A Hybrid Large Neighborhood Search for Pickup and Delivery Problem with Time Windows

Anh M. NGUYEN[1], Kazushi SANO[2], Kiichiro HATOYAMA[3] and Vu Tu TRAN[4]

[1] Master Student, Urban Transport Engineering and Planning Lab, Nagaoka University of Technology
(1603-1 Kamitomiokamachi, Nagaoka, Niigata Prefecture, 940-2188, Japan)
E-mail: s165081@stn.nagaokaut.ac.jp
[2] Member of JSCE, Professor, Urban Transport Engineering and Planning Lab, Nagaoka University of Technology
(1603-1 Kamitomiokamachi, Nagaoka, Niigata Prefecture, 940-2188, Japan)
E-mail: sano@nagaokaut.ac.jp
[3] Member of JSCE, Project Associate Professor, Urban Transport Engineering and Planning Lab, Nagaoka University of Technology
(1603-1 Kamitomiokamachi, Nagaoka, Niigata Prefecture, 940-2188, Japan)
E-mail: kii@vos.nagaokaut.ac.jp
[4] PhD, Department of Transportation Engineering, Ho Chi Minh City University of Technology and Education
(1 Vo Van Ngan, Thu Duc, Ho Chi Minh City, 708000, Vietnam)
E-mail: tutv@hcmute.edu.vn

This paper focuses on solving the Pickup and Delivery Problem with Time Windows, in which number of pickup and delivery requests that have to be served by a fleet of vehicles. The problem also associates with a number of tight constraints regarding time windows and vehicle capacity must be satisfied. We introduces an extension of Large Neighborhood Search (LNS) by hybridizing with Path Relinking in order to enhance heuristic intensification. In addition, we also introduced an efficient and straightforward feasibility checking procedure. The performance of proposed heuristic is test on set of 100-location problems. Computational results show a very good performance of the hybridization.

Key Words : *Pickup and delivery problem, Time windows, Large neighborhood search, Metaheuristics, Path relinking*

1. INTRODUCTION

Transportation and logistics operation has been considered as a competitive differentiator in business due to not only vast cost it incur, but also its important to service satisfaction goal. Especially in the context of highly sophisticated transport system, there are always needs for transportation optimization to provide efficient strategies relating to the design and management of distribution systems that incur least cost while still maintain a desired service level. These facts have been inspiring and motivating the strong emerge of routing problem. Not only its practical applications, but also its beautifully challenging complexity that draw attention and effort of researchers.

Traveling salesman problem (TSP) is the most famous classic NP-hard problem and is one of the most intensively studied problem in computational mathematics. Three other most well known and extensively studied routing problems are the extension of TSP. In the Pickup and Delivery Problem (PDP) each transportation request specifies a single origin and a single destination and all vehicles depart from and return to a central depot. The Dial a Ride Problem (DARP) is a PDP in which the loads to be transported represent people and all customer requests have load sizes equal to one. The Vehicle Routing Problem (VRP) is a PDP in which either all the origins or all the destinations are located at the depot. Inspired by the wide range of practical application and its generalization, we further consider the pickup and delivery problem with time windows (PDPTW).

As a generalization of the Traveling Salesman Problem (TSP), PDP is inherently known to be NP-hard, and the presence of many related constraints makes the problem even more complicated.

Significant progress has occurred in the past five years, with the development of new exact and approximate algorithms for several types of PDPs. These exact algorithms employ decomposition techniques such as branch-and-cut and branch-and-cut-and-price, while the new heuristics are based on tabu search, simulated annealing and variable neighborhood search.

Regardless of the superior quality of solution, there are only few papers on exact approaches for the PDP. Branch-and-cut in the work of [2, 9], as well as branch-

and-price-and-cut algorithm in [8] have been used in the attempt to generate optimal solution. However, as their high complexity nature, the largest instances that have been solved to optimality so far have no more than 96 requests and 8 vehicles, and have tight time windows and ride time constraints, along with a vast amount of computational time required.

When it comes to approximation methods, it usually refer to metaheuristics for their success and important contribution in solving NP-hard combinatorial problems. They have the capability to provide enough satisfactory results for large-scale instances in different types of optimization problems within reasonable time limits. As for solving PDPTW, there are several successful works using various kind metaheuristics.

