Investigating Genetic Algorithms for Solving 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

One important problem in transportation and logistics systems is the Multiple Vehicle Pickup and Delivery Problem with Time Windows (MV-PDPTW). The problem deals with a number of pickup and delivery (P&D) requests that should be served by a fleet of vehicles. A delivery location must be served after its corresponding pickup location. The vehicle's journey should start and end at a central depot, while the load carried by the vehicle should not exceed its capacity. In addition, each request must be served within a certain pre-determined time window (TW) interval. If the vehicle arrives at a location before its due service time, it must wait until the beginning of the specified period. A solution to the problem should assign requests to vehicles and generate a feasible route for each vehicle, such that the total service cost is minimized, and all problem constraints are adhered with. A formal problem definition can be found in [6]. Among the practical applications of the MV-PDPTW are the e-commerce activities, the transport of raw materials from suppliers to factories, and the important dial-a-ride services, in which people instead of goods are transported.

Similar to the Vehicle Routing Problem with Time Windows (VRPTW) [1], the MV-PDPTW is *NP-hard* [11]. However, the presence of many constraints adds to the complexity of the problem, and makes even finding a feasible solution a difficult task for many solution algorithms. Since exact algorithms are too slow for large size problems, approximation algorithms are often used to find a reasonable solution. The solution process usually starts by constructing one or more initial solutions to the problem, and then these solutions are improved using heuristics or meta-heuristics.

Genetic Algorithms (GAs) have been successfully used for solving routing and scheduling problems. However, research using GAs for solving the MV-PDPTW is generally scarce, and the results reported by most GA techniques attempted are often disappointing one way or another. The MV-PDPTW consists of two related problems. The *grouping* or the *clustering* problem tries to find the best allocation of requests to vehicles, while the *routing* problem is concerned with finding the best feasible route for each vehicle, given the requests assigned to it. When trying to solve this problem using a GA, it is often hard to tackle these two problems simultaneously. Moreover, a major issue is finding a suitable genetic encoding and designing intelligent genetic operators that

[1] In the VRPTW it is assumed that either pickup or delivery is performed for all requests, but not both.

are capable of handling all the difficult problem constraints [8]. In the MV-PDPTW, it is not clear how an 'ideal' encoding could be achieved, and without the help of such encoding, the researcher should design genetic operators that are smart enough to transfer the favorable genetic traits to the offspring, while avoiding at the same time the generation and evaluation of infeasible problem solutions. Infeasible solutions are usually handled using a repair method to fix the infeasibility during the search, which will inevitably increase the processing time and complicate the algorithm. Most previous GA research, for example [8] for the MV-PDPTW and [5] for the dial-a-ride, tried to solve the problems encountered in the GA encoding and operators by allowing the GA to handle only the grouping aspect. The routing aspect, on the other hand, was handled by an independent routing algorithm that is hidden from the GA and is called when a chromosome is decoded. The genetic operators in that case are usually general-purpose and do not apply any problem-specific knowledge. Other attempts to use a GA for both the grouping and the routing aspects, for example [2] for the MV-PDPTW and [3] for the dial-a-ride problem, generally produced discouraging results.

We present in this paper our early experimentation with a new GA for solving the MV-PDPTW. Our GA tries to face the challenge of handling both the routing and the grouping aspects of the problem simultaneously. Unlike the most popular approaches, in which the GA is only aware of how requests are clustered, but is not aware of how they are routed, our chromosome representation more naturally accommodate each groups of requests together with their suggested routes. By explicitly monitoring and manipulating all the solution information, we aim to preserve the distinctive characteristic of GAs in identifying the desirable genetic material and transferring it from generation to generation during the evolutionary process. Our GA, thus, does not rely on a separate decoder for interpreting the chromosome contents and creating the subordinate routing information for each cluster. Instead, the algorithm has an embedded simple routing heuristic that allows individual routes to dynamically change during the search within the chromosome itself. Also, with our solution representation in mind, we developed new simple genetic operators. Using problem-specific knowledge, such as the quality of the generated routes, these operators try to create good quality feasible solutions throughout the search. In addition, since no parallel repair method is needed to fix the infeasibility of solutions, the overall algorithm is simple and elegant, a feature often missing from most up-to-date solution algorithms.

