The Single Vehicle Pickup and Delivery Problem with Time Windows: Intelligent Operators for Heuristic and Metaheuristic Algorithms

Manar I. Hosny and Christine L. Mumford
Cardiff School of Computer Science,
Cardiff University,
Cardiff, CF24 3AA, UK.
Note: Manar is now at King Saud University, Riyadh, Saudi Arabia.
Email: manar_hosny@hotmail.com, C.L.Mumford@cs.cardiff.ac.uk

Published 2008 online and 2010 in print

Abstract

The single vehicle pickup and delivery problem with time windows is an important practical problem, yet only a few researchers have tackled it. In this research, we compare three different approaches to the problem: a genetic algorithm, a simulated annealing approach, and a hill climbing algorithm. In all cases, we adopt a solution representation that depends on a duplicate code for both the pickup request and its delivery. We also present an intelligent neighborhood move, that is guided by the time window, aiming to overcome the difficult problem constraints efficiently. Results presented herein beat those that have been previously published.

Keywords

Pickup and Delivery Problem with Time Windows– Dial-a-Ride–Vehicle Routing– Genetic Algorithms– Simulated Annealing– Hill Climbing

1 Introduction

The pickup and delivery problem with time windows (PDPTW) is an important practical problem that is likely to assume even greater prominence

in the future. Current concerns over global warming, resource depletion and the social impact of traffic congestion and pollution (and resulting legislation, increase in cost, and changes in public perceptions) are driving companies, government organizations and researchers to improve the efficiency of logistics and distribution operations. In addition, the rapid growth in parcel transportation as a result of e-commerce is likely to have an increasing impact. More cooperation between manufacturers, shippers and carriers in supply chains could greatly reduce the environmental impact of transport (the number of "empty running" journeys undertaken by heavy goods vehicles, for example, is currently recorded at 27.4% by UK government statistics (Transport-Statistics, 2006)). In addition, an important variant of the PDPTW, known as dial-a-ride, can provide an effective and efficient means of transport for delivering people from door to door. This model is frequently adopted for the disabled, but could provide a greener alternative to the private car and traditional taxi cab for the wider community.

The single vehicle PDPTW deals with a number of customer requests that must be satisfied by one vehicle with a known capacity. The route of the vehicle usually starts and ends with a central depot. A request must be collected from a pickup location before being dropped off at a corresponding delivery location, and every pickup and delivery request is associated with a specific time window during which it must be served. If the vehicle arrives earlier than the beginning of the designated time window interval, it must wait until the service time begins. All requests must be served in a way that minimizes the total travel cost of the vehicle, without violating precedence, capacity and time windows constraints (Savelsbergh and Sol, 1995).

In addition to being a sub-problem in the general pickup and delivery problem, the single vehicle PDPTW has also practical applications, where small scale companies or individuals may operate one vehicle to serve a set of clients, as for example in a dial-a-ride service. In other applications, the underlying vehicle could be a helicopter or a small ship. Also, in some multiple vehicle applications, the assignment of requests to vehicles may be restricted by some commodity, vehicle, driver or client conditions. In such cases, certain requests must be assigned to a specific vehicle, making the optimization of a single vehicle route necessary for reducing the overall cost of the logistic operation.

As a constrained version of the traveling salesman problem (TSP), the single vehicle PDPTW is known to be *NP-Hard* (Landrieu et al., 2001), with the presence of time windows making the problem particularly complicated. Since exact algorithms are too slow for large problem sizes, heuristic and metaheuristic approaches seem to be good alternatives.

The main goals of our research are, first of all to devise a good representation for the PDPTW, and secondly to develop effective operators and intelligent neighborhood moves capable of directing a heuristic or a metaheuristic search towards high quality solutions. A major challenge is to avoid inefficiencies that can too easily result if large numbers of infeasible solutions (that violate one or more of the hard constraints) are generated.

Several heuristics and metaheuristics are potentially suitable for this problem. Amongst metaheuristics, genetic algorithms (GAs) are well known for their robustness, parallelism, and their ability to perform reasonably well on a wide variety of problems, including ordering and grouping problems, as well as highly constrained problems (Goldberg, 1989; Man et al., 1996). Thus, exploring GAs to solve the PDPTW would seem to be a justified option.

Another alternative metaheuristic approach that could be suitable for this problem is simulated annealing, whose technique is analogous to the annealing of solids. This approach has been widely applied to many optimization problems, successfully transforming "random", low quality solutions to stable high quality optimized solutions (Rutenbar, 1989). One appealing feature of simulated annealing is that it is very easy to implement, since it only requires a method for generating a move in the neighborhood of the current solution, and an appropriate annealing schedule.

A third solution alternative is simple hill climbing that first creates a candidate solution and then iteratively tries to perturb this solution to improve it.

Our research investigates all three approaches to the single vehicle pickup and delivery problem. The challenge is to design a simple, yet intelligent, solution methodology that can handle all problem constraints efficiently. Ultimately we aim to obtain feasible and high quality solutions in an acceptable amount of time.

The rest of this paper is organized as follows: Section 2 provides a brief literature review. Section 3 formally defines the single vehicle pickup and delivery problem with time windows. Section 4 is an overview of our current research, its adopted representation, neighborhood strategy, and the underlying objective function. Section 5 is a description of our first solution methodology, the genetic algorithm and its operators. Section 6 is a description of our second solution approach, simulated annealing. Section 7 describes our final solution approach, hill climbing. Section 8 explains the creation of test cases for the problem, and the computational results obtained when all approaches are compared, and finally Section 9 concludes with our future plans.

2 Literature Review

There are many variations of the pickup and delivery problem (*PDP*) in the literature, and approaches developed to handle them tend to be rather variation-dependent. One classification distinguishes *static* from *dynamic* problems: static problems requiring that all requests are determined in advance of the route construction process, with dynamic problems allowing some requests to arrive during the execution of the route(s) (Savelsbergh and Sol, 1995).

Some PDP problems have neither capacity nor time window constraints, for example (Renaud et al., 2000). Some researchers, as will be detailed shortly, deal with only a single vehicle, while others deal with a more general multiple vehicle case (Nanry and Barnes, 2000; Lau and Liang, 2001; Li and Lim, 2001; Tam and Tseng, 2003; Lu and Dessouky, 2006). An important variant of the PDP is the dial-a-ride problem, in which people instead of goods are transferred, giving rise to customer inconvenience issues that should be taken into consideration during the construction of a solution (Healy and Moll, 1995; Diana and Dessouky, 2004; Jørgensen et al., 2007).

As previously mentioned, there are various approaches to handle the PDP problems: some are *exact* and guarantee to solve the problem to optimality, while others are *approximation* and attempt to find an acceptable solution in a reasonable amount of time (Savelsbergh and Sol, 1995).

