New Solution Construction Heuristics for the Multiple Vehicle Pickup and Delivery Problem with Time Windows

Manar Hosny* Christine Mumford*

*Cardiff School of Computer Science, Cardiff University
Queen's Buildings, 5 The Parade, Roath, Cardiff, CF24 3AA, UK
(M.I.Hosny, C.L.Mumford)@cs.cardiff.ac.uk

1 Introduction

The Multiple Vehicle Pickup and Delivery Problem with Time Windows (MV-PDPTW) is an important problem in logistics and transportation management. Yet, this problem is less studied than the classical vehicle routing problems, possibly due to the large number of constraints involved and the difficulty in handling them. Even constructing a feasible solution to this hard problem is a challenge in itself.

The MV-PDPTW is a variant of the well-known Vehicle Routing Problem with Time Windows (VRPTW)[1]. The problem deals with a number of customer requests that are to be served by a fleet of vehicles, while a number of constraints must be observed. Each vehicle has a limited capacity (the capacity constraint). A vehicle route usually starts and ends at a central depot. A request must be picked up from a pickup location to be delivered to a corresponding delivery location. Naturally, the pickup and delivery pair must be served by the same vehicle (the coupling constraint) and the pickup must precede the delivery (the precedence constraint). In addition, every request must be served within a predetermined time window (TW) interval (the time window constraint). If the vehicle arrives earlier than the allowed service time, it should wait until the beginning of the specified period. A solution to the problem should assign requests to vehicles and find a route for each vehicle, such that the total service cost is minimized and all problem constraints (coupling, precedence, capacity and time windows) are adhered with. A formal problem definition can be found in [4]. Possible practical applications of the MV-PDPTW include: transportation of raw materials from suppliers to factories, Internet-based pickup from sellers and delivery to buyers, pickup and delivery of charitable donations from homes to different organizations, and the transport of medical samples from medical offices to laboratories. In addition, an important related variant is the dial-a-ride services, where people instead of goods are transported.

As a generalization of the traveling salesman problem, the MV-PDPTW is known to be *NP-hard* [10], and the presence of many constraints makes the problem particularly complicated. Exact algorithms are too slow for large problem sizes. In addition, generating feasible and good quality solutions to the problem in a reasonable amount of time is often a hard challenge for researchers. The

[1]In the VRPTW all requests are of the same type, either pickup or delivery.

MV-PDPTW is both a ***grouping problem*** (assigning requests to vehicles), and a ***routing problem*** (finding the best route for each vehicle). Thus, an intelligent solution methodology should be able to handle these two aspects efficiently. Researchers in the area usually try to solve the problem in two stages: the first stage constructs one or more initial solutions to the problem, while the second stage tries to improve these solutions using a heuristic or a meta-heuristic approach.

To construct a solution for the MV-PDPTW, each step of the algorithm usually selects an un-assigned customer whose insertion causes the least increase in the overall cost of the solution. The selected customer is then inserted in its best (least cost) feasible insertion position found among all available routes. This kind of insertion may require complicated calculations to estimate the effect of the insertion, in terms of the increase in travel distance and time delay, on all customers already existing in the route who could be affected by the insertion. Additional decisions during the construction of the solution include whether to build routes sequentially or in parallel, and possibly the selection of seed customers to initialize the routes. Some construction algorithms order customers before the insertion, and the initial order is also an important factor that may affect the quality of the generated solution. Common approaches include sorting customers according to the distance from the depot, or according to the time window. Finally, the selection of a cost function to assess the quality of the solution during the construction is sometimes needed, so that insertions which greatly affect the solution cost could be identified and appropriately handled.

