2007－AL－113 (7)
2007／7／3

多制約配送計画問題に対する集合被覆アプローチ

橋本 英樹[1], 江崎 洋一[2], 柳浦 睦憲[3],
野々部 宏司[4], 茨木 俊秀[5], Arne Løkketangen[6]
[1] 京都大学, [2] キヤノンシステムソリューションズ, [3] 名古屋大学,
[4] 法政大学, [5] 関西学院大学, [6] Molde College

各ルートに様々な制約を課すことを許した配送計画問題を考え、集合被覆アプローチに基づく近似解法を提案する。この問題は、すべての顧客を実行可能なルート集合で被覆するという集合被覆問題として定式化できる。本解法では、まず実行可能なルート集合を生成し、そのルート集合を修正するという操作を反復し、最後に、求めたルート集合に対応する集合被覆問題を解いて、本問題の解を得る。よい解を得るためには、ルート集合のサイズが大き過ぎずかつ十分な多様性を持つことが期待される。本解法では、ルート集合の修正にラグランジュ緩和の情報を利用することで、その実現を図る。計算実験により、様々な制約を課した問題例に対して、本解法の効果を確認する。

A set covering approach for a generalization of the pickup and delivery problem with time windows by allowing general constraints on each route

Hideki Hashimoto[1], Youichi Ezaki[2], Mutsunori Yagiura[3],
Koji Nonobe[4], Toshihide Ibaraki[5], and Arne Løkketangen[6]
[1] Kyoto University, [2] Canon System Solutions Inc., [3] Nagoya University
[4] Hosei University, [5] Kwansei Gakuin University, [6] Molde College

We consider a generalization of the pickup and delivery problem with time windows by allowing general constraints on each route, and propose a heuristic algorithm based on the set covering approach, in which all requests are required to be covered by a set of feasible routes. Our algorithm first generates a set of feasible routes, and repeats reconstructing of the set by using information from a Lagrangian relaxation of the set covering problem corresponding to the set. The algorithm then solves the resulting set covering problem instance to find a good feasible solution for the original problem. We conduct computational experiments for instances with various constraints and confirm the flexibility and robustness of our algorithm.

1 Introduction

The *pickup and delivery problem with time windows* (PDPTW) is one of the useful variants of the vehicle routing problem with time windows (VRPTW) [11, 13]. In this problem, we are given a set of requests, where each request signifies the delivery of a demand from the origin to the destination. The origin and destination of each request must be visited by the same vehicle in the order of origin and destination. Each service (i.e., pickup at an origin or delivery at a destination) must start within a given time window (time window constraint). Each vehicle has a capacity and the total amount of loads of a vehicle cannot exceed its capacity (capacity constraint).

This problem has been widely studied recently. Savelsbergh and Sol [12] proposed a branch and price algorithm based on a set partitioning formulation. Dumas, Desrosiers and Soumis [3] proposed a column generation scheme using a constrained shortest path as a subproblem. Many heuristic algorithms have also been proposed. Nanry and Barnes [7] presented a reactive tabu search approach. A variant of the genetic algorithm called a grouping genetic algorithm was presented by Pankratz [8]. Li and Lim [6] proposed a tabu-embedded simulated annealing. They also generated new instances, and tested the performance of their algorithm on them. Bent and Van Hentenryck [2] and Ropke and Pisinger [10] proposed large neighborhood search based algorithms, and obtained good results on the benchmark instances of Li and Lim.

In this paper, we consider a generalization of the pickup and delivery problem with time windows by allowing general constraints on each route (abbreviated as PDP-GCER). We assume that the constraints on each route satisfy the monotone property:

If a route consisting of a set of requests satisfies a constraint, then any subroute (i.e., consisting of a subset of the requests) also satisfies the constraint.

Any monotone constraint is allowed provided that we can determine its feasibility in a reasonable time. We also assume the traveling times satisfy the triangle inequality, which implies that capacity and time window constraints satisfy the monotone property. We note that many constraints that appear in practical situations are often monotone.