We compare in this paper our results with what seems to be the only other two GA attempts for solving the MV-PDPTW. The computational experimentation indicate the success of our representation and operators in improving upon similar previous attempts to integrate both the routing and the grouping tasks in the GA search. Nevertheless, certain aspects of our approach could still benefit from some improvement to reach the anticipated standard. We also highlight and explain those aspects in our paper. The rest of the paper is organized as follows: Section 2 provides a brief literature review. Section 3 describes the suggested solution representation, the objective function used, and the creation of the initial population. Section 4 explains the genetic operators used in this research. Section 5 reports the experimental results of the algorithm when tested on published benchmark data. Finally, Section 6 gives some concluding remarks and our future plans.

2 Literature Review

In general Simulated Annealing (SA) and Tabu Search have been the most popular approaches for solving the MV-PDPTW, for example they have been successfully applied in [7] and [6]. Also, both

[1] and [11] use SA as part of their approach, together with a Large Neighborhood Search (LNS) technique, which was successful in producing many new best-known results.

As mentioned previously, GAs have not been very popular for solving the MV-PDPTW. The authors in [2] use an evolutionary algorithm to solve the MV-PDPTW. The solution representation is a list of vehicles' routes. The genetic operators used are adaptations of the operators used in [10] for the VRPTW. Another GA attempt is presented in [8], where the technique is based on a Grouping Genetic Algorithm (GGA). Each gene represents a group of requests that are assigned to one vehicle. Thus an individual solution only covers the grouping aspect of the problem. The routing aspect, on the other hand, is handled by an independent data structure associated with the chromosome[2]. GAs have also been tried for the related dial-a-ride problem. For example, the authors in [5] use a cluster-first route-second approach, where only the clustering is handled using a GA. On the other hand, the work in [3] tries to use a GA for both the grouping and the routing phases. An important survey of the general pickup and delivery problem and approaches developed to handle it was presented in [12]. A more recent surveys is presented in [9]. In the following sections, we describe in detail the GA approach suggested in this research.

3 The Solution Representation and the Initial Population

One of our goals in this research is to develop a GA representation that facilitates dealing with the difficult problem constraints. Following our approach in [4], for solving the Single Vehicle PDPTW (SV-PDPTW), we adopt a simple representation for each individual route. A route is simply a list of visited requests in order. However, when we assign requests to each route, both the pickup and the delivery locations are given the same code. A simple parser is then used to traverse the route and retrieve the information of the pickup location if this is the first encounter of the code, and retrieve the delivery information if this is the second occurrence. This way, the precedence constraint, between the pickup and the delivery, will always be satisfied, and will not be disturbed by any neighborhood move, attempted to improve the route.

The chromosome in our GA represents a complete problem solution. It is simply a collection of individual routes. Both the route (gene) length and the chromosome length are variable depending on the number of requests to be visited and the number of vehicles in the solution. Thus our representation is not actually an encoding in the usual GA sense, rather it is just a problem solution upon which the genetic operators are directly applied. Similar to most solution methods in the literature, for example [6], our objective function tries first to minimize the number of vehicles used in the solution followed by both the total distance traveled and the total route duration. We used the following objective function of a solution s to achieve this goal :

$$O(s) = N^2 \times TotDist(s) \times TotDur(s).$$

Where N is the number of vehicles, $TotDist(s)$ is the total distance traveled by all vehicles, and $TotDur(s)$ is the total schedule duration, which includes the total travel time, the waiting time of the vehicles, and the service time at each location.

The initial population is generated using a sequential construction algorithm to create each solution. The algorithm utilizes a simple routing heuristic to generate fast and feasible individual vehicle routes. This sequential construction algorithm is in fact used in many parts of our GA during the evolutionary process as will be explained later while addressing the GA operators.