One of the first attempt to solve PDPTW using metaheuristic belongs to [7] work. Their approach based on reactive tabu search that combines several standard neighborhoods. Soon later, [6] developed a hybrid metaheuristic which combines simulated annealing and tabu search. Based on Solomon's VRPTW benchmark [13], they also modified and proposed 56 new benchmark instances for PDPTW, which would be used to test our algorithm in the later section. Afterward, the success of metaheuristics by [1] based on large neighborhood search has proven to be efficient for classes of the PDP compare to the previous. In the same year, an important extension for LNS, adaptive large neighborhood search (ALNS) was developed in the work of [10] in which an adaptive weight adjustment was introduced to choose the most suitable pair of removal and insertion heuristics each iteration based on its historical performance, instead of choosing randomly like in previous studies. That adaptive feature significantly influence quality of solutions and has made it become probably the most effective metaheuristic for PDPTW so far, with results reported for up to 1000 locations. Since then, there are several more related works and most of them sole focus on modifying and improve this adaptive mechanism and there are no significant improvement introduced to the LNS. What's more, to the best of our knowledge, there are no LNS which takes advantages of the pool of historical good solutions.

For the sake of improving the LNS while still keeping it straightforward and easy to be implemented, we are not going to integrate the an adaptive weight adjustment component originally proposed in [10]; rather, we improve the heuristic in a different way by making use of path relinking inside LNS to improve its intensification. Similar to LNS, path relinking is a fairly new approach conceptually introduced in [4] and has been applied to VRP with great success in [5].

Our main contributions in this study are as follows:

- We introduce a new adaptive layer using path relinking for improving classical LNS
- We consider a number of realistic constraints and provide an efficient feasibility checking procedure with waiting strategy.
- We consider the trade-off between the cost for hiring vehicles and cost for total distance.

2. PROBLEM DESCRIPTION

In PDPTW, there are n shipments that need to be served and m vehicles available. The problem is defined on a complete graph, $P = \{1,\ldots, n\}$ is the set of pickup nodes, $D = \{n+1,\ldots, 2n\}$ is the set of delivery nodes. Then the pair of nodes $\{i, i+n\}$ represents for pickup and delivery nodes of shipment i. K represents for the set of vehicles, $|K| = m$. The graph $G = (V, A)$ consists of V nodes and $A = V\times V$ arcs. Vertex V_0 represents depot at which is based fleet of vehicles. Each edge $(i, j) \in A$ is assigned a distance $d_{ij} \geq 0$ and travel time $t_{ij} \geq 0$.

Each node $i \in V$ has its amount of goods l_i that need to be picked up/delivered. $l_i \geq 0$ for $i \in P$, and $l_i = -l_{i-n}$ for $i \in$ D. We denote $CapV$ as the capacity of vehicles.

The PDPTW in this study considers these following practical constraints:

- Every route starts and ends at the same depot;

- Pairing and precedence: for every shipment i, the pickup and delivery points belong to the same route and the pickup point is visited earlier than its correspond delivery;

- The load of vehicle k does not exceed capacity C at any time along the route;

- The total duration of route k does not exceed the maximum route duration;

- The total distance of route k does not exceed the maximum route distance;

- The service time at every node in route is fall within the time windows interval.

We denote A_i as the arrival time of a vehicle at node i, then $B_i \geq \max(e_i, A_i)$ as the beginning of the service at node i, and $D_i = B_i + s_i$ as the departure time from i to the next visit. Each node $i \in V$ has a service duration s_i and a time window $[e_i, l_i]$. s_i represents the time needed for serving point i. A service must be started within its time window, from e_i to l_i. Vehicles are allowed to arrive at node i before its earliest time e_i; however, it have to wait until e_i to start it service with waiting time $W_i = B_i - A_i$. And clearly, time window constraint at node i is violated if $B_i > l_i$ and any late service after latest time l_i is not allowed. The total driving time of a vehicle thus corresponds to the amount of time between the last node (at depot) arrival and the departure from the first node (at depot). It can be computed as $L_k = A_0' - D_0$, where A_0' is the time when vehicle k come back at depot.

The objective is to minimize the weighted sum consisting of: (i) the sum of cost for the distance traveled by vehicles, and (ii) the total of fixed cost hiring vehicles with weighted coefficients α, and β, respectively. In real-life applications, these coefficient is usually assigned with real cost unit so that to minimize the correspond total operating cost. In benchmark, these coefficients are adjusted accordingly depending on instances and objectives.

