# Single Vehicle Pickup and Delivery with Time Windows: Made to Measure Genetic Encoding and Operators

Manar I. Hosny
M.I.Hosny@cs.cardiff.ac.uk

Christine L. Mumford
C.L.Mumford@cs.cardiff.ac.uk

Cardiff School of Computer Science
Cardiff University, Queen's Buildings, 5 The Parade, Roath, Cardiff
CF24 3AA, UK

## ABSTRACT

To the best of our knowledge, only a few researchers have experimented with genetic algorithms (GAs) to tackle the single vehicle pickup and delivery problem with time windows, possibly due to the large number of constraints involved and the difficulty in handling them. In particular, there is the difficulty in designing an appropriate genetic representation and intelligent genetic operators that are able to transfer the ordering characteristic of the parents to the offspring, while preserving the feasibility of the solution. In this research, we will experiment with a genetic encoding and operators specially designed to deal with the problem in hand. We will present a *duplicate gene encoding* that guarantees the satisfaction of the the precedence constraint, between the pickup and the delivery requests, throughout the search. We aim to show that GAs, if guided by some problem-specific information, will be able to handle this hard problem and possibly other similarly highly constrained problems.

## Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search—*Heuristic methods*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Routing and Layout*

## General Terms

Performance

## Keywords

Pickup and Delivery with Time Windows, Dial-a-Ride, Genetic Algorithms, Vehicle Routing

## 1. INTRODUCTION

The single vehicle pickup and delivery problem with time windows ($PDPTW$) is an important practical problem, having applications such as mail delivery, newspaper distribution, school bus routing and the transportation of employees. Indeed, in the view of the present authors the PDPTW will assume even greater prominence in the future, if we are to design more efficient transport systems and reduce the environmental impact of the ever increasing volumes of goods transported on our roads. Greater cooperation between the various stakeholders in manufacturing and distribution supply chains could significantly reduce the number of "empty running" journeys undertaken by heavy goods vehicles, for example, which is currently recorded at 27.4 % by UK government statistics in [19].

The 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.

The PDPTW is known to be *NP-Hard*, with the presence of time windows making the problem particularly complicated. Since exact algorithms are too slow for large problem sizes, heuristic and meta-heuristic approaches seem to be good alternatives. Amongst meta-heuristics, 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. Thus, exploring GAs to solve the PDPTW would seem to be a justified option. Besides their ability to widely explore the search space, GAs are easy to implement and do not depend so much on the quality of the initial solution as some other heuristic and meta-heuristic techniques. Nevertheless, we aim to explore alternative heuristics and meta-heuristic options as part of our future research. The present paper focusses only on the first stage of our planned programme of work.

## 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 problem-specific. One classification distinguishes *static* from *dynamic* problems:


*GECCO'07,* July 7–11, 2007, London, England, United Kingdom.

static problems requiring that all requests are determined in advance of the route construction process, with dynamic allowing some requests to arrive during the execution of the route(s).

Some PDP problems have neither capacity nor time window constraints, for example [16]. Some researchers, as will be detailed shortly, deal with only a single vehicle, while others deal with a more general multiple vehicle case [10, 11, 12, 13, 18]. 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 [4, 5, 8].

As previously mentioned, approaches to handle the PDP problems tend to be rather problem-specific: 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. Examples of approximation algorithms for the multiple vehicle case are the evolutionary approach of [2] and the grouping genetic algorithm of [14].

For the single vehicle case, an exact algorithm is the dynamic programming approach of Psaraftis in [15]. However this technique has a time complexity of $O(n^2 3^n)$, and for this reason is limited to solving small problems of up to about 10 requests (20 locations). On the other hand, Desrosiers, Dumas and Soumis in [3] solve the single vehicle dial-a-ride problem, with precedence, capacity and time windows constraints, exactly using dynamic programming. This algorithm can solve problem sizes 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.

Approximation algorithms, in which a heuristic or a metaheuristic is developed to deal with the problem, make it possible to cope with much larger problem sizes even than this.

For the single vehicle PDPTW, the heuristic in [20] 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, 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, time constraints may be violated as long as precedence and capacity constraints are satisfied. In addition, a simulated annealing alternative approach is used as a precaution against poor or infeasible solutions that may result from the above heuristic algorithm.

The work reported in [7] 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 [1]. The mutation operator is applied only when the offspring is identical to one of its parents.

In a more recent work, the same genetic operators used in [7] are examined in the context of a family competition genetic algorithm (FCGA) [6]. 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.

