Guided Ejection Search for the Pickup and Delivery Problem with Time Windows

Yuichi Nagata and Shigenobu Kobayashi

Interdisciplinary Graduate School of Science and Engineering,
Tokyo Institute of Technology,
4259 Nagatsuta Midori-ku Yokohama, Kanagawa 226-8502, Japan
nagata@fe.dis.titech.ac.jp, kobayasi@dis.titech.ac.jp

Abstract. This paper presents an efficient route minimization heuristic for the pickup and delivery problem with time windows (PDPTW) based on guided ejection search (GES). GES is a recently proposed metaheuristc framework and was first applied to the vehicle routing problem with time windows. The existence of the pickup and delivery constraint makes the feasible solution space tightly constrained and then makes the design of effective metaheuristics more difficult. We demonstrate that GES can be successfully applied to such an complicated problem. Experimental results on the Li and Lim's benchmarks demonstrate that the proposed GES algorithm outperforms existing algorithms and is able to improve 105 best-known solutions out of 298 instances.

Keywords: vehicle routing, pickup and delivery problem, ejection chain, guided local search, metaheuristics.

1 Introduction

The pickup and delivery problem with time windows (PDPTW) can be described as the problem of designing least cost routing plan to satisfy a set of transportation requests by a given identical vehicle fleet. Each request consists of delivering goods from a predefined location (pickup customer) to another one (delivery customer). The routing plan must be designed in such a way that all vehicles start and end at the depot, the amount of goods must not exceed the capacity of the vehicle (capacity constraint), each customer must be serviced within a given time interval (time window constraint), and for each request the corresponding pickup customer must be visited before the corresponding delivery customer by the same vehicle (pickup and delivery constraint). In standard benchmarks, a hierarchical objective of minimizing the number of routes (primary objective) and the total travel distance (secondary objective) is frequently used.

Given the hierarchical objective, recent metaheuristics for the PDPTW use a two-stage approach where the number of routes is minimized in the first stage and the total travel distance is then minimized in the second stage [3][1][11]. The two-stage approach allows us to independently develop algorithms for the route minimization and for the distance minimization. In this paper, we focus

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 202–213, 2010.

on the route minimization for the PDPTW because efficient algorithms for the route minimization and for the distance minimization would be different from each other.

In early work on the metaheuristics for the PDPTW, tabu search and simulated annealing algorithms were proposed for minimizing the number of routes [9][1] where new neighborhood structures suited for the PDPTW were proposed. The current state-of-the-art algorithm by Ropke and Pisinger [11] is based on large neighborhood search (LNS) and it has shown good results for both the route and distance minimization. In the LNS algorithm [11], a move is defined by a fairly large change in a solution where up to 30–40 % of all requests are removed and re-inserted in a single move. For an extensive review of the metaheuristics as well as exact methods for the PDPTW, the reader is referred to Parragh *et al.* [10].

Recently, we proposed a powerful route minimization heuristic for the vehicle routing problem with time windows (VRPTW) [7]. This heuristic has dominated all of the previous route minimization heuristics applied to the VRPTW. Then we proposed a generalized metaheuristc framework based on this heuristic, which is called guided ejection search (GES), for solving a wide variety of combinational optimization problems and GES was successfully applied to the job shop scheduling problem [8]. In this paper we develop a route minimization heuristic for the PDPTW based on GES. The existence of the pickup and delivery constraint makes the feasible solution space tightly constrained and the design of an effective algorithm more difficult compared with the VRPTW case. Therefore, it is worthwhile to develop an effective GES algorithm for the PDPTW in order to investigate the effectiveness and robustness of GES to such an complicated problem.

The GES algorithm was tested on the standard benchmark problems of Li and Lim [4]. Computational results showed that the proposed algorithm is robust and highly competitive. It improved 105 best-known solutions out of 298 benchmark instances. The remainder of this paper is arranged as follows. First, the problem definition and notation are described in Section 2. The problem solving methodology based on GES is described in Section 3. The computational analysis of GES and its comparison with other algorithms are presented in Section 4. Conclusions are presented in Section 5.