3. SOLUTION METHODOLOGY

This section describes components of ALNS heuristic solving the PDPTW. Some components are mainly based on the works of [10]. However, compare to [10], the heuristic in this paper is extended in several ways:

- We integrate path relinking as an important intensification component inside LNS.
- We propose an efficient and straightforward feasibility checking routine for realistic constraints in terms of spatial and temporal aspects along with waiting strategy at depot to minimize tour duration.
- We consider the trade off between the vehicle cost and travel cost, rather than utilize two-stage method in [10] to separately minimize number of vehicles and distance, respectively.

The overall pseudocode LNS is presented in the algorithm below.

Algorithm 1: LNS heuristic

```
1    Function LNS ($s \in \{solution\}, q \in \mathbb{N}$)
2        Solution $s_{best} = s$
         $E = \phi$
3        repeat
4            $s' = s$
5            Remove $q$ shipment from $s'$
6            Reinsert removed shipments into
                                 partial solution $s'$
7            if $(f(s')) < f(s_{best}))$ then
8                $s_{best} = s'$
9            if accept($s'$, $s$) then
10               $s = s'$
11           if $|E| > 0$ then
12               Randomly select an elite solution
                                          $s^*$ from $E$
13               $s$ = PathRelinking($s$, $s^*$)
14           Update elite set ($s$, $E$)
15       until stop-criterion are met
16   return $s_{best}$
```

(1) Destroy Procedures

This section present four removal heuristics. All of them take a solution and the amount of shipments to be removed q as input. The output of these heuristics is a solution which has q shipments removed and unassigned. What's more, Shaw removal and Worst removal are acquired randomized feature characterized the level of randomization by parameter p.

a) Random Removal

Random Removal is the simplest heuristic among all. It simply randomly picks shipments and removes origin and destination out from the route. The heuristic repeatedly run until a certain degree of destruction is reached. Due to the natural simplicity, it can be implemented significantly fast. It is also a very important heuristic for the LNS algorithm because of being able to maintain diversification for exploring a large search space, avoiding stuck in local optimum.

b) Worst Removal

Given a shipment i served in solution s, we define the cost for serving shipment i as the difference between cost of s and cost of the solution without shipment i, $f_{-i}(s)$. The heuristic tries to remove the shipment that cost most with the hope that the shipment would be reinserted in another position with better cost incurred.

$$\text{argmax}[cost(i, s) = f(s) - f_{-i}(s)]$$

A shipment includes pickup and delivery points. Therefore, two node in route are removed in each removal. That leads to two possible scenarios when it comes to cost computing. If pickup and delivery points of a shipment next to each other in route order, $j = i+1$, $cost = d_{R_{i-1},R_i} + d_{R_i,R_j} + d_{R_j,R_{j+1}} - d_{R_{i-1},R_{j+1}}$. Otherwise, if $j > i+1$, $cost = d_{R_{i-1},R_i} + d_{R_i,R_{i+1}} - d_{R_{i-1},R_{i+1}} + d_{R_{j-1},R_j} + d_{R_j,R_{j+1}} - d_{R_{j-1},R_{j+1}}$.

The procedure is repeated until enough shipments are removed. It is worth to notice that the cost differences change after every removal done. However, not all cost differences have to be recalculated after every iteration due to the fact that the change only occur on the route at which the shipment was removed. With that in mind, only the cost of those shipments in previously destroyed route would be recalculated after each removal.

Besides, in order to make sure the heuristic keep repeatedly removing the same options and create diversification, instead of always removing the first ranked request, the process can be randomized with parameter $p \geq 1$. A random number x in interval [0,1) would be determined in each selection, the candidate at $x^p|L|$ th place in descending list L is selected. The higher p is set, the wider range selection would be due to the highly sensitive of x^p with p.

Algorithm 2: Worst Removal

```
1   Function WorstRemoval ($s \in \{solution\}$,
        $q \in \mathbb{N}$, $p \in \mathbb{R}+$)
2       while $q > 0$ do
3           List $L$ = list of shipments sorted by
                $cost(i, s)$ in descending order
4           Pick a random value of $x$
                in the interval [0,1)
5           $r\_shipment = L[x^p|L|]$
6           Remove $r\_shipment$ from $s$
7           $q = q - 1$
8       end while
```

c) Shaw Removal

This heuristic was originally proposed by [12] and slightly modified by [10] to suit the PDPTW. The heuristic is a combination of service time-oriented and distance-oriented approach. The basic idea is to attempt to remove shipments to some extend similar to each other in terms of distance and time. And expect the later insertion procedure would likely shuffle these shipments around with a probability of generating a new and might be better solution.