In our algorithm, the problem is formulated as the set covering problem (abbreviated as SCP), in which all requests must be covered by a set of feasible routes. That is, an element and a set of SCP correspond to a request and a route, respectively. Since enumerating all feasible routes is not realistic for reasonable problem sizes, we try to construct a set of good routes which is of manageable size but has sufficient diversity. We construct the initial set of routes by an insertion method, and then repeats reconstruction procedure. In the reconstruction procedure, we estimate the attractiveness of each route by the relative cost of the Lagrangian relaxation of the set covering problem with the current set of routes. We generate new routes from those with small relative costs by applying five types of operations. The algorithm then solves the resulting set covering problem instance to find a good feasible solution of PDP-GCER. Although a solution of SCP may cover a request more than once, we can transform it into a feasible solution of the original problem because of the monotone property of constraints. This type of approach, called column generation, tends to be efficient for problems with complicated or tight constraints. Note that our algorithm is a heuristic algorithm though the column generation method is usually used for exact algorithms. For PDPTW, Savelsbergh and Sol [12] and Dumas, Desrosiers and Soumis [3] proposed exact algorithms using the column generation approach based on a set partitioning formulation of the problem.

To confirm the flexibility and efficiency of our algorithm, we conduct computational experiments. We first confirm the efficiency of using the information from the Lagrangian relaxation, and then compare our algorithm with a local search type algorithm which we prepared for the purpose of comparison, and confirm the flexibility of our algorithm.

2 Problem Definition

Let $G = (V, E)$ be a complete directed graph with vertex set $V = \{0, 1, \ldots, 2n\}$ and edge set $E = \{(i, j) \mid i, j \in V, i \neq j\}$. In this graph, vertex 0 is the depot and other vertices are customers where a load is picked up or delivered. Each edge $(i, j) \in E$ has a traveling cost $c_{ij} \geq 0$ and a traveling time $t_{ij} \geq 0$. The traveling costs and times satisfy the triangle inequalities, $c_{ik} + c_{kj} \geq c_{ij}$ and $t_{ik} + t_{kj} \geq t_{ij}$ for $i, j, k \in V$. Let $H = \{1, 2, \ldots, n\}$ be the given set of requests. Each request $h \in H$ signifies the delivery from the origin $h \in V$ to the destination $h + n \in V$ (for convenience, we call a request and its origin by the same name h). The vertices h and $h + n$ must be visited by the same vehicle (coupling constraint), and h must be visited before $h + n$ (precedence constraint). We consider the problem of serving all requests by a fleet of homogeneous vehicles. Each vehicle must start from the depot and return to the depot. Let S_r be the set of requests served in route r, $m_r = |S_r|$, and σ_r be the sequence of customers to be visited, where $\sigma_r(k)$ denotes the kth customer in r. We assume $\sigma_r(0) = \sigma_r(2m_r + 1) = 0$.

In this paper, we consider various constraints on each route. Each customer $i \in V$ has a handling time s_i for the service and a time window $[e_i, l_i]$, where e_i is the release time to serve i and l_i is the deadline of the service. Each request h consumes q^{re}_{hp} units of renewable resources ($p = 1, 2, \ldots, \rho$) while it is loaded, and consumes $q^{\mathrm{non}}_{hp'}$ units of nonrenewable resources ($p' = 1, 2, \ldots, \pi$). Each vehicle has capacities Q^{re}_p for renewable resource p and $Q^{\mathrm{non}}_{p'}$ for nonrenewable resource p'. The total load of each renewable resource p at each customer in route r must not exceed the capacity Q^{re}_p; i.e.,

$$\sum_{h \in S_r : \sigma_r^{-1}(h) \leq k < \sigma_r^{-1}(h+n)} q^{\mathrm{re}}_{hp} \leq Q^{\mathrm{re}}_p \text{ for any } k = 0, 1, \ldots, 2m_r.$$

The total load of each nonrenewable resource p' must be within $Q^{\mathrm{non}}_{p'}$; i.e.,

$$\sum_{h \in S_r} q^{\mathrm{non}}_{hp'} \leq Q^{\mathrm{non}}_{p'}.$$

We further introduce Last-In First-Out (abbreviated as LIFO) constraint. That is, if a request h is picked up before a request h', either h is delivered before the pickup of h' or after the delivery of h'; i.e., if $\sigma_r^{-1}(h) < \sigma_r^{-1}(h')$, either $\sigma_r^{-1}(h) < \sigma_r^{-1}(h') < \sigma_r^{-1}(h'+n) < \sigma_r^{-1}(h+n)$ or $\sigma_r^{-1}(h) < \sigma_r^{-1}(h+n) < \sigma_r^{-1}(h') < \sigma_r^{-1}(h'+n)$ must hold. Note that LIFO constraint satisfies the monotone property. PDPTW has the time window constraint and one dimensional renewable resource (i.e., $\rho = 1$ and $\pi = 0$). Hence, in this paper, we assume that the problem has the time window constraint and and $\rho \geq 1$. As for the LIFO constraint, we consider both cases where the constraint is imposed and not. In addition to the above constraints, any monotone constraint can be introduced, assuming that we have an algorithm to efficiently test its feasibility.