The authors in [9] present a tabu search to solve the single vehicle PDPTW. The algorithm first creates a route respecting precedence and capacity constraints, using a simple insertion heuristic. Then, two local search methods, simple and probabilistic tabu search, are conducted to construct a feasible route with minimum distance traveled. To construct an initial route, pickup locations are sorted in increasing order of earlier time window intervals. Locations are then inserted in the route as they are encountered in that order, with a delivery location inserted immediately after its corresponding pickup location, while taking the capacity constraint into consideration. The generated route may be infeasible in terms of the time window constraint, however. A tabu search is then conducted to improve the initial solution. Two tabu search heuristic algorithms were applied: a deterministic tabu search, in which classical neighborhood moves are performed, and a probabilistic tabu search, which 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.

For the interested reader [17] provides an excellent and detailed survey on the general pickup and delivery problem and existing techniques to handle it in the literature.

# 3. THE SINGLE VEHICLE PDPTW

Assume we have a set of nodes $N = \{n_0, n_1, n_2, ..., n_m\}$, $n_0$ denotes the depot and the last index $m$ is an even number. 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 $[e_i, l_i]$ during which the location must be served.

For each possible edge $< n_i, n_j >$ a distance $d_{ij}$ or a travel time $t_{ij}$ are specified. Only edges satisfying the time window ($TW$) constraint should be considered, i.e., $t_{0i} + t_{ij} < l_j$.

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 TW $e_i$, 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

In this research we investigate the potential of using genetic algorithms to solve the single vehicle pickup and delivery problem with time windows. In particular, we experiment with a *duplicate gene encoding* that guarantees the satisfaction of the the precedence constraint, between the pickup and the delivery, throughout the search. In addition, several problem-specific genetic operators are tested and compared on a number of data sets ranging from 10 to 200 requests. The operators tested are: a modified 2-child merge crossover operator guided by two time window precedence criteria, a new crossover operator that depends on the order of pickup and delivery locations appearing in parent solutions, a simple gene swap mutation, and a time window oriented swap mutation. The results obtained indicate 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. These operators were able to obtain feasible solutions even for large data sets tested in our research.

## 4.1 The Encoding

The genetic encoding of this kind of problem is not as obvious as it seems. The PDP problem is an ordering problem in which a genetic solution (a chromosome) could be encoded 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 encoding of a 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 a genetic operator to the chromosome. For example, any simple genetic operator (like a simple gene swap mutation) 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; and this, of course, would increase the processing time and complicate the algorithm.

We have developed a genetic encoding 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. This simple and obvious encoding will eliminate the problem of backtracking to repair an infeasible solution, and solves the precedence constraint issue in a simple and effective way. As a result, more effort can be directed towards harder constraints such as capacities and time windows. An example of a chromosome with 4 requests following this encoding is: (**2 1** *1* **3 4** *2 3 4*), where pickups are shown in boldface and deliveries in italics.

[1]It is assumed here that the vehicle's journey is open path, to enable a comparison with the results reported in [6] that follow this assumption. An alternative would be for the vehicle to return to the depot after servicing all requests.

## 4.2 The Fitness Function

Following [6], the fitness function treats the constraints as soft constraints, meaning that an infeasible solution that violates either the capacity and/or the time window constraint will be penalized by an added term in the fitness function. The fitness function of a route $r$ is:

$$F(r) = w_1 \times D(r) + w_2 \times TWV(r) + w_3 \times CV(r)$$

Where $D(r)$ is the total route duration including the waiting time, $TWV(r)$ is the total number of time window violations in the route, and $CV(r)$ is the total number of capacity violations; $w_1, w_2,$ *and* $w_3$ are weights in the range $[0, 1]$ assigned to each term in the fitness function, and $w_1 + w_2 + w_3 = 1$.

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

## 4.3 The Operators

### *4.3.1 Crossover*

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 [1] for the the vehicle routing with time window problems, and used in [6] for the PDPTW. Unlike traditional crossover operators for ordering problems, which depend only on chromosome contents and local ordering of genes in the 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 in these types of problems [1, 6, 7].

In the current research we have slightly modified the **MX1** operator used by [6]. 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 [1] 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 lower bound of 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.

