社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2008－AL－118　(9)
2008／5／27

時間枠つき配送計画問題に対する パス再結合と適応的パラメータ調整

橋本 英樹 [1], 柳浦 睦憲 [2]
[1] 京都大学, [2] 名古屋大学

時間枠つき配送計画問題に対してパス再結合アプローチを提案する．本解法は主にパス再結合，局所探索法，適応的パラメータ調整の 3 つから構成される．パス再結合では，これまでの探索中に見つけた良い解を組み合わせて新しい有望な解集合の生成法を提案する．生成された解は局所探索法により改善を行う．局所探索法の近傍としては 2-opt*, cross exchange, Or-opt を用い，さらに効率的に近傍探索を行うために近傍リストを定義し，それを利用して近傍のサイズを制限する．本解法は実行不可能解も探索の対象とし，制約違反度をペナルティとして評価関数に重み付きで足し合わせることで解を評価する．アルゴリズムの性能はその重みに大きく依存するため，探索状況をフィードバックしつつ重みを適応的に調整する方法を提案する．最後に，ベンチマーク問題に対して計算実験を行い，本解法の効果を確認する．

Path Relinking and Adaptive Parameter Control for the Vehicle Routing Problem with Time Windows

Hideki Hashimoto[1], Mutsunori Yagiura[2]
[1] Kyoto University, [2] Nagoya University

We propose a path relinking approach for the vehicle routing problem with time windows. In our algorithm, solutions generated by path relinking operations are improved by a local search whose neighborhood consists of the representative neighborhoods called 2-opt*, cross exchange and Or-opt. To make the search more efficient, we propose a neighbor list that prunes the neighborhood search heuristically. Infeasible solutions are allowed to be visited during the search, while the amount of violation is penalized. As the performance of the algorithm crucially depends on penalty weights that specify how such penalty is emphasized, we propose an adaptive mechanism to control the penalty weights. The computational results on well-studied benchmark instances with up to 1000 customers revealed that our algorithm is highly efficient especially for large instances. Moreover, it updated 41 best known solutions among 356 instances.

1 Introduction

The vehicle routing problem with time windows (VRPTW) is the problem of minimizing the total traveling distance of a number of vehicles, under capacity and time window constraints, where every customer must be visited exactly once by a vehicle. The capacity constraint signifies that the total load on a route cannot exceed the capacity of the assigned vehicle. The time window constraint signifies that each vehicle must start the service at each customer in the period specified by the customer. The VRPTW has a wide range of applications such as bank deliveries, postal deliveries, school bus routing and so on, and it has been a subject of intensive research focused mainly on heuristic and metaheuristic approaches.

We propose a path relinking approach for the VRPTW. Our algorithm invokes a path relinking operation for generating new candidate solutions, which are then improved by a local search whose neighborhood consists of slight modifications of the representative neighborhoods called 2-opt*, cross exchange and Or-opt. To reduce the computation time for searching these neighborhoods, we propose a neighbor list that prunes the neighborhood search heuristically. In our algorithm, infeasible solutions are allowed to be visited during the search, while the amount of violation is penalized. The amount of violation for the capacity constraint is estimated by the amount of capacity excess. To estimate the amount of violation of time window constraints of each route, we consider the total amount of traveling time to be shortened to satisfy the constraints. We also incorporate in our algorithm a frequency-based penalty, in which a customer who often appears in an infeasible route of locally optimal solutions is penalized to direct the search to make those routes with many heavily penalized customers feasible. As the evaluation of these penalties takes time if naively implemented, we propose an efficient algorithm, which enables us to evaluate each neighborhood solution in $O(1)$ time. We also propose an adaptive mechanism to control the weights of these penalties. Finally we report computational results on well-studied benchmark instances with up to 1000 customers. The results show the high competence of our algorithm against existing methods; it updates 41 best known results among 356 instances within a reasonable amount of computation time.

2 Problem Definition

Here we formulate the vehicle routing problem with time windows. Let $G = (V, E)$ be a complete directed graph with vertex set $V = \{0, 1, \ldots, n\}$ and edge set $E = \{(i,j) \mid i,j \in V, i \neq j\}$, and $M = \{1, 2, \ldots, m\}$ be a vehicle set. In this graph, vertex 0 is the depot and other vertices are customers. Each customer i and each edge $(i,j) \in E$ are associated with: (1) fixed quantity a_i (≥ 0) of goods to be delivered to i, (2) a time window $[e_i, l_i]$, (3) a traveling time $t_{ij}(\geq 0)$ and a traveling distance $c_{ij}(\geq 0)$ from i to j. We assume $a_0 = 0$ and $e_0 = 0$ without loss of generality. Each vehicle has an identical capacity u.