While these difficulties can also apply to the general VRPTW, the pickup and delivery problem in itself entails other difficult considerations, due to the presence of a pair of related locations for each individual request and the precedence and coupling issues resulting thereof. For example, the decision regarding the best insertion position for a certain request should ideally take both the pickup and the delivery into consideration. The sorting criteria for requests may likewise be based on either the pickup or the delivery location, or perhaps combine both. It is also often in the MV-PDPTW that the initial solution is drastically changed during the improvement phase. For example [1] and [10] reported very good results using an algorithm that is based on a Large Neighborhood Search (LNS). The algorithm removes and then relocates a large number of requests (30% - 40%) in each iteration. This could possibly indicate that sophisticated construction algorithms, that are usually time consuming, parameter dependent, and hard to implement, may not actually warrant their cost, as opposed to more straightforward and faster algorithms.

Trying to overcome the difficulties inherent in the construction of a feasible solution, which are mainly due to the hard problem constraints and the many complex problem-specific decisions, we propose in this paper four different construction heuristics that aim to build initial feasible solutions to the MV-PDPTW. All our algorithms utilize a simple and efficient routing algorithm to generate feasible individual vehicle routes. These algorithms, nevertheless, differ in whether the construction of vehicle routes is performed sequentially or in parallel. They also differ in the criteria according to which an un-routed request is selected next for insertion in a particular route. The aim of the research is to decide which construction algorithm has more potential as a preliminary step towards a complete solution methodology to the problem. We have tested several benchmark problem instances and the experimental results are reported in this paper.

The rest of the paper is organized as follows: Section 2 summarizes some related work. Section 3 explains the routing algorithm embedded within the different construction heuristics used. Section 4 details the construction heuristics suggested in this research. Section 5 reports the experimental results of the algorithms tested. Finally, Section 6 concludes with our future plans.

2 Related Work

Solution construction can be done either sequentially or in parallel. A sequential construction builds routes one after another, while a parallel construction builds a number of routes simultaneously. To construct initial solutions for the MV-PDPTW sequentially, researchers usually adapted Solomon's sequential insertion heuristics of the VRPTW [12]. A weighted sum of the extra travel distance and total time delay resulting from the insertion is often used to estimate the cost of the insertion. This type of construction was used by [4] for the MV-PDPTW, and was followed by a solution improvement phase called a tabu-embedded simulated annealing.

A parallel construction heuristic, on the other hand, was first introduced in [9] for the VRPTW. In a parallel construction, several routes are initialized with seed customers and requests are subsequently inserted into any of the initialized routes. Accordingly, the algorithm needs an initial estimate of the number of vehicles to be used. Routes are later added as needed if the initial estimate does not yield a feasible solution. The authors also introduced an additional complex measure in the cost function, which is a generalized regret value comparing the difference between the cost of an immediate insertion verses a postponed insertion. Customers with a large regret value must be considered first. This regret measure was also used by [10] for the MV-PDPTW, and was embedded within an adaptive large neighborhood search technique to improve the solution quality.

The work in [5] presents a construction algorithm for the MV-PDPTW. The algorithm repeats a cycle of three components. The first component is a constructor, which uses a sequential greedy algorithm to add pairs of customers in the order they appear in a priority sequence that is initially random. The analyzer afterwards analyzes the solution and assigns a certain 'blame' value for each customer based on its contribution to the total solution cost. Finally, the prioritizer reorders the customers, such that customers with a high blame value are moved forward in the priority sequence.

Another construction heuristic that solves the MV-PDPTW is the algorithm in [6], which takes into consideration the effect of insertion on both the classical increase in distance measure, and also the remaining time window slack in the route, i.e., priority is given to insertions that do not use much of the available time slack, allowing for more feasible latter insertions. The authors also use a non-standard measure of the visual attractiveness of the route to select the most desired insertions.

An important survey of the general pickup and delivery problem and approaches developed to handle it was presented in [11]. A more recent surveys is presented in [8]. A survey of the important related dial-a-ride problem is in [2].

We noticed during our literature survey that researchers who adopt a 2-phase approach to the problem often pay more attention to the solution improvement phase, such that the results of the initial solution construction phase are seldom reported, if at all. This makes it difficult to assess the contribution of the construction method to the success or the failure of the overall algorithm. It is also important to note that the role of the construction algorithm is not only limited to the initialization phase. The construction algorithm is often utilized at various stages during the improvement phase to create or modify new or partial solutions, as done for example in [1] and [7]. Thus, using a simple, fast and effective construction algorithm is an important factor in any successful solution methodology.