In addition, we designed a new problem-specific crossover operator for the PDP problem. The idea is that, since any crossover operator roughly tries to create a child that inherits half of its genes from the first parent and half of its genes from the second parent, we will try to transmit pickup locations to the child in an order close to the order of their appearance in the first parent, and try to transmit delivery locations to the child in an order close to the order of their appearance in the second parent. A second child is created by reversing the role of parents. This kind of crossover might prove useful in satisfying the capacity constraint in particular, since the satisfaction of this constraint is mainly dependent on the ordering of pickups and deliveries. To illustrate this crossover operator, which we will call **PDPX**, consider the following two parent solutions:
P1: **2 1 3** *3 1 2* **4** *4*
P2: **3 1** *1* **2** *2* **4** *4 3*
First all pickups (first occurrences of genes) will be copied to the child in the exact order and locations as the first parent.
C1: **2 1 3 - - - 4 -**
Now, we want to take the order of delivery locations from the second parent. Since a delivery is always the second occurrence, we will start processing parent 2 from the last gene rather than the first. If we encounter a gene that appeared in the child only once, i.e., it has already been picked up, we will copy it in the first available location in the child (also processed starting from the last gene). If, on the other hand, we encounter a gene that has already appeared twice in the child, this gene will be ignored because pickup and delivery has already occurred for this gene, and we move on to the next gene in order. Following this we obtain the child:
C1: **2 1 3** *1 2* **4** *4 3*
the second child is obtained by taking the pickups order from the second parent and the deliveries order from the first parent, as shown below.
C2: **3 1 - 2 - 4 - -**
C2: **3 1** *3* **2** *1* **4** *2 4*

#### 4.3.2 Mutation

We experiment with two mutation operators 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 swap.

We also implemented, a new problem oriented mutation operator, named **directed mutation**. As mentioned above, it appears that the time window constraint is the hardest to satisfy among all other problem constraints, given the precedence constraint is dealt with using the simple duplicate gene encoding, and the capacity constraint is easy to satisfy, because each delivery request represents a negative load that is removed from the vehicle. To deal with the time window constraint, a mutation operator could attempt to bring a location that may be more urgent earlier in the visiting order. This may result in a better ordering of locations, which should be beneficial in satisfying the time window constraint. 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.

### 4.4 The Genetic Algorithm Implementation

The genetic algorithm was implemented in C++ with the aid of an MIT genetic algorithm library, GALIB. A steady state GA was selected, with a replacement percentage of 100% and a population size of 1000. More details will be given of the other important parameters we used for the GA in Section 6.

## 5. TEST DATA

The genetic algorithm was tested on a data set obtained from the authors of [6]. 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 have created 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, we must ensure the existence of at least one solution that satisfies all problem constraints. The following steps were followed to create the test data:

1. For the set of nodes $N = \{n_0, n_1, n_2, ..., n_m\}$, where $m$ is an even index, assume $n_0$ is the depot, 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 associated delivery location $n_{i+(m/2)}$.
2. Generate a random vehicle capacity within a certain predetermined range.
3. Generate random $x$ *and* $y$ coordinates for the depot, in the range $[0, 200]$.

4. 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 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.

5. Run the genetic algorithm to obtain a feasible solution. Note that, due to the nature of the problem, the satisfaction of the capacity constraint can be easily accomplished by any of our genetic operators. The time window constraint can also be easily satisfied because the TW intervals are very large at this point.

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

7. Create a random time window interval for each location such that the arrival time falls within the created time window. The width of the time window interval should be determined in advance[2].

# 6. EXPERIMENTAL RESULTS

To test the genetic algorithm, different combinations of crossover and mutation operators were compared on two data set samples. The first sample, which we will call **SET 1**, is the sample obtained from [6] and includes 30, 80, 90 and 100 customer requests. The second sample, which we will call **SET 2**, is obtained from the data set created by us, and includes 130, 170, and 200 customer requests. Note that the number of locations is always double the number of requests.

The algorithm was run 10 times on each test case, with a crossover rate of 1.0 and a mutation rate of 0.4[3]. The following combinations of crossover and mutation operators were compared:

1. MX1 crossover and random swap mutation.
2. MX1 crossover and directed mutation.
3. PDPX crossover and random swap mutation.
4. PDPX crossover and directed mutation.
5. Directed mutation without any crossover.
6. MX1 crossover without any mutation.

[2]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.

[3]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.