2 Problem Definition and Notation

The PDPTW is defined on a complete directed graph $G = (V, E)$ with a set of vertices $V = \{0, 1, \ldots, N\}$ and a set of edges $E = \{(v, w)|v, w \in V(v \neq w)\}$. Node 0 represents the depot and the set of nodes $\{1, \ldots, N\}$ represents the customers. Let $H = \{0, 1, \ldots, N/2\}$ be a set of the requests, and for each request $h \in H$, let p_h and d_h be the corresponding pickup and delivery customers, respectively. Let $P = \{p_h|h = 1, \ldots, N/2\}$ be a set of the pickup customers and $D = \{d_h|h = 1, \ldots, N/2\}$ a set of the delivery customers. Here, we assume that $P \cap D = \emptyset$ and $P \cup D = V \backslash \{0\}$. With each node v $(\in V)$ are associated a

demand q_v (with $q_0 = 0$), a non-negative service time s_v (with $s_0 = 0$), and a time window $[e_v, l_v]$. For each request h, the demand of the pickup customer, q_{p_h}, must be positive and the demand of the delivery customer is defined by $q_{d_h} = -q_{p_h}$. Each edge (v, w) has the non-negative travel distance d_{vw} and travel time c_{vw}. The capacity of the identical vehicles is given by Q.

Given a route r, let $< v_0, v_1, \ldots, v_n, v_{n+1} >$ be a sequence of the customers in this route where v_0 and v_{n+1} represent the depot and n refers to the number of customers in this route. The travel distance of the route is defined by $\sum_{i=0}^{n} d_{v_i v_{i+1}}$. A route is called feasible if the time window, capacity, and pickup and delivery constraints are all satisfied. These constraints are defined as follows. The earliest departure time at the depot, a_{v_0}, the earliest start time of service at a customer v_i, a_{v_i} $(i = 1, \ldots, n)$, and the earliest arrival time to the depot, $a_{v_{n+1}}$, are defined recursively as follows:

$$\begin{aligned} &a_{v_0} = e_0, \\ &a_{v_i} = \max\{a_{v_{i-1}} + s_{v_{i-1}} + c_{v_{i-1}v_i},\ e_{v_i}\}\ (i = 1, \ldots, n+1). \end{aligned} \tag{1}$$

The route satisfies the time window constraint if

$$a_{v_i} \leq l_{v_i}\ (i = 0, \ldots, n+1). \tag{2}$$

Let $Q_{v_i} (= \sum_{s=1}^{i} q_{v_s})$ $(i = 0, \ldots, n+1)$ denote the amount of goods in a vehicle at a location v_i. The route satisfies the capacity constraint if

$$Q_{v_i} \leq Q\ (i = 0, \ldots, n+1). \tag{3}$$

The route satisfies the pickup and delivery constraint if each pickup (delivery) customer in the route is visited before (after) the corresponding delivery (pickup) customer (the two customers must belong to the same route).

A feasible solution σ is defined as a set of feasible routes such that all customer are visited exactly once. In addition, we define a *partial* solution as a set of routes that pass through a subset of all customers exactly once. A partial solution is called feasible if it consists of feasible routes, and infeasible otherwise.

In standard benchmarks, the objective of the PDPTW consists of finding a feasible solution σ that minimizes the number of routes m (primary objective) and, in case of ties, minimizes the total distance traveled (secondary objective).

3 Problem Solving Methodology

This section first presents the GES algorithm for the PDPTW followed by the description of the difference from the previous work. Core parts of the GES algorithm are then described in more detail.

3.1 The GES Framework for the PDPTW

The GES algorithm (Procedure DELETE-ROUTE(σ)) is shown in Algorithm 1. The route minimization procedure starts with an initial solution where each

request (a pair of pickup and delivery customers) is served individually by a separate route. Procedure DELETE-ROUTE(σ) is then repeatedly applied to an incumbent solution σ to reduce the number of routes one by one until the termination condition is met.

Procedure DELETE-ROUTE(σ) is started by selecting and removing a route randomly from the current solution σ (line 1). Thus, σ is a feasible partial solution. An ejection pool (*EP*) [5] is then initialized with the set of the requests in the removed route (line 2). Here, the EP is a list that stores a set of temporarily unserved requests, currently missing from σ.

In each iteration (lines 5-14), a request h_{in} is selected from the EP with the last-in first-out (LIFO) strategy and is removed from the EP (line 5). The selected request h_{in} is then inserted into σ without violating the capacity, time window, and pickup and delivery constraints, if possible. More formally, the insertion of h_{in} is determined as follows. Let $\mathcal{N}^{fe}_{insert}(h_{in}, \sigma)$ be the set of feasible partial solutions that are obtained by inserting $p_{h_{in}}$ and $d_{h_{in}}$ into σ in all possible ways. If there is a possible feasible insertion (*i.e.*, $\mathcal{N}^{fe}_{insert}(h_{in}, \sigma) \neq \emptyset$), the insertion is executed by randomly selecting a solution from $\mathcal{N}^{fe}_{insert}(h_{in}, \sigma)$ and update the incumbent solution (lines 6-7).