To the best of our knowledge, our research is the first attempt to compare different initial solution construction methods for the MV-PDPTW. The research will help identify the construction heuristic(s) that seems to be most appropriate for this problem, and decide whether sophisticated and computationally expensive methods actually perform a better job in constructing good quality problem solutions, as opposed to other simpler and less expensive algorithms. In the following section we explain the routing algorithm embedded within the different construction algorithms proposed. Section 4 then discusses in detail these construction algorithms.

3 The Routing Algorithm

A crucial part of the MV-PDPTW is the routing algorithm that will generate a feasible route for each individual vehicle, a major concern is how to handle all problem constraints efficiently. Our routing algorithm, first introduced in [3], was proven very effective for solving the Single Vehicle PDPTW. The main difference between our algorithm and other routing (insertion) heuristics in the literature is that our algorithm is greedy in nature. The algorithm does not try to find the best insertion position for each request in the route, but accepts any feasible insertion. As a result, many complex calculations and problem-specific decisions, that are related to the association between the pickup and the delivery, can be avoided. For example, our algorithm eliminates the bias towards either the pickup or the delivery location, which is one of the major drawbacks of 'classical' insertion methods. Clearly, when the best insertion position for one request (pickup or delivery) is chosen first, the choices available for its partner will be restricted accordingly.

Our routing algorithm adopts a simple route representation. Rather than representing the visiting order of requests by a one-dimensional permutation of different locations, we treat both the pickup location and its associated delivery as one unit. In other words, we assign the same code (number) to both the pickup and its delivery. We then rely on a simple decoder to always identify the first occurrence as the pickup and the second as the delivery. Also, to deal with the hard time window constraint, our routing algorithm adopts an intelligent neighborhood move that uses the time window as a guidance. The idea is to try to improve the current route by creating a new neighboring route. To avoid creating and evaluating infeasible routes, though, our neighborhood move only swaps locations that are out of order in terms of their late time window bounds, i.e., if the latter location has a deadline that precedes the earlier one. Having dealt with the precedence and the time windows constraints, the capacity constraint is the only remaining issue. However, due to the nature of the problem, the capacity constraint can often be easily satisfied, since half of the locations in the route are delivery locations whose loads are removed from the vehicle. This simple representation and neighborhood move are employed in a simple Hill-Climbing (HC) route-improvement heuristic, which tries to gradually modify the current route until no further improvement is possible. Algorithm 1 describes this simple heuristic.

The cost function used in the HC algorithm to evaluate the quality of each route tries to minimize the total route duration as well as the degree of infeasibility in capacity and time windows constraints. The objective function of a route r is described by the following equation:

$$F(r) = w_1 \times D(r) + w_2 \times TWV(r) + w_3 \times CV(r) \tag{1}$$

where $D(r)$ is the total route duration, including the waiting time and the service time at each location. $TWV(r)$ is the total number of time window violations in the route, and $CV(r)$ is the total number of capacity violations. The constants $w_1, w_2, and\ w_3$ are weights in the range $[0, 1]$, and $w_1 + w_2 + w_3 = 1.0$. The choice of appropriate weights depends on the importance of each

Algorithm 1 The HC Routing Algorithm

```
1: Given a route $r$
2: repeat
3:    for (Each possible pair of locations in $r$) do
4:       if (The latter location is more urgent in its upper time window bound) then
5:          Swap the current 2 locations in $r$ to get a new route $r'$
6:          $\Delta \leftarrow cost(r') - cost(r)$
7:          if ($\Delta < 0$) then
8:             Replace $r$ with $r'$
9: until (Done){Stop when no improvement has been achieved in the previous pass}
```

term in the objective function. We found that in order to get feasible solutions, the largest penalty should be imposed on the time window violations.