For the single vehicle case, an exact algorithm is the dynamic programming approach of (Psaraftis, 1983). However this technique has a time complexity of $O(n^2 3^n)$ (where n is the number of locations) and for this reason is limited to solving small problems of up to about 10 requests (20 locations). The authors in (Desrosiers et al., 1986) are also able to provide exact solutions to the single vehicle dial-a-ride problem, with precedence, capacity and time windows constraints, using dynamic programming. This algorithm can solve instances with up to 40 requests (80 locations), where the capacity and time windows constraints are rather tight. Narrowing the constraints seems to transform the exponential running time of dynamic programming to a linear running time. On the other hand, approximation algorithms, in which a heuristic or a metaheuristic is developed to deal with the problem, make it possible to cope with much larger instances even than this.

For the single vehicle PDPTW, the heuristic in (Bruggen et al., 1993) involves a 2-phase approach. In the initial phase a feasible solution is constructed, and in the second phase this solution is improved. In both phases a variable depth arc exchange procedure is performed, as a neighborhood

move, in which the number of arcs to be exchanged is not determined in advance, but calculated dynamically during the search. In the route construction phase, the time constraint may be violated as long as the precedence and capacity constraints are satisfied. In the improvement phase, however, only feasible solutions are permitted and route duration is used as an objective function. During the improvement phase, the feasibility of a tour (in terms of precedence and capacity) is verified using a set of global variables and an algorithm of $O(n^2)$. To check the feasibility of the time constraint and determine the promising arc exchanges, a forward time slack is computed at each node to identify the permissible shift in departure time that can be introduced without causing violations of time windows for other nodes in the route. As indicated by (Landrieu et al., 2001), the algorithm of (Bruggen et al., 1993) can reach near optimal solutions for problem sizes up to 38 customers, in less than 150 seconds. Unfortunately, success is not guaranteed, and low quality or even infeasible solutions can result, if the 2-phase approach gets trapped in poor local optima. To handle this possible situation, the authors present an alternative approach, which uses simulated annealing. This algorithm accepts time window violations in the early stages, but penalizes them more severely as the search progresses. The simulated annealing approach is able to obtain good quality solutions, albeit with a relatively high processing time.

The work reported in (Jih and Hsu, 1999) also deals with a single vehicle PDPTW. This time, however, a hybrid strategy is proposed which combines an exact method with a genetic algorithm and both static and dynamic cases are considered. The approach consists of three consecutive stages: a pre-planned module, a dynamic programming module, and a genetic algorithm module (GA). The pre-planned module arranges requests and prepares information for the dynamic programming module. The role of the dynamic programming module is to create a set of sub-routes, which it will eventually pass on to a temporary result pool, where the genetic algorithm module will pick them up, installing these unfinished sub-routes to establish its initial population. In the GA module, a solution is encoded as a permutation of locations, and four crossover operators are compared: two traditional order-based crossover operators, and two merge crossover operators, MX1 and MX2, that use a global precedence vector to guide the inheritance process, as explained in (Blanton and Wainwright, 1993). The mutation operator is applied only when the offspring is identical to one of its parents. Three mutation operators are used: one swaps two genes selected at random, the second selects two random sites in the chromosome and reverses the sub-route between the selected sites, finally the third mutation

shuffles the genes that precede a location for which a violation of constraints is observed. If an infeasible solution, violating the precedence constraint, is generated as a result of any genetic operator, this solution is repaired by swapping the corresponding pickup and delivery pair. The experimental results on data sets ranging from 10 to 50 tasks indicate the merge crossover operator MX1 generally performed better than the other crossover operators tested. The first two mutation operators also achieved better results than the third mutation operator.

In a more recent work (Jih and Hsu, 2004), similar genetic operators to the ones used in (Jih and Hsu, 1999) are examined in the context of a family competition genetic algorithm (FCGA). Here again, an order-based representation is adopted. The idea is to allow each individual of the population to play the role of a family father in turn. Another randomly selected individual plays the role of mate for the family father. The two individuals are combined to produce an offspring in a regular GA fashion. The selection of the mate and the reproduction is repeated for a chosen number of iterations to produce a family of offsprings. Only the best offspring in the family survives and is added to a temporary population of champions. The new generation is chosen from among the best individuals in both the original population and the champions of the families. Comparing the performance of the FCGA and a traditional GA on data sets created by the authors (ranging from 10 to 100 requests), the results indicate that the merge crossover operators MX1 and MX2 generally worked better in the context of a traditional GA than a FCGA, possibly due to premature convergence in the latter case. On the other hand, a traditional uniform order-based crossover (UOX) worked better within the framework of a FCGA than a traditional GA, possibly due to its uniform, non greedy, nature of exploring the search space.

The authors in (Landrieu et al., 2001) present a tabu search based heuristic to solve the single vehicle PDPTW. The algorithm first creates a route respecting precedence and capacity constraints, using a simple insertion heuristic. The generated route may be infeasible in terms of the time window constraint, however. Then, two tabu search methods are conducted to transform the initial route to a feasible route with minimum total distance. The two tabu search methods presented are: a regular deterministic tabu search, and a probabilistic tabu search. The probabilistic search is based on the same principals as the deterministic one with the addition of a buffered memory of potential moves and introducing some probabilistic criteria for the selection of next move. The creation of a neighboring solution in both tabu searches is based on classical neighborhood moves, a swap operation

and an insertion operation, both of which respect the precedence and the capacity constraints. Test results on problem instances created by the authors (from 10 to 40 customers) indicate that for these problem sizes, a feasible solution could be reached in a reasonable amount of time. However, for larger problem sizes (more than 60 locations), reaching a feasible solution could possibly take more than one hour. According to the authors, an improvement in processing time could be achieved by adapting the parameters of the method.

In a previous work (Hosny and Mumford, 2007), we investigated the potential of using genetic algorithms to solve the single vehicle PDPTW. We experimented with a duplicate gene encoding that guarantees the satisfaction of the precedence constraint, between the pickup and the delivery during the genetic search. In addition, several problem-specific genetic operators were tested and compared. We experimented with a merge crossover operator (MX1) guided by two time window precedence criteria, which we adopted, with a slight modification, from (Blanton and Wainwright, 1993) and (Jih and Hsu, 2004). In addition we also introduced a new crossover operator that depends on the order of pickup and delivery locations appearing in parent solutions. Two mutation operators were tested: a simple gene swap mutation, and a new time window directed swap mutation. Test results, on a number of data sets ranging from 10 to 200 requests, indicated that due to the difficulty in satisfying the time window constraint, the most successful operators are the ones that take into account the time window precedence order while manipulating genes, namely the merge crossover, MX1, and the directed mutation. These operators were able to obtain feasible solutions even for large data sets tested in our research. For some test cases, these operators were able to beat the best known results reported in (Jih and Hsu, 2004) in terms of both quality and processing speed.

For the interested reader (Savelsbergh and Sol, 1995) provides an excellent and detailed survey on the general pickup and delivery problem and the main techniques to handle it in the literature.