If the request h_{in} cannot be inserted into σ, we remove (eject) requests from a route in σ in order to make room to insert h_{in} without violating the constraints. Let $\mathcal{N}^{fe}_{EJ}(h_{in}, \sigma)$ be the set of feasible partial solutions that are obtained by inserting h_{in} into σ in all possible ways, and for each insertion, ejecting at most k_{max} requests from the resulting infeasible route in all possible ways. Note that we allow ejection of h_{in} itself and $\mathcal{N}^{fe}_{EJ}(h_{in}, \sigma)$ always includes σ itself. The insertion positions for $p_{h_{in}}$ and $d_{h_{in}}$ and the ejecting requests, denoted by $\{h^{(1)}_{out}, \ldots, h^{(k)}_{out}\}$ ($k \leq k_{max}$), are determined by selecting a solution from $\mathcal{N}^{fe}_{EJ}(h_{in}, \sigma)$ so that the sum of penalty counters of the ejecting requests, $P_{sum} = p[h^{(1)}_{out}] + \ldots + p[h^{(k)}_{out}]$, is minimized (line 11). Here, the penalty counter $p[h]$ ($h \in \{1, \ldots, N/2\}$) refers to how many times an attempt to insert request h has failed during the current DELETE-ROUTE procedure (lines 3 and 10). This criterion is motivated to usually eject fewer requests at a time (*e.g.*, $k = 1$). In addition, this criterion is also motivated to avoid *cycling*. If a few requests conflict with each other due to the constraints, cycling will occur (*i.e.*, insertions and ejections of requests are repeated within a few conflicting requests) and the penalty counter of the conflicting requests will increase. Therefore, GES can escape from a cycling by ejecting a relatively large number of requests at a time that would not be so hard for the subsequent re-insertion when *cycling* occurs.

Each time after one or more requests are ejected from σ, the resulting feasible partial solution σ is perturbed by procedure PERTURB(σ) to diversify the search (line 13). Here, random local search moves are iterated inside the partial solution subject to the constraint that σ is a feasible partial solution. The detailed procedure is presented in Section 3.2.

The basic GES framework for the PDPTW is almost the same as that for the VRPTW. The main difference is that we employ the request as the fundamental component for the insertion and ejection whereas the customer was employed in

Algorithm 1. Procedure DELETE-ROUTE(σ)

1: Select and remove a route randomly from σ;
2: Initialize EP with the requests in the removed route;
3: Initialize all penalty counters $p[h] := 1$ $(h = 1, \ldots, N/2)$;
4: **while** $EP \neq \emptyset$ or termination condition is not met **do**
5: Select and remove request h_{in} from EP with the LIFO strategy;
6: **if** $\mathcal{N}^{fe}_{insert}(h_{in}, \sigma) \neq \emptyset$ **then**
7: Select $\sigma' \in \mathcal{N}^{fe}_{insert}(h_{in}, \sigma)$ randomly; Update $\sigma := \sigma'$;
8: **end if**
9: **if** h_{in} cannot be inserted in σ **then**
10: Set $p[h_{in}] := p[h_{in}] + 1$;
11: Select $\sigma' \in \mathcal{N}^{fe}_{EJ}(h_{in}, \sigma)$ such that $P_{sum} = p[h^{(1)}_{out}] + \ldots + p[h^{(k)}_{out}]$ is minimized; Update $\sigma := \sigma'$;
12: Add the ejected requests $\{h^{(1)}_{out}, \ldots, h^{(k)}_{out}\}$ to EP;
13: $\sigma :=$ PERTURB(σ);
14: **end if**
15: **end while**
16: **if** $EP \neq \emptyset$ **then** Restore σ to the input state;
17: **return** σ;

the previous work for the VRPTW. This choice is natural because the pickup and delivery constraint is imposed in the PDPTW. In the GES algorithm, the procedure for finding the best insertion-ejection combination that minimize P_{sum} (line 11) requires an efficient algorithm. As for the VRPTW, all possible insertion positions for a customer to be inserted, and for each insertion all possible ejections consisting of at most k_{max} customers were considered for finding the best inserting-ejection combination. Here, the parameter k_{max} was introduced in order to reduce the number of all possible insertion-ejection combinations. However, the number of all possible insertion-ejection combinations is usually very large (even if k_{max} is two) and it is therefore impractical to test all of them. So, we developed an efficient algorithm for finding the best insertion-ejection combination in the VRPTW case [7]. As described above, we employ the request as the fundamental component in the GES algorithm for the PDPTW. However, it makes more difficult to find the best insertion-ejection combination because the number of all possible insertion positions (for $p_{h_{in}}$ and $d_{h_{in}}$) is much greater than that in the VRPTW case. In Section 3.3, we present an efficient algorithm for finding the best insertion-ejection combination in the PDPTW case.

