Efficient algorithms for optimal pickup-point selection in the selective pickup and delivery problem with time-window constraints

Yosuke TAKADA*, Masaru SHIMAZAKI*, Yannan HU** and Mutsunori YAGIURA*

* Department of Mathematical Informatics, Graduate School of Informatics, Nagoya University
  Furo-cho, Chikusa-ku, Nagoya, 464-8601, Japan
  E-mail: y-takada@nagoya-u.jp

** Department of Applied Mathematics, Tokyo University of Science
  Kagurazaka, Shinjuku, Tokyo, 162-8601, Japan

Received: 23 December 2019; Revised: 19 April 2020; Accepted: 29 May 2020

Abstract
We address the selective pickup and delivery problem with time-window constraints, which is a problem of finding vehicle routes that pick up commodities at stores and deliver them to customers so as to minimize the total distance of the routes under capacity and time-window constraints. In this paper, to design a local search method for this problem, we consider the following: Given the order of customers in a route, we determine the pickup stores and optimal times of visiting at customers so that the total distance is minimized. This is called the optimal pickup point problem and is a subproblem of the selective pickup and delivery problem with time-window constraints. We show that the optimal pickup point problem is NP-hard in general, and then we propose dynamic programming methods, which can obtain upper and lower bounds in linear time, and optimal solution in pseudo-polynomial time.

Keywords: Vehicle routing problem, Selective pickup and delivery problem, Dynamic programming

1. Introduction

There is a recent trend of starting shopping substitute services, targeting people who have difficulties or feel inconvenience in going out for shopping. In this paper, we define a shopping substitute service as a selective pickup and delivery problem with time-window constraints (SPDPTW), which aims to find the shortest routes of vehicles to pick up goods at stores and deliver those to all customers without violating customers’ time windows. The pickup and delivery of goods are carried out by using multiple vehicles, each having a capacity. Because such a capacity is not sufficient to carry all goods to be delivered to the customers, each vehicle needs to pick up goods at stores between deliveries. Accordingly, each vehicle departs from the depot, delivers goods to customers, picking up goods at stores when necessary, and then returns to the depot. Each vehicle can collect goods requested by customers at any store. Every customer needs to be visited exactly once by a vehicle, while each store does not necessarily have to be visited and can be visited many times.

This problem is an extension of the vehicle routing problem with time-window constraints (VRPTW) (Solomon, 1987). In VRPTW, every vehicle starts and ends at the same depot, and vehicles cannot visit the depot during their trips. Some studies consider multiple use of a vehicle (Fleischmann, 1990; Taillard, Laporte, and Gendreau, 1996). In these studies, vehicles must collect goods at their depots, not at stores. In the pickup and delivery problem (Savelsbergh, 1995), vehicles pick up goods at stores and deliver them to customers; however a request of goods delivery by a customer specifies a single store to pick up goods. The SPDPTW cannot be expressed by the above problems.

There are some papers that consider the selective pickup and delivery problem with a single vehicle (Ho and Szeto, 2016; Ting and Liao, 2013), and with multiple vehicles (Ting, Liao, Huang, and Liaw, 2017); however, they mainly focus on applications that replace bikes in rental bike services, and therefore they do not consider the time window constraints.
For many variants of the vehicle routing problem (VRP) such as VRPTW and SPDPTW, in addition to determining the order of customers to visit, it is also required to determine some other variables such as the timings to visit customers. There are some metaheuristic algorithms that incorporate mechanisms to solve subproblems to determine such variables, which are necessary to evaluate neighborhood solutions efficiently. Examples of such algorithms include those for the VRP with general or convex piecewise linear time-window penalty functions (Ibaraki, Imahori, Kubo, Masuda, Uno, and Yagiura, 2005; Ibaraki, Imahori, Nonobe, Sobue, Uno, and Yagiura, 2008), with flexible travel time (Hashimoto, Ibaraki, Imahori, and Yagiura, 2006), and with time-dependent travel time (Hashimoto, Yagiura, and Ibaraki, 2008). For more about heuristics for VRP variants with additional settings, see the survey by Hashimoto, Yagiura, Imahori, and Ibaraki (2010), and Vidal, Crainic, Gendreau, and Prins (2013).