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

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

Note that each customer i is included in exactly one route σ_k, and is visited by vehicle k exactly once. We denote by n_k the number of customers in σ_k. For convenience, we define $\sigma_k(0) = 0$ and $\sigma_k(n_k + 1) = 0$ for all k (i.e., each vehicle $k \in M$ departs from the depot and comes back to the depot). Moreover, let s_i be the start time of service at customer i (by exactly one of the vehicles) and s_k^{a} be the arrival time of vehicle k at the depot. Note that each vehicle is allowed to wait at customers before starting services.

Let us introduce 0-1 variables $y_{ik}(\boldsymbol{\sigma}) \in \{0,1\}$ for $i \in V \setminus \{0\}$ and $k \in M$ by

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

That is, $y_{ik}(\boldsymbol{\sigma}) = 1$ holds if and only if vehicle k visits customer i. The traveling distance of a vehicle k is expressed as $d(\sigma_k) = \sum_{h=0}^{n_k} c_{\sigma_k(h),\sigma_k(h+1)}$. Then the problem we consider in this paper is formulated as follows:

$$\begin{array}{llll}
\text{minimize} & \displaystyle\sum_{k\in M} d(\sigma_k) & & (1)\\
\text{subject to} & \displaystyle\sum_{k\in M} y_{ik}(\boldsymbol{\sigma}) = 1, & i \in V \setminus \{0\} & (2)\\
& \displaystyle\sum_{i\in V\setminus\{0\}} a_i y_{ik}(\boldsymbol{\sigma}) \leq u, & k \in M & (3)\\
& t_{0,\sigma_k(1)} \leq s_{\sigma_k(1)}, & k \in M & (4)\\
& s_{\sigma_k(i)} + t_{\sigma_k(i),\sigma_k(i+1)} \leq s_{\sigma_k(i+1)}, & 1 \leq i \leq n_k - 1,\ k \in M & (5)\\
& s_{\sigma_k(n_k)} + t_{\sigma_k(n_k),0} \leq s_k^{\mathrm{a}} \leq l_0, & k \in M & (6)\\
& e_i \leq s_i \leq l_i & i \in V \setminus \{0\} & (7)\\
& y_{ik}(\boldsymbol{\sigma}) \in \{0,1\}, & i \in V \setminus \{0\},\ k \in M. & (8)
\end{array}$$

Constraint (2) means that every customer $i \in V \setminus \{0\}$ must be served exactly once by a vehicle. Constraint (3) means a capacity constraint for vehicle k. Constraints (4)–(6) require that each vehicle cannot serve a customer before arriving at the customer. Constraint (7) is a time window constraint for each customer.

3 Local Search

In this section, we describe our local search (LS). Our LS searches a visiting order $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_m)$, which can be infeasible with respect to the capacity and time window constraints. The algorithm evaluates each route σ_k by a function $p(\sigma_k)$, which is the sum of its traveling distance $d(\sigma_k)$ and the penalty for violation of constraints if σ_k is infeasible, and it evaluates a solution $\boldsymbol{\sigma}$ by $\sum_{k\in M} p(\sigma_k)$. The details of function $p(\sigma_k)$ will be discussed in Section 4. Our LS starts from an initial solution $\boldsymbol{\sigma}$ and repeats replacing $\boldsymbol{\sigma}$ with a better solution (with respect to $\sum_{k\in M} p(\sigma_k)$) in its neighborhood $N(\boldsymbol{\sigma})$ until no better solution is found in $N(\boldsymbol{\sigma})$. To define the neighborhood $N(\boldsymbol{\sigma})$, we use the 2-opt*, cross exchange and Or-opt neighborhoods with slight modifications. For the 2-opt* and cross exchange neighborhoods, we propose a neighbor list to prune the neighborhood search heuristically.

3.1 Neighbor List

We consider a neighbor list for each customer i, which is a set of customers preferable to visit immediately after i. Each customer j that can be visited after i (i.e., $e_i + t_{ij} \leq l_j$) is evaluated by $\max\{t_{ij}, e_j - l_i\}$. When a vehicle visits j immediately after i, it takes at least $\max\{t_{ij}, e_j - l_i\}$ time between the start times of i and j. Hence, if this value is small, it is preferable to visit j immediately after i. The algorithm computes these values once at the beginning and stores the best N_{nlist} (a parameter) customers as a neighbor list of i. We set $N_{\text{nlist}} = 20$ in the experiments.

3.2 Neighborhoods

We use the 2-opt* [10], cross exchange [14] and Or-opt neighborhoods [12] with slight modifications, wherein we restrict the 2-opt* and cross exchange neighborhoods by using the neighbor lists.