The Genetic algorithm was run to convergence or for a maximum of 3000 generations if no convergence can be reached. The results for the 10 runs are recorded as follows, for each test case: the best score found (total route duration), the number of generations needed to reach this score, and the percentage of feasible solutions obtained during the 10 runs. The results are summarized in tables 1 through 6, where the last column of each table shows the best known result. For SET 1, the best known results are the ones reported in [6][4], while for SET 2, the best known results are the best results achieved by our genetic algorithm. A score followed by a * indicates that this solution is infeasible in terms of time window constraint violation.

As can be observed, the best results are obtained from the MX1 crossover and random swap mutation as shown in table 1. Together they achieved a 100% feasibility rate in all test cases except for the largest task (200 requests) in which the feasibility rate was 80%. In two cases the results obtained were even better than the best known results reported in [6]. These are the results for test cases 30 and 80. Our GA with MX1 combined with random swap also converges quite quickly, in terms of the number of generations needed to reach the best solution. For example, only 13 generations for test case 100.

The worst results are obtained by PDPX crossover and swap mutation, as shown in table 3. These two operators together failed to achieve any feasible results, in terms of time window constraint, even for small sized tasks. This can be explained by noticing that both PDPX crossover and swap mutation lacked any guidance towards the time window ordering. Although they were able to satisfy the capacity constraint in all test cases, they failed to satisfy the time window constraint. Clearly, they worked blindly with respect to the timing order, and consequently were not able to find an ordering that services all requests in their desired intervals.

When directed mutation replaced the random swap mutation, however, the results dramatically improved for PDPX. Table 4 shows that PDPX together with directed mutation were able to obtain feasible solutions in all test cases. The feasibility ratio ranged from 40% for the largest task to 100% for the smallest task. The results, though, were not able to beat the results obtained from the MX1 crossover and swap mutation. Figure 1 shows how directed mutation was able to guide the search towards much better solutions for the largest task tested.

The situation is different, however, when directed mutation replaced random swap mutation with the MX1 crossover. As shown in table 2, these two operators together produced lower quality solutions than those obtained by MX1 crossover with the random swap mutation. Possibly, the introduction of directed mutation here caused the algorithm to rapidly converge to a local optimum without a chance for improvement. Figure 2 bears this out, and shows clearly the rapid conversion of the MX1 crossover with directed mutation compared to the gradual conversion of the MX1 crossover with the random swap mutation. The MX1 crossover seems capable of guiding the GA towards feasible solutions without the help of an "intelligent" mutation operator. Its success is possibly due to the fact that it creates not only one but two children that have a better visiting order than

[4]the results shown are truncated as they appear in [6].

**Table 1: Results for MX1 & Swap Mutation**

| Task | Best | Gener | % Feas | Best Known |
|---|---|---|---|---|
| 30 | 3696.51 | 43 | 100 | 3713 |
| 80 | 7838.21 | 47 | 100 | 7849 |
| 90 | 8619.01 | 15 | 100 | 8618 |
| 100 | 10600.1 | 13 | 100 | 10600 |
| 130 | 13856.2 | 377 | 100 | 13856.2 |
| 170 | 19861.3 | 414 | 100 | 19861.3 |
| 200 | 24512.8 | 919 | 80 | 24512.8 |

**Table 2: Results for MX1 & Directed Mutation**

| Task | Best | Gener | % Feas | Best Known |
|---|---|---|---|---|
| 30 | 3738.73 | 13 | 100 | 3713 |
| 80 | 7858.72 | 13 | 100 | 7849 |
| 90 | 8619.01 | 14 | 100 | 8618 |
| 100 | 10600.1 | 12 | 100 | 10600 |
| 130 | 15913.1 | 167 | 20 | 13856.2 |
| 170 | 20674.5 | 705 | 60 | 19861.3 |
| 200 | 26058.1 | 300 | 60 | 24512.8 |

their parents. These children are biased towards the more urgent requests in terms of the allowed service time. All that this crossover needs is a simple random swap to introduce a bit of a diversity in the population. Any more intelligent interference does not seem to help and actually appears to drive the search away from the better solutions. The importance of a random swap mutation to help introduce diversity can also be shown in table 6 in which MX1 worked alone without any mutation. The results show that MX1 failed to find some previously achieved optimum results in the small tasks (SET 1), and even failed to achieve any feasible results in all large tasks (SET 2).