In this paper, we focus on a subproblem of SPDPTW, in which given the order of customers for a vehicle to visit, it is required to determine the pickup stores to pick up goods, and the points between customers to visit the stores so that the total distance is minimized under the capacity and time-window constraints. We call this problem the optimal pickup point problem (OPP). In devising heuristic algorithms for the SPDPTW, it is important to design efficient methods to evaluate candidate routes, and for such purposes, it is advantageous to have efficient algorithms to solve this subproblem. Even though only one route is considered and the order of customers in the route is given in this problem, we show that the OPP is NP-hard in general. We then propose dynamic programming methods that obtain upper and lower bounds on the optimal value of this problem. It can be computed in linear time by exploiting the characteristics of the recurrence formulae. We also propose a dynamic programming algorithm that solves the OPP in pseudo-polynomial time. For designing local search algorithms for the SPDPTW, it is advantageous to have such an efficient method to solve the subproblem OPP and to compute their lower and upper bounds.

2. Problem

2.1. Selective pickup and delivery problem with time-window constraints

Let $G = (V, E)$ be a complete directed graph with a vertex set $V = \{v_0, v_1, \ldots, v_n\}$ and an edge set $E = \{(v_i, v_j) | v_i, v_j \in V\}$, and $M = \{1, 2, \ldots, m\}$ be a set of vehicles. Vertex $v_0$ is the depot, and other vertices belong to a store set $V^*$ or a customer set $V^* = V^* \cap V = 0, V^* \cup V^* = V \setminus \{v_0\}$. For convenience, a vertex $v_i$ is denoted as $v_i^+$ (resp., $v_i^-$) when we emphasize that $v_i$ is in $V^+$ (resp., $V^-$). The following parameters are associated with vertices, edges, and vehicles:

- an amount $a_i(\geq 0)$ of the goods that store $v_i^+$ can supply or to be delivered to customer $v_i^-$ ($q_0 = 0$),
- a service time $u_i(\geq 0)$ at customer/store $v_i$ ($u_0 = 0$),
- a time window $[a_i, b_i]$ within which we can start the service at customer/store $v_i$,
- a distance $d_{ij}(\geq 0)$ for edge $(v_i, v_j)$,  
- a travel time $t_{ij}(\geq 0)$ for edge $(v_i, v_j)$,
- a capacity $Q(\geq 0)$ for each vehicle.

A route is a sequence of the vertices visited by one vehicle. Let $\sigma_k$ denote the route traveled by vehicle $k$, and $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$. We denote by $\sigma_k(h)$ the index of the $h$th vertex in $\sigma_k$, and we define $\sigma_k(0) = \sigma_k(n_k + 1) = 0$, where $n_k$ is the number of customers visited by vehicle $k \in M$. Let $s_k(h)$ be the start time of service of vehicle $k$ at the $h$th vertex of $\sigma_k$, $s_k(0)$ be the departure time of vehicle $k$ from the depot and $s_k(n_k + 1)$ be the arrival time of vehicle $k$ at the depot. We define $w_k(h)$ as the amount of goods that vehicle $k$ has when the vehicle arrives at the $h$th vertex of $\sigma_k$ (before starting the service at the vertex).

We consider the following constraints:

- The amount $w_k(h) \geq 0$ at all $h$ and $k$.
- Time $s_k(h)$ to start the service must be between $a_{\sigma(h)}$ and $b_{\sigma(h)}$.
- Each customer $v_i^+$ must be visited exactly once by exactly one vehicle.
- Each store $v_i^-$ can be visited multiple times by multiple vehicles and is not necessarily visited.
- $s_k(h + 1) \geq s_k(h) + u_{\sigma(h)} + t_{\sigma(h)\sigma(h+1)}$.

In this paper, we assume the following conditions for convenience:

- The service time $u_i$ at store $v_i^-$ depends neither on the amount of goods the vehicle loads, nor on the number of vehicles that visit the store at the same time.
- Stores can be visited at any time (i.e., $a_i = 0, b_i = \infty, \forall v_i^+ \in V^+$).
- The amount $q_i$ that each store $v_i^-$ can supply is infinite.

The objective of SPDPTW is to determine a collection $\sigma$ of routes that minimizes the total distance of the routes.
2.2. Optimal pickup point problem

In this paper, as a subproblem of the SPDPPTW, we consider the problem of determining at which point between customers vehicle $k$ should visit stores in a given route $\sigma_k$ consisting of only customer vertices so that the total distance is minimized. We call this problem the optimal pickup point problem (OPP). We have to determine whether the vehicle should visit a store immediately before or not, and which store the vehicle should visit. Note that the vehicle does not have to visit any store between the last customer and the depot because no goods is needed at the depot. This problem can be solved independently for each route because we assume that multiple vehicles can be supplied at the same store at the same time. We show that the OPP is NP-hard in Section 2.3, and we propose dynamic programming algorithms to solve this problem, and to obtain lower and upper bounds in Section 3.