Another important feature of GES is the perturbation procedure (procedure PERTURB(σ)). As we will show in the experiments, this procedure significantly affects the performance. Ropke and Pisinger [11] suggested that very large moves (*e.g.*, up to 30–40 % of all requests are rearranged in a single iteration) are required to move a solution from one promising area of solution space to another, when faced with tightly constrained problems such as the PDPTW, even when embedded in metaheuristics. We believe that the perturbation procedure can efficiently move an incumbent solution σ to other promising area by a certain

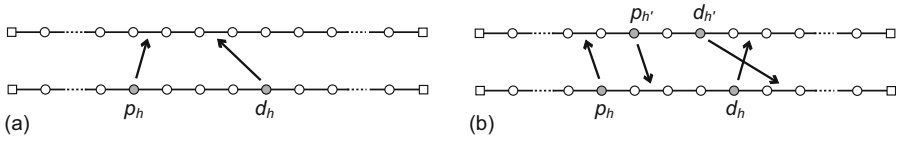

Fig. 1. The neighborhoods. (a) The pair relocation neighborhood is defined as a set of feasible solutions (feasible partial solutions in GES) that are obtained from the current solution by ejecting a request and re-inserting it in other possible ways (the request can be inserted into the same route). (b) The extended pair exchange neighborhood is defined as a set of feasible (partial) solutions that are obtained from the current solution by ejecting two requests from two different routes and inserting them into the two routes in all possible ways (each request must be inserted into another route).

number of iterations of small moves because an incumbent solution σ would not be so tightly constrained when requests are temporarily ejected. In this paper, we first employ the simple pair relocation neighborhood (See Fig. 1), which has been frequently used in previous metaheuristic algorithms for the PDPTW [9][1]. However, we have confirmed that possible feasible moves by the pair relocation neighborhood is still limited and the incumbent solution σ cannot be fully moved even though some requests are temporarily ejected. So, we introduce an extended pair exchange neighborhood (See Fig. 1) to enhance the perturbation procedure. To the best of our knowledge, this neighborhood has not been used in the previous works.

3.2 The Perturbation Procedure

Let $\mathcal{N}^{fe}_{relocate}(h, \sigma)$ be the set of feasible partial solutions defined by the relocation moves of request h. Let $\mathcal{N}^{fe}_{exchange}(h, \sigma)$ be the set of feasible partial solutions defined by the extended exchange moves of request h (request h and other one are exchanged). In the perturbation procedure, the input solution σ is perturbed by the iteration of random moves for a given number of times, I_{rand}. At each iteration, a request h currently served in σ is randomly selected and a move is executed by randomly selecting a solution from $\mathcal{N}^{fe}_{relocate}(h, \sigma)$ or $\mathcal{N}^{fe}_{exchange}(h, \sigma)$, which are selected with a given probability. Here, the probability of selecting the extended pair exchange neighborhood is given by p_{ex}. Note that for a selected h if no feasible move is found in the selected neighborhood, no move is executed.

3.3 Algorithm for Finding the Best Insertion-Ejection Combination

Given a request h_{in} to be inserted, the procedure for finding the best insertion-ejection combination (insertion positions for $p_{h_{in}}$ and $d_{h_{in}}$ and ejecting requests) that minimizes P_{sum} proceeds as follows. All insertion positions in σ are considered for $p_{h_{in}}$. For each insertion of $p_{h_{in}}$, a sub procedure is performed where all possible insertion positions for $d_{h_{in}}$ (inserting $d_{h_{in}}$ between $p_{h_{in}}$ and the end of the route (depot)) and ejections of at most k_{max} requests from the resulting infeasible routes are tested; check the feasibility of the resulting routes and record

the insertion positions for $p_{h_{in}}$ and $d_{h_{in}}$ and the ejected requests if a resulting route is feasible and P_{sum} is less than P_{best} (the minimum value of P_{sum} found so far).