Finally, when directed mutation was tested without any crossover, the results obtained were, to some extent, surprising. As shown in table 5, this mutation alone was able to obtain feasible solutions for all test cases. The feasibility rate was 90% or 100% for all test cases except for the largest one in which the feasibility rate was 70%. Although the quality of the results obtained are slightly worse than the those obtained when directed mutation operated together with one of the two crossovers tested, the feasibility ratio indicates that this mutation is an intelligent operator that has a great potential. Clearly, it was able to obtain good quality and feasible solutions without the help of any crossover operator, albeit with a larger number of generations. It should be noted, though, that when directed mutation was tested with MX1 crossover, only one swap movement was needed to inject the necessary level of diversity and to reduce the chance of being trapped in a local optimum. On the other hand, when directed mutation was tested alone or with the PDPX crossover, it was found that a larger number of swaps was needed to achieve good results. For these cases, the number of swaps was taken to be a random number between 1 and the number of customer requests in the data set.

Table 7 shows processing time in seconds needed to obtain best and feasible solutions for data sets 100 and 200, on a Pentium (R) 3.40 GHz processor.

**Table 3: Results for PDPX & Swap Mutation**

| Task | Best | Gener | % Feas | Best Known |
|---|---|---|---|---|
| 30 | 3863.27* | 633 | 0 | 3713 |
| 80 | 9163.92* | 932 | 0 | 7849 |
| 90 | 10057* | 1958 | 0 | 8618 |
| 100 | 11886.1* | 1482 | 0 | 10600 |
| 130 | 19185.4* | 1000 | 0 | 13856.2 |
| 170 | 25817.1* | 2000 | 0 | 19861.3 |
| 200 | 32009.7* | 2000 | 0 | 24512.8 |

**Table 4: Results for PDPX & Directed Mutation**

| Task | Best | Gener | % Feas | Best Known |
|---|---|---|---|---|
| 30 | 3734.54 | 46 | 100 | 3713 |
| 80 | 7838.21 | 294 | 100 | 7849 |
| 90 | 8623.23 | 375 | 100 | 8618 |
| 100 | 10608.2 | 634 | 80 | 10600 |
| 130 | 14791.7 | 789 | 60 | 13856.2 |
| 170 | 20359.3 | 1418 | 90 | 19861.3 |
| 200 | 24911.2 | 2245 | 40 | 24512.8 |

**Table 5: Results for Directed Mutation**

| Task | Best | Gener | % Feas | Best Known |
|---|---|---|---|---|
| 30 | 3747.47 | 32 | 100 | 3713 |
| 80 | 7879.35 | 275 | 90 | 7849 |
| 90 | 8662.2 | 444 | 100 | 8618 |
| 100 | 10608.2 | 405 | 90 | 10600 |
| 130 | 14679.2 | 1854 | 100 | 13856.2 |
| 170 | 20470.2 | 2235 | 90 | 19861.3 |
| 200 | 25091.7 | 2773 | 70 | 24512.8 |

**Table 6: Results for MX1**

| Task | Best | Gener | % Feas | Best Known |
|---|---|---|---|---|
| 30 | 3741.1 | 6 | 100 | 3713 |
| 80 | 7867.68 | 14 | 80 | 7849 |
| 90 | 8619.01 | 12 | 100 | 8618 |
| 100 | 10600.1 | 13 | 90 | 10600 |
| 130 | 21626* | 65 | 0 | 13856.2 |
| 170 | 30575.6* | 46 | 0 | 19861.3 |
| 200 | 37148.5* | 51 | 0 | 24512.8 |

**Table 7: Processing Time for Tasks 100 and 200**

| Operators | Task 100 | Task 200 |
|---|---|---|
| MX1 & Swap | 12 sec | 988 sec |
| MX1 & Directed | 11 sec | 317 sec |
| PDPX & Swap | 1148 sec | no feasible result |
| PDPX& Directed | 493 sec | 2158 sec |
| Directed | 253 sec | 1939 sec |
| MX1 | 12 sec | no feasible result |