A 2-opt* operation removes two edges from two different routes (one from each) to divide each route into two parts and exchanges the second parts of the two routes. Our algorithm searches only those solutions obtainable by a 2-opt* operation in which at least one of the newly added edges is in the neighbor list. The size of this neighborhood is $O(N_{\text{nlist}}n)$.

A cross exchange operation removes two paths from two routes (one from each) of different vehicles, whose length (i.e., the number of customers in the path) is at most L^{cross} (a parameter), and exchanges them. Our algorithm searches only those solutions obtainable by a cross exchange operation in which a newly added edge linking the former part of a route and the path from another route is in the neighbor list. The size of this neighborhood is $O((L^{\text{cross}})^2 N_{\text{nlist}}n)$. We set $L^{\text{cross}} = 3$ in the experiments.

The cross exchange and 2-opt* operations always change the assignment of customers to vehicles. We also use an intra-route neighborhood to improve individual routes. An intra-route operation removes a path of length at most $L^{\text{intra}}_{\text{path}}$ (a parameter) and inserts it into another position of the same route, where the position is limited within length $L^{\text{intra}}_{\text{ins}}$ (a parameter) from the original position. The size of the intra-route neighborhood is $O(L^{\text{intra}}_{\text{path}} L^{\text{intra}}_{\text{ins}} n)$. We set $L^{\text{intra}}_{\text{path}} = 3$ and $L^{\text{intra}}_{\text{ins}} = 10$ in the experiments.

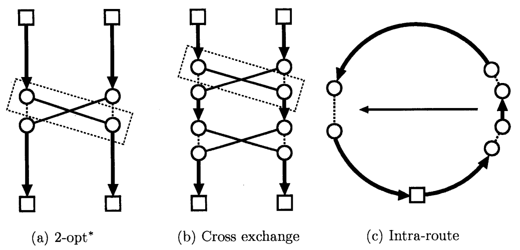

Figure 1: Neighborhood operations in our local search

Figure 1 is an illustration of the neighborhoods. In Figure 1, squares represent the depot (which is duplicated at each end) and small circles represent customers in the routes. A thin line represents a route edge and a thick line represents a path (i.e., more than two customers may be included). The dotted boxes mean that edges in them are in the neighbor lists.

Our LS searches the above intra-route, 2-opt* and cross exchange neighborhoods, in this order. Whenever a better solution is found, the LS immediately accepts it (i.e., we adopt the first admissible move strategy) and resumes the search from the intra-route neighborhood.

4 Evaluation Function $p(\sigma_k)$

We first define the function $p(\cdot)$ to evaluate a route σ_k. For convenience, throughout this section, we assume that vehicle k visits customers $1, 2, \ldots, n_k$ in this order and let customer $n_k + 1$ represent the arrival at the depot (i.e., $s_{n_k+1} = s_k^{\mathrm{a}}$). The function we adopt is

$$p(\sigma_k) = \begin{cases} d(\sigma_k), & \text{if } \sigma_k \text{ is feasible} \\ d(\sigma_k) + \alpha p_{\mathrm{c}}(\sigma_k) + \beta p_{\mathrm{t}}(\sigma_k) + \sum_{h=1}^{n_k} \gamma_h, & \text{otherwise,} \end{cases} \tag{9}$$

where $p_{\mathrm{c}}(\sigma_k)$ is the amount of capacity excess (i.e., $p_{\mathrm{c}}(\sigma_k) = \max\{0, \sum_{i=1}^{n_k} a_i - u\}$) and $p_{\mathrm{t}}(\sigma_k)$ is the minimum total amount of traveling times to be shortened to satisfy the constraints; i.e.,

$$p_{\mathrm{t}}(\sigma_k) = \min\left\{ \sum_{h=1}^{n_k+1} \tau_h \;\middle|\; \begin{array}{l} s_0 \geq 0,\ s_{h-1} + t_{h-1,h} - \tau_h \leq s_h, \\ \tau_h \geq 0,\ e_h \leq s_h \leq l_h,\ h = 1, \ldots, n_k + 1 \end{array} \right\}.$$

In function p, α, β and γ_i for each $i \in V$ are parameters, which are controlled adaptively (see Section 5). Parameters α and β are controlled reflecting the difficulties in satisfying the capacity constraint and the time window constraint, respectively. Parameter γ_i reflects the difficulty in visiting customer i by a feasible route. In the evaluation of (9), each traveling time can be shortened by an arbitrary amount (i.e., the resulting traveling time $t_{h-1,h} - \tau_h$ can be negative) to satisfy time window constraints while the shortened amount is penalized as $p_{\mathrm{t}}(\sigma_k)$. This idea of defining p_{t} was proposed by Nagata [8]. The algorithm computes $p(\sigma_k)$ by each term separately. In the rest of this section, we focus on the computation of $p_{\mathrm{t}}(\sigma_k)$, since the other terms can be efficiently computed by using standard data structures.