Let ν be the number of vehicles used in a solution. A feasible solution is a set $\{\sigma_1, \sigma_2, \ldots, \sigma_\nu\}$ of routes such that each σ_r satisfies all the given constraints and each request is serviced exactly once. In the literature, it is often considered that the primary objective is to reduce the number of vehicles, and the secondary objective is to minimize the total traveling cost. However, for convenience, we adopt the following objective function:

$$\sum_{r=1}^{\nu} C_r,$$

where $C_r = \alpha + \sum_{i=0}^{2m_r} c_{\sigma_r(i)\sigma_r(i+1)}$ (i.e., C_r is the sum of a fixed cost α for using a vehicle and the traveling cost of r). If we need to reduce the number of vehicles, we set α to a large value compared with the traveling cost.

We remark that the following property holds because the traveling cost satisfies the triangle inequality and constraints on each route satisfy the monotone property:

Property 2.1 *Given a feasible route, any request can be deleted from the route without violating the constraints and without increasing the cost.*

3 Set Covering Approach

The PDP-GCER can be formulated as the following set covering problem:

$$\begin{array}{lll} \mathrm{SCP}(R^*) & \text{minimize} & \displaystyle\sum_{r \in R^*} C_r x_r \\ & \text{subject to} & \displaystyle\sum_{r \in R^*} a_{hr} x_r \geq 1, \quad \forall h \in H \\ & & x_r \in \{0,1\}, \qquad \forall r \in R^* \end{array}$$

where R^* is the set of all feasible routes, and $a_{hr} = 1$ if request h is in route $r \in R^*$, otherwise $a_{hr} = 0$. Note that in this formulation we can write $\sum_{r \in R^*} a_{hr} x_r \geq 1$ instead of $\sum_{r \in R^*} a_{hr} x_r = 1$ by Property 2.1.

However, enumerating all feasible routes is not realistic because the size of R^* is exponentially large. We therefore choose a subset R of manageable size from all feasible routes R^* and solve the corresponding set covering problem SCP(R). In order to solve SCP(R), we use the algorithm proposed by Yagiura et al. [14]. Finally we obtain a solution of PDP-GCER from the solution of SCP(R). The solution to SCP(R) may contain more than one route serving the same requests. In this case, based on Property 2.1, we can remove the over-covered requests one by one in a greedy way until no such request remains.

The following is the outline of our algorithm:

1. Generate a set R of feasible routes.
2. Solve the resulting instance of SCP(R).
3. Construct a feasible solution of PDP-GCER from the solution of SCP(R).

The main part of our algorithm is how to determine the set R. To obtain a good solution, we need to choose R very carefully. For instance, if we generate a large set R that has only similar routes, the quality of a solution will be poor. On the other hand, if we can construct a small set R of good routes having sufficient diversity, then we can expect a good solution. The details of generating routes will be described in Section 4.

4 Route Generation

The generation of a set of routes consists of two phases. The first phase is the initial construction phase, which generates a certain number of routes for each request by an insertion method. The second phase is the reconstruction phase, which chooses good routes from the current set of routes, and generates their neighbor routes.

In our algorithm, we may possibly generate a route that has the same requests with that of a route already in R. To avoid such duplication, we use a hash table, and check whether such a route exists in R or not whenever we add a new route into R. If a route with the same requests exits, we choose the route having the lower cost. Thus we keep a single route which has the lowest cost among those which have already been generated and have the same requests.

4.1 Initial Construction Phase

The initial construction phase starts from the empty set $R = \emptyset$, and generates a certain number of routes for each request by an insertion method. The insertion method first prepares a route that contains a single request and the depot, and then repeats inserting requests into the route by the criteria described below. When the route becomes maximal (i.e., no more request can be inserted to it), we add it to R.