3 The Single Vehicle PDPTW

Assume we have a set of nodes $N = \{n_0, n_1, n_2, ..., n_m\}$, where n_0 denotes the depot, and each $n_i, i = 1, 2, ...m$ denote a customer location. The last index m is an even number. Since for each customer request we have a pair of pickup and delivery locations, we can assume, without loss of generality, that the set $\{n_1, ..., n_{m/2}\}$ represents pickup locations, and the set

$\{n_{(m/2)+1}, ..., n_m\}$ represents delivery locations, such that the pickup location n_i has the corresponding delivery location $n_{i+(m/2)}$. Each location n_i, $i \neq 0$ is associated with:

- A customer demand q_i, such that $q_i > 0$ for a pickup location, $q_i < 0$ for a delivery location and $q_i + q_j = 0$ for the same customer's pickup and delivery locations.
- A time window (TW) $[e_i, l_i]$ during which the location must be served, and $l_i \geq e_i$.

For each possible edge $< n_i, n_j >$ a travel time t_{ij} is specified, and we assume that $t_{ij} = t_{ji}$.

The vehicle has a limited capacity C. The capacity constraint ensures that the total load carried by the vehicle at any given time does not exceed its capacity.

The vehicle's journey should start from the depot and could end at any of the delivery locations[1], while each location should be visited exactly once.

A location must be serviced within the specified time window, i.e., if the vehicle reaches the location before the earliest service time e_i, it must wait until e_i. The precedence constraint requires that each pickup location must precede the corresponding delivery location.

The objective function varies depending on the application. In general, one or more of the following parameters are included: the total traveling distance, the total route duration, or the driver's total waiting time.

4 The Research

The main challenge that we are faced with in this research is the development of a good solution representation, reflecting the problem and its constraints in a simple way to avoid complicating our algorithms. Represented solutions should also be fast to evaluate and easy to interpret. In addition, the representation should be coupled with intelligent neighborhood operators that are capable of directing the search towards high quality and feasible solutions. Taking these challenges into consideration, we investigate and compare three approaches to the single vehicle PDPTW:

[1] It is assumed here that the vehicle's journey is open path, to enable a comparison with the results reported in (Jih and Hsu, 2004) that follow this assumption. An alternative would be for the vehicle to return to the depot after servicing all requests.

The first approach is a genetic algorithm approach (GA), first introduced in (Hosny and Mumford, 2007), which adopts a solution representation having the same code for both the pickup and its associated delivery, and using duplicated entries in the representation to guarantee the precedence feasibility throughout the search. During the evolutionary process, we apply some problem-oriented operators. These operators are: a 2-child merge crossover operator guided by both time window bounds, a simple gene swap mutation, and an intelligent time window directed swap mutation. Details are presented in Section 5.

The second solution methodology applied is a simulated annealing approach (SA), which adopts the same solution representation as the one used in the GA. Also inspired by the GA mutations, two SA neighborhood strategies are tested: a random blind move, and an intelligent move that is directed by the time window. Details are presented in Section 6.

The third approach is a simple hill climbing heuristic (HC) that creates a starting solution and then tries to improve it, replacing the current solution with better solutions generated during the search. The same solution representation and the directed neighborhood move used in both the GA and SA are also adopted in the HC. Details are presented in Section 7.

4.1 The Solution Representation

A good solution representation for this kind of problem is not as obvious as it seems. The pickup and delivery problem (PDP) is an ordering problem in which a solution could be represented as a permutation of locations, representing an order in which these locations will be visited. In the PDP problem, however, the issue of precedence must be addressed in the representation of the solution, because no delivery location is allowed to precede its corresponding pickup location. Nevertheless, this precedence order may not be maintained following the application of any neighborhood move to a solution. For example, a simple swap of locations could disturb the precedence order and result in an infeasible solution. Consequently, a repair method would be needed to restore the feasibility of the solution, as done, for example, in (Jih and Hsu, 1999) and (Jih and Hsu, 2004). Inevitably, such approach would increase the processing time and complicate the algorithm.

We have developed a solution representation which avoids the precedence issue: we simply assign the same code to both the pickup and its associated delivery location, and rely on a simple decoder to identify its first occurrence as the pickup and the second as the delivery. While processing a certain solution, the decoder simply retrieves the information of the pickup location,

if this is the first time the location code is encountered, and retrieves the information of the delivery location if this is the second occurrence of the location code. This straightforward representation eliminates the problem of backtracking to repair an infeasible solution, and solves the precedence constraint issue in an effective way. As a result, more effort can be directed towards harder constraints such as the vehicle capacity and time windows. An example of a solution with 4 requests following this representation is: (**2** **1** *1* **3** **4** *2 3 4*), where pickups are shown in boldface and deliveries in italics.

4.2 The Neighborhood Move

While creating a neighboring solution during the search process, we are faced with a major challenge: the possible violation of one or more of the problem constraints that may follow such move. A neighborhood move should be intelligent enough to direct the search towards high quality and feasible solutions, and thus avoid valuable time being wasted evaluating vast quantities of infeasible solutions. When designing such a move, all problem constraints should be considered. Nevertheless, in our research we found the time window constraint the most difficult to deal with. Recall that our duplicate encoding scheme renders the precedence constraint trivial. Furthermore, the capacity constraint tends to be easy to satisfy in most problem instances, because half of the locations visited in any route involve delivery requests which reduce the load on the vehicle.

In our research we adopt neighborhood search operators that apply "randomness", yet at the same time take account of the time windows, and bring the more "urgent" locations earlier in the visiting order, where this is possible. This neighborhood idea is adopted in all our search algorithms, with slight variations depending on the context in which it is applied. Details are presented in Sections 5, 6 and 7.

4.3 The Objective Function

The objective function will try to minimize the total route duration as well as the degree of infeasibility in capacity and time windows constraints. In our objective function, we treat the constraints as soft constraints, meaning that an infeasible solution that violates either the capacity and/or the time windows constraints will be penalized by an added term in the objective function. We slightly modified the objective function used in (Jih and Hsu, 2004) and (Hosny and Mumford, 2007), by additionally penalizing the amount of time delay together with the number of time window and

capacity violations. Adding a penalty for the extent of total tardiness in the route could possibly direct the search towards better quality and feasible solutions. Thus, the objective function of a route r is described by the following equation:

$$F(r) = w_1 \times D(r) + w_2 \times TWV(r) + w_3 \times CV(r) + w_4 \times TD(r) \qquad (1)$$

Where $D(r)$ is the total route duration including the waiting time, if the vehicles arrives at a location before the start of its service time, $TWV(r)$ is the total number of time window violations in the route, $CV(r)$ is the total number of locations that after being served result in overloading the vehicle in the current route. Finally, $TD(r)$ is the total amount of delay if the vehicle arrives at a location later than the specified deadline; $w_1, w_2, w_3 \, and \, w_4$ are weights in the range $[0, 1]$ assigned to each term in the objective function, and $w_1 + w_2 + w_3 + w_4 = 1.0$.

The choice of appropriate weights depends on the importance of each term in the objective function. In this research we found that in order to get feasible solutions, more penalty should be imposed on the time window violations and the total delay than the capacity violations or the total route duration.