A key observation to the efficient computation is that each route σ_k of a neighborhood solution is a recombination of a few paths of the current solution. Hence we consider a speeding up approach that stores some useful information of paths from the depot to customers and those from customers to the depot, among those paths of the current routes. For each customer h in a new route σ_k, let $\mathcal{F}_h$ (resp., $\mathcal{B}_h$) be some data structure that contains the information of the path (of σ_k) from the depot to h (resp., from h to the depot). Note that $\mathcal{F}_h$ and $\mathcal{B}_h$ signify the information of the paths of the new route σ_k. For example, if σ_k is generated by a 2-opt* operation, and the path from the depot to h and the path from $h+1$ to the depot are from the current solution, then $\mathcal{F}_h$ and $\mathcal{B}_{h+1}$ are available from the stored information when they are used to compute $p(\sigma_k)$. On the other hand, for the cross exchange and intra-route neighborhoods, $\mathcal{F}_h$ and $\mathcal{B}_h$ for customers h in inserted paths need to be recomputed, because in the new route σ_k the path from the depot to such an h and that from h to the depot are different from those in the current route. What is important in this approach is to execute the followings efficiently for a given σ_k:

1. construction of $\mathcal{F}_{h+1}$ from $\mathcal{F}_h$ (the forward computation),
2. construction of $\mathcal{B}_h$ from $\mathcal{B}_{h+1}$ (the backward computation), and
3. computation of $p_{\mathrm{t}}(\sigma_k)$ from $\mathcal{F}_h$ and $\mathcal{B}_{h+1}$.

It is not hard to show that each neighborhood solution can be evaluated in $O(T)$ time, if the above operations can be done in $O(T)$ time for any h $(0 \leq h \leq n_k)$. However, to accomplish this, the neighborhood need to be searched in an appropriate search order. The detailed description of such a search order is explained in Ibaraki et al. [4]. Below we show that the forward and backward computation can be done in $O(1)$ time and the computation of $p_{\mathrm{t}}(\sigma_k)$ from $\mathcal{F}_h$ and $\mathcal{B}_{h+1}$ can also be done in $O(1)$ time. Hence the algorithm can evaluate each neighborhood solution in $O(1)$ time.

Let f_h be the minimum total amount of traveling times to be shortened to satisfy the time window constraints for customers $1, 2, \ldots, h$ when vehicle k visits them along the route. Let s_h^{f} be the start time of service at h that attains f_h together with $s_1^{\mathrm{f}}, \ldots, s_{h-1}^{\mathrm{f}}$, and let $\mathcal{F}_h = (f_h, s_h^{\mathrm{f}})$. Then the forward computation can be done by:

$$s_{h+1}^{\mathrm{f}} = \min\{l_{h+1}, \max\{s_h^{\mathrm{f}} + t_{h,h+1}, e_{h+1}\}\} \tag{10}$$

$$f_{h+1} = f_h + \max\{s_h^{\mathrm{f}} + t_{h,h+1}, e_{h+1}\} - s_{h+1}^{\mathrm{f}}. \tag{11}$$

In (10), if $l_{h+1} < \max\{s_h^{\rm f} + t_{h,h+1}, e_{h+1}\}$ holds, the traveling time is shortened to satisfy the time window constraint and this amount is added to f_{h+1} in (11).

The backward computation can be done similarly. Let b_h be the minimum total amount of traveling times to be shortened to satisfy the time window constraints for customers $h, h+1, \ldots, n_k+1$ when vehicle k starts from h and returns to the depot along the route. Let $s_h^{\rm b}$ be the start time of service at h that attains b_h together with $s_{h+1}^{\rm b}, \ldots, s_{n_k+1}^{\rm b}$, and let $\mathcal{B}_h = (b_h, s_h^{\rm b})$. Then the backward computation can be done by:

$$s_h^{\rm b} = \max\{\min\{l_h, s_{h+1}^{\rm b} - t_{h,h+1}\}, e_h\} \qquad (12)$$

$$b_h = b_{h+1} + s_h^{\rm b} - \min\{l_h, s_{h+1}^{\rm b} - t_{h,h+1}\}. \qquad (13)$$

We can compute $p_{\rm t}(\sigma_k)$ from $\mathcal{F}_h = (f_h, s_h^{\rm f})$ and $\mathcal{B}_{h+1} = (b_{h+1}, s_{h+1}^{\rm b})$ by

$$s_{h+1}^{\rm f} = \min\{l_{h+1}, \max\{s_h^{\rm f} + t_{h,h+1}, e_{h+1}\}\} \qquad (14)$$

$$p_{\rm t}(\sigma_k) = f_h + b_{h+1} + \max\{0, s_{h+1}^{\rm f} - s_{h+1}^{\rm b}\}. \qquad (15)$$

5 Adaptive Mechanism to Control Parameters

In this section, we describe an adaptive mechanism to control the parameters α, β and γ_i for each customer i. The algorithm (in which the local search (LS) is executed many times) updates these parameters whenever the LS outputs a locally optimal solution. We set their initial values to $\alpha = 1000$, $\beta = 1000$ and $\gamma_i = 100$ in the experiments.