[2]More details about the algorithms of [2] and [8] will be presented in Section 5, when our results are compared with their results.

The sequential construction algorithm starts with only one route. Requests are first placed in a relocation pool in a random order. A request from the pool (P&D pair) is then inserted at the end of the current route. The routing algorithm (Algorithm 1) is then called to improve the current route and return a new route. Afterwards, the sequential construction algorithm will check the new route for feasibility. If the new route is feasible, the insertion is accepted and the next request in order will be tried for insertion. On the other hand, if the route is not feasible, the newly inserted couple will be removed from the route and remain in the relocation pool to wait for a later insertion attempt in a new route. The generated solution simply consists of the set of all created routes.

Algorithm 1 The 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 two locations in r to get a new route r'
6:          Δ ← cost(r') − cost(r)
7:          if (Δ < 0) then
8:             Replace r with r'
9: until (Done){Stop when no improvement has been achieved in the previous pass}
```

Our ***routing algorithm***, described in Algorithm 1, was first introduced in [4], and has been very successful in generating fast and feasible routes for the SV-PDPTW. Unlike most other routing (insertion) algorithms in the literature (see for example [13]), our routing heuristic does not try to calculate the insertion cost at each and every possible insertion position, and select the one with the least cost. Rather, it only tries to improve the current route by using a simple Hill-Climbing (HC) heuristic. It is thus a greedy algorithm that is very fast and much simpler than the 'classical' insertion heuristics. The neighborhood move used in the HC algorithm is a regular swap of two locations. However, in order to satisfy the hard time window (TW) constraint, our neighborhood move only swaps locations if the latter location has a TW deadline that precedes the earlier location. The *cost* function used by the algorithm to evaluate the quality of the current route tries to minimize the total route duration, as well as the number of violations in capacity and TW constraints in the route. Our routing algorithm allows routes to dynamically change during the search, i.e., previous routing decisions for some requests already existing in the route may be altered as requests are added or removed from the route. The new routing information is copied back to the chromosome whenever a change in the route occurs.

4 The Genetic Operators

Mutation*:** The mutation operator, which we will call the ***Vehicle Merge Mutation (VMM), simply tries to merge two vehicles selected at random. The idea is to try to reduce the number of vehicles by distributing the requests among already existing vehicles, or possibly combining the two vehicles into one. For example, assume that the current solution contains the following vehicles:

$v1:$	0	1	2	3	3	2	1	4	4	0
$v2:$	0	5	5	6	6	7	7	0		
$v3:$	0	8	9	8	9	0				
$v4:$	0	10	11	12	11	10	12	0		

Now Assume that vehicles $v2$ and $v3$ were selected for merging. The requests belonging to them will now be placed in a relocation pool in a random order.

Relocation Pool: 8 7 6 5 9

The remaining requests in the solution, i.e., vehicles $v1$ and $v4$ will be copied to the new solution to form a partial solution.

$v1:$ 0 1 2 3 3 2 1 4 4 0
$v2:$ 0 10 11 12 11 10 12 0

The requests in the relocation pool are then re-inserted into the partial solution using the sequential solution construction algorithm, described in Section 3, and the final solution is constructed.

$v1:$ 0 **8** 1 2 3 1 3 **7** 2 **8** 4 **7** 4 0
$v2:$ 0 10 11 12 **9** 11 10 **9** 12 **6** **6** 0
$v3:$ 0 **5** **5** 0

