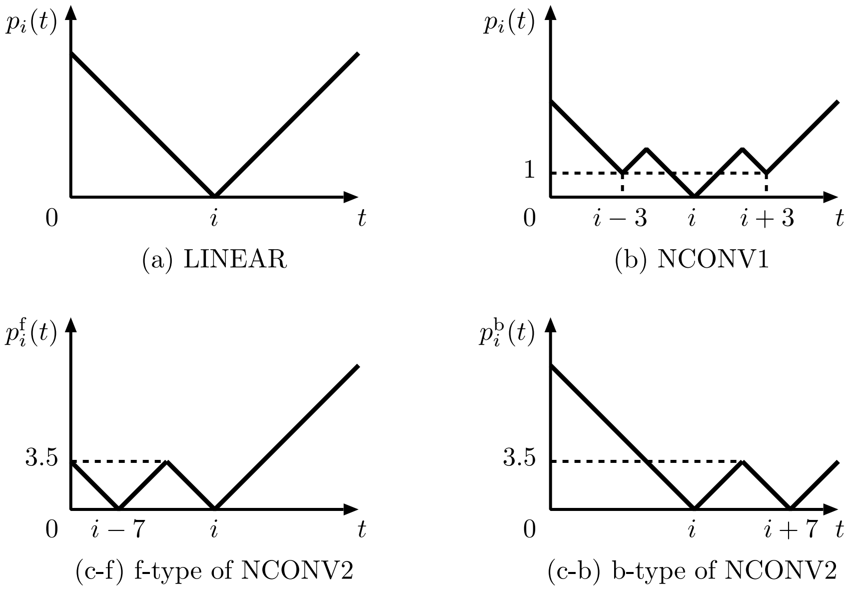


Figure 11. Penalty functions $p_i(t)$: (a) LINEAR, (b) NCONV1, (c-f) f-type of NCONV2 and (c-b) b-type of NCONV2

Table 1. The default parameter values of our algorithms

N^{cross}:	$L^{\text{cross}} = 3$
N^{intra}:	$L^{\text{intra}}_{\text{path}} = 3$, $L^{\text{intra}}_{\text{ins}} = 30$
N^{cyclic}:	$L^{\text{cyclic}} = 3$, $\nu^{\text{cyclic}} = 5$
Neighbor-lists:	$\nu^{\text{nlist}} = 20$, $\mu^{\text{nlist}} = 3$, $L^{\text{nlist}}_{\text{cross}} = 20$ ($L^{\text{nlist}}_{\text{2opt}^*} = \min\{(L^{\text{cross}})^2 L^{\text{nlist}}_{\text{cross}}, n\} = 100$)
ILS:	$\nu^{\text{ILS}}_{\text{ptb}} = 1$
AMLS:	$\nu^{\text{AMLS}}_{\text{pop}} = 10$

Table 2. Comparison of the solution quality on Solomon's instances

problem class		ILS 2000s	ILS 15000s	AMLS 2000s	AMLS 15000s	H&G (1999)	G&H (2002)	BBB (2003)	Br (2003)	†BVH (2001)	H&G (2003)
C1	MNV	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00
	MTD	828.38	828.38	828.38	828.38	828.38	828.63	828.48	828.38	828.38	828.38
C2	MNV	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00
	MTD	589.86	589.86	589.86	589.86	589.86	590.33	589.93	589.86	589.86	589.86
R1	MNV	12.00	11.92	11.92	11.92	11.92	12.00	11.92	11.92	11.92	11.92
	MTD	1215.83	1214.26	1220.02	1217.40	1228.06	1217.57	1221.10	1222.12	1213.25	1212.73
R2	MNV	2.73	2.73	2.73	2.73	2.73	2.73	2.73	2.73	2.73	2.73
	MTD	978.84	967.03	961.64	959.11	969.95	961.29	975.43	975.12	966.37	955.03
RC1	MNV	11.50	11.50	11.63	11.50	11.63	11.50	11.50	11.50	11.50	11.50
	MTD	1385.89	1385.42	1378.72	1391.03	1392.57	1395.13	1389.89	1389.58	1384.22	1386.44
RC2	MNV	3.25	3.25	3.25	3.25	3.25	3.25	3.25	3.25	3.25	3.25
	MTD	1147.38	1131.24	1132.17	1122.79	1144.43	1139.37	1159.37	1128.38	1141.24	1123.17
All	CNV	406	405	406	405	406	406	405	405	405	405
	CTD	57798	57516	57480	57444	57876	57641	57952	57710	57567	57309