For convenience, we assume without loss of generality that $\sigma_k = (1, 2, \ldots, n_k)$ (i.e., $\sigma_k(h) = h$ holds for every $h \in \{1, 2, \ldots, n_k\}$, and as a result, we have $i \geq n_k + 1$ for every store $v_i^+ \in V^+$ because we assumed $V^+ \cap V^- = \emptyset$ and $\sigma_k$ consists of only customer vertices in $V^-$) throughout the reminder of this paper unless otherwise stated in order to simplify notations. For example, with this assumption, $t_{\sigma_k(h-1), \sigma_k(h)} = t_{h-1, h}$ and $t_{\sigma_k(h-1), i} = t_{h-1, i}$ hold for every $h \in \{1, 2, \ldots, n_k\}$ and every store $v_i^+$, and we can simply write “$t_{h-1, h}$” and “$t_{h-1, i}$” instead of “$t_{\sigma_k(h-1), \sigma_k(h)}$” and “$t_{\sigma_k(h-1), i}$” to signify the travel time between the $(h-1)$st and the $h$th customers and that between the $(h-1)$st customer and store $v_i^+$. For store $v_i^+ \in V^+$ and customer $y_h$, we define $t_i^{I_1} = u_{h-1} + t_{h-1, i} + u_i + t_{h, i}$. Then $a_{h-1} + t_i^{I_1}$ is the earliest time that the vehicle can visit customer $v_i^+$ after picking up goods at $v_i^+$ immediately before arriving at $v_h^+$ (i.e., between $v_h^+$ and $y_h$). If $a_{h-1} + t_i^{I_1} > b_h$, it is impossible to pick up at store $v_i^+$ immediately before $v_h^+$. We define $V_h^+ = \{v_i^+ \in V^+ | a_{h-1} + t_i^{I_1} \leq b_h\}$ as the set of stores that can be visited between customers $v_{h-1}^+$ and $v_h^+$.

2.3. NP-hardness of OPP

In this section, we show that the OPP is NP-hard. It can be proved by a reduction from the knapsack problem, which is known to be NP-hard.

We consider an arbitrary instance of the knapsack problem $\text{KP}_p$: given a knapsack with capacity $C$ and a set of items $1, 2, \ldots, n$, where each item $i$ has a weight $p_i$ and a value $c_i$, the objective of this problem is to find a solution $x = (x_1, x_2, \ldots, x_n) \in \{0, 1\}^n$ that maximizes the total value $\sum_{i=1}^{n} c_i x_i$ satisfying the capacity constraint $\sum_{i=1}^{n} p_i x_i \leq C$. We transform $\text{KP}_p$ to an instance $\text{OPP}_{I_{\text{KP}}}$ of OPP.

We consider a route $(v_0, v_1^+, v_2^+, \ldots, v_n^+, v_0)$. The depot has a time window $[0, C]$, and each customer has a demand $Q$ that equals to the capacity of a vehicle; thus we need to visit a store to pickup goods before every customer. Before each customer, the vehicle can choose a store to visit from two candidate stores. For the store to be visited between customers $v_{i-1}^+$ and $v_i^+$, if the first store is chosen, the distance and the time from $v_{i-1}^+$ to $v_i^+$ equal to $c_{\text{max}}$ and zero, respectively, where $c_{\text{max}} = \max_i c_i$. If the second store is chosen, the distance and the time equal to $c_{\text{max}} - c_i$ and $p_i$, respectively. If the vehicle do not visit any stores, both the distance and the time are 0. Because any stores do not have to be visited between $v_0^+$ and $v_n^+$, the distance and the time from $v_0^+$ to $v_n^+$ are always 0. A solution of $\text{OPP}_{I_{\text{KP}}}$ can be represented by $y = (y_1, y_2, \ldots, y_n) \in \{0, 1, 2\}^n$, where $y_i = 0$ if the vehicle does not visit any store, $y_i = 1$ if the vehicle visits the first store, and $y_i = 2$ if the vehicle visits the second store between customers $v_{i-1}^+$ and $v_i^+$. Because the vehicle has to visit a store before every customer in this instance $\text{OPP}_{I_{\text{KP}}}$, $y$ is a feasible solution only if $y_i \neq 0$ for all $i$.