The relatedness of two shipments *i* and *j* uses the measure *R*(*i*, *j*). The lower *R*(*i*, *j*), the more related the two shipments. *R*(*i*, *j*) consists of two measured terms: distance and time window. They are weighted with theirs corresponding weights *ρ*, *η*, respectively. The formulation is given by:

$$R(i,j) = \rho(d_{i,j} + d_{i,j+n} + d_{i+n,j} + d_{i+n,j+n}) + \eta(|T_{P(i)} - T_{P(j)}| + |T_{D(i)} - T_{D(j)}|)$$

P(*i*) and *D*(*i*) represent the pickup and delivery point of shipment *i*. T_i represents the average of earliest and latest time of location *i*.

Algorithm 3: Shaw Removal

```
1   Function ShawRemoval ($s \in \{solution\}$,
        $q \in N$, $p \in R+$)
2       Randomly pick a rshipment
3       Set list $D = \{rshipment\}$
4       while $|D| < q$ do
5           $r$ = a randomly selected shipment in $D$
6           List $L$ = list of shipments $i$ not in $D$
                sorted by $R(r, i)$ in ascending order
7           Pick a random value of $x$
                in the interval [0,1)
8           Append shipment $L[x^p|L|]$ to list $D$
9       end while
10      Remove all shipments in $D$ from $s$
```

Similar to Worst removal, randomization is also introduced to generate a certain degree of diversification for the heuristic through a determined parameter $p \geq 1$.

(2) Repair Procedures

Heuristics in this section are going to repair given partial routes destroyed in the previous step by removal heuristics. It is worth to note that these heuristics operate in parallel way in which they construct routes in the same time. They not only rebuild the partial route, but also can be used to build entire solution from scratch as a construction heuristic.

Algorithm 4: Repair Heuristic

```
1   Function RepairHeuristic (partial solution $s$,
    set of unassigned shipments $D$)
2       while $|D| > 0$ do
3           Ascending order insertion cost list $L$
4           Determine top-ranked request-vehicle
                pair $(r, i)$ from $L$
5           Insert shipment $i$ to its best position in
                route $r$
6           Remove $i$ from set $D$
7           Update insertion cost
8       end while
```

a) Greedy Insertion

The heuristic bases on the simple idea of inserting shipments into their best position that have cheapest cost at the moment. Denote $f_{r,s,i,j}$ as the change in objective function value shipment s would incur when being inserted in route *r* at *i*, *j*. We calculate all possible *f*.

In general sense, we would pick the insertion that cost *c* least overall. Define $c = \min(f)$. This operator continues until all shipments assigned.

During the operator, if no feasible route found, a new route is created in order to satisfy all remaining requests.

Similar to Worst removal, it is important to note that each insertion only affect one route in which a new shipment has just been inserted. Therefore, there is no need to recalculate insertion cost in all the other untouched routes.

One of obviously natural drawback this heuristic carrying from the Greedy algorithms family is its myopic. In fact, it often postpones the placement of the customers with higher cost increments to the last iterations where no many available spots left for inserting. Therefore, there are high risks of showing up more unexpected costs at last which likely to quickly escalate the overall cost.

The following heuristic will try to address this drawback.

b) Regret-2 Insertion

The Regret insertion is an improvement of the Greedy insertion. To overcome inherent disadvantage of the latter, instead of inserting the shipment that has the cheapest cost, the Regret insertion looks further information through the so-called regret values, then give decisions accordingly based on this measure.

Regret heuristic uses a variable $x_{ik} \in \{1,\ldots,m\}$, indicates the route for which the customer *i* has the k_{th} lowest insertion cost. In the regret-2 heuristic, a regret value is defined as $c_i = \Delta f_{i,x_{i2}} - \Delta f_{i,x_{i1}}$, where $\Delta f_{i,xm}$ is the change in objective function value incurred by inserting customer *i* into the m^{th} best position. In order words, the regret value defines the difference in the cost of inserting the customer in its best position and its second-best position.

In each iteration, the heuristic chooses the shipment *i* that maximizes:

$$\max c_i$$

In normal sense, we pick the insertion that we would regret most if we would have not done it.

The regret heuristic can be extended to *k* to help it increase its sight pool, thereby improve solution quality. However, we are not going to focus on such thing in this paper, these heuristics, even are imprecise, are enough to compose precise local search heuristics according to [10].