†Better results with CNV 405 and CTD 57273 were reported without specifying the CPU time in Bent and Van Hentenryck (2001). The results in column BVH (2001) were taken from Tables 4 and 5 of Bent and Van Hentenryck (2001), which are the best of five runs where each run takes 7200 CPU seconds on a Sun Ultra 10 440 MHz.

Table 3. The best costs, average costs and the number of calls to algorithm LS by six algorithms applied to the parallel machine scheduling problem

instance		MLS	MLS$^-$	ILS	ILS$^-$	AMLS	AMLS$^-$
	best cost	0	4	0	0	0	2
LINEAR	avg. cost	0.0	6.7	0.0	0.7	0.0	2.0
	#LS	101.7	1048.0	113.7	985.7	120.0	936.3
	best cost	0	13	0	0	0	8
NCONV1	avg. cost	0.0	15.3	0.0	1.0	0.0	9.3
	#LS	65.7	635.0	94.3	588.3	70.3	395.3
	best cost	4	23	0	12	3	21
NCONV2	avg. cost	5.3	25.0	0.0	17.7	4.0	21.3
	#LS	55.3	720.7	94.0	677.7	58.0	403.0

Table 4. Comparison of schedules obtained by ILS and those used in the company

data	n	cost (yen) of the real schedule	ILS cost (yen)	ILS CUP secs.	reduction ratio (%)
1999.11	66	1443758	1094932	794	24.2
1999.12	67	1400603	919898	523	34.3
2000.01	57	1258431	913148	316	27.4
2000.02	60	1106430	758573	348	31.4
2000.03	59	1177601	858588	334	27.1
2000.04	66	1119503	698220	458	37.6

Table 5. The number of vehicles m and parameter α for Solomon's benchmark instances

instance	m	α	instance	m	α	instance	m	α	instance	m	α
R101	19	50	R201	4	10	RC101	14	100	RC201	4	10
R102	17	50	R202	3	10	RC102	12	100	RC202	3	10
R103	13	100	R203	3	1	RC103	11	10	RC203	3	1
R104	10	10	R204	2	10	RC104	10	10	RC204	3	1
R104	9	10	R205	3	1	RC105	14	100	RC205	4	10
R105	14	10	R206	3	1	RC105	13	5000	RC206	3	1
R106	12	10	R207	2	5	RC106	11	100	RC207	3	1
R107	10	10	R208	2	1	RC107	11	10	RC208	3	1
R108	9	10	R209	3	1	RC108	10	10			
R109	11	50	R210	3	5						
R110	10	50	R211	2	10						
R111	10	10									
R112	9	50									