We construct a solution $x$ for $\text{KP}_p$ from a feasible solution $y$ for $\text{OPP}_{I_{\text{KP}}}$ with the following rule: $x_i = 0$ if $y_i = 1$ and $x_i = 1$ if $y_i = 2$. Then, $\sum_{i=1}^{n} p_i x_i \leq C$ holds because $y$ is a feasible solution, and hence we have $\sum_{i=1}^{n} p_i x_i \leq C$. Therefore the solution $x$ for $\text{KP}_p$ constructed with the above rule is feasible if $y$ is feasible for $\text{OPP}_{I_{\text{KP}}}$. Similarly, we can construct a solution $y$ from $x$ by setting $y_i = 1$ if $x_i = 0$ and $y_i = 2$ if $x_i = 1$, and such solution $y$ is feasible for $\text{OPP}_{I_{\text{KP}}}$ if $x$ is feasible for $\text{KP}_p$. The objective value of $y$ for $\text{OPP}_{I_{\text{KP}}}$ is $\sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} (c_{\text{max}} - c_i) = n c_{\text{max}} - \sum_{i=1}^{n} c_i$; here $n c_{\text{max}}$ is a constant value that does not depend on $y$. The objective value of $x$ for $\text{KP}_p$ is $\sum_{i=1}^{n} c_i x_i$, and this value equals $\sum_{i=1}^{n} c_i x_i$ for the $x$ corresponding to $y$.

From the above argument, if a solution $y$ for $\text{OPP}_{I_{\text{KP}}}$ is an optimal solution, the corresponding solution $x$ for $\text{KP}_p$ is an optimal solution. Therefore the OPP is NP-hard.

3. Algorithms for OPP

In this section, we propose dynamic programming algorithms for the OPP. We can assume that every vehicle loads full amount of goods whenever it visits a store without loss of generality because the service time $u_i$ does not depend on the amount of the goods vehicle loads.
3.1. Algorithm for lower and upper bounds

We propose dynamic programming algorithms to obtain lower and upper bounds and show that they can be computed in linear time of the number of customers in the route.

3.1.1. Algorithm for lower bound

In this section, we consider the OPP without time window constraints. The optimal value of this problem can be regarded as a lower bound on the optimal value of the OPP.

We define $T(h, w)$ as the minimum increase of distance from that of the route consisting only of customer vertices when vehicle $k$ has an amount $w$ of goods at the time it arrives at the $h$th vertex (i.e., $w_k(h) = w$) and serves from the $h$th vertex to the $n_k$th vertex satisfying all constraints (excluding the time window constraint). Then $T(h, w)$ can be computed by

$$T(n_k + 1, w) = \begin{cases} \infty, & (w < 0) \\ 0, & \text{otherwise} \end{cases}$$

$$T(h, w) = \begin{cases} \infty, & (w < 0) \\ \min(T(h + 1, Q) + \Delta d_{h+1}, T(h + 1, w - q_h)), & \text{otherwise} \end{cases}, \quad h = 0, 1, 2, \ldots, n_k, \quad (2)$$

where $\Delta d_h$ is the increase of distance when the vehicle visits the nearest store before the $h$th vertex (i.e., $\Delta d_h = \min(d_{h-1,j} + d_{j,k} - d_{k-1,h} | v^*_h \in V^*_h)$). This recurrence can be solved in $O(n_k Q)$ time by calculating $T(h, w)$ for all combinations of $h = 0, 1, 2, \ldots, n_k$ and $w = 0, 1, \ldots, Q$ in the descending order of $h$ (it is not necessary to actually compute $T(h, w)$ for $w < 0$). Then $T(0, w)$ gives the optimal value of this special case (i.e., the OPP without time window constraints) when the vehicle departs from the depot having an amount $w$ of goods.

3.1.2. Efficient lower bound computation

The recurrence in Section 3.1.1 can be computed in $O(n_k)$ time by exploiting the fact that the function $T(h, w)$ is a monotonically non-increasing step function of $w$, as shown in Fig. 1, when $h$ is fixed. Equation (2) can be computed as follows. We can obtain $T(h + 1, w - q_h)$ by shifting $T(h + 1, w)$ to the direction of $w$-axis by $q_h$. An example of this operation, which is applied to the function in Fig. 1, is shown in Fig. 2 in solid lines. The function $T(h + 1, Q) + \Delta d_{h+1}$ is a horizontal line at the height $\Delta d_{h+1}$ above the segment of $T(h + 1, w)$ that contains $w = Q$ between the left and right endpoints as illustrated by a dash-dotted line in Fig. 2. Then, we can calculate $\min(T(h + 1, Q) + \Delta d_{h+1}, T(h + 1, w - q_h))$ by taking their lower envelope (thick solid lines in Fig. 3).