5 The Genetic Algorithm

Several problem-specific genetic operators were considered potentially suitable for this kind of problem. The first genetic operator we tried follows the merge crossover operators suggested in (Blanton and Wainwright, 1993) for the vehicle routing with time window problems, and used in (Jih and Hsu, 1999, 2004) for the PDPTW. Unlike traditional crossover operators for ordering problems, which depend only on the order of genes in a chromosome, the merge crossover operators depend on a global precedence among genes, such as the time window or distance ordering.

Traditional order-based crossover operators are not very effective for highly constrained problems like the PDPTW problem, since they frequently produce infeasible solutions. Merge crossover operators, however, were shown to be superior to the traditional ones for these types of problems (Blanton and Wainwright, 1993; Jih and Hsu, 1999, 2004).

In the current research we have slightly modified the **MX1** operator used by (Jih and Hsu, 1999, 2004). Instead of creating just one child, giving priority to the parent's gene having an earlier time window lower bound, we have created two children: the first child favoring genes that have an

earlier lower bound, while the second child favors genes that have an earlier upper bound. The idea is that visiting a location just before its deadline could be more beneficial than visiting it as early as possible in its allowed interval. This may help to reduce the waiting time that would result if the vehicle arrives too early at a location, and, as a consequence, could reduce the total route duration. Creating two children instead of one was suggested in (Blanton and Wainwright, 1993) and may also improve the quality of the new generation and speedup the optimization process.

To illustrate how the MX1 operator works, assume that the following vector defines the precedence order, in terms of the lower bound of the time window, among all pickup and delivery locations, (for clarity, pickups are followed by a $_+$ and deliveries are followed by a -)
(2_+ 1- 3_+ 1_+ 4- 2- 3- 4_+)
Now, assume we have the following two parent solutions:
P1: 2_+ 1_+ 3_+ 3- 1- 2- 4_+ 4-
P2: 3_+ 1_+ 1- 2_+ 2- 4_+ 4- 3-
Since 2_+ has a higher precedence than 3_+, the child will inherit 2_+ as the first gene.
C1: 2_+ **- - - - - - -**
To maintain feasibility, 2_+ in P2 will be swapped to the first location.
P1: 2_+ 1_+ 3_+ 3- 1- 2- 4_+ 4-
P2: 2_+ 1_+ 1- 3_+ 2- 4_+ 4- 3-
The second gene in both parents is identical, so it is copied to the child and we move on to the next gene in order.
C1: 2_+ 1_+ **- - - - - -**
1- has a higher precedence than 3_+, so 1- is copied to the child, and 1- is swapped with 3_+ in P1.
P1: 2_+ 1_+ 1- 3- 3_+ 2- 4_+ 4-
P2: 2_+ 1_+ 1- 3_+ 2- 4_+ 4- 3-
C1: 2_+ 1_+ 1- **- - - - -**
Continuing in the same manner, we obtain the child:
C1: 2_+ 1_+ 1- 3_+ 2- 3- 4- 4_+
The last 2 genes are out of order, but this is of no concern since the first one is automatically considered as the pickup. So, the child in its final form will be:
C1: **2 1** *1* **3** *2 3* **4** *4*
The second child is created in a similar manner but with the precedence vector defined by the upper bound of time window intervals instead.

Two mutation operators were tested in our genetic algorithm. The first is a random gene swap mutation, which selects 2 genes at random and

swaps them. The use of duplicate gene encoding eliminates the possibility of infeasible solutions, in terms of precedence order, that may result following such a swap.

We also implemented a new problem-oriented mutation operator, named **directed mutation**. As mentioned above, to deal with the hard time window constraint, a mutation operator could attempt to bring a location that may be more urgent earlier in the visiting order. Instead of a traditional random swap of two genes, this new mutation operator only swaps genes if they are out of order in terms of their late time window bounds, i.e., if the later one has a deadline that precedes the earlier one. We chose the upper bound of the time window as our mutation guidance to minimize the infeasibility that could result if the vehicle arrives at a location after its deadline.

The genetic algorithm was implemented in C++ with the aid of an MIT genetic algorithm library GALIB (Wall, 1996). A steady state GA was selected, with a replacement percentage of 100%, a population size of 1000, crossover rate of 1.0, and a chromosome mutation rate of 0.4[2].

6 The Simulated Annealing

The theoretical foundation of simulated annealing (SA) was established in (Kirkpatrick et al., 1983), and the theory is based on the statistical mechanics of physical systems. The term simulated annealing is adopted from the annealing of solids where we try to minimize the energy of the system using slow cooling until the atoms reach a stable state. The initial temperature and the rate at which the temperature is reduced together make up the annealing schedule. In solving a combinatorial optimization problem using an SA, we start with an arbitrary solution to the problem. We then try to optimize this solution using a method analogous to the annealing of solids. A neighbor of this solution is generated using an appropriate method, and the cost (or the fitness) of the new solution is calculated. If the new solution is better than the current solution in terms of reducing cost (or increasing fitness) the new solution is accepted. However, if the new solution is not better than the current solution, the new solution is accepted with a certain probability which is usually set to $\exp^{(-\Delta/T)}$, where Δ is the change in cost between the new and the old solution and T is the current temper-

[2]A higher than usual mutation rate was found necessary to avoid being trapped in a local optimum. One reason could be the duplicate encoding, a side effect of which is that mutation may swap identical genes producing the same offspring.

ature. Thus, the procedure is less likely to get stuck in a local optimum since bad moves still have a chance of being accepted. The annealing temperature is first chosen to be high so that the probability of acceptance will also be high, and almost all new solutions are accepted. The temperature is then gradually reduced so that the probability of acceptance will be very low and the algorithm works more or less like hill climbing. The process is repeated until the temperature approaches zero or no further improvement can be achieved, which is analogous to the atoms of the solid reaching a crystallized state (Rutenbar, 1989).

To construct an initial solution for the PDPTW, some researchers choose to apply different heuristic techniques to create a route with minimum constraint violations, for example (Bruggen et al., 1993; Landrieu et al., 2001), and then try to improve this solution using a heuristic or a metaheuristic technique. In our research, we construct an initial solution s by simply generating a large number of random routes and selecting the route with minimum cost as our starting solution[3]. This technique is straightforward to implement and also computationally inexpensive. Although the quality of the initial solution may be very poor, the main focus of the algorithm is on the route improvement phase, accomplished during the SA search. A good neighborhood move should be capable of producing a high quality solution at the end of the SA search, irrespective of the starting solution.

The choice of an appropriate annealing schedule is critical to the performance of SA. Ideally it is desirable to devise a scheme that is adaptable for all test cases and problem sizes, eliminating the need for (arbitrary) parameter tuning, which can be very time consuming. Following (Dorband et al., 2004), we created the annealing parameters for each test case individually as shown in Algorithm 1. To generate a neighboring solution s' in this algorithm, a simple random swap between two different locations in s is performed.

The main SA procedure is described in Algorithm 2. The starting solution s for the main SA algorithm is the same as the starting solution used to calculate the annealing parameters. This time, however, to get a new state s' from the current state s, an intelligent neighborhood move, that depends on the upper bound of the time window, is used. First two locations in the current solution are selected at random. Then, these two locations are swapped, only if their deadlines are out of order, i.e., if the later has a more urgent deadline than the earlier one. This is exactly the same idea adopted in the directed mutation used in the GA, and its purpose is to

[3] 10000 random routes were initially generated.