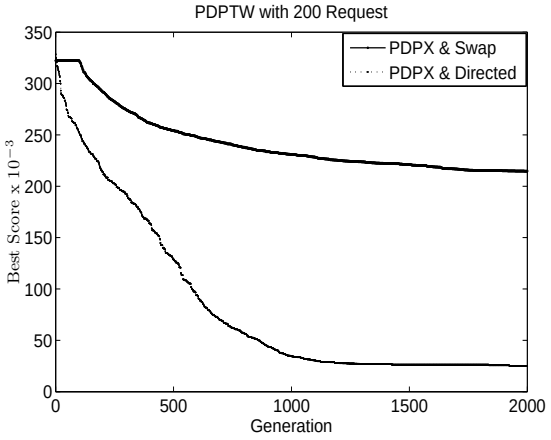


**Figure 1: PDPX & Swap vrs. PDPX & Directed**

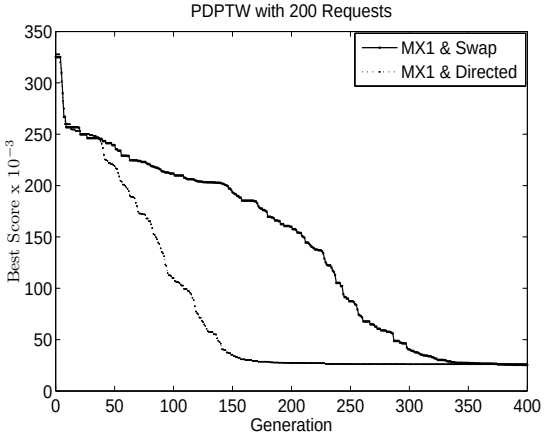


**Figure 2: MX1 & Swap vrs. MX1 & Directed**

# 7. CONCLUSIONS AND FUTURE WORK

This research investigated problem-specific genetic encoding and operators for the single vehicle pickup and delivery problem with time windows, aiming to efficiently conquer the difficult constraints of the problem. The order based representation we used is a duplicate gene encoding that guarantees the satisfaction of the precedence constraint throughout the search, alleviating the need for backtracking to solve the infeasibility that may result following any genetic manipulation.

Four genetic operators were tested: a modified 2-child merge crossover (MX1) guided by precedence of both time window bounds, a new PDPX crossover operator that depends on the order in which the pickup and delivery locations appear in parent solutions, a regular random swap mutation, and a directed mutation that swaps genes according to urgency in terms of the deadline of service time. The experimental results on the two data sets indicate that MX1 crossover and directed mutation are each effective as genetic operators (although they do not work very well together). We believe their success is due to the guidance they take from the time window information. On the other hand, the PDPX crossover was not able to find feasible solutions to any of the test instances, when tested in combination with the random swap mutation. This was probably due to the absence of guidance towards the desired service time. However, the results for PDPX dramatically improved, when used in combination with directed mutation. Indeed, the directed mutation operator seems to show great promise, and would appear to be useful in guiding the search towards feasible solutions, in cases where a crossover operator is disruptive, ineffective or absent. Future research will attempt to investigate the potential of this operator for the PDPTW in the context of other meta-heuristic algorithms, such as simulated annealing and tabu search. We also plan to experiment with simple hill-climbing.

# 8. ACKNOWLEDGMENTS

We would like to thank W. Jih the author of [6] for his cooperation, and for providing us with the data sets to test our algorithm.

# 9. REFERENCES

[1] J. L. Blanton and R. L. Wainwright. Multiple vehicle routing with time and capacity constraints using genetic algorithms. In *Proceedings of the Fifth International conference on Genetic Algorithms and their Applications*, pages 452–459, 1993.

[2] J.-C. Créput, A. Koukam, J. Kozlak, and J. Lukasik. An evolutionary approach to pickup and delivery problem with time windows. *Lecture Notes in Computer Science*, 3038:1102–1108, 2004.

[3] J. Desrosiers, Y. Dumas, and F. Soumis. A dynamic programming solution of the large-scale single vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6:301–325, 1986.

[4] M. Diana and M. Dessouky. 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, 2004.

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

[6] W. Jih and Y. Hsu. A family competition genetic algorithm for the pickup and delivery problems with time window. *Bulletin of the College of Engineering*, 90:121–130.

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

[8] R. M. Jorgensen, J. Larsen, and K. B. Bergvinsdottir. Solving the dial-a-ride problem using genetic algorithms. *Journal of Operational Research Society*, 2006.

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

[10] H. Lau and Z. Liang. 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, November 2001.

[11] 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 (ICTAI)*, 2001. Dallas, TX, USA.

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

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

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

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

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

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

[18] V. Tam and L. C. Tseng. Effective heuristics to solve pickup and delivery problems with time windows. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, 2003.

[19] Transport Statistics. *Road Freight Statistics 2005*. A national statistics publication produced for the departement of transport, United Kingdom, 29th June 2006. (This document can be downloaded from www.dft.gov.uk/transtat.)

[20] L. J. J. Van-der-Bruggen, J. Lenstra, and P. Schuur. Variable-depth search for the single-vehicle pickup and delivery problem with time windows. *Transportation Sience*, 27:298–311, 1993.