We prepare a double-ended queue (deque) $A$, of which elements can be added to or removed from either the front or back. The deque $A$ stores the segments of the function, where each element in the deque stores the information of a segment $r$ (a segment is one of the steps of the step function), including the function value $a_r$ and the length $\delta_r$ in the $w$-axis direction. The function can be restored by the information in $A$. For example, suppose for simplicity that the function $T(h, w)$ for an $h$ consists of segments $r = 1, 2, \ldots$; then deque $A$ consists of the elements corresponding to segments $r = 1, 2, \ldots$, which are stored in $A$ in this order from the front, and for each $r$, the element in $A$ corresponding to

\[ T(h, w) = \text{lower envelope of } \min(T(h + 1, Q) + \Delta d_{h+1}, T(h + 1, w - q_h)), \quad h = 0, 1, 2, \ldots, n_k, \quad (2) \]
segment $r$ signifies that $T(h, w) = \alpha_r$ holds for all $w \in \left[\sum_{r=1}^{h-1} \delta_r, \sum_{r=1}^{h-1} \delta_r + \delta_h\right)$.

We also prepare a value $q_{\text{back}}$ that remembers the total shifted value of the segment at the back of $A$ and enables us to restore the information of the last element in $A$ in constant time.

We first add the segment of $T(n_k + 1, w)$ for $w \in [0, Q]$ (i.e., function value 0, and length $Q$) into empty double-ended queue $A$ and initialize $q_{\text{back}} := 0$. Then we calculate $T(h, w)$ for $h = n_k, n_k - 1, \ldots, 1$ in this order. The calculation for each $h$ can be done efficiently by the following steps.

1. For the element $r_b$ at the back of $A$, let $T_{\text{new}} := \alpha_{r_b} + \Delta d_{h+1}$ (i.e., $T_{\text{new}} := T(h + 1, Q) + \Delta d_{h+1}$ (the first term in “min” in Eq. (2)), and initialize $\delta' := q_{h}$.

2. Compare the value $\alpha_{r_f}$ of the element $r_f$ at the front of $A$ (i.e., the value of $T(h + 1, w - q_h)$ for all $w \in [\delta', \delta' + \delta_h]$) with $T_{\text{new}}$. If $\alpha_{r_f}$ is larger than or equal to $T_{\text{new}}$, remove element $r_f$ from the front of $A$ and add $\delta_{r_f}$ to $\delta'$; otherwise proceed to Step 4.

3. Return to Step 2.

4. Add the element that has the value $T_{\text{new}}$ and the length $\delta'$ to the front of $A$.

5. Add $q_h$ to $q_{\text{back}}$.

6. If $q_{\text{back}} > Q$, remove the element at the back of $A$. Then, for the element $r_b$ currently at the back of $A$, subtract the value $\alpha_{r_b}$ from $q_{\text{back}}$; otherwise stop.

7. Return to Step 6.

The computation time depends on the number of segments considered in Steps 2 and 6. The total number of added segments for $h = n_k, n_k-1, \ldots, 1$ is at most $n_k$ because at most one segment is added for each $h$, and each segment is removed at most once in Step 2 or 6. Thus the total time to compute the recurrence is $O(n_k)$. 

---

Fig. 2 Function $T(h + 1, w)$ shifted to the right by $q_h$ (i.e., $T(h + 1, w - q_h)$) and a horizontal line at a height of $T(h + 1, Q) + \Delta d_{h+1}$.

Fig. 3 Taking the minimum of the two functions in Fig. 2 (i.e., $\min T(h + 1, Q) + \Delta d_{h+1}, T(h + 1, w - q_h)$).
For simplicity, we explained the above algorithm by using a double-ended queue; however, with this method, we can only obtain the optimal value and cannot restore an optimal solution because the information of some segments of functions \( T(h, w) \) for \( h \geq 2 \) is lost when such segments are removed (and hence are deleted) in Step 2 or 6. Thus we actually use a doubly linked list to implement a double-ended queue. In Step 4, we make bidirectional links between the new element and the element \( r_t \) at the front of \( A \) and do not delete the removed element in Step 2. Similarly, we make a pointer that points an element at the back of \( A \) and do not delete the removed element in Step 6. Thus we keep the functions for all \( h \). It is easy to see that the time complexity remains the same.