Algorithm 1 Calculating Annealing Parameters

Generate a starting solution s
{Initialize $Pstart$ the starting acceptance probability and $Pend$ the final acceptance probability}
Let $Pstart$ = a large value {For example 0.999}
Let $Pend$ = a small value { For example 0.001}
$\Delta_{avg} \leftarrow 0$
{Generate n neighboring solutions of s}
for $(i = 0; i < n; i++)$ **do**
  Select two random locations in s
  Swap the current 2 locations in s to get a new solution s'
  $\Delta \leftarrow |cost(s') - cost(s)|$
  $\Delta_{avg} \leftarrow \Delta_{avg} + \Delta$
$\Delta_{avg} \leftarrow \Delta_{avg}/n$
$T_0 \leftarrow -\Delta_{avg}/\log(Pstart)$ {$T_0$is the initial temperature}
$T_N \leftarrow -\Delta_{avg}/\log(Pend)$ {$T_N$is the final temperature}
$\alpha \leftarrow \exp^{(\log(T_N)-\log(T_0))/N}$ {α is the temperature reduction factor, and N is the number of iterations desired}

Algorithm 2 The Main SA Algorithm

Start with an initial solution s
$T \leftarrow T_0$ {Initialize the current temperature}
repeat
  Select two random locations in s
  if (The later location is more urgent in its time window) **then**
    Swap the current 2 locations in s to get a new solution s'
    $\Delta \leftarrow cost(s') - cost(s)$
    if $(\Delta < 0)$ **then**
      Replace s with s'
      $T \leftarrow \alpha \times T$ {Reduce the current temperature}
    else
      With probability $\exp^{(-\Delta/T)}$
      Replace s with s'
until (Frozen){Stop when no improvement is achieved for a pre-specified number of iterations}

arrange requests in a way that will best satisfy the timing constraint.

During any iteration of the main SA algorithm, if the currently selected locations are not out of sequence in terms of their deadlines, however, no swap of locations will take place. As a result, the current solution s will remain intact, and no change in cost is evaluated. The temperature value T also remains unchanged and is carried forward to the next iteration. In fact, as will be explained shortly, the present temperature value is only reduced following the replacement of the current solution s with a new improved solution s'.

Nevertheless, since the final solution obtained after the main SA procedure may still be a low quality or an infeasible solution, we extended our algorithm with two further SA stages, in order to more fully exploit all the guidance that the time windows can give us. The second stage adopts a neighborhood move depending on the *lower bound* of the time window instead of the upper bound, while the third stage uses a neighborhood move based on the *center* of the time window interval. Each of these different neighborhood moves may help introduce some improvement to the fitness of the current solution, for example by reducing the total delay or the total waiting time, which could ultimately lead to obtaining high quality solutions. Each new SA stage starts from the final solution reached by the end of the previous stage, and its starting temperature is the final temperature reached by the previous stage. We chose to start with a route improvement move that is guided by the upper bound of time window, in order to reduce the total number of time window violations in the route by visiting the more urgent requests first. However, a different improvement order could also be attempted.

In the first two stages of the SA procedure (in which the neighborhood move depends on the upper and then the lower bound of the time window) the temperature reduction schedule was slow. As shown in Algorithm 2, we chose to reduce the temperature only when a better solution was found, i.e, several neighboring solutions are explored for the same temperature value. The idea is to allow the algorithm a thorough exploration of the neighborhood by accepting a large number of worse moves during the early stages of the search. However, during the final stage of the SA (the one that adopts the center of the time window to perturb the solution), the solution was approaching stability, and there was a danger of loosing the best solution obtained if the temperature was slowly reduced. Thus, during this final stage, the temperature reduction was made fast. This was achieved by reducing the temperature at every iteration of the search, which will make the chance of accepting worse solutions very slim.

To evaluate the performance of the 3-stage SA algorithm described above, it was compared with another simple SA algorithm in which the same annealing schedule and a slow cooling is used, but the neighborhood move was a simple random swap of locations that does not take the timing order into consideration. The comparison results are reported in Section 8.2.

7 The Hill Climbing

The final approach we used to solve the PDPTW problem is a simple hill climbing heuristic (HC). The algorithm basically has two phases: a route construction phase, and a route improvement phase. As with our SA algorithm, the initial solution is simply created by generating a large number of random solutions and selecting the best generated (minimum cost) solution. Again this procedure will avoid unnecessary complications and increased processing time that may result if we try to generate a high quality solution with minimum infeasibility. A guided route improvement phase, which repeatedly replaces the current solution with better solutions generated during the search, should be able to transform the starting low quality and possibly infeasible solution to a high quality and feasible solution. Our main hill climbing algorithm is described in Algorithm 3.

Algorithm 3 The Main HC Algorithm

```
1: Start with an initial solution s
2: repeat
3:   for (Each possible pair of locations in s) do
4:     if (The later location is more urgent in its time window) then
5:       Swap the current 2 locations in s to get a new solution s'
6:       Δ ← cost(s') − cost(s)
7:       if (Δ < 0) then
8:         Replace s with s'
9: until (Frozen){Stop when no improvement is achieved for a pre-specified
   number of iterations}
```

Again, in our hill climbing, we adopt the same solution representation and neighborhood moves that were used in both the GA and the SA. Similar to the SA algorithm, the HC algorithm was divided into three stages. The first stage generates a neighboring solution by swapping the two locations currently under consideration, only if their deadlines (i.e., the upper bound of the time windows) are out of order. On the other hand, if the deadlines are

in sequence, the current solution remains intact, and two new locations are considered in the next iteration. When no further improvement is possible using this neighborhood move, the second stage starts from the final solution achieved in the first stage and repeats the same HC procedure, but with a neighborhood move that adopts the lower bound of the time window to decide the swapping. Finally, the third stage starts with the final solution obtained in the second stage, but with a neighborhood move that swaps locations if they are out of order in terms of the center of the time window interval.

8 Computational Experimentation

8.1 Test Data

In our previous work (Hosny and Mumford, 2007), the suggested genetic algorithm was tested on a data set obtained from the authors of (Jih and Hsu, 2004). This data set has a number of customer requests ranging from 10 to 100 (20 to 200 locations). In addition, for a more extensive and thorough testing of the operators, we also created in (Hosny and Mumford, 2007) a new data set with larger numbers of customer requests ranging from 100 to 200 (200 to 400 locations).

To create test data for the PDPTW, it is essential to ensure the existence of at least one solution that satisfies all problem constraints. Similar to (Jih and Hsu, 2004), our algorithm first creates a route that respects precedence and capacity constraints, then a time window interval is generated for each location based on the arrival time realized in the created route. Unlike (Jih and Hsu, 2004), however, we rely on our GA to create the initial feasible solution, while they create their starting feasible solution randomly.