(3) Choosing removal and insertion heuristics

In the previous section, we defined three removal heuristics and two insertion heuristics. Technically, one just need one removal and one insertion heuristic for the search. However, an obvious fact and big issue one usually face when works with optimization problem is that one heuristic could be very efficient in one kind of instance but very bad in others. By alternating between the different heuristics, we could make up a more robust heuristic. That is the reason why we are going to use all mentioned heuristics. On the other hand, how to choose heuristics is also a very important aspect that profoundly influence the quality of solutions. The work of [10] has been proved its success with the adaptive weight adjustment which operates and tunes weight based on the previous performance of heuristics. However, that is not what we want to focus on in this paper.

Rather, to select the heuristic to use in each iteration, we use roulette wheel selection principle and the selection of insertion heuristic and removal heuristic are independent to each other. Instead, we want to approach the adaptive aspect under a different method that we will detail in the next section.

(4) Acceptance and Stopping Criteria

Purpose of acceptance is to decide whether to accept the newly created solution or to keep the current solution to continue manipulating. It provides the search opportunities to escape local optimum and to further explore other search spaces that might have promising solutions. Therefore, the acceptance method is one important part that decides the success of every state-of-the-art metaheuristic.

There are several types of acceptance method, for instance, Greedy acceptance, Threshold acceptance, Simulated Annealing, e.g. to name a few. They all share a same common that is acceptance of any improving solutions. The rule of acceptance of non-improving solutions is the only difference.

Among all, we are inspired by Simulated Annealing acceptance method, which has been playing the key role in the success of this metaheuristic. Obviously by that, it will accept any improved solution. Besides, we will also accept a non-improvement solution *s'* given current solution *s* with probability $e^{-\frac{(f(s') - f(s))}{T}}$ where *T* is temperature.

At the beginning, the initial temperature T_{start} is set so that a solution *w* % worse than the current one would be accepted with probability 0.5. Therefore, T_{start} is dependent on and is calculated according to the value of initial solution. To do so, we can deduce from: $e^{\frac{-f(s_{\text{initial}}).w}{T_{start}}} = 0.5$, hence, $T_{start} = \frac{f(s_{initial}).w}{\log 2}$. The temperature decreases after every iteration $T = T.c$ with a cooling rate c ($0 < c < 1$).

The algorithm stops when certain amount of iterations have been done.

(5) Path Relinking

Metaheuristic algorithms could also have some limitations as the premature convergence, which may cause the algorithm to trap in local optima or to stagnate and therefore it is a challenging problem for the metaheuristics approaches.

The aim of the path relinking phase is to introduce progressively attributes of the guiding solution into solutions obtained by moving away from the initial solution. In order to generate desired path, an initial solution and a guiding solution, representing for start point and end point, respectively, will be needed and be picked from a so-called reference list which contains all elite solutions. Attributes of guiding solution are gradually introduced and replace those from initial one in intermediate solutions along the path. As a consequence, a solution contains more and more attributes of the destination as one moves to the end.

In this section, we manage to hybridized path relinking with LNS. A path relinking is rarely used alone, it is either used as an external component for post-optimization or as an internal procedure within metaheuristic. In this study, we integrate path relinking inside LNS as an intensification method by exploring the path which links elite solutions randomly chosen together in the reference set built during iteration.

At first, we start with an empty elite set and limit it to contain a most n_E solutions inside. It is formed by a set of diverse high-quality solutions found during the search. However, it should represent different potentially good regions and therefore should not include solution that are too similar, even if they are high quality. Every intermediate solution from path relinking is considered to be inserted in elite set. Solutions resulting from path relinking would be considered as a candidate to be inserted in the elite set. The procedure is described in algorithm below.

Algorithm 5: Path Relinking

```
1   Function PathRelinking (initial solution $S^i$, guiding solution $S^g$)
2       $S = S^i$
3       $S^* = S$
4       $f^* = f(S)$
5       while $|N(S:S^g)| \geq 1$ do
6           $S = \text{argmin}\{f(S'):S' \in N(S:S^g)\}$
7           if $f(s) < f^*$ then
8               $S^* = S$
9               $f^* = f(S)$
10      end-while
11  return $S^*$, $f(S^*)$
```

The algorithm relinks the locally optimal solution produced in each LNS iteration with a single solution which is randomly chosen from the elite set. It follows the backward path relinking strategy in which the initial solution is better the guiding solution, contrast to the forward strategy. The reason is that PR explore more thoroughly the neighborhood of the initial solution than that of guiding one as it moves along the path, the size of the neighborhood progressively decreases. Since it is more likely to find an improving solution in the restricted neighborhood of the better solution than in that of the worse, backward path relinking usually tends to perform better than forward path relinking.