Crossover*:** Two crossover operators have been used in our research. The first crossover operator, which we will call the ***Vehicle Merge Crossover (VMX), is similar to the mutation operator described previously. However, instead of merging two vehicles from the same solution, the *VMX* tries to merge two vehicles selected at random, one from each parent solution. The second crossover operator, which will call the ***Vehicle Copy Crossover (VCX)***, tries to copy complete routes from the parent to the child. The number of routes to be copied is a random number between 1/4 to 1/2 the number of routes in the first parent. To select routes for inheritance, the *VCX* tries to select the 'good' routes. It is generally desirable to copy routes that serve a large number of requests, since our main objective is to reduce the number of vehicles. Accordingly, the *VCX* first ranks routes based to the number of nodes served in each route. The larger the number of nodes served the higher the rank of the route. Routes with the same number of nodes are ranked according to the total distance traveled, in which case routes with a shorter distance are more favorable than the longer ones. To illustrate the *VCX*, consider the following example:

Parent1:

$v1:$ 0 1 2 3 3 2 1 4 4 0
$v2:$ 0 5 5 6 6 7 7 0
$v3:$ 0 8 9 8 9 0
$v4:$ 0 10 11 12 11 10 12 0

Parent2:

$v1:$ 0 **5** **6** **6** 1 1 **5** 0
$v2:$ 0 2 **7** 2 3 3 **7** 0
$v3:$ 0 4 4 12 12 0
$v4:$ 0 **8** **9** **8** **9** 0
$v5:$ 0 10 10 11 11 0

Now assume that vehicles $v1$ and $v4$ were selected from Parent1, depending on the ranking criterion described above. These two vehicles will now be copied to the first child.

Child1:

$v1:$ 0 1 2 3 3 2 1 4 4 0
$v2:$ 0 10 11 12 11 10 12 0

The remaining requests that have not been included in Child1 (shown in boldface in Parent2) will be copied in the same order of their appearance in Parent2, and placed in a relocation pool.

Relocation Pool: 5 6 7 8 9

The requests in the relocation pool are sent to the solution construction algorithm and used to form a set of new routes. These new routes will then be appended to the routes already existing in Child1, which were inherited from Parent1.

Child1:

$v1:$ 0 1 2 3 3 2 1 4 4 0
$v2:$ 0 10 11 12 11 10 12 0
$v3:$ **0** **5** **6** **8** **9** **8** **9** **5** **6** **0**
$v4:$ **0** **7** **7** **0**

Child2 is created similarly by reversing the roles of parents. Our early experimentation indicated

that the presence of both crossover operators was necessary for improving the results and satisfying the objective function.

5 Experimental Results

Using Visual C++, we implemented a steady state GA with a 95% replacement. The following parameters were used: population size= 500, crossover probability= 0.6, mutation probability= 0.05, and the number of generations= 300. In cases where crossover is performed either *VCX* or *VMX* is selected at random. We used the 56 (100-customers) benchmark instances, created by Li & Lim in [6]. There are 6 different categories of problem instances: *LC1*, *LC2*, *LR1*, *LR2*, *LRC1* and *LRC2*. Problems in the *LC* category have clustered customers, problems in the *LR* category have randomly distributed customers, and problems in the *LRC* category have both random and clustered customers. Also, problems identified with '1' have a tight TW width, while problems identified with '2' have a large TW width. The data together with the best-known results can be downloaded from: `http://www.top.sintef.no/vrp/benchmarks.html`. We used Windows XP on Intel Pentium (R) CPU, 3.40 GHz and 2 GB RAM. The algorithm was run 10 times on each problem instance.

We compared our algorithm, which we will call the **Grouping-Routing GA** (***GRGA***), with the GA in [2], denoted by ***CKKL***[3], and the grouping GA in [8], denoted by ***GGA***. To the best of our knowledge, they are the only GA approaches that have been attempted in the literature for the MV-PDPTW and applied to the published benchmark data of [6]. The results in [8] are also very close to the best-known results, so we found that a comparison with their results will be sufficient for the purpose of this paper. Before we report our experimental findings, though, we present in Table 1 a comparison of the most distinctive features of the three algorithms under consideration.

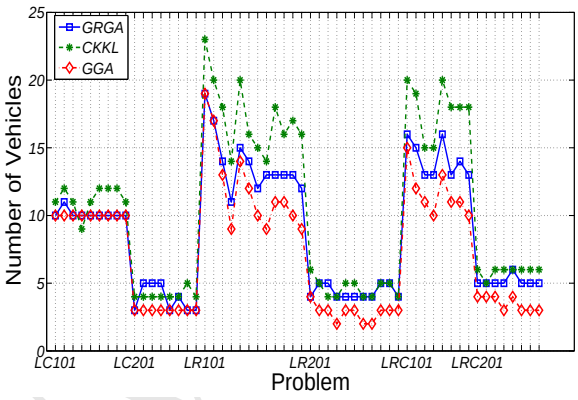

Figure 1: Total number of vehicles

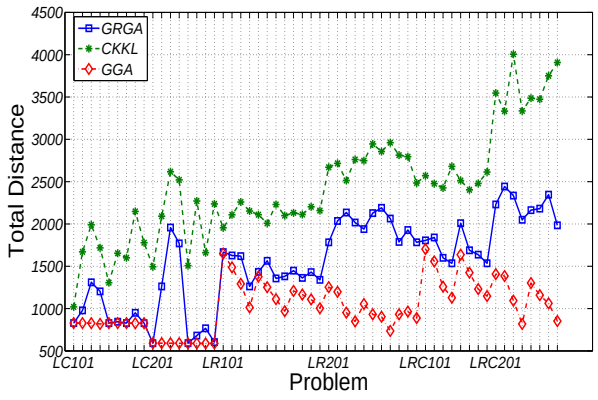

Figure 2: Total travel distance

Figures 1 and 2 show the best results achieved by the three algorithms in terms of the number of vehicles and the total distance traveled. The two figures show that our GA clearly achieves better results than the CKKL algorithm in almost all test cases. There are only 5 cases in which our algorithm produced one more vehicle than the number of vehicles produced by CKKL. Moreover,

[3]We thank the authors of [2] for providing us with the data files containing their detailed results.