The following steps were followed to create the test data:

1. Generate a random vehicle capacity within a certain predetermined range.
2. Generate random $x\,and\,y$ coordinates for the depot, in the range $[0, 200]$.
3. For all pickup and delivery locations:
   (a) Generate random $x\,and\,y$ coordinates, each in the range $[0, 200]$.
   (b) Generate a random demand (load) within a certain allowable range, such that the demand of a delivery is the same as the demand of the corresponding pickup but with a negative sign.

(c) Assume a very large time window interval that could not possibly be violated.

4. Run the genetic algorithm to obtain a feasible solution. Note that, the precedence constraint is always satisfied in any generated solution, thanks to our duplicate gene encoding. Moreover, due to the nature of the problem, the satisfaction of the capacity constraint can be easily accomplished by our genetic operators. The time window constraint can also be easily satisfied because the TW intervals are very large at this point.

5. Calculate the arrival time at each location in the feasible route obtained.

6. Create a random time window interval for each location such that the arrival time falls within the created time window. The width of the permitted time window interval should be determined in advance[4].

8.2 Experimental Results

To test our three algorithms, we used two data set samples. The first sample, which we will call **SET 1**, is the sample obtained from (Jih and Hsu, 2004) and includes 30, 80, 90 and 100 customer requests [5]. The second sample, which we will call **SET 2**, is obtained from the data set created by us in (Hosny and Mumford, 2007), and includes 130, 170, and 200 customer requests [6]. Note that the number of locations is always double the number of requests.
Each of the following algorithms was run 10 times on each test case:

1. The GA with MX1 crossover and random swap mutation (GA1).

2. The GA with directed mutation only (GA2) [7].

[4]The allowable ranges for the random values were determined empirically. Due to lack of space, no more elaboration on the ranges can be given here. However, all the ranges were scaled according to the number of requests currently generated, i.e., the larger the number of requests, the larger the allowable range for the vehicle capacity, the demand and the width of time window interval.

[5]This data set can be downloaded from:
http://wrjih.wordpress.com/2006/12/09/pdptw-test-data/

[6]This data set can be downloaded from:
http://users.cs.cf.ac.uk/M.I.Hosny/PDP.zip

[7]The selection of the genetic operators to be tested was based on their promising performance in our previous work(Hosny and Mumford, 2007).

3. The 3-stage SA (SA1).

4. The simple random-move SA (SA2).

5. The hill climbing algorithm (HC).

The results for the 10 runs are recorded in Table 1 as follows: for each test case under each algorithm, the best result found (in terms of total route duration only), and the percentage of feasible solutions obtained during the 10 runs. The last column of the table shows the previous best known results, reported in (Jih and Hsu, 2004) and (Hosny and Mumford, 2007).

Table 2 shows the processing time in seconds needed to obtain the best result in each test case under the different algorithms tested. The processor used was a Pentium (R) 3.40 GHz processor.

The best results were achieved by the 3-stage SA (SA1) as shown in Table 1. In all test cases the best results achieved by this version of SA were better than the previous best known results. The results were also superior to the results achieved by all the other algorithms tested in the current research. A very high feasibility rate and good quality solutions were obtained in all tasks tested, and also in a very reasonable amount of time, as indicated by Table 2. For example the best result was achieved in less than 90 seconds for the largest task of 200 requests (400 locations).

It appears that this algorithm is able to escape the trap of local optima and gradually find better solutions by progressing from one stage to another. Each stage of the SA seems to contribute to the improvement process. To illustrate this, consider the best result obtained for our largest task of 200 requests: The starting solution before the beginning of the 3-stage SA had: total route duration = 52969.3, number of capacity violations = 64, and number of time window violations= 387. After the first stage of the SA, the solution obtained had: total route duration = 37768.2, number of capacity violations = 0, and number of time window violations= 344. After the second stage of the SA, the solution obtained had: total route duration = 23841.1, number of capacity violations = 0, and number of time window violations= 3. After the third stage of the SA, the solution obtained had: total route duration = 23841.1, number of capacity violations = 0, and number of time window violations= 0.

It also appears in Table 1, that the best results obtained by both versions of the GA were slightly worse than the results obtained by the random-move SA and the HC in most test cases, although the GAs achieved a slightly better feasibility ratio. Moreover, Table 2 shows that, in terms of processing time, both GA versions were very slow in reaching their best results, which

Table 1: Results Summary

Data	Task	GA1		GA2		SA1		SA2		HC		Prev Best
		Best	%Feas	Best	%Feas	Best	%Feas	Best	%Feas	Best	%Feas	
SET1	30	3738.73	100	3751.21	100	3299.4	100	3684.21	100	3772.82	70	3696.51
	80	7838.21	100	7867.68	100	7267	100	7852.25	80	7847.67	90	7838.21
	90	8619.01	100	8619.01	100	8292.67	100	8619.01	100	8671.54	100	8618
	100	10600.1	100	10600.1	90	10544.6	90	10611.1	70	10600.1	90	10600
SET2	130	14041	100	14343	100	13826	100	14081.2	90	14031.1	100	13856.2
	170	20618.1	100	20333.8	100	19690.9	100	19964.8	100	20003.6	100	19861.3
	200	25799.5	100	24730	100	23841.1	100	24461.6	100	24304.3	100	24512.8

Table 2: Processing Time in Seconds

Data Set	Task	GA1	GA2	SA1	SA2	HC
SET 1	30	11	15	6	6	1
	80	75	74	35	13	5
	90	15	78	24	17	5
	100	14	90	53	15	6
SET 2	130	673	570	21	25	8
	170	507	970	71	49	14
	200	1302	791	87	59	21

is expected due to the need to maintain a large population of individuals throughout the search. The SA and HC algorithms, on the other hand, progress from the focus of a single individual.

Figure 1 bears out the above observations. This graph shows the average objective value (as calculated by Equation 1) achieved during the 10 runs, by each of the algorithms tested. For all test cases, the graph demonstrates the superiority of the 3-stage SA, since its average objective value is the lowest compared to the other algorithms tested. From this graph, we can also notice that the hill climbing algorithm seems to achieve a better average solution quality than the SA algorithm that uses the random swaps (SA2) in some test cases, for example tasks 100, 130 and 200. Despite this observation, Table 1 indicates that some best results obtained by the HC were worse than the best results obtained by the random-move SA.

It should be noted that infeasible solutions are seldom produced by most of our algorithms. However, since an infeasible solution usually has a very large objective value, the presence of one or more such solutions often results in a large increase in the average objective value (as an example, notice the average objective value produced by the random-move SA algorithm (SA2) for tasks 100 and 130 shown in Figure 1).