(6) Feasibility Evaluation and Waiting Strategy

The repair steps need to evaluate the insertion of single request into given feasible routes. Guaranteeing pairing and precedence is straightforward and is certainly for granted thanks to the consideration of insertion procedures, while verification capacity constraint and temporal constraints is quite tricky and time consuming. Especially during the LNS, millions of insertions must be checked for its feasibility, so that the check should be as efficient as possible.

We apply the concept of Forward Time Slack (FTS) originally proposed by [11)] in the context of TSPTW. We assume that the service times are scheduled as early as possible. Denote $TWT_{(u,v)} = \sum_{u<i\leq v} w_i$ is the total waiting time on path from point u to point v. Then we get:

$$B_v = B_u + \sum_{u\leq i<v} (s_i + \theta_{i,i+1}) + TWT_{(u,v)}$$

Given a feasible schedule T for route R, the FTS F_i at a node v_i gives the maximum amount of time by which the service time at v_i can be delayed so that the resulting schedule is not violated time windows.

$$F_u = \min_{u\leq i\leq q} \left\{ TWT_{(u,i)} + \left(l_i - B_i\right) \right\}$$

The algorithm 6 interprets the basic course of a feasibility test for the PDPTW.

Algorithm 6: Feasibility Check

```
1   Function FeasibilityCheck (given shipment s, route r, position i and j)
2       c = items carrying at location [i-1]
3       if c + item of s at pickup point > capacity then
4           return False
5       else:
6           if j > i + 1 then
7               $c_{max}$ = max amount of item carrying from position [i] to [j-1]
8               if item of s at pickup point + $c_{max}$ > capacity then
9                   return False
10      if j > i + 1 then
11          $B_p$ = Service time at position [i-1]
12          $ts_i$ = CheckInsertion(P(s), i, $B_p$)
13          if $ts_i$ = False then
14              return False
15          else:
16              $B_d$ = new service time at position [j-2]
17              $ts_j$ = CheckInsertion(D(s), j-1, $B_d$)
18              if $ts_j$ = False then
19                  return False
20      else:
21          $B_p$ = Service time at position [i-1]
22          $ts_j$ = CheckInsertion(s, i, $B_p$)
23          if $ts_j$ = False then
24              return False
25      new_end = new arrival time at depot after insertion
26      if new_end – $D_0$ > maxduration then
27          return False
28      return True
```

Algorithm 7: Check Insertion

```
1   Function CheckInsertion (given pickup/delivery point v, inserted position i, service time at precedence position B)
```

```
2    $p$ = precedence point, location at [$i$-1]
3    $B_v = \max(e_v, B + s_p + t_{p,v})$
4    **if** $B_v > l_v$ **then**
5        **return** False
6    **else**:
7        $s$ = successor point, location at [$i$]
8        $B_s' = B_v + s_v + t_{v,s}$
9        *timeshift* = $\max(B_s' - B_s, 0)$
10       **if** *timeshift* $> F_s$ **then**
11           **return** False
12       **else**:
13           **return** *timeshift*
```

It is clear that let the vehicle leave depot as soon as possible and service every vertex as fast as possible as long as vehicle arrive is the safe way to satisfied time windows. In other words, $D_0 = e_0$ and $s_i = \max(a_i, e_i)$. However, according to [3], in fact, setting $D_0 = e_0 + F_0$ instead of $D_0 = e_0$ will thus yield a modified route of minimal total duration with equal violations of time window constraints and equal or smaller violations of total duration constraints.

4. COMPUTATIONAL EXPERIMENTS

This section describes computational results to assess the performance of the proposed algorithm. We use the 100-location instances constructed by [6]. The instances are single depot pickup and delivery problem with time windows with primary objective to minimize number of homogeneous vehicle used and the secondary objective is to minimize the total distance travel.

There are 2 main objectives for this section:

- To determine whether any problem properties can influence the LNS heuristics ability to obtain good solutions.
- To check the performance of LNS as applying in large instances.