Table 6. Detailed results by ILS for Solomon's instances

		time limit = 2000 secs			time limit = 15000 secs			to find feasible	
			#LS in	time to		#LS in	time to	#LS to	time to
	m	distance	total	best (s)	distance	total	best (s)	feas.	feas. (s)
R101	19	1650.80	760	190.94	1650.80	5632	190.94	21	43.01
R102	17	1487.88	598	440.49	1486.96	4073	14246.96	2	5.06
R103	13	1297.99	775	1072.59	1293.05	5338	6343.37	96	171.39
R104	10	982.02	899	1247.93	982.02	6512	1247.93	2	5.78
R104	9	—	—	—	1013.13	5313	2032.78	718	2012.69
R105	14	∗ 1377.11	786	486.91	∗ 1377.11	5319	486.91	6	13.82
R106	12	1261.94	704	736.88	1257.96	5107	9416.67	6	13.96
R107	10	1125.40	712	855.11	1120.25	5079	4869.27	92	217.28
R108	9	980.32	763	994.55	962.34	5761	4013.62	49	102.82
R109	11	1197.64	761	557.19	∗ 1194.73	5423	4700.72	92	210.70
R110	10	1119.00	763	884.47	1119.00	5499	884.47	60	126.45
R111	10	1106.30	714	190.10	1101.91	5119	12034.23	25	60.61
R112	9	1003.55	930	1984.91	993.82	6634	14486.12	817	1776.35
R201	4	1262.64	555	1893.11	1253.23	3805	3306.99	1	3.22
R202	3	1201.24	338	1671.67	1200.40	2213	4555.79	5	27.13
R203	3	953.98	313	1017.88	952.29	2208	11018.32	1	7.38
R204	2	853.86	253	200.95	853.86	1786	200.95	24	164.66
R205	3	1026.25	308	1026.25	1009.10	2190	10856.82	1	6.10
R206	3	913.18	394	1355.17	912.01	2473	9251.25	1	7.30
R207	2	977.54	240	1616.77	898.64	1672	12453.91	82	623.46
R208	2	736.43	278	1942.15	734.67	1787	4639.49	1	8.80
R209	3	919.58	331	944.87	919.58	2202	944.87	3	14.05
R210	3	967.94	359	526.71	962.45	2508	8211.05	1	5.49
R211	2	954.65	248	1093.23	941.10	1678	9217.80	45	314.73
RC101	14	1696.95	903	348.62	1696.95	6608	348.62	110	215.14
RC102	12	1554.84	873	682.01	1554.84	6150	682.01	327	672.98
RC103	11	1265.80	968	471.14	1263.54	7031	5248.24	2	4.25
RC104	10	1137.03	978	1816.49	1136.88	6800	14816.71	13	30.49
RC105	14	1543.38	869	1452.53	1540.18	6165	3312.70	6	14.00
RC105	13	1633.72	812	1550.40	1633.72	5726	1550.40	109	219.51
RC106	11	∗ 1424.73	1124	573.76	∗ 1424.73	8039	573.76	130	193.17
RC107	11	1232.26	1093	181.31	1232.26	7956	181.31	4	6.20
RC108	10	1141.76	1096	700.90	1140.40	7984	7149.54	28	44.35
RC201	4	1423.60	602	589.51	1413.52	4011	9425.99	1	2.91
RC202	3	1459.40	344	1596.03	1388.86	2519	8374.82	42	197.19
RC203	3	1096.15	410	1842.24	1077.93	2642	10083.76	4	16.18
RC204	3	799.16	458	355.70	799.16	3169	355.70	1	6.32
RC205	4	1314.40	645	1337.54	1307.15	4443	10381.77	1	2.88
RC206	3	1167.28	378	306.16	1165.04	2553	6164.89	2	10.84
RC207	3	1075.92	353	1118.53	1064.05	2399	2902.25	2	14.86
RC208	3	843.15	424	1936.64	834.23	2859	5116.93	1	4.58