The insertion cost of request h into route r when h is inserted between $\sigma_r(k)$ and $\sigma_r(k+1)$ and $n+h$ is inserted between $\sigma_r(k')$ and $\sigma_r(k'+1)$ $(k' \geq k)$ is defined by

$$\delta_r(h,k,k') = \begin{cases} c_{\sigma_r(k)h} + c_{h,h+n} + c_{h+n,\sigma_r(k+1)} - c_{\sigma_r(k)\sigma_r(k+1)}, & \text{if } k = k' \\ c_{\sigma_r(k)h} + c_{h\sigma_r(k+1)} - c_{\sigma_r(k)\sigma_r(k+1)} & \\ \quad + c_{\sigma_r(k'),h+n} + c_{h+n,\sigma_r(k'+1)} - c_{\sigma_r(k')\sigma_r(k'+1)}, & \text{otherwise.} \end{cases}$$

Let $\delta_r^{\min}(h)$ be the minimum insertion cost of request h into r among those k and k' whose resulting routes are feasible. If all combinations of k and k' are infeasible, we set $\delta_r^{\min}(h) = \infty$. However, if we always insert a request which achieves the minimum insertion cost, the resulting set of routes may not have sufficient diversity, which is not desirable in order to achieve high performance of PDP-GCER. We therefore incorporate randomness in the algorithm based on the idea often used in GRASP (greedy randomized adoptive search procedure) [4]. Let D_r be the set of requests h with the κ (κ is a parameter) smallest values of $\delta_r^{\min}(h)(< \infty)$ among those in $H \setminus S_r$ (i.e., the requests not in route r). Then, in each iteration, we choose a request h randomly from D_r and insert it into r at its best position achieving $\delta_r^{\min}(h)$.

4.2 Reconstruction Phase

In the reconstruction phase, we modify the set of routes by using the Lagrangian relaxation of the set covering problem. We first calculate the Lagrangian multipliers by a subgradient method, and select routes from the current set R by using the relative cost. Next we generate routes by applying five types of operations to the selected routes, and update the set R of routes.

4.2.1 Selection of Routes.

From the current set R, we first select some number of routes for use in generating new routes. We estimate the attractiveness of a route by its relative cost for the Lagrangian relaxation problem of SCP(R).

The Lagrangian relaxation problem of SCP(R) for a given nonnegative $n = |H|$ dimensional Lagrangian multiplier vector $\boldsymbol{u}= (u_1, u_2, \ldots, u_n)$ is defined as follows:

$$\begin{aligned} L(\boldsymbol{u}) &= \min_{\boldsymbol{x}\in\{0,1\}^{|R|}} \sum_{r\in R} C_r x_r + \sum_{h\in H} u_h \left(1 - \sum_{r\in R} a_{hr} x_r\right) \\ &= \min_{\boldsymbol{x}\in\{0,1\}^{|R|}} \sum_{r\in R} c_r(\boldsymbol{u}) x_r + \sum_{h\in H} u_h, \end{aligned} \tag{1}$$

where $c_r(\boldsymbol{u}) = C_r - \sum_{h \in H} a_{hr} u_h$ is the relative cost associated with r. An optimal solution $\boldsymbol{x}(\boldsymbol{u})$ to problem (1) is obtained by

$$x_r(\boldsymbol{u}) = \begin{cases} 1 & \text{if } c_r(\boldsymbol{u}) < 0 \\ 0 \text{ or } 1 & \text{if } c_r(\boldsymbol{u}) = 0 \\ 0 & \text{if } c_r(\boldsymbol{u}) > 0. \end{cases}$$

It is known that $L(\boldsymbol{u})$ gives a lower bound on the optimal value of problem SCP(R). If a good Lagrangian multiplier vector $\boldsymbol{u}$ is obtained, the relative cost $c_r(\boldsymbol{u})$ gives reliable information on the attractiveness of fixing $x_r = 1$, because any r with $x_r = 1$ in an optimal solution of SCP tends to have a small $c_r(\boldsymbol{u})$ value.