Table 1: Comparison between the 3 algorithms

GA Component	GRGA	GGA	CKKL
General Approach	- A GA handles both the grouping and the routing aspects of the problem. - All problem information is explicity monitored and manipulated by the GA.	- A GA only handles the grouping aspect of the problem. - The routing information is hidden from the GA and created when the chromosome is decoded.	- A GA handles both the grouping and the routing aspects of the problem. - All problem information is explicity monitored and manipulated by the GA.
Representation	- A Chromosome has a variable number of genes. - Each gene is a vehicle route (a sequence of visited nodes). - Same code for P&D, and a parser to traverse the route and identify each.	- A Chromosome has a variable number of genes. - Each gene is a cluster of requests assigned to one vehicle. - A sperate data structure and an insertion heuristic are used to create individual routes.	- A Chromosome has a variable number of genes. - Each gene is a vehicle route (a sequence of visited nodes).
Routing or Insertion Heuristic	- Insert P&D pair at end of route, and improve the route using an HC algorithm. - The routing decisions of requests in the route may change during the search. - New routes are copied back to the chromosomes, during recombination and mutation.	- Examine all feasible insertions for the P&D pair in all routes, and select the insertion that causes minimal additional cost. - The routing decisions of nodes already existing in the route are static and do not change during the search.	- The P&D pair is inserted in a feasible route position. - Position of insertion could be modified later using a local search mutation.
Crossover	- Vehicle Copy Crossover (VCX) & Vehicle Merge Crossover (VMX). - Crossover operators are aware of, and utilize, the routing information of each gene. - Offspring is always feasible, and no repair method needed.	- Adaptation of the general GGA crossover, where crossover is not aware of the routing information of each gene. - Consecutive set of clusters are selected from the first parent and inserted in the second parent. - Chromosome cleanup is needed to correct infeasibility of offspring.	- Sequence Based Crossover (SBX): fragments of two routes are selected from each parent and joined together to form a new route. - Route Based Crossover (RBX): two selected routes are exchanged between the two parents. - If possible, infeasibility of offspring is repaired. Otherwise, offspring is discarded.
Mutation	- Vehicle Merge Mutation (VMM). - Mutation is performed on the offspring created by crossover with a certain probability.	- Remove one vehicle and reassign its requests. - Mutation is performed on the offspring created by crossover with a certain probability.	- One-Level exchange Mutation ($1M$): removes one vehicle and reassigns its requests. - Local Search Mutation (LSM): tries to find better locations for requests in a randomly selected route. - Mutation is performed on a randomly chosen individual.
Objective Function	- Minimize the number of vehicles, followed by total distance and total duration.	- Minimize total travel distance, irrespective of the number of vehicles.	- Minimize a weighted sum of the number of vehicles, total distance and total duration (equal weights are assigned).

all our total distance results were better than the results of CKKL. On average, the improvement of our results compared to the results of CKKL in the number of vehicles is approximately 16%, while the average improvement in the total travel distance is approximately 36%. On the other hand, our results are also close to the results of the GGA in the number of vehicles produced, with only few exceptions. Nevertheless, the resulting total distance is larger than the resulting total distance of the GGA in most test cases. This is even more noticeable in instances with a long time window width, i.e, instances of category '2'.