We propose an efficient algorithm for the sub procedure. For each insertion of $p_{h_{in}}$, let $< 0, 1, \ldots, n, n+1 >$ be a sequence of the customers in the resulting infeasible route (we call it *original* route) where the customers are labeled according to the order in which they appear in the original route to simplify the notations (0 and $n+1$ refer to the depot). Although for any insertion-ejection combination, the feasibility of the resulting route can be checked in $O(n)$ time by a naive calculation, the feasibility check can be possible in constant time by testing insertion-ejection combinations in a particular order. Let us represent the possible ejections of requests as the corresponding customers. The combinations of all possible insertion positions for $d_{h_{in}}$ and ejections of at most k_{max} requests ($2k_{max}$ customers) can be denoted as $\{i(1), \ldots, i(j_d), \ldots, i(j)\}$ ($1 \leq i(1) < \ldots < i(j_d) < \ldots < i(j) \leq n$), $1 \leq j_d \leq j \leq 2k_{max}+1$, $1 \leq j_d \leq 2k_{max}+1$) (See Fig. 2). Here, the insertion position for $d_{h_{in}}$ is denoted as $i(j_d)$, meaning that $d_{h_{in}}$ is inserted just after $i(j_d)$ in the original route. The requests to be ejected are denoted as a set of the corresponding customers $\{i(1), \ldots, i(j_d-1), i(j_d+1) \ldots, i(j)\}$ where j refers to the number of temporarily ejected customers plus one. The basic idea is to search the possible insertion-ejection combinations denoted as $\{i(1), \ldots, i(j_d), \ldots, i(j)\}$ in the lexicographic order (we call it *lexicographic search*) (See Fig. 2), making it possible to check the feasibility of the resulting routes in $O(1)$ time for each route. Note that the lexicographic search is performed for each j_d ($1 \leq j_d \leq 2k_{max}+1$).

For each j_d ($1 \leq j_d \leq 2k_{max}+1$), the lexicographic search is performed as follows. Before starting the lexicographic search, we calculate the latest possible arrival times [6], z_i ($i = 1, \ldots, n+1$), for the original route (*i.e.*, if the vehicle arrives at customer i no later than time z_i, the time window constraints of i and the subsequent customers in the route are satisfied). Let a_i^{new} and Q_i^{new} denote, respectively, a_i and Q_i at location i in a route obtained through the lexicographic search. These values are dynamically updated. For example, when $i(j)$ is ejected (See Fig. 3 (a)), $a_{i(j)+1}^{new}$ can be updated in $O(1)$ time according to Eq. (1) because a_p^{new} (p is the predecessor of $i(j)$ in the current route) has already been updated. In the same way, when $i(j)$ is incremented (See Fig. 3 (b)), $a_{i(j)-1}^{new}$ can be updated in $O(1)$ time. In the same way, $a_{i(j_d)+1}^{new}$ (or $a_{i(j_d)}^{new}$) can be updated in $O(1)$ time when $d_{h_{in}}$ is inserted just after $i(j_d)$ (or $i(j_d)$ is incremented) (See Fig. 3). Q_i^{new} can also be updated in the same manner. When customer $i(j)$ is ejected, the resulting route is feasible in terms of the time window and capacity constraints if both $a_{i(j)+1}^{new} \leq z_{i(j)+1}$ and $Q_{i(j)+1}^{new} \leq Q$ hold, and the pickup and delivery constraint can be easily checked. Therefore, the feasibility of the resulting route can be checked in $O(1)$ time. Similarly, when $p_{h_{in}}$ is inserted just after $i(j_d)$, the feasibility of the resulting route can be checked in $O(1)$ time.

In fact, most of the lexicographic search can be pruned if one of the following conditions (i)–(v) is met and the efficiency is greatly improved.

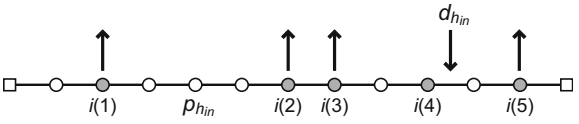

Fig. 2. An example of an insertion-ejection combination: $\{i(1), i(2), i(3), i(4), i(5)\} = \{2, 6, 7, 9, 11\}$ $(j_d = 4, j = 5)$. If $k_{max} = 2$, the lexicographic search proceeds in the following order: $\{1\}$, $\{1, 2\}$, $\{1, 2, 3\}$, $\{1, 2, 3, 4\}$, $\{1, 2, 3, 4, 5\}$, $\{1, 2, 3, 4, 6\}$ ..., $\{1, 2, 3, 4, n\}$, $\{1, 3\}$, $\{1, 3, 4\}$, $\{1, 3, 4, 5\}$, $\{1, 3, 4, 5, 6\}$, $\{1, 3, 4, 5, 7\}$, ..., $\{1, n-3, n-2, n-1, n\}$, $\{2\}$, $\{2, 3\}$, ...