Let $p_{\rm c}^{\rm sum}(\boldsymbol{\sigma}) = \sum_{k \in M} p_{\rm c}(\sigma_k)$ and $p_{\rm t}^{\rm sum}(\boldsymbol{\sigma}) = \sum_{k \in M} p_{\rm t}(\sigma_k)$, and let $p_{\rm c}^{\rm min}$ (resp., $p_{\rm t}^{\rm min}$) be the minimum $p_{\rm c}^{\rm sum}(\boldsymbol{\sigma})$ (resp., $p_{\rm t}^{\rm sum}(\boldsymbol{\sigma})$) of the solutions in the current reference set R of good solutions, where rules for maintaining R are described in Section 6. Let $P_{\rm c}$ (resp., $P_{\rm t}$) be the number of moves, during the last call to the LS, to a solution $\boldsymbol{\sigma}$ whose $p_{\rm c}^{\rm sum}(\boldsymbol{\sigma})$ (resp., $p_{\rm t}^{\rm sum}(\boldsymbol{\sigma})$) is less than $p_{\rm c}^{\rm min}$ (resp., $p_{\rm t}^{\rm min}$) or equals to 0. Let $N_{\rm total}$ be the total number of moves during the last call to LS, and let $N_{\rm c} = N_{\rm total} - P_{\rm c}$ and $N_{\rm t} = N_{\rm total} - P_{\rm t}$. We use parameters $\delta_{\rm inc}$, $\delta_{\rm dec}$, $\delta_{\rm inc}^{\rm cust}$ and $\delta_{\rm dec}^{\rm cust}$, and in the experiments, we set $\delta_{\rm inc} = 0.05$, $\delta_{\rm dec} = 0.1$, $\delta_{\rm inc}^{\rm cust} = 0.1$ and $\delta_{\rm dec}^{\rm cust} = 0.01$. If the LS found, during last call, a solution $\boldsymbol{\sigma}$ that satisfied $p_{\rm c}^{\rm sum}(\boldsymbol{\sigma}) < p_{\rm c}^{\rm min}$ and $p_{\rm t}^{\rm sum}(\boldsymbol{\sigma}) < p_{\rm t}^{\rm min}$, the parameters α and β are decreased by

$$\alpha := \left(1 - \frac{P_{\rm c}}{\max\{P_{\rm c}, P_{\rm t}\}}\delta_{\rm dec}\right)\alpha, \qquad \beta := \left(1 - \frac{P_{\rm t}}{\max\{P_{\rm c}, P_{\rm t}\}}\delta_{\rm dec}\right)\beta.$$

Even if the LS did not find such a solution, if $N_{\rm c} = 0$ (resp, $N_{\rm t} = 0$) holds, α (resp., β) is decreased by the same equation. Otherwise they are increased by

$$\alpha := \left(1 + \frac{N_{\rm c}}{\max\{N_{\rm c}, N_{\rm t}\}}\delta_{\rm inc}\right)\alpha, \qquad \beta := \left(1 + \frac{N_{\rm t}}{\max\{N_{\rm c}, N_{\rm t}\}}\delta_{\rm inc}\right)\beta.$$

In the locally optimal solution, if a route violates the capacity or time window constraint, γ_i of each customer i in the route is increased by $\gamma_i := (1 + \delta_{\rm inc}^{\rm cust})\gamma_i$. For each customer i who is in a feasible route, γ_i is decreased by $\gamma_i := (1 - \delta_{\rm dec}^{\rm cust})\gamma_i$.

6 Path Relinking Approach

Let R be a reference set of solutions. Initially R is prepared by applying the LS to randomly generated solutions. Then it is updated by reflecting outcome of the LS. During the search, the algorithm always keeps the size of R to ρ (a parameter). We set $\rho = 10$ in the experiments. Good solutions with respect to p are kept in R, excluding at most two solutions: One which achieves $p_{\rm c}^{\rm min}$ and the other which achieves $p_{\rm t}^{\rm min}$. After a feasible solution is found (i.e., $p_{\rm c}^{\rm min} = 0$ and $p_{\rm t}^{\rm min} = 0$), the best feasible solution is always stored as a member of R. Other solutions in R are maintained as follows. Whenever the LS stops, the locally optimal solution $\boldsymbol{\sigma}_{\rm lopt}$ is exchanged with the worst (with respect to p) solution $\boldsymbol{\sigma}_{\rm worst}$ in R (excluding the above solutions), provided that $\boldsymbol{\sigma}_{\rm lopt}$ is not worse than $\boldsymbol{\sigma}_{\rm worst}$ and is different from all solutions in R.