When we try to analyze the computational results in the light of the differences between the three algorithms summarized in Table 1, we will realize the following. First, since both the GRGA and the CKKL algorithms have the same chromosome structure and the same components of the objective function, then the obvious success of the GRGA compared to the CKKL algorithm must be due to the routing algorithm and/or the genetic operators used in the former. The success of the GRGA to produce solutions with less vehicles in most test cases could be attributed to the presence of two genetic operators the are specifically designed for this purpose and heavily utilized during the search, namely the *VMM* mutation and the *VMX* crossover. On the other hand, the *1M* mutation, used for reducing the number of vehicles in CKKL, is only performed occasionally during the search. Also, the noticeable success of the GRGA in terms of reducing the total travel distance, could be attributed to the routing algorithm that is called whenever a route is created or modified to try to improve the quality of the route by reducing its overall cost. This again is in contrast to the *LSM* mutation of CKKL that tries to improve the route, but is only called occasionally during the search. The *VCX* crossover, used in the GRGA, also seems to do a better job than the *RBX* crossover used in CKKL. The *RBX* merely exchanges one route between parents, while the *VCX* tries to be selective when transferring routes from the parent to the child, by choosing routes that serve a large number of nodes with the smallest possible distance. It also seems that the *SBX* crossover, used in CKKL, may not be suitable for the genetic representation used. Since the gene is actually a complete route, it would seem more appropriate to transfer a collection of routes rather than route fragments between parents.

We will now try to analyze the reasons behind the sub-optimal results achieved by the GRGA compared to the GGA, specially in terms of the total travel distance[4]. First of all, one of the major differences between the two algorithms is the objective function. The objective function of the GRGA includes the number of vehicles, the total distance, and the total duration, while the objective function of the GGA only includes the total distance. Thus, the best solution as far as the GRGA is concerned must balance all three components, i.e., it takes into consideration other parameters involved in the routing schedule like the total waiting time of the vehicle at each location and the total service time. The second main difference is the routing heuristic used in both algorithms. The routing (insertion) algorithm of the GGA tries to find the best insertion position for each newly inserted request in all available routes, while we use a simple greedy algorithm that accepts any feasible insertion. The attempt to improve the route, using the HC algorithm, is only local to each route and does not involve comparing the insertion cost with other routes. It seems that more work should be done by our algorithm, or through an additional local search operator, to try to improve the routes, which in turn will also help reduce the number of vehicles. In our opinion, these two reasons are the main reasons for the inability of our approach to reach the solution quality of the GGA. On the other hand, the genetic operators in both seem to be comparable, since they either try to reduce the number of vehicles, or copy complete routes from parents to children.

[4]It should also be noted that our algorithm was run only 10 times on each test case, while the GGA of [8] was run 30 times and the best result was selected.

However, we think that our *VCX* crossover may be more suitable than the crossover of the GGA for this problem type, because transferring a sequence of routes is not really meaningful, since the order of routes in the chromosome is irrelevant.

Also, as mentioned previously, our algorithm seems not to be able to cope with instances of type '2', as evident by the large gap between our results and the results of the GGA in the total travel distance. This could be explained if we recall that our routing algorithm had a neighborhood move, which was guided by the TW. It seems that the routing algorithm was thus capable of dealing better with instances in which the TW constraint is hard to satisfy, i.e., those with a tight TW width. This neighborhood move may not be sufficient to improve the route in problems with a large TW width, because of the availability of many different feasible orders of nodes. An alternative neighborhood move may be needed in that case.