4 Solution Construction Heuristics

In all our construction heuristics we first start by sorting customers according to the farthest distance from the depot. However, since in our approach we deal with customers as a pickup and a delivery pair, the distance measure should take into consideration this problem-specific property. In our research, we found, experimentally, that the distance separating the depot and the delivery location gives an appropriate estimate of the required distance.

4.1 The Sequential Construction Algorithm (*SEQ*)

The sequential construction heuristic tries to build routes one after another. Requests are taken one by one in order, and each request (pickup and delivery) is inserted at the end of the current route. Our HC routing heuristic (Algorithm 1) is then called to try to improve the current route. If the HC algorithm returns an improved route that can feasibly accommodate the newly inserted pair, this route is accepted and we move on to the next request. However, if the pair cannot be feasibly inserted in the current route, the pair is removed from the route, and a new route is allocated to insert the pair. Algorithm 2 describes the sequential construction procedure.

Algorithm 2 The Sequential Construction (*SEQ*)

```
1: Let $M \leftarrow 0$ {$M$ is the number of vehicles used}
2: repeat
3:    Initialize a new route $r$
4:    $M = M + 1$
5:    for (All unassigned requests) do
6:       Get the next unassigned request $i$
7:       Insert the request $i$ at the end of the current route $r$
8:       Call the HC routing heuristic (Algorithm 1) to improve $r$
9:       if ($r$ is a feasible route) then
10:         Mark $i$ as inserted
11:      else
12:         Remove $i$ from $r$
13: until (All requests have been inserted)
```

4.2 The Parallel Construction Algorithms

In our research, we adapted the parallel construction method of [9], explained in Section 2, to the MV-PDPTW. However, we estimated the initial number of vehicles using a simple formula that divides the total load of the pickup requests by the capacity of the vehicle. We also initialized each route with a seed request (pickup and delivery pair) from the sorted list of requests. We then take the remaining requests in order and attempt to insert each request in the partial routes created. If a request cannot be feasibly inserted in any of the already created routes, a new route is added to accommodate this request. Following is an explanation of the different parallel construction algorithms used in our research.

Parallel Construction: First Route (*PFR*): In this parallel construction algorithm, the next request in order is inserted in the first route in which a feasible insertion of this request is found, i.e., no attempt is made to find the best route for the current request.

Parallel Construction: Best Route (*PBR*): In our second parallel construction algorithm, the next request in order is inserted in the best route in which a feasible insertion of this request is found. The best route for each request is the route that causes the least increase in the overall cost of the solution after the insertion process. To calculate the cost of the solution, we used an objective function suggested by [1]. The objective function consists of 3 weighted components: the first component tries to minimize the number of vehicles used in the solution, the second component tries to minimize the total distance traveled, while the third component is a measure that tries to ***maximize*** the square of the number of nodes visited by each vehicle. This last component is intended to favor routes that are rather full and those that are rather empty, as opposed to an even distribution of nodes among routes. The idea is to try to get rid of some vehicles that are under-occupied during subsequent route improvement phases. Minimizing the number of vehicles is usually the primary objective for most solution algorithms, followed by the total travel distance.

Parallel Construction: Best Request (*PBQ*): This parallel construction heuristic does not only try to find the best route for each request, but also tries to select the best un-routed request to be inserted next. The best un-routed request is the one whose insertion (in its best route) causes the least increase in the overall cost of the solution. To evaluate the cost of the solution, the same cost function used in the *PBR* algorithm is used. Algorithm 3 describes this procedure. [2]

5 Computational Experimentation

To test our algorithms, we used several instances from the benchmark data created by Li and Lim in [4]. There are 6 different categories of problem instances in this data set: *LR1*, *LR2*, *LC1*, *LC2*, *LRC1*, and *LRC2*. Problems in the *LR* category have randomly distributed customers, problems in the *LC* category have clustered customers, and problems in the *LRC* category have partially random and partially clustered customers. On the other hand, problems identified with the number '1' have a tight time window width, while problems identified with the number '2' have a long time window width. Each category has problem sizes ranging from 100 to 1000 customers. The total number of files in the data set is 354. The data together with the best known results can be

[2]Due to lack of space, the detailed algorithms of the *PFR* and *PBR* heuristics could not be included in this paper.