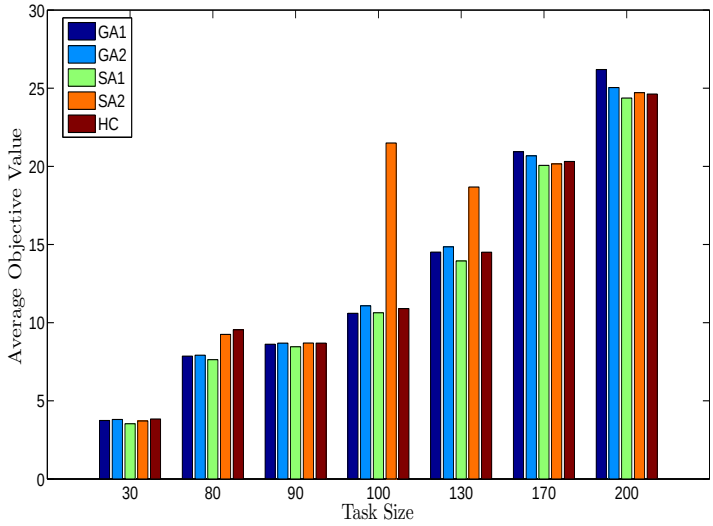

Figure 1: Average Objective Value for All Algorithms

Similar to the 3-stage SA, the three stages applied in the HC algorithm seem to contribute to guiding the search towards better solutions. To see this, consider again the largest task of 200 requests: The starting solution before the beginning of the 3-stage HC had: total route duration = 55013, number of capacity violations = 117, and number of time window violations= 387. After the first stage of the HC, the solution obtained had: total route duration = 26462.9, number of capacity violations = 1, and number of time window violations= 1. After the second stage of the HC, the solution obtained had: total route duration = 24315.9, number of capacity violations = 0, and number of time window violations= 0. After the third stage of the HC, the solution obtained had: total route duration = 24304.3, number of capacity violations = 0, and number of time window violations= 0.

Figure 2 shows the best run of the hill climbing algorithm (HC) vs. the best run of 3-stage SA (SA1) algorithm for the largest task of 200 requests. The graph demonstrates the current objective value, as calculated by Equation 1, in each iteration of the run. It seems in this graph that the 3-stage SA was able to explore a wider area of the search space, albeit with a larger number of iterations needed to reach convergence, and inevitably longer processing time.

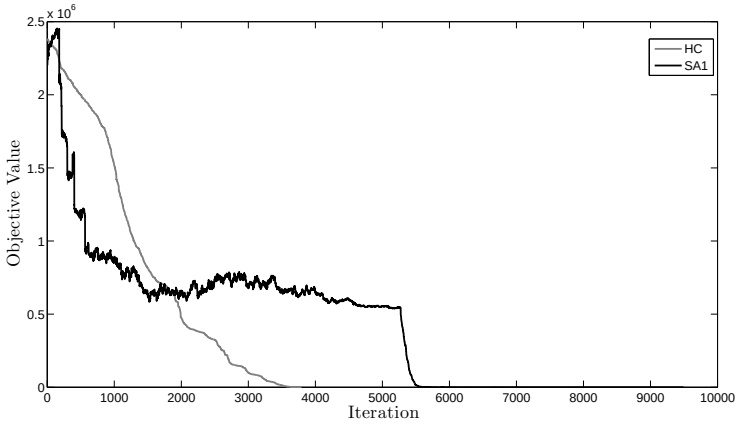

Figure 2: Hill Climbing vs. 3-Stage SA(SA1) for Task 200

Figure 3 shows the best run of the random-move SA algorithm (SA2) vs. the best run of the 3-stage SA (SA1), again showing the current objective value in each iteration of the run for the 200-requests task. In this graph, it is clear that the 3-stage SA was immediately directed towards lower cost solutions during the early phases of the run. The random-move SA, however, spent quite a long time during these early phases investigating low quality solutions, and only started discovering the promising areas of the search almost halfway through the run. However, since the random move SA algorithm seems to reach convergence earlier in the run, the processing time of the 3-stage SA, in most test cases, was longer than the processing time of the random-move SA, which can be seen in Table 2.

Figure 4 compares the average processing time for our fastest algorithms, the 3-stage SA (SA1), the random-move SA (SA2) and the HC, during the 10 runs for all tasks tested in the research. This graph shows that the HC algorithm has the fastest average processing time among the three algorithms in all test cases.

Figure 5 shows box plots for all our algorithms pertaining to the 200-requests task. An analysis of variance shows that the five algorithms produce significantly different results at $p = 0.0001$, and the graph indicates the superiority of the 3-stage SA.

Moreover, a post analysis of variance was performed on our algorithms, using a Tukey HSD test, to compare the resulting sample means for the 200-requests task. The test results indicated that the 3-stage SA produced