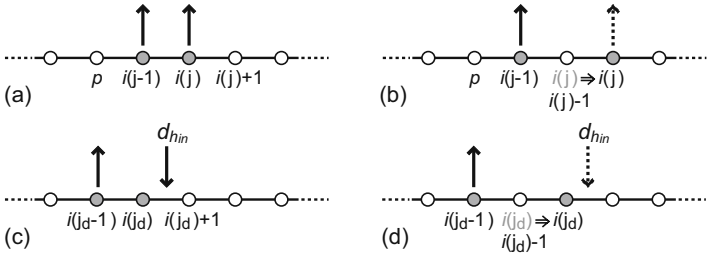

Fig. 3. Examples for the lexicographic search: (a) $i(j)$ is ejected ($i(j-1)$ is already ejected), (b) $i(j)$ is incremented ($i(j)$ is not yet ejected), (c) $d_{h_{in}}$ is inserted just after $i(j_d)$, (d) $i(j_d)$ is incremented ($d_{h_{in}}$ is not yet inserted).

- After $i(j)$ $(j \neq j_d)$ is ejected,
  - (i) P_{sum} (it is updated each time a pickup customer is ejected) $\geq P_{best}$
  - (ii) $i(j) \in P$ and the number of temporarily ejected pickup customers is greater than k_{max}
  - (iii) $i(j) \in D$ and the paired pickup customer is not temporarily ejected
- After $i(j)$ $(j \neq j_d)$ is incremented,
  - (iv) $l_{i(j)-1} < a^{new}_{i(j)-1}$ or $Q < Q^{new}_{i(j)-1}$
  - (v) $i(j) - 1 \in D$ and the paired pickup customer is temporarily ejected

In the cases (i)-(iii), the lower-level lexicographic search ($\{i(1), \ldots, i(j), *, *, \ldots\}$) can be pruned because (i) these insertion-ejection combinations never improve P_{best}, or (ii)(iii) the pickup and delivery constraint is never satisfied. In the cases (iv) and (v), the lower-level lexicographic search ($\{i(1), \ldots, i(j-1), v, *, *, \ldots\}$ $(v > i(j)-1)$) can be pruned because (iv) the time window or capacity constraint is violated already at customer $i(j)-1$, or (v) the pickup and delivery constrained is never satisfied. In addition, the lexicographic search can be further pruned in the similar manner when $d_{h_{in}}$ is inserted just after $i(j_d)$ or $i(j_d)$ is incremented (conditions are easily derived).

4 Experimental Results

The proposed GES algorithm was implemented in C++ and was executed on an AMD Opteron 2.6GHz (4GB memory) computer. Several computational

experiments have been conducted to analyze the behavior of the proposed GES algorithm. We also present results along with a comparison to the state-of-the-art metaheuristics for the PDPTW.

4.1 Benchmark Set

The GES algorithm was tested on the benchmark set constructed by Li and Lim [4]. The data set and the best-known solutions are available at `http://www.sintef.no/projectweb/top`. We used this benchmark set because recent state-of-the-art metaheuristics for the PDPTW are compared on this benchmark set. The benchmark set consists of five sets of 200, 400, 600, 800 and 1000 customers, with 60 instances in each set [1], resulting in 298 instances. For each size, the 60 instances are divided into six groups: R1, R2, RC1, RC2, C1, and C2, each consisting of 10 instances. The C1 and C2 classes have customers located in clusters and in the R1 and R2 classes the customers are at random positions. The RC1 and RC2 classes contain a mix of both random and clustered customers. The C2, R2 and RC2 classes (Class 2) have longer scheduling horizons and larger capacities than the C1, R1 and RC1 classes (Class 1), meaning that each vehicle can service a larger number of customers in the Class 2 instances.

4.2 Analysis of GES

First, we focus on the perturbation procedure in order to analyze the importance of this procedure where several parameter values of I_{rand} and p_{ex} are tested. Next, we investigate the effect of parameter k_{max} because this parameter would have a great impact on the behavior of GES. We test 12 different GES configurations, which are depicted in Table 1, where the following parameter values are combined: $k_{max} = 0, 1, 2, 3$, $I_{rand} = 0, 10, 100, 1000$, and $p_{ex} = 0, 0.5$. The GES algorithm with each configuration was applied to the benchmarks five times with the time limit of 600 seconds for each run.

Due to space limitation, Table 1 reports the results for only the 200 and 600-customer instances where the results are averaged over Class 1 and Class 2 instances (each consisting of 30 instances), respectively. For each problem class, the number of vehicles (m), the number of iterations to reach the best solution in each run ($iter$), and the computation time in seconds to reach the best solution in each run ($time$) are reported. Here, an iteration is defined as the sequence of procedures to insert a request h_{in} (lines 5-14) in Algorithm 1.

First, let us focus on the first setting ($k_{max} = 2$, $p_{ex} = 0$, $I_{rand} = 0, 10, 100$, 1000). We can see that m tends to improve (decrease) with increasing the value of I_{rand} from 0 to 100, but the improvement may be saturated around $I_{rand} = 10$ on 200-customer instances. Moreover, $iter$ also tends to decrease with increasing the value of I_{rand} from 0 to 100 even though $iter$, in general, tends to increase to reach higher quality solutions. However, the computational effort for the perturbation procedure is not negligible when I_{rand} is large. Therefore, the solution

[1] Two instances in the 1000-customer benchmark set are not available.