The algorithm is coded in Python 3.6 and run on computers with a Core i3-6100 3.7 GHz CPU and 16 GB of RAM.

(1) Data Sets and Parameter Setting

All problem have 100 customers, a central depot, vehicle capacity constraint, precedence constraints together with coupling constraints. The benchmark was created to simulated different scenarios in real life practice. Instances are categorized based on spatial distribution of customers and schedule horizon characteristics. Each group carries certain difficulties, in order to test the all-rounded of search algorithm. In terms of spatial distribution, the customers are clustered in *LC* problems, while those in *LR* problems, in contrast, are randomly distributed; what's more, customers are also partially clustered and partially randomly distributed in *LRC* problems. Furthermore, in terms of schedule horizon characteristics, *LC1*, *LR1* and *LRC1* problems have tight and short scheduling horizon; while *LC2*, *LR2* and *LRC2* have longer scheduling horizon. Hence, there are 6 types of instances being solved here.

In each case, the problem is solved 5 times to compute average number of vehicle used and average total distance obtained. Since the primary objective of the instances is to minimize number of vehicle used, the fix cost of a vehicle is set to H = 1,000, we set α = 1, while $\beta = H$. Besides, due to the fact that instances in the benchmark do not concern about maximum route duration and total route distance, we assign very big values to parameters *maxduration* and *maxdistance* so that the algorithm can fit benchmark criteria.

Since LNS algorithm is composed by several procedures and each procedure has it own parameters, parameters setting should be determined in preliminary experiments. Most of parameters we select from [10] works, while several are modified several to consider trade-off between solution quality and CPU time. Besides, the parameter q that defines the number of vertices removed from solution at each iteration is randomly determined in fixed interval [0.05n, 0.4n]. Parameters are gathered in table 1.

Table 1 Parameter configuration

Parameter	Role	Value
N	Total iteration per run	5000
α	Distance weight	1
β	Fixed-cost vehicle weight	1000
c	SA cooling factor	0.9996
p	Randomized parameter	6
w	Tolerance threshold	0.015
q	Degree of destruction	random

(2) Results

The results on 100-location problems of [6] benchmark are shown in table 2. The name of problems and the best solutions so far are placed in first three columns. Columns two and three give the total number of vehicles and the total traveled distance of the best known solutions. Likewise, the next two columns show the same information of the best solutions obtained in 5 runs of LNS. The last two columns display average values of 5 runs.

As can be seen, the proposed hybrid heuristic shows promising results. The objective function value out of 51 over 56 instances was found equal the best known solution collected so far from previous studies.

Table 2 Result on 100-location problems benchmark [6)]

Instance	Best known solutions		Best of 5 runs		Average of 5 runs	
	Vehicles	Distance	Vehicles	Distance	Vehicles	Distance
LC101	10	828.94	10	828.94	10	828.94
LC102	10	828.94	10	828.94	10	828.94
LC103	9	1035.35	9	1035.35	9.4	959.25
LC104	**9**	**860.01**	**9**	**861.65**	**9**	**868.394**
LC105	10	828.94	10	828.94	10	828.94
LC106	10	828.94	10	828.94	10	828.94
LC107	10	828.94	10	828.94	10	828.94
LC108	10	826.44	10	826.44	10	826.44
LC109	**9**	**1000.6**	**10**	**827.82**	**10**	**827.82**
LC201	3	591.56	3	591.56	3	591.56
LC202	3	591.56	3	591.56	3	591.56
LC203	3	591.17	3	591.17	3	591.17
LC204	3	590.6	3	590.6	3	590.942
LC205	3	588.88	3	588.88	3	588.88
LC206	3	588.49	3	588.49	3	588.49
LC207	3	588.29	3	588.29	3	588.29
LC208	3	588.32	3	588.32	3	588.32
LR101	19	1650.8	19	1650.8	19	1650.8
LR102	17	1487.57	17	1487.57	17	1487.57
LR103	13	1292.68	13	1292.68	13	1292.68
LR104	**9**	**1013.39**	**9**	**1016.93**	**9.8**	**1040.628**
LR105	14	1377.11	14	1377.11	14	1377.11
LR106	12	1252.62	12	1252.62	12	1252.62
LR107	10	1111.31	10	1111.31	10	1111.31
LR108	9	968.97	9	968.97	9	968.97
LR109	11	1208.96	11	1208.96	11	1208.96
LR110	**10**	**1159.35**	**11**	**1193.18**	**11**	**1183.412**
LR111	10	1108.9	10	1108.9	10	1108.9
LR112	9	1003.77	9	1003.77	9.6	1020.172
LR201	4	1253.23	4	1253.23	4	1253.23
LR202	3	1197.67	3	1197.67	3	1197.67
LR203	3	949.4	3	949.4	3	949.4
LR204	2	849.05	2	849.05	2	849.05
LR205	3	1054.02	3	1054.02	3	1054.02
LR206	3	931.63	3	931.63	3	931.63
LR207	2	903.06	2	903.06	2	903.06
LR208	2	734.85	2	734.85	2	736.46
LR209	3	930.59	3	930.59	3	930.59
LR210	3	964.22	3	964.22	3	964.22
LR211	**2**	**911.52**	**3**	**884.29**	**3**	**902.82**
LRC101	14	1708.8	14	1708.8	14	1708.8
LRC102	12	1558.07	12	1558.07	12	1558.07
LRC103	11	1258.74	11	1258.74	11	1258.74
LRC104	10	1128.4	10	1128.4	10	1128.4
LRC105	13	1637.62	13	1637.62	13	1637.62
LRC106	11	1424.73	11	1424.73	11	1424.73
LRC107	11	1230.14	11	1230.14	11	1230.14
LRC108	10	1147.43	10	1147.43	10	1147.43
LRC201	4	1406.94	4	1406.94	4	1406.94
LRC202	3	1374.27	3	1374.27	3	1374.27
LRC203	3	1089.07	3	1089.07	3	1089.07
LRC204	3	818.66	3	818.66	3	818.66
LRC205	4	1302.2	4	1302.2	4	1302.2
LRC206	3	1159.03	3	1159.03	3	1159.03
LRC207	3	1062.05	3	1062.05	3	1062.05
LRC208	3	852.76	3	852.76	3	852.76