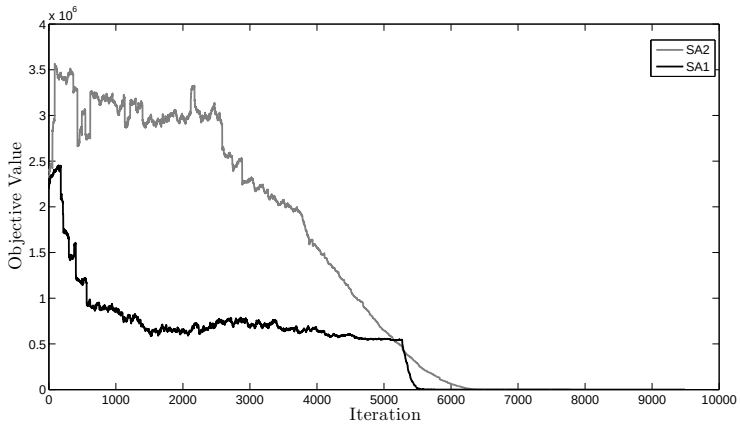

Figure 3: Random-move SA(SA2) vs. 3-Stage SA(SA1) for Task 200

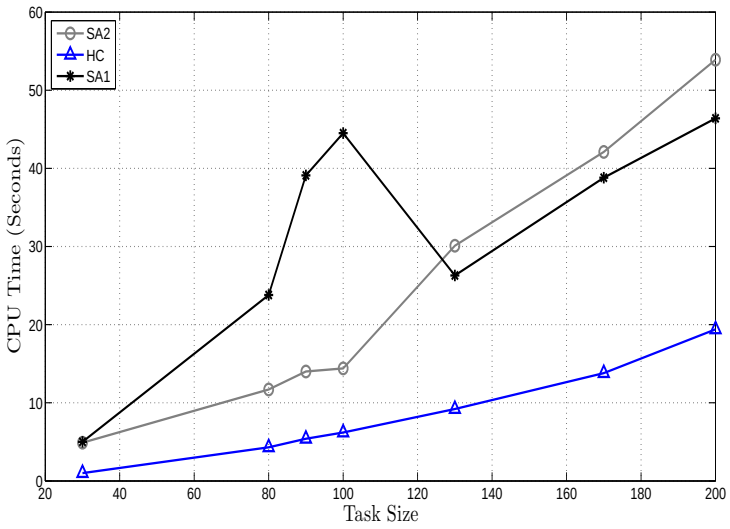

Figure 4: Average CPU Time For SA1, SA2 & HC

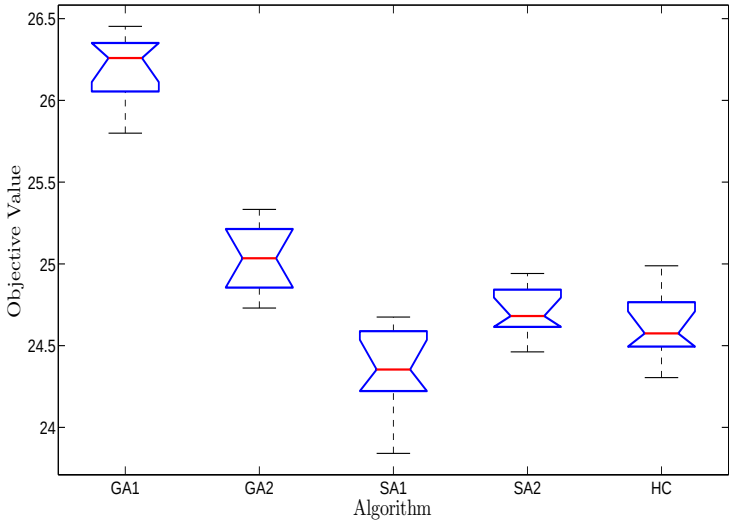

Figure 5: Analysis of Variance for All Algorithms (Task 200)

a mean value that is significantly different, beyond the 0.01 level, from all mean values produced by the other algorithms, with only one exception, the mean produced by the HC algorithm. This can be explained by realizing that both algorithms operated in a similar manner, adopting three stages and employing intelligent neighborhood moves during the search. Their final mean result, for this particular task, did not show a significant difference. All other mean comparisons, pertaining to the 200-requests task, showed that the mean produced by the 3-stage SA was significantly better than all other means.

9 Conclusions and Future Work

In this research, we investigated three different approaches to the single vehicle pickup and delivery problem with time windows. First, a genetic algorithm approach equipped with problem-specific genetic operators was implemented. One GA version had a time window directed merge crossover operator working together with a random swap mutation, while the other depended only on an intelligent directed mutation that is guided by the time windows, without any crossover operator. Second, two versions of

simulated annealing were tested: the first version operated in three stages and adopted neighborhood moves that are guided by the time windows, and a second SA version adopted a random unguided neighborhood move. The third approach is a hill climbing heuristic which also operated in a manner similar to the 3-stage SA, but only accepted better solutions encountered during the search.

The experimental results on two data set samples indicated that both GA versions had more or less comparable results in terms of both quality and processing speed. This could only indicate that the directed mutation operator is an intelligent operator that can guide the search towards better solutions without the help of any crossover, which is usually the major GA operator. However, the performance of the GA in general was considerably inferior to the other algorithms tested in this research, in terms of the quality of the solution in most test cases, and more so in terms of the processing speed.

On the other hand, the 3-stage SA seems to be superior to all other algorithms tested in this research in terms of the quality of the solutions obtained. Moreover, its best results were also able to beat the best known results from previous research on the problem in all test cases. However, this SA version was slightly slower than the random-move SA version and the hill climbing heuristic tested.

The success of the 3-stage SA is possibly due to its dependence on intelligent neighborhood moves that were able to transform a random low quality starting solution to a feasible high quality solution, within a reasonable amount of time. These neighborhood moves were also successful in the context of hill climbing but with a less dramatic effect than their effect in the context of simulated annealing, possibly due to the trap of local optima. However, using hill climbing can still give us acceptable quality solutions in a very short processing time, which may be preferable in real world applications.

Our future work will further examine the 3-stage SA algorithm, possibly by attempting a different order of stages, or modifying the initial solution strategy or the annealing schedule. We also plan to broaden our scope and start investigating different heuristics and metaheuristics for the more general multiple vehicle pickup and delivery problem. The work done so far for the single vehicle case could be easily incorporated into our future plan of study.

Acknowledgement

The first author's PhD program in Cardiff University, U.K. is funded by King Saud University (KSU), Riyadh, Kingdom of Saudi Arabia.

References

Blanton, J. and Wainwright, R. (1993). Multiple vehicle routing with time and capacity constraints using genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 452–459, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Bruggen, L. J. J. V. D., Lenstra, J., and Schuur, P. (1993). Variable-depth search for the single-vehicle pickup and delivery problem with time windows. *Transportation Sience*, 27(3):298–311.

Desrosiers, J., Dumas, Y., and Soumis, F. (1986). A dynamic programming solution of the large-scale single vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6(3-4).

Diana, M. and Dessouky, M. (2004). A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows. *Transportation Research Part B: Methodological*, 38(6):539–557.

Dorband, J., Mumford, C., and Wang, P. (2004). Developing an ace solution for two-dimentional strip packing. In *18th International Parallel and Distributed Processing Symposium Workshop on Massively Parallel Processing*. Santa Fe, New Mexico.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Healy, P. and Moll, R. (1995). A new extension of local search applied to the dial-a-ride problem. *European Journal of Operational Research*, 83(1):83–104.

Hosny, M. I. and Mumford, C. L. (2007). Single vehicle pickup and delivery with time windows: made to measure genetic encoding and operators. In *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, pages 2489–2496, New York, NY, USA. ACM Press.

Jih, W. and Hsu, Y. (1999). Dynamic vehicle routing using hybrid genetic algorithms. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, volume 1, pages 453–458. Detroit, Michigan.

Jih, W. and Hsu, Y. (2004). A family competition genetic algorithm for the pickup and delivery problems with time window. *Bulletin of the College of Engineering, N.T.U.*, 90:8998.

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

Kirkpatrick, S., Gelatt, C. D., J., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598):671–680.

Landrieu, A., Mati, Y., and Binder, Z. (2001). A tabu search heuristic for the single vehicle pickup and delivery problem with time windows. *Journal of Intelligent Manufacturing*, 12(5):497–508.

Lau, H. and Liang, Z. (2001). Pickup and delivery with time windows: algorithms and test case generation. In *Proceedings of the 13th International Conference on Tools with Artificial Intelligence*, pages 333–340.

Li, H. and Lim, A. (2001). 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. Dallas, TX, USA.

Lu, Q. and Dessouky, M. M. (2006). A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, 175(2):672–687.

Man, K., Tang, K., and Kwong, S. (1996). Genetic algorithms: concepts and applications [in engineering design]. *Industrial Electronics, IEEE Transactions on*, 43(5):519–534.

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

Psaraftis, H. (1983). An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportaion Science*, 17(3):351–357.

Renaud, J., Boctor, F. F., and Quenniche, J. (2000). A heuristic for the pickup and delivery traveling salesman problem. *Computers and Operations Research*, 27(9):905–916.

Rutenbar, R. (1989). Simulated annealing algorithms: an overview. *Circuits and Devices Magazine, IEEE*, 5(1):19–26.

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

Tam, V. and Tseng, L. C. (2003). Effective heuristics to solve pickup and delivery problems with time windows. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, pages 184–188.

Transport-Statistics (2006). Road freight statistics 2005. A national statistics publication produced for the department of transport, U.K. (www.dft.gov.uk/transtat).

Wall, M. (1996). Galib: A C++ library of genetic algorithm components. Mechanical Engineering Department, MIT. (http://lancet.mit.edu/ga).