Finally, the average processing time needed by our algorithm, over all 56 problem instances, was 176.9 seconds, which is comparable to the processing time of the grouping GA in [8] having an average of 167.1 seconds. Nevertheless, the separation of the data structure used for individual routes in [8] from the actual chromosome, seems to be favorable than our representation which includes all vehicle routes in the chromosome, as far as processing time is concerned, since the transfer of complete routes during the recombination and mutation operators is definitely time consuming. The authors in [2], on the other hand, do not report their processing time.

6 Conclusions and Future Work

In this research we investigated GAs for solving the MV-PDPTW. Our research tried to face the challenge of allowing the GA and its operators to be aware of and manipulate both the grouping and the routing aspects of the problem. A challenge that most previous GA research on this problem has been mostly unsuccessful with, or has been avoided by allowing the GA to tackle only one problem aspect. We first tried a simple representation and an intelligent neighborhood move to handle the routing part of the problem. For the grouping part, on the other hand, we designed new genetic operators that try to exploit problem-specific information and create new solutions from existing ones, while maintaining the feasibility of solutions throughout the search. Our operators overcome the difficult problem constraints, and avoid at the same time the need for a repair method to fix the infeasible solutions, a technique that previous GAs and most other heuristic and meta-heuristic techniques have been relying on to maintain feasibility. Overall, though, our algorithm is a simple and straightforward GA technique.

We compared our results with two previous GA attempts to solve the problem, the CKKL algorithm of [2] and the GGA algorithm of [8]. The experimental results show that our algorithm was able to greatly improve upon the results of the CKKL algorithm, using just a few simple modifications in the routing algorithm and the genetic operators, and exploiting problem-specific information. The improvement was evident in both main objectives, the number of vehicles and the total travel distance. However, our results are still behind the results of the GGA in most test cases. We analyzed all three algorithms and tried to identify possible reasons behind the differences in the results obtained. In summary, it seems that our representation and genetic operators are doing their job in guiding the search towards better solutions. However, to cope with the difficulty of the problem and the different types of problem instances, our approach still needs further improvement. For example, different neighborhood moves could be attempted in the route improvement heuristic,

or another local search method could be added to improve the quality of the routes and reduce the total travel distance. In our future work, we will experiment with some of these improvement techniques. We also plan to try the representation and some of the operators suggested in this research, within other heuristic and meta-heuristic algorithms for solving this 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. Créput, A. Koukam, J. Kozlak, and J. Lukasik. *Computational Science- ICCS 2004*, volume 3038/2004, chapter An Evolutionary Approach to Pickup and Delivery Problem with Time Windows, pages 1102–1108. Springer, 2004.

[3] C. Cubillos, N. Rodriguez, and B. Crawford. *Bio-inspired Modeling of Cognitive Tasks*, chapter A Study on Genetic Algorithms for the DARP Problem, page 498. 507, 2007.

[4] 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 (to appear),* `http://www.springerlink.com/content/f54u5618w5241816`, 2008.

[5] R. Jørgensen, J. Larsen, and K. Bergvinsdottir. Solving the dial-a-ride problem using genetic algorithms. *The journal of the Operational Research Society*, 58:1321–1331, 2007.

[6] 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.

[7] W. Nanry and J. Barnes. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B: Methodological*, 34(2):107–121, February 2000.

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

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

[10] J.-Y. Potvin and S. Bengio. The vehicle routing problem with time windows part ii: Genetic search. *INFORMS Journal on Computing*, 8(2):165–172, 1996.

[11] 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.

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

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