Algorithm 3 Parallel Construction: Best Request (*PBQ*)

```
1: Calculate $M$ (the initial estimate of the number of vehicles)
2: Initialize $M$ routes with seed customer pairs from the sorted list of customers
3: repeat
4:    Initialize $GlobalMin$ to an arbitrary large value
5:    for (All remaining unassigned requests) do
6:       Initialize $LocalMin$ to an arbitrary large value
7:       for ($r = 0; r < M; r++$) do
8:          Get the next unassigned request $i$
9:          Insert the request $i$ at the end of the current route $r$
10:         Call the HC routing heuristic (Algorithm 1) to improve $r$
11:         if ($r$ is a feasible route) then
12:            calculate $\Delta cost$ {$\Delta cost$ is the change in solution cost due to the insertion}
13:            if ($\Delta cost < LocalMin$) then
14:               $LocalMin = \Delta cost$
15:               $r* = r$ { $r*$ is the current best route for request $i$}
16:         Remove $i$ from $r$ {temporarily remove $r$ until all insertion costs have been calculated}
17:      if ($r*$ is found) then
18:         if ($LocalMin < GlobalMin$) then
19:            $GlobalMin = LocalMin$
20:            $i* = i$ {$i*$ is the current best request}
21:            $v* = r*$ {$v*$ is the best vehicle (route) for $i*$}
22:      else
23:         Initialize a new route $r'$ {because no feasible insertion is found for $i$ in any of the available routes}
24:         $M = M + 1$
25:         Insert $i$ in the new route $r'$
26:         Mark $i$ as inserted
27:   if ($i*$ is found) then
28:      Insert $i*$ in $v*$
29:      Mark $i*$ as inserted
30: until (All requests have been inserted)
```

downloaded from `http://www.top.sintef.no/vrp/benchmarks.html`. For the purpose of testing our algorithms we selected the first 6 files from each category of each problem size. The total number of files used to test our algorithms is 216. The algorithms were implemented using Visual C++ under a Windows XP operating system, on Intel Pentium (R)D CPU 3.40 GHz and 2 GB RAM. Since the construction algorithms are all deterministic, each algorithm was run only once on each test file. Table 1 shows the average number of vehicles, and the average total distance produced by each algorithm for each problem size separately.

Analyzing the results in Table 1, we notice that regarding the number of vehicles generated, *SEQ* and *PBQ* produced the best results, , with *SEQ* producing better results than *PBQ* in large size problems, while both *PFR* and *PBR* were slightly inferior in this respect. On the other hand, in terms of the total distance traveled, *PBQ* was able to beat all other algorithms, followed by *PBR* and *SEQ*. *PFR* produced the worst average distance in all test cases, but it was slightly better than *PBR* in the number of vehicles used. As a result, *PFR* and *PBR* can be eliminated from further consideration, and we can focus our attention on *SEQ* and *PBQ*.

As can be noticed from the average results in the last row of Table 1, *SEQ* produced better results than *PBQ* in the number of vehicles used. The *PBQ* algorithm, however, was able to beat

Table 1: Average results for all algorithms

Problem	**SEQ**		**PFR**		**PBR**		**PBQ**	
	Veh	**Dist**	**Veh**	**Dist**	**Veh**	**Dist**	**Veh**	**Dist**
100	11.78	2662.92	11.83	2767.19	11.83	2711.89	11.69	2564.09
200	17.33	8887.08	17.69	8954.06	18.17	8816.33	17.14	8132.84
400	33.56	22215.14	34.64	23010.53	34.69	21898.96	33.72	19758.38
600	48.22	44949.4	49.89	46644.49	50.69	45234.96	49.53	41791.82
800	63.53	74650.07	65.94	77895.32	66.44	74056.45	64.89	68713.31
1000	77.25	108513.19	81.97	115106.93	81.75	108662.01	81.58	103751.31
Avg	**41.95**	**43646.30**	**43.66**	**45729.75**	**43.93**	**43563.43**	**43.09**	**40785.29**