A path relinking operation is applied to two solutions $\boldsymbol{\sigma}_{\mathrm{A}}$ (initiating solution) and $\boldsymbol{\sigma}_{\mathrm{B}}$ (guiding solution) randomly chosen from R, where a random perturbation is applied to $\boldsymbol{\sigma}_{\mathrm{B}}$ with probability $1/2$ before applying the path relinking (for the purpose of keeping the diversity of the search), and the resulting solution is redefined to be $\boldsymbol{\sigma}_{\mathrm{B}}$. We use a cyclic operation, which exchanges partial paths between different routes cyclically, as a random perturbation. In the path relinking operation, we focus on route edges which are used in vehicle routes of a solution. Let $dist(\boldsymbol{\sigma}, \boldsymbol{\sigma}')$ be the number of different route edges between two solutions $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}'$. It is not difficult to see that the distance $dist(\boldsymbol{\sigma}, \boldsymbol{\sigma}')$ between two different solutions $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}'$ can be shortened by at least one by applying an appropriate 2-opt* operation or intra-route operation to $\boldsymbol{\sigma}$. The path relinking operation generates a sequence of solutions $(\boldsymbol{\sigma}_{\mathrm{A}} = \boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \ldots, \boldsymbol{\sigma}_q, \ldots, \boldsymbol{\sigma}_{\mathrm{B}})$ by repeating the following procedure starting from $q = 1$ until $\boldsymbol{\sigma}_q = \boldsymbol{\sigma}_{\mathrm{B}}$ holds: Let $\boldsymbol{\sigma}_{q+1}$ be the best solution with respect to p among those that satisfy $dist(\boldsymbol{\sigma}_{q+1}, \boldsymbol{\sigma}_{\mathrm{B}}) < dist(\boldsymbol{\sigma}_q, \boldsymbol{\sigma}_{\mathrm{B}})$ and obtainable from $\boldsymbol{\sigma}_q$ by a 2-opt* or intra-route operation, and then let $q := q + 1$.

We call a solution $\boldsymbol{\sigma}_q$ locally minimal in the sequence if $p(\boldsymbol{\sigma}_q) < \min\{p(\boldsymbol{\sigma}_{q-1}), p(\boldsymbol{\sigma}_{q+1})\}$ holds. Let S be the best π (a parameter) solutions among the locally minimal solutions in the sequence. Every solution in S is used as an initial solution of the LS. We set $\pi = 20$ in the experiments. The next path relinking is initiated whenever all solutions in S are exhausted as the starting solutions for the local search.

The proposed algorithm is summarized in Algorithm 1. The algorithm stops when it reaches a given time limit. In Algorithm 1, a call to the local search starting from a solution $\boldsymbol{\sigma}$ is denoted by $LS(\boldsymbol{\sigma})$, whose output is the obtained locally optimal solution.

Algorithm 1 Path Relinking Approach

```
 1: Construct the neighbor lists.
 2: Let $R$ be $\rho$ randomly generated solutions. For each $\boldsymbol{\sigma} \in R$, let $\boldsymbol{\sigma}_{\mathrm{lopt}} := LS(\boldsymbol{\sigma})$ and then let $R := (R \setminus \{\boldsymbol{\sigma}\}) \cup$
    $\boldsymbol{\sigma}_{\mathrm{lopt}}$.
 3: Let $S := \emptyset$.
 4: while the stopping criterion is not satisfied do
 5:   while $S = \emptyset$ do
 6:     Randomly choose two solutions $\boldsymbol{\sigma}_{\mathrm{A}}$ and $\boldsymbol{\sigma}_{\mathrm{B}}$ from $R$ ($\boldsymbol{\sigma}_{\mathrm{A}} \neq \boldsymbol{\sigma}_{\mathrm{B}}$).
 7:     With probability $1/2$, apply a cyclic operation to $\boldsymbol{\sigma}_{\mathrm{B}}$.
 8:     Apply the path relinking operation to $\boldsymbol{\sigma}_{\mathrm{A}}$ and $\boldsymbol{\sigma}_{\mathrm{B}}$, and then let $S$ be the set of best $\pi$ locally minimal
        solutions in the generated sequence.
 9:   end while
10:   Randomly choose $\boldsymbol{\sigma} \in S$, and let $S := S \setminus \{\boldsymbol{\sigma}\}$ and $\boldsymbol{\sigma}_{\mathrm{lopt}} := LS(\boldsymbol{\sigma})$.
11:   Update the penalty weights.
12:   Choose the worst $\boldsymbol{\sigma}_{\mathrm{worst}} \in R$ among those that satisfy (1) $\boldsymbol{\sigma}_{\mathrm{worst}}$ is not the unique feasible solution in $R$,
      (2) $\exists \boldsymbol{\sigma}_{\mathrm{c}} \in R \setminus \{\boldsymbol{\sigma}_{\mathrm{worst}}\}$, $p_{\mathrm{c}}^{\mathrm{sum}}(\boldsymbol{\sigma}_{\mathrm{c}}) \leq p_{\mathrm{c}}^{\mathrm{sum}}(\boldsymbol{\sigma}_{\mathrm{worst}})$ and (3) $\exists \boldsymbol{\sigma}_{\mathrm{t}} \in R \setminus \{\boldsymbol{\sigma}_{\mathrm{worst}}\}$, $p_{\mathrm{t}}^{\mathrm{sum}}(\boldsymbol{\sigma}_{\mathrm{t}}) \leq p_{\mathrm{t}}^{\mathrm{sum}}(\boldsymbol{\sigma}_{\mathrm{worst}})$.
13:   if $p(\boldsymbol{\sigma}_{\mathrm{lopt}}) \leq p(\boldsymbol{\sigma}_{\mathrm{worst}})$ and $\boldsymbol{\sigma}_{\mathrm{lopt}}$ is different from all solutions in $R$ then
14:     $R := (R \setminus \{\boldsymbol{\sigma}_{\mathrm{worst}}\}) \cup \boldsymbol{\sigma}_{\mathrm{lopt}}$
15:   end if
16: end while
```