Table 1. The results of GES with the different configurations on 200 and 600-customer instances. The values of m calculated from the best-known solution are listed in the first column.

200-customer	$k_{max}=2$, $p_{ex}=0$				$k_{max}=2$, $p_{ex}=0.5$				$I_{rand}=100$, $p_{ex}=0.5$			
	I_{rand}	m	*iter*	*time*	I_{rand}	m	*iter*	*time*	k_{max}	m	*iter*	*time*
Class1 (15.60)	0	15.81	52418	20.0	0	15.77	34279	12.6	0	15.68	16763	58.3
	10	15.49	14610	6.3	10	15.47	5190	3.6	1	15.49	3357	12.7
	100	15.49	7721	7.1	100	15.47	1418	4.7	2	15.47	1418	4.7
	1000	15.50	1956	10.5	1000	15.47	840	22.6	3	15.46	1480	5.0
Class2 (4.60)	0	4.99	4954	9.0	0	4.97	8160	9.3	0	4.70	6449	45.6
	10	4.58	5865	12.9	10	4.57	5135	18.4	1	4.59	2029	17.2
	100	4.58	4142	14.7	100	4.57	2262	21.7	2	4.57	2262	21.7
	1000	4.57	1913	14.8	1000	4.61	751	36.5	3	4.58	2474	24.9
600-customer	$k_{max}=2$, $p_{ex}=0$				$k_{max}=2$, $p_{ex}=0.5$				$I_{rand}=100$, $p_{ex}=0.5$			
	I_{rand}	m	*iter*	*time*	I_{rand}	m	*iter*	*time*	k_{max}	m	*iter*	*time*
Class1 (43.1)	0	44.39	63750	86.0	0	44.36	61457	83.0	0	43.66	20530	200.8
	10	42.77	61603	106.7	10	42.52	38748	102.0	1	42.54	12463	121.1
	100	42.69	32858	118.7	100	42.45	12950	124.6	2	42.45	12950	124.6
	1000	42.90	8697	161.6	1000	42.76	3555	249.0	3	42.50	12251	110.8
Class2 (12.36)	0	14.43	16461	69.6	0	14.43	15928	62.5	0	12.42	6344	144.9
	10	12.45	13896	131.5	10	12.40	10741	115.1	1	12.33	5383	131.2
	100	12.38	7883	82.8	100	12.27	5026	122.2	2	12.27	5026	122.2
	1000	12.33	3826	101.1	1000	12.46	1898	230.0	3	12.29	4974	133.8

quality with $I_{rand}=1000$ sometimes inferior to those with $I_{rand}=10$ and 100 because of the termination condition given by the fixed time limit (600 seconds).

Next, we focus on the second setting ($k_{max}=2$, $p_{ex}=0.5$, $I_{rand}=0, 10, 100, 1000$). Here, the extended pair exchange neighborhood and pair relocation neighborhood are randomly selected in the perturbation procedure while only the pair relocation neighborhood is used in the first setting. The results show the same tendency as those in the first setting. But it should be noted that both of m and *iter* tend to be better than those in the first setting, indicating that the introduction of the extended pair exchange neighborhood enhances the ability of the perturbation procedure. However, *time* was not improved (compared with the first setting) even though *iter* was decreased because the computational effort for applying the extended pair exchange neighborhood is fairly greater than that for the pair relocation neighborhood. We can conclude from the results of the two settings that the more the incumbent solution is perturbed in the perturbation procedure, the better the solution quality is. If the extended pair exchange neighborhood can be applied in a more deliberate way, the performance of the proposed GES algorithm will be improved.

Finally, let us focus on the third setting ($I_{rand}=100$, $p_{ex}=0.5$, and $k_{max}=0, 1, 2, 3$). The solution quality (m) tends to improve with increasing the value of k_{max}, but the improvement is saturated at $k_{max}=2$ whereas $k_{max}=5$ seemed to be a good value in the VRPTW case.