the *SEQ* algorithm in minimizing the total distance traveled. This was obviously due to the fact that the *SEQ* algorithm was more concerned with fitting the largest possible number of requests in each vehicle before allocating a new one, while the *PBQ* algorithm relied on a cost function that has the total travel distance among its components. The *PBQ* algorithm was, nevertheless, much slower than the *SEQ* algorithm. The average processing time of the *SEQ* algorithm ranged from 0.02 seconds for 100-customers problems to 1.88 seconds for 1000-customers problems. The *PBQ* algorithm, on the other hand, had a processing time ranging from 0.34 seconds to 952.34 seconds for the same problem types, which indicates beyond doubt the huge difference in the computational effort needed for both algorithms.

We also performed a one-way analysis of variance of the average results produced by both the *SEQ* and the *PBQ* algorithms. The analysis showed that there is no statistically significant difference in the average results produced by the two algorithms, both in terms of the number of vehicles and the total distance. This further indicates that the SEQ algorithm, despite its simplicity and its exceptional speed, produced comparable results to the results of the *PBQ* algorithm. It should also be noted that the *SEQ* algorithm neither requires an initial estimate of the number of vehicles, nor does it need a solution evaluation mechanism during the construction process. The only advantage that the *PBQ* algorithm offers, which is a slight reduction in the total travel distance, does not seem to justify its added cost in terms of the complexity of the algorithm and the increase in processing time. Another advantage of the *SEQ* algorithm is that it can be easily adapted to population-based heuristics or meta-heuristics by randomizing the initial order of request to generate different diverse solutions. The *PBQ* algorithm, on the other hand, is expected to produce a limited diversity, even if the initial order of requests is randomized, because of the selection criteria and the cost function it relies on during the insertion process. Most likely, requests that are hard to insert, and thus cause a large increase in the solution cost, will always remain the same, despite the change in the insertion order.

Although our algorithms are not intended to provide final good quality solutions to the MV-PDPTW, it would still be useful to compare our results with the best known solutions. This would give us a general idea about the expected effort needed in the solution improvement phase. We tried to analyze the relative gap (difference) to best known results, produced by the *SEQ* algorithm for each benchmark category separately. Figure 1 shows the average gap, in the number of vehicles, produced by the *SEQ* algorithm for all problems, organized by problem categories. For example, a gap of 0.5 means that the algorithm produced, on average, 50% more vehicles than the best known results. Figure 2 shows the average gap produced by the same algorithm with respect to the distance traveled.

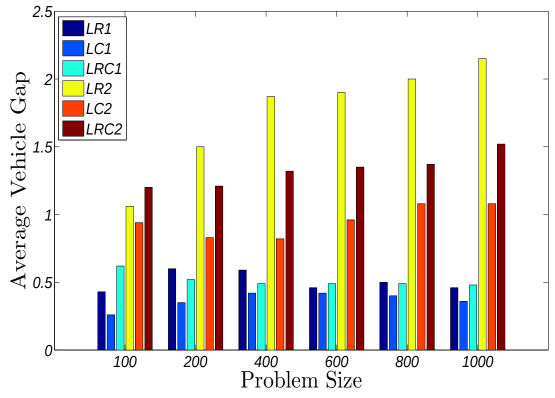

Figure 1: Average vehicle gap (*SEQ*)

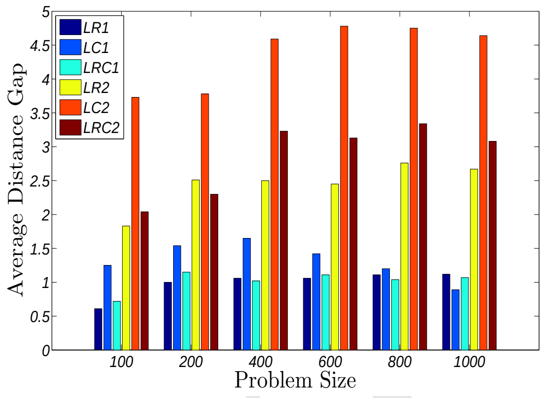

Figure 2: Average distance gap (*SEQ*)