7 Computational Experiments

We conducted computational experiments to evaluate the proposed algorithm. The parameter setting of the algorithm was determined by preliminary experiments on several instances, in which we observed that the performance of the algorithm was not sensitive to parameter values.

We used Solomon's benchmark instances [13] and Gehring and Homberger's benchmark instances [3]. There are 356 instances in total, and all of them have been widely used in the literature. In Solomon's instances, the number of customers is 100, and in Gehring and Homberger's instances, which are the extended instances from Solomon's instances, the number of customers is from 200 to 1000. The customers are distributed in the plane and the distances between customers are measured by Euclidean distances. For these instances, the number of vehicles m is also a decision variable, and the objective is to find a solution with the minimum vehicle number and the total traveling distance in the lexicographical order (i.e., a solution is better than another (1) if its vehicle number is smaller or (2) if the vehicle numbers are the same but the distance is smaller).

As our algorithm deals with the problem with a fixed number of vehicles, we first set the number of vehicles in each instance to the known smallest number to the best of our knowledge, and repeat the followings. If the algorithm found a feasible solution and the number of vehicles is larger than a lower bound $\lceil \sum_{i \in V} a_i/u \rceil$, we ran the algorithm again after decrementing the number of vehicles by one. On the other hand, if the algorithm was not able to find a feasible solution, we ran the algorithm again after incrementing the number of vehicles by one. Among the 356 instances, the algorithm found a feasible solution in the first run for every instance except for six instances. Among the remaining six instances, it was able to find feasible solutions with one more vehicle for five instances and with two more vehicles for the one. The time limit for each run of the algorithm for 100, 200, 400, 600, 800 and 1000-customer instances are 1000, 2000, 4000, 6000, 8000 and 10000 seconds, respectively.

Table 1: Comparison of our results with the existing methods for benchmark instances

References		100	200	400	600	800	1000
Hashimoto et al.	CNV	405	**692**	**1381**	2069	2746	3430
(2008) [2]	CTD	57,282	171,223	406,646	847,470	1,444,513	2,204,728
	P4 2.8GHz	17	33	67	100	133	167
Ibaraki et al.	CNV	407	694	1387	2070	2750	3431
(in press) [5]	CTD	57,545	170,484	398,938	825,172	1,421,225	2,155,374
	P4 2.8GHz	17	33	67	100	133	167
Prescott-Gagnon et al.	CNV	405	694	1385	2071	2745	3432
(2007) [11]	CTD	**57,240**	168,556	389,011	800,797	1,391,344	2,096,823
	Opt 2.3GHz	5×30	5×53	5×89	5×105	5×129	5×162
Pisinger and Ropke	CNV	405	694	1385	2071	2758	3438
(2007) [9]	CTD	57,322	169,042	393,210	807,470	1,358,291	2,110,925
	P4 3GHz	10×2	10×8	5×16	5×18	5×23	5×27
Mester and Bräysy	CNV	–	694	1389	2082	2765	3446
(2005) [7]	CTD	–	168,573	390,386	796,172	1,361,586	2,078,110
	P 2GHz	–	8	17	40	145	600
Le Bouthillier et al.	CNV	405	694	1389	2086	2761	3442
(2005) [6]	CTD	57,360	169,959	396,612	809,494	1,443,400	2,133,645
	5×P 850MHz	12	10	20	30	40	50
Gehring and Homberger	CNV	406	696	1392	2079	2760	3446
(2001) [1]	CTD	57,641	179,328	428,489	890,121	1,535,849	2,290,367
	4×P 400MHz	5×14	3×2	3×7	3×13	3×23	3×30
Homberger and Gehring	CNV	408	699	1397	2088	2773	3459
(2005) [3]	CTD	57,422	180,602	431,089	890,293	1,516,648	2,288,819
	P 400MHz	5×17	3×2	3×5	3×10	3×18	3×31
Ours	CNV	405	694	1383	**2068**	**2737**	**3420**
	CTD	57,484	169,070	392,507	800,982	1,367,971	2,085,125
	Xeon 2.8GHz	17	33	67	100	133	167