### 3.1.3. Algorithm for upper bound

We propose an algorithm to obtain a feasible solution for the OPP such that the vehicle arrives at the depot earliest satisfying the time window constraints (i.e., an algorithm to solve the problem of minimizing the arriving time at the depot under the same constraints as the OPP). Although such a solution is not always an optimal solution of OPP, it gives an upper bound on the optimal value of OPP since it is feasible. By solving this problem, we can also confirm whether a feasible solution for the instance exists. Additionally, it is expected that we can obtain a good feasible solution if there is a high correlation between distance and time (this is often the case because time is almost proportional to distance in many real-world situations).

In this algorithm, among candidate stores between customers \( v_{j-1} \) and \( v_j \), we can restrict our attention to the store \( v_j^* \) that minimizes the increase of the time \( t_{f-1,j} + t_{j} - t_{f-1,j} \). We calculate such a store for each customer in advance.

We define \( F(h, s) \) as the maximum amount of goods when the vehicle arrives at the \( h \)th customer on the route and starts the service for this customer by time \( s \), satisfying all constraints at each vertex from the depot to the \( h \)th customer. If it is impossible for the vehicle to satisfy all of these customers on or before time \( s \) satisfying all constraints, the value of \( F(h, s) \) is \(-\infty\). The value of \( F(h, s) \) can be computed by

\[
F(h, s) = \begin{cases} 
  f(h, b_h), & (s > b_h) \\
  -\infty, & (s < a_h) \\
  f(h, s), & (\text{otherwise}) 
\end{cases}
\]

where \( f(h, s) \) is the maximum amount of goods when the vehicle arrives at the \( h \)th customer and starts the service by time \( s \) without considering the time window constraint of customer \( v_h^* \). The function \( f(h, s) \) can be computed by

\[
f(1, s) = \begin{cases} 
  -\infty, & (s < \min(t_1, t_1')) \\
  Q, & (s \geq t_1') \\
  0, & (\text{otherwise}) 
\end{cases}
\]

\[
f(h, s) = \max\{g(h, s), g'(h, s)\} 
\]

\( h = 2, 3, \ldots, n_k + 1, \quad \) where \( g(h, s) \) (resp., \( g'(h, s) \)) is the amount of goods the vehicle has when arriving at customer \( v_h^* \) without (resp., with) a pick up of goods immediately before \( v_h^* \). The function \( g(h, s) \) and \( g'(h, s) \) can be computed by

\[
g(h, s) = \begin{cases} 
  -\infty, & (F(h - 1, s - t_h) - q_{h-1} < 0) \\
  F(h - 1, s - t_h) - q_{h-1}, & (\text{otherwise}) 
\end{cases}
\]

\[
g'(h, s) = \begin{cases} 
  -\infty, & (F(h - 1, s - t'_h) - q_{h-1} < 0) \\
  Q, & (\text{otherwise}), 
\end{cases}
\]

where \( t_h \) is the time to serve \( v_h^* \) and travel from \( v_{h-1}^* \) to \( v_h^* \) directly, and \( t'_h \) is the time to serve \( v_h^* \) and travel from \( v_{h-1}^* \) to \( v_h^* \) via the store (i.e., \( t_h = t_{h-1} + t_{h-1,h} \) and \( t'_h = \min(t_{h-1} + t_{h-1,j} + t + t_{j,h} \mid v_j^* \in V_h^* \)).

For each \( h \), the functions (3)–(7) are monotonically non-decreasing step functions considering \( s \) as a variable in the domain \([0, b_0]\). We can compute these recurrences in \( O(n_k) \) time exploiting this fact and using a similar idea as in Section 3.1.2.

### 3.2. Algorithm for optimal solution

In this section, we propose a dynamic programming algorithm to obtain an optimal solution for the OPP. In Section 3.1.1–3.1.2 (resp., Section 3.1.3), we only have to consider a store \( v_j^* \) that minimizes \( \Delta f(h, s) \) as a candidate...
for visiting immediately before a customer $v_h^i$, where $\Delta d_h^{(i)} = d_{h-1,j} + d_{j,h} - d_{h-1,h}$ and $t_h^{(i)} = u_{h-1} + t_{h-1,j} + u_j$. However, in the OPP, we cannot fix the candidates to such stores. This is because optimal solutions do not always consist of such stores due to the fact that in the OPP, both distance and travel time have to be considered, while only one of them is considered in Sections 3.1.1–3.1.3. We define $n_{\text{max}}^s$ as the maximum number of stores the vehicle can visit before each store (i.e., $n_{\text{max}}^s = \max_i |V_{h}^s|$).