5. CONCLUSION

In this study, the authors proposed an extension for the large neighborhood search to solve the pickup and delivery problem with time windows by hybridizing with an intensification method, path relinking. The heuristic was tested on 100-location benchmark and could obtain good results, in which almost problems were solved optimally. It shows the success of this hybridization.

In addition, an efficient and straightforward feasibility checking procedure was also presented in the study. It not only facilitates the identification of feasible insertions but also helps improve the quality of solutions by allowing delaying the departure time at depot before vehicles leaving for servicing. This strategy eventually helps minimize the total driving time of each tour.

REFERENCES

1) Bent, R., Hentenryck, P.V. : A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Comput. Oper. Res. 33, 875–893, 2006.
2) Cordeau, J.-F. : A Branch-and-Cut Algorithm for the Dial-a-Ride Problem. Oper. Res. 54, 573–586, 2006.
3) Cordeau, J.-F., Laporte, G. : A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transp. Res. Part B Methodol. 37, 579–594, 2003.
4) Glover, F., Laguna, M., Taillard, E., de Werra, D. : Tabu search. Baltzer Basel, 1993.
5) Ho, S.C., Gendreau, M. : Path relinking for the vehicle routing problem. J. Heuristics 12, 55–72, 2006.
6) Li, H., Lim, A. : A Metaheuristic for the Pickup and Delivery Problem with Time Windows. Int. J. Artif. Intell. Tools 12, 173–186, 2003.
7) Nanry, W.P., Barnes, J.W. : Solving the pickup and delivery problem with time windows using reactive tabu search. Transp. Res. Part B Methodol. 34, 107–121, 2000.
8) Ropke, S., Cordeau, J.-F. : Branch and Cut and Price for the Pickup and Delivery Problem with Time Windows. Transp. Sci. 43, 267–286, 2009.
9) Ropke, S., Cordeau, J.-F., Laporte, G. : Models and branch-and-cut algorithms for pickup and delivery problems with time windows. Networks 49, 258–272, 2007.
10) Ropke, S., Pisinger, D. : An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. Transp. Sci. 40, 455–472, 2006.
11) Savelsbergh, M.W.P. : The Vehicle Routing Problem with Time Windows: Minimizing Route Duration. ORSA J. Comput. 4, 146–154, 1992.
12) Shaw, P. : Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems, in: Principles and Practice of Constraint Programming — CP98. Presented at the International Conference on Principles and Practice of Constraint Programming, Springer, Berlin, Heidelberg, pp. 417–431, 1998.
13) Solomon, M.M. : Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. Oper. Res. 35, 254–265, 1987.