∗: a tie with the best known solution

Table 7. Detailed results by AMLS for Solomon's instances

		time limit = 2000 secs			time limit = 15000 secs			to find feasible	
	m	distance	#LS in total	time to best (s)	distance	#LS in total	time to best (s)	#LS to feas.	time to feas. (s)
R101	19	1651.97	909	222.56	1650.80	7217	2999.98	2	7.00
R102	17	1487.97	771	180.45	1487.97	7098	180.45	5	20.88
R103	13	1293.05	952	1718.30	1293.05	8205	1718.30	61	137.66
R104	10	981.23	1268	1927.28	981.23	9922	1927.28	2	6.32
R104	9	1013.13	920	1986.56	1013.13	7718	1986.56	912	1986.56
R105	14	$*$ 1377.11	884	466.13	$*$ 1377.11	7213	466.13	5	16.58
R106	12	1258.48	852	1683.83	1257.96	7034	2915.93	13	39.10
R107	10	1120.98	1093	1165.17	1118.04	9293	5962.73	46	145.89
R108	9	977.28	922	1495.42	963.99	8699	2550.13	14	39.44
R109	11	1197.42	1215	1690.89	1197.42	10338	1690.89	29	91.40
R110	10	1142.85	1260	1650.77	1137.10	10579	2460.66	89	210.39
R111	10	1104.54	803	1913.35	1096.73	7747	4275.11	29	79.20
R112	9	1015.50	1118	1768.87	1015.50	9168	1768.87	434	903.86
R201	4	1256.11	1063	1900.43	1253.23	8508	2359.64	1	3.21
R202	3	1201.69	1044	398.19	1200.18	8326	4573.36	6	31.77
R203	3	946.20	599	1407.59	942.86	5060	13230.44	1	7.09
R204	2	848.59	678	1808.52	844.20	5740	9664.77	41	259.32
R205	3	1012.32	519	1175.57	1004.04	4115	4749.60	1	5.95
R206	3	914.28	599	1533.23	913.26	4877	13551.43	1	7.03
R207	2	909.80	909	1574.27	908.77	7894	2601.94	54	333.98
R208	2	728.03	601	1898.09	726.82	5476	2334.02	1	8.37
R209	3	916.99	536	1543.06	913.32	4591	2309.21	2	13.11
R210	3	939.91	719	1184.84	939.37	5127	7045.56	1	5.30
R211	2	904.14	785	1670.15	904.14	4537	1670.15	53	277.88
RC101	14	1696.95	1293	228.75	1696.95	10052	228.75	45	98.62
RC102	12	1554.84	1154	1350.84	$*$ 1554.75	10365	4089.89	118	285.01
RC103	11	1263.54	1634	1973.22	$*$ 1261.67	12480	8887.18	2	5.98
RC104	10	1138.95	1533	566.86	$*$ 1135.48	12248	9584.67	5	16.44
RC105	14	1542.29	911	1342.22	1540.18	11671	3133.75	49	121.21
RC105	13	—	—	—	1681.56	7207	5220.97	2453	5220.97
RC106	11	1453.59	1425	1831.30	$*$ 1424.73	10440	13621.57	514	733.54
RC107	11	1232.26	1735	821.58	1232.26	13306	821.58	8	18.08
RC108	10	1147.35	1681	1789.81	1140.83	12928	4134.77	11	20.06
RC201	4	1413.52	1061	734.08	1406.94	8366	6877.76	1	2.95
RC202	3	1378.42	899	1826.62	1376.03	7932	3001.17	23	100.63
RC203	3	1088.06	580	410.42	1063.68	4805	2262.31	12	70.40
RC204	3	800.83	849	662.97	799.16	7522	8742.92	1	6.31
RC205	4	1302.60	1066	375.94	1297.65	9141	4668.77	1	2.87
RC206	3	1152.03	642	1798.40	$*$ 1146.32	5459	3986.83	6	36.49
RC207	3	1093.74	540	1528.29	1064.40	4917	4274.48	2	13.74
RC208	3	$**$ 828.14	749	1522.56	$**$ 828.14	7232	1522.56	1	4.58

$*$: a tie with the best known solution; $**$: a better solution than the best known solution

Table 8. The best known values for Solomon's instances according to http://www.sintef.no/static/am/opti/projects/top/vrp/bknown.html (the data was taken on March 6, 2003)

instance	m	distance	instance	m	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	1367.09
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

CNV: 405, CTD: 57192

†The best solution of RC208 was found by us.

Table 9. Benchmark results of various CPUs

CPU	SPECint2000	SPECint95	LINPACK	Mflops	Mips
Pentium III 1 GHz	408–442				
Pentium III 933 MHz			192–234		
Pentium III 800 MHz	344	38.0			
Sun Ultra 10 440 MHz		18.1	(73)		
Pentium 400 MHz			(54)	239	940
Pentium 200 MHz		5.00–5.47	(24)	107	349
Sun Sparc 10			10		

Table 10. Estimated running time of various algorithms

algorithm	CPU	estimated speed	avg. CPU seconds for types R & RC	number of runs	number of CPUs	estimated time on our CPU (s)
ILS and AMLS	Pentium III 1 GHz	1	2000 or 15000	1	1	2000 or 15000
G&H (2002)	Pentium 400 MHz	0.24	1088–1420	5	4	5222–6816
H&G (1999)	Pentium 200 MHz	0.1	660–990	10	1	660–990
BBB (2003)	Pentium 400 MHz	0.24	1800	3	1	1300
Br (2003)	Pentium 200 MHz	0.1	3245–7203	1	1	325–720
BVH (2001)	Sun Ultra 10 440 MHz	0.4	7200	5	1	14000