We propose in Section 3.2.1 the recurrence formula of a dynamic programming method to solve the OPP, and in Section 3.2.2, we propose a method to compute the recurrence efficiently.

### 3.2.1. Recurrence formula of dynamic programming

We define $T(h, w, s)$ as the minimum increase of distance from that of the route consisting only of customer vertices when vehicle $k$ has an amount $w$ of goods at the time it arrives at the $h$th vertex (i.e., $w_k(h) = w$), starts the service at time $s$ or later (i.e., $s_k(h) \geq s$), and serves from the $h$th vertex to the $n_k$th vertex satisfying all constraints. We can obtain an optimal value when the vehicle departs from the depot by time $s$ having an amount $w$ of goods by calculating $T(0, w, s) + \sum_{h=0}^{n_k} d_{h,h+1}$. Then $T(h, w, s)$ can be computed by

$$T(n_k + 1, w, s) = \begin{cases} \infty, & (w < 0 \lor s > b_h) \\ 0, & \text{(otherwise)} \end{cases}$$

(8)

$$T(h, w, s) = \begin{cases} \infty, & (w < q_h \lor s > b_h) \\ \min \{ \min[T(h + 1, Q, \max(s, a_h) + t_h^{(i)} + \Delta d_h^{(i)} | v_i^s \in V_{h+1}^s), T(h + 1, w - q_h, \max(s, a_h) + t_h^{(i)})] \} & \text{(otherwise)} \\ h = 0, 1, 2, \ldots, n_k \} 

This recurrence can be solved in $O(b_h n_{\text{max}}^s n_k)$ time by calculating $T(h, w, s)$ for all combinations of $h \in \{0, 1, 2, \ldots, n_k + 1\}$, $w \in \{0, 1, \ldots, Q\}$, and $s \in \{0, 1, \ldots, b_h\}$ in an appropriate order (decreasing order of $h$). We can improve this computation time using a similar idea as in Section 3.1.2.

### 3.2.2. Efficient dynamic programming

The recurrence in Section 3.2.1 can be computed more efficiently by exploiting the fact that the function $T(h, w, s)$ is a monotonically non-increasing step function of $w$ with at most $n_k + 1$ segments when $h$ and $s$ are fixed. We use a double-ended queue (a linked list to be more precise) as in Section 3.1.2 to represent function $T(h, w, s)$ for each $s$. Note that the algorithm in Section 3.2.2 is based on the fact that we can obtain the value $T_{\text{new}} = T(h + 1, Q) + \Delta d_{h+1}$ by referring to the back of the double ended queue; thus the algorithm runs in $O(n_k)$ time. However, we cannot use the same idea to compute the value $T(h + 1, Q, \max(s, a_h) + t_h^{(i)} + \Delta d_h^{(i)}$ because for each $h$, we need to keep more than one function $T(h, w, s)$ for different values of $s$, and the segment corresponding to $T(h, Q, s)$ for any $h$ and $s$ is reachable only by tracing the linked list of function $T(h, w, s)$ from the front (i.e., in ascending order of $w$). Although this takes $O(n_k)$ time, with this idea, we can improve the computation time of $O(b_0 n_{\text{max}}^s n_k)$ to $O(b_0 n_{\text{max}}^s n_k^2)$.

We can further reduce the computation time by using the following arrays. Let each segment $r \in \{1, 2, \ldots\}$ of a step function $T(h, w, s)$ for fixed $h$ and $s$ have an array $B_r$ that has links to segments in larger domain. The $j$th element of $B_r$ has a link to the $2^j$th element from $r$. The length of an array $B_r$ is $O(\log n_k)$ for each $r$. We can construct such an array for a segment added to the front of the double-ended queue in $O(\log n_k)$ time if each segment in functions $T(h', w, s')$ for all $h' \geq h + 1$ and $s'$ has such an array. When we compute Eq. (9), we can calculate the value $T(h + 1, Q, \max(s, a_h) + t_h^{(i)} + \Delta d_h^{(i)}$ for each $v_i^s \in V_{h+1}^s$ in $O(\log n_k)$ time by using binary search for the array. Therefore, the computation time of the recurrence can be reduced to $O(b_0 n_{\text{max}}^s n_k \log n_k)$.