We calculate the Lagrangian multiplier $\boldsymbol{u}$ for SCP(R) by a heuristic approach called the subgradient method [1, 5, 14], because computing an optimal $\boldsymbol{u}^*$ that maximizes $L(u)$ directly is usually quite expensive. We evaluate a route r by its relative cost $c_r(\boldsymbol{u})$ of the obtained Lagrangian multiplier $\boldsymbol{u}$. Let R' be the set of routes with an (a is a parameter) smallest values of $c_r(\boldsymbol{u})$ among those in R. Furthermore, for each request $h \in H$, let R''_h be the set of routes with the b (b is a parameter) smallest values of $c_r(\boldsymbol{u})$ among those in R that include h, and then let $R'' = \bigcup_{h \in H} R''_h$. We use $R' \cup R''$ for two purposes: (1) to choose the set of routes from which new routes are generated, and (2) to reduce the number of routes to be kept in R when the size of R becomes too large.

4.2.2 Neighbor Routes of a Route.

We introduce three methods to generate neighbor routes of a route r.

Insertion This operation inserts a request h into r at the best position (i.e., at the pair of positions that achieves $\delta_r^{\min}(h)$). We apply this operation for each request (which is not in r), and all feasible routes obtained by these operations are generated.

Deletion This operation deletes one request from r. We apply this operation for each request in r, and all routes obtained by these operations are generated. Note that the feasibility of the route is preserved by Property 2.1.

Swap This operation deletes one request from r and then inserts one request which is not in r at the best position. We apply this operation for all pairs of a request in r and another not in r. All feasible routes obtained by these operations are generated.

4.2.3 Neighbor Routes of Two Routes.

In addition, we use two operations to generate neighbor routes of two routes r and r'.

2-opt* method This operation is similar to the 2-opt* neighborhood operation proposed in [9]. This operation is applied to two routes r and r' that satisfy $S_r \cap S_{r'} = \emptyset$. This operation first constructs a route by concatenating a former part of r and a latter part of r' at k and k':

$$(\sigma_r(0), \sigma_r(1), \ldots, \sigma_r(k), \sigma_{r'}(k'), \sigma_{r'}(k'+1), \ldots, \sigma_r(2m_{r'}+1)).$$

We choose a random position k of r (i.e., $\sigma_r(k)$), and then choose the minimum k' such that the resulting concatenated route is feasible with respect to the time window constraint. However, the resulting route may not satisfy the coupling or other constraints, and some modification may be necessary for remedy. To satisfy the coupling constraints, for each violating customer in the route, it inserts the corresponding customer not in the route at the best position if the feasibilities of other constraints are preserved; otherwise it deletes the violating customer from the route. Repeat this process until all requests in the route satisfy the coupling constraint.

Mixing two routes Given two routes r and r', and a Lagrangian multiplier vector $\boldsymbol{u}$, this operation starts from $\sigma_{\text{mix}} := \sigma_r$ and repeats modifying the current route σ_{mix} so that the set of requests in it becomes closer to that of $\sigma_{r'}$ by inserting or deleting different requests between the two routes σ_{mix} and $\sigma_{r'}$. Similarly to $\delta_r^{\min}(h)$, we denote by $\delta_{\text{mix}}^{\min}(h)$ the minimum increase in the cost when request h is inserted into σ_{mix}. In each iteration, the operation chooses the request h that minimizes

$\delta^{\min}_{\mathrm{mix}}(h) - u_h$ (i.e., the increase in the relative cost) among those request which are in $\sigma_{r'}$ but not in the current route σ_{mix}, and inserts it at the best position of the route provided that the resulting route is feasible. If there is no such request or all inserting positions make the resulting route infeasible for all such requests, then it moves to the deletion phase described below. Let

$$\delta^{-}_{\mathrm{mix}}(h) = \begin{cases} c_{\sigma_{\mathrm{mix}}(k-1)\sigma_{\mathrm{mix}}(k+2)} - c_{\sigma_{\mathrm{mix}}(k-1),h} \\ \quad - c_{h,h+n} - c_{h+n,\sigma_{\mathrm{mix}}(k+2)}, & \text{if } k' = k+1 \\ c_{\sigma_{\mathrm{mix}}(k-1)\sigma_{\mathrm{mix}}(k+1)} + c_{\sigma_{\mathrm{mix}}(k'-1)\sigma_{\mathrm{mix}}(k'+1)} \\ \quad - c_{\sigma_{\mathrm{mix}}(k-1),h} - c_{h,\sigma_{\mathrm{mix}}(k+1)} \\ \quad - c_{\sigma_{\mathrm{mix}}(k'-1),h+n} - c_{h+n,\sigma_{\mathrm{mix}}(k'+1)}, & \text{otherwise.} \end{cases}$$