4.3 Comparisons with Other Algorithms

We compare the GES algorithm with state-of-the-art route minimization heuristics for the PDPTW. According to the analysis conducted in the previous section,

Table 2. Comparisons with state-of-the art algorithms

	Best	BH (5 runs)		RP (10 runs)		GES (5 runs)			
N	known	Best	Time	Best	Time	Best	Average	Time	#new
200	606	614	300	606	(264)	601	601.2	600	5
400	1157	1188	600	1158	(881)	1139	1140.0	600	18
600	1664	1718	600	1679	(2221)	1636	1641.8	600	27
800	2175	2245	900	2208	(3918)	2135	2147.4	600	30
1000	2644	2759	900	2652	(5370)	2613	2624.8	600	25
total	8266	8524		8303		8124	8155.2		105
Computer		Athlon 1.2GHz		Penti.IV 1.5GHz		Opteron 2.6GHz			

the parameters of the GES algorithm were set as follows: $k_{max} = 2$, $p_{ex} = 0.5$, and $I_{rand} = 100$. The GES algorithm was applied five times to all of the benchmark instances where the time limit was set to 600 seconds for each run. We selected two algorithms for comparisons that have shown the best results from the literature: BH (Bent and Hentenryck [1][2]) and RP (Ropke and Pisinger [11]). BH and RP heuristics are both based on the two-stage approach where BH and RP heuristics, respectively, use simulated annealing and adaptive large neighborhood search for the route minimization phase. BH and RP heuristics were executed five and ten times, respectively, on the Li and Lim's benchmarks. We compare the results in terms of the number of routes.

Table 2 shows the results for each problem size. The solution quality is evaluated by the cumulative number of vehicles where the columns "Best" and "Average" show the best and average results over the number of runs. The column "Best known" shows the cumulative number of vehicles calculated from the best-known solutions. The column "Time" shows the average computation time in seconds for a run spent on each instance. Here, the computation time for the RP heuristic presented in the table includes both route- and distance- minimization phases. The column "#new" shows the number of new -best solutions obtained by the five runs of the GES algorithm. At the bottom of the table, specifications on the computers used in the experiments are shown.

As can be seen from the table, the GES algorithm outperforms the compared heuristics in terms of the cumulative number of vehicles in all problem sizes although the GES algorithm may be assigned longer computation time than the compared heuristics. However, these results are very good because our result are clearly better than the best-known solutions and 105 new best-known solutions were found by the five runs of the GES algorithm.

5 Conclusions

In the paper we extended a GES algorithm, which was first developed for the VRPTW, to the PDPTW. We showed that the perturbation procedure has a great impact on both the solution quality and computation time, and developed an effective GES algorithm for the PDPTW by giving a careful attention to

the perturbation procedure. We believe that this feature comes from the tightly constrained solution space of the PDPTW because the GES algorithm applied to the VRPTW was not so sensitive to the structure of the perturbation procedure. In addition, we presented an efficient algorithm for finding the best insertion-ejection combination from numerous candidates in terms of the given criterion. In the future work, a more innovative criterion for selecting a insertion-ejection move would also be possible and the proposed efficient algorithm will also be available.

References

1. Bent, R., Hentenryck, P.V.: A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Computers and Operations Research 33, 875–893 (2006)
2. Bent, R., Hentenryck, P.V.: A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows, Appendix, `http://www.cs.brown.edu/people/rbent/pickup-appendix.ps`
3. Lau, H., Liang, Z.: Pickup and Delivery with Time Windows: Algorithms and Test Case Generation. In: Proc. of the 13th IEEE International Conference on Tools with Artificial Intelligence, pp. 333–340 (2001)
4. Li, H., Lim, A.: A Metaheuristic for the Pickup and Delivery Problem with Time Windows. In: Proc. of the 13th IEEE International Conference on Tools with Artificial Intelligence, pp. 160–170 (2001)
5. Lim, A., Zhang, X.: A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. Informs Journal on Computing 19, 443–457 (2007)
6. Kindervater, G.A.P., Savelsbergh, M.W.P.: Vehicle routing: handling edge exchanges. In: Local Search in Combinatorial Optimization, pp. 337–360. Wiley, Chichester (1997)
7. Nagata, Y., Bräysy, O.: A Powerful Route Minimization Heuristic for the Vehicle Routing Problem with Time Windows. Operations Research Letters 37, 333–338 (2009)
8. Nagata, Y., Tojo, S.: Guided Ejection Search for the Job Shop Scheduling Problem. In: Cotta, C., Cowling, P. (eds.) EvoCOP 2009. LNCS, vol. 5482, pp. 168–179. Springer, Heidelberg (2009)
9. Nanry, W.P., Barnes, J.W.: Solving the pickup and delivery problem with time windows using reactive tabu search. Transportation research. Part B 34, 107–121 (2000)
10. Parragh, S.N., Doerner, K.F., Hartl, R.F.: A survey on pickup and delivery problems, Part II: Transportation between pickup and delivery locations. Journal für Betriebswirtschaft 58, 81–117 (2008)
11. Ropke, S., Pisinger, D.: An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. Transportation Science 40, 455–472 (2006)