Both figures show that the *SEQ* construction heuristic seems to be more 'successful' in instances with a short schedule horizon, i.e., instances identified with '1' in the data set, since these instances always have a smaller gap than instances of type '2'. Regarding the primary objective, which is the number of vehicles used, the algorithm seems to do a better job for instances that have clustered customers, as opposed to instances that have random or partially random customers. It is clear that instances in the *LC* category always have the smallest gap compared to the other problem types. Problems with random customers and a long time window interval appear to be the most challenging for the *SEQ* algorithm, and possibly all solution algorithms. The reason could be that the solution space for these problems seems to be larger, due to the randomness of locations and the large width of time windows involved in this case. It also appears from both graphs that the gap in the number of vehicles is inversely proportional to the gap in the total travel distance, in most test cases. This indicates that a solution that uses more vehicles may result in an overall shorter travel distance compared to a solution that uses less number of vehicles.

6 Conclusions and Future Work

In this research we investigated several initial solution construction heuristics for the multiple vehicle pickup and delivery problem with time windows. The experimental results on a large number of benchmark instances indicate that the sequential construction heuristic (*SEQ*) seems to be the most favorable solution construction method, which can be by easily embedded in a heuristic or a meta-heuristic technique to reach final good quality solutions. With just a few simple lines of code, and without a pre-determined number of vehicles or a solution evaluation mechanism, this algorithm produced good quality results, that are sometimes even better than the results obtained by the most sophisticated parallel algorithm tested in our research (the *PBQ* algorithm). The *SEQ* algorithm also had an impressive speed, with a processing time that is at most 2% of the time needed by the *PBQ* algorithm, making it even more suitable for population-based solution algorithms. The experimental results, nevertheless, show that a costly improvement phase is still needed to achieve final good quality solutions, as evident by the relatively large gap to best known results produced by the *SEQ* construction algorithm. This, however, further supports the need for a fast solution construction method to achieve an overall reasonable computation time for the complete solution algorithm.

In our future plans, we will start investigating appropriate heuristic and meta-heuristic methods to handle the solution improvement phase. Information obtained in the present research could be utilized in our future work, for example by adapting the improvement algorithm to different problem instances based on their difficulty and the expected amount of effort required to reach good quality solutions to the problem.

References

[1] R. Bent and P. Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers and Operations Research*, 33(4):875–893, 2006.

[2] J.-F. Cordeau and G. Laporte. The dial-a-ride problem (DARP): Variants, modeling issues and algorithms. *4OR: A Quarterly Journal of Operations Research*, 1(2):89–101, 2003.

[3] M. Hosny and C. Mumford. The single vehicle pickup and delivery problem with time windows: Intelligent operators for heuristic and metaheuristic algorithms. *Journal of Heuristics,* `http://www.springerlink.com/content/f54u5618w5241816`, 2008.

[4] H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. In *Proceedings of the 13th IEEE International Conference on Tools with Artificial Intellegence*, pages 160–167, November 2001. Dallas, TX, USA.

[5] H. Lim, A. Lim, and B. Rodrigues. Solving the pickup and delivery problem with time windows using squeaky wheel optimization with local search. In *AMCIS 2002 Proceedings*, 2002.

[6] Q. Lu and M. Dessouky. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, 172(2):672–687, December 2006.

[7] G. Pankratz. A grouping genetic algorithm for the pickup and delivery problem with time windows. *OR Spectrum*, 27:21–24, 2005.

[8] S. Parragh, K. Doerner, and R. Hartl. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(2):81–117, 2008.

[9] J.-Y. Potvin and J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, May 1993.

[10] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, November 2006.

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

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