### 4. Local search

In this section, we consider a method to use dynamic programming algorithms for the OPP as a part of local search algorithms for the SPDPTW. A possible search space would be the sets of routes that consist of only customers. Then we have to determine at which point between customers vehicle should visit stores and which stores to be visited for each neighbor solution to check the feasibility and the solution quality. Because it is computationally expensive to solve an instance of OPP by computing Eq. (8)–(9) for each neighbor solution, it is nicer to have methods to reduce the computational effort for this part. One possible method is to compute Eq. (3) first to check the feasibility before computing Eq. (8)–(9). If the solution is infeasible, we do not have to compute Eq. (8)–(9). Even if the solution is feasible, we do
not have to solve the OPP optimally unless the lower bound that can be obtained by computing Eq. (1)–(2) is less than the objective value of the current solution. Thus we can omit expensive computation and improve the efficiency of local search by computing upper and lower bounds first.

5. Conclusion

We considered the selective pickup and delivery problem with time-window constraints (SPDPTW). We defined the optimal pickup point problem (OPP) as a subproblem of SPDPTW, and we proposed dynamic programming algorithms to obtain upper and lower bounds in linear time, and an optimal solution in pseudo-polynomial time. We also discussed some possible methods to make local search algorithms for SPDPTW efficient by using the dynamic programming algorithms for the OPP.

Acknowledgments

The authors are grateful to the anonymous referees for their valuable comments. This work was supported by JSPS KAKENHI Grant Numbers JP15H02969, JP20H02388, JP19K04900, JP20K04983.

References

Fleischmann, B, The vehicle routing problem with multiple use of the vehicles, Working paper (1990).
Hashimoto, H., Ibaraki, T., Imahori, S., and Yagiura, M., The vehicle routing problem with flexible time windows and traveling times, Discrete Applied Mathematics, Vol. 154, No. 16 (2006), pp. 2271–2290.
Hashimoto, H., Yagiura, M., and Ibaraki, T., An iterated local search algorithm for the time-dependent vehicle routing problem with time windows, Discrete Optimization, Vol. 5, No. 2 (2008), pp. 434–456.
Hashimoto, H., Yagiura, M., Imahori, S., and Ibaraki, T., Recent progress of local search in handling the time window constraints of the vehicle routing problem, 4OR, Vol. 8 (2010), pp. 221–238.
Ho, S. C. and Szeto, W. Y., GRASP with path relinking for the selective pickup and delivery problem, Expert Systems with Applications, Vol. 51 (2016), pp. 14–25.
Ibaraki, T., Imahori, S., Kubo, M., Masuda, T., Uno, T., and Yagiura, M., Effective local search algorithms for routing and scheduling problems with general time-window constraints, Transportation Science, Vol. 39, No. 2 (2005), pp. 206–232.
Ibaraki, T., Imahori, S., Nonobe, K., Sobue, K., Uno, T., and Yagiura, M., An iterated local search algorithm for the vehicle routing problem with convex time penalty functions, Discrete Applied Mathematics, Vol. 156, No. 11 (2008), pp. 2050–2069.
Savelsbergh, M. W. P. and Sol, M., The general pickup and delivery problem, Transportation Science, Vol. 29 (1995), pp. 17–29.
Solomon, M. M., Algorithms for the vehicle routing and scheduling problems with time window constraints, Operations Research, Vol. 35 (1987), pp. 254–265.
Taillard, É., Laporte, G., and Gendreau, M., Vehicle routing with multiple use of vehicles, Journal of the Operational Research Society, Vol. 47, No. 8 (1996), pp. 1065–1070.
Ting, C. K. and Liao, X. L., The selective pickup and delivery problem: Formulation and a memetic algorithm, International Journal of Production Economics, Vol. 141, No. 1 (2013), pp. 199–211.
Ting, C. K., Liao, X. L., Huang, Y. H. and Liaw, R. T., Multi-vehicle selective pickup and delivery using metaheuristic algorithms, Information Sciences, Vol. 406–407 (2017), pp. 146–169.
Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C., Heuristics for multi-attribute vehicle routing problems: A survey and synthesis, European Journal of Operational Research, Vol. 231, No. 1 (2013), pp. 1–21.