where $\sigma_{\mathrm{mix}}(k) = h$ and $\sigma_{\mathrm{mix}}(k') = h + n$. In the deletion phase, it chooses the request h with the minimum $\delta^{-}_{\mathrm{mix}}(h) + u_h$ (i.e., the increase of the relative cost) among those not in $\sigma_{r'}$ but in σ_{mix}, and removes it from σ_{mix}. Letting σ_{mix} be the new route obtained either by the insertion or the deletion, we go back to the insertion phase again. All routes obtained during the modifications are considered as candidates to be added into R.

5 Computational Experiment

We conducted computational experiments to evaluate the proposed algorithm. The algorithm was coded in C language and run on a handmade PC (Intel Pentium4, 2.8 GHz, 1 GB memory).

5.1 Efficiency of Using Lagrangian Multiplier

In the reconstruction phase of route generation method, relative cost is used to choose a set of routes to be used in generating new routes. To confirm the efficiency of this method, we tested two other methods for selecting a set of routes in the reconstruction phase. For comparison purpose, we solved SCP(R) with the algorithm by Yagiura et al. [14] (denoted YKI) whenever algorithm Reconstruction outputs R, and observe the quality of the solution. First method selects the set of routes appearing in the solution of SCP(R). Second method selects a set of routes randomly from the current R. We conducted the comparison of these two methods with the method in Section 4.2 that uses the relative cost.

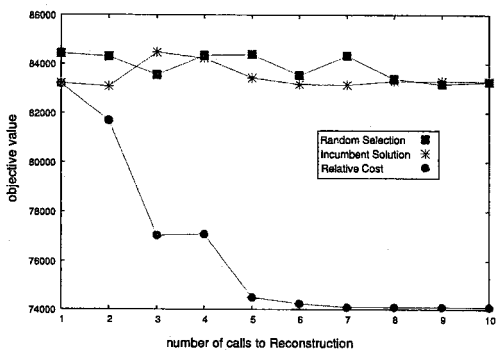

Figure 1: Three selection methods for type-C

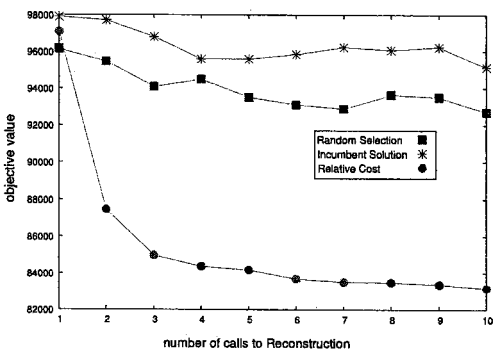

Figure 2: Three selection methods for type-R

Figures 1 and 2 show the objective values of the solutions of SCP(R) obtained by YKI against the number of calls to algorithm Reconstruction. Figure 1 shows the result on an instance whose customers are distributed as clusters (type-C), and Figure 2 shows the result on an instance whose customers are distributed randomly (type-R). In both figures, the results of "Relative Cost" are better than the others. We therefore use the method based on the relative cost in the algorithm Reconstruction.

Table 1: Comparison for GC1–GC6

	Resource		Capacity				Ours		LS	
INST	ρ	π	$Q1$	$Q2$	TW	LIFO	CNV	CDIST	CNV	CDIST
GC1	1	0	200	1000	–	0	208	65624.54	224	72422.65
GC2	3	1	200	1000	–	1	278	95016.41	313	92170.04
GC3	1	0	200	1000	−4%	0	142	48421.68	155	56234.36
GC4	1	1	200	1000	$[0, \infty)$	0	234	79763.98	212	59545.98
GC5	1	1	200	1000	$[0, \infty)$	1	238	84378.57	212	55065.95
GC6	2	0	200	200	–	0	271	84785.49	276	82716.75

5.2 Comparison of Our Algorithm with LS Type Algorithm

Next, we conducted experiments to confirm the flexibility and performance of our algorithm for PDP-GCER. We compared our algorithm with a metaheuristic algorithm coded in reference to the algorithm proposed for PDPTW by Li and Lim [6]. It is based on the simulated annealing and tabu search procedure based on the same objective function as ours; that is, the primary objective is to reduce the number of vehicles and the secondary objective is to minimize the total traveling cost. We modify it so that it can deal with PDP-GCER. The modified algorithm executes the local search in a feasible region under the constraints of PDP-GCER.