Table 1 shows the comparison of our results with those obtained by existing methods. A number in the first row shows the number of customers. Our results are denoted by "Ours." For each method, we provide the cumulative number of vehicles (CNV), the cumulative total distance (CTD), the CPU, and the average computation time in minutes for solving an instance. In the notation of the CPU, "P," "P4," and "Opt" mean Pentium, Pentium 4 and Opteron, respectively. Marks "×" in the second column mean the number of CPUs (e.g., "4×P 400MHz" means four CPUs of Pentium 400MHz), and those in other columns mean the number of runs (e.g., "5×30" means five runs each with 30 minutes of computation time). A number in bold in rows CNV indicates that the value is the best among all the algorithms in the table and there is no tie. When there are ties for the best CNV, the corresponding distance value that is the smallest among those ties is indicated by boldface.

From Table 1, the CNV obtained by our algorithm is much smaller than those of the other methods for large instances with 600 customers or more, and the computation time spent by our algorithm seems to be reasonable; e.g., for instances with $n = 1000$, the computation times spent by recent algorithms by Hashimoto et al. [2], Ibaraki et al. [5], Prescott-Gagnon et al. [11], Pisinger and Ropke [9], and Mester and Bräysy [7] are similar to or sometimes larger than ours even if the difference of CPUs are taken into consideration. Moreover, our algorithm updated 41 best known solutions among the 356 instances. This indicates that our algorithm is highly efficient.

8 Conclusion

We proposed a path relinking approach for the vehicle routing problem with time windows with an adaptive mechanism to control parameters. The generated solutions in the path relinking are improved by a local search. In the local search, each neighborhood solution is evaluated in $O(1)$ time and the neighborhood search is pruned heuristically by the neighbor list. During the search, infeasible solutions are allowed to be visited while the amount of violation is penalized. We also proposed an adaptive mechanism to control the penalty weights. The computational results on representative benchmark instances indicate that the proposed algorithm is highly efficient, and furthermore, the algorithm updated 41 best known solutions among 356 instances.

References

[1] H. Gehring and J. Homberger. A parallel two-phase metaheuristic for routing problems with time-windows. *Asia-Pacific Journal of Operational Research*, 18:35–47, 2001.

[2] H. Hashimoto, M. Yagiura, and T. Ibaraki. An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. *Discrete Optimization*, 5:434–456, 2008.

[3] J. Homberger and H. Gehring. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, 162:220–238, 2005.

[4] T. Ibaraki, S. Imahori, M. Kubo, T. Masuda, T. Uno, and M. Yagiura. Effective local search algorithms for routing and scheduling problems with general time-window constraints. *Transportation Science*, 39(2):206–232, 2005.

[5] T. Ibaraki, S. Imahori, K. Nonobe, K. Sobue, T. Uno, and M. Yagiura. An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. *Discrete Applied Mathematics*, in press.

[6] A. Le Bouthillier, T. G. Crainic, and P. Kropf. A guided cooperative search for the vehicle routing problem with time windows. *IEEE Intelligent Systems*, 20(4):36–42, 2005.

[7] D. Mester and O. Bräysy. Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Computers and Operations Research*, 32:1593–1614, 2005.

[8] Y. Nagata. Effective memetic algorithm for the vehicle routing problem with time windows: Edge assembly crossover for the VRPTW. In *Proceedings of the Seventh Metaheuristics International Conference (MIC2007)*, 2007.

[9] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34:2403–2435, 2007.

[10] 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.

[11] E. Prescott-Gagnon, G. Desaulniers, and L.-M. Rousseau. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. Technical report, GERAD (Group for Research in Decision Analysis), 2007.

[12] S. Reiter and G. Sherman. Discrete optimizing. *Journal of the Society for Industrial and Applied Mathematics*, 13(3):864–889, 1965.

[13] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[14] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186, 1997.