We generated the PDP-GCER instances consisting of six groups GC1–GC6 from Li and Lim's PDPTW instances [6] by adding various constraints to them. Li and Lim's instances are categorized into the type-C1, C2, R1, R2, RC1, RC2. The types C, R and RC represent the distribution of customers. The types 1 and 2 represent the severeness of the time window and the capacity constraints of the instances; the type 1 instances have severer constraints than the type 2 instances (hence more vehicles are needed). We chose three instances from those of Li and Lim for each type, and generated new instances from them; hence each of GC1–GC6 contains 18 instances. In Table 1, columns "ρ" and "π" represent the number of renewable and nonrenewable resources. Column "$Q1$" (resp., "$Q2$") represent the vehicle capacities of type 1 (resp., type 2) instances; that is, we set $Q_p^{\mathrm{re}} := Q1$ and $Q_{p'}^{\mathrm{non}} := Q1$ (resp., $Q_p^{\mathrm{re}} := Q2$, $Q_{p'}^{\mathrm{non}} := Q2$) for all type 1 (resp., type 2) instances. Column "TW" shows the information about the time window constraint. In GC4 and GC5, we set the all time windows (i.e., that of all customers and the depot) to $[0, \infty)$. That is, the instances are equivalent to those with no time window. On the other hand, in GC3, we cut 4% from the original time windows by setting $[e_i, l_i]$ to $[e'_i, l'_i]$ such that $e'_i = e_i + 0.02(l_i - e_i)$ and $l'_i = l_i - 0.02(l_i - e_i)$. For the rest (i.e., GC1, GC2 and GC6), we adopted the time windows of the original instances. We imposed the LIFO constraint to GC2 and GC5 as shown in the LIFO column by 1. The time limit of constructing routes is set to 2400 seconds and the time limit of solving the set covering problem is set to 1200 seconds. We set the time limit to 3600 seconds for the metaheuristic algorithm.

Table 1 compares the results of our algorithm and those of metaheuristic algorithm. In Table 1, column "LS" represents the results of metaheuristic algorithm, column "CNV" means the cumulative number of vehicles and column "CDIST" means the cumulative traveling cost. The results show that for GC1, GC2, GC3 and GC6 whose instances have multiple constraints or severe constraints, our algorithm works efficiently, but for GC4 and GC5 whose instances have weaker constraints, the metaheuristic algorithm works better than ours. These results confirm our consideration that our algorithm works well on the instances with tighter constraints, because the number of feasible routes is limited in such cases.

6 Conclusion

We generalized the pickup and delivery problem with time windows by allowing general constraints. Our generalization can treat any monotone constraints imposed on each route. In our algorithm, we generate a set of feasible routes and apply the set covering approach. We construct an initial set of routes by an insertion method and reconstruct the set of routes repeatedly by modifying the routes using various types of neighborhood operations while reducing the candidate routes by utilizing the Lagrangian relative costs. The computational results indicated that our algorithm works more efficiently than a

metaheuristic algorithm, if the instances which have tighter constraints. We also confirmed the flexibility of our algorithm by applying it to instances with various constraints.

References

[1] E. Balas and A. Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. *Mathematical Programming Study*, 12:37–60, 1980.

[2] R. Bent and P. V. Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers and Operations Research*, 33:875–893, 2006.

[3] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54:7–22, 1991.

[4] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[5] M. L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.

[6] H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools*, 12(2):173–186, 2003.

[7] W. P. Nanry and J. W. Barnes. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B*, 34:107–121, 2000.

[8] G. Pankratz. A grouping genetic algorithm for the pickup and delivery problem with time windows. *OR Spectrum*, 27:21–41, 2005.

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

[10] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Technical report, Department of Computer Science, University of Copenhagen, 2004.

[11] M. W. P. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

[12] M. Savelsbergh and M. Sol. Drive: Dynamic routing of independent vehicles. *Operations Research*, 46(4):474–490, 1998.

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

[14] M. Yagiura, M. Kishida, and T. Ibaraki. A 3-flip neighborhood local search for the set covering problem. *European Journal of Operational Research*, 172:472–499, 2006.