Pick-up and delivery with due times and time windows

Georgina Carmody and Philip Neame

Department of Mathematical Sciences,
University of Technology, Sydney.

Philip.Neame@uts.edu.au

Abstract:

We consider a variant on a pick-up and delivery problem with time windows, where both early and late delivery of items is penalized. This problem arises from an application in the area of defence, but has wider applications in delivery of perishable goods. We develop a heuristic for this problem — decomposing the problem into a higher level problem assigning items to be delivered to trips of vehicles (which we solve using tabu search) and a lower level problem choosing optimal delivery times for these trips (which we solve by an exact iterative approach). We compare this approach with an integer programming model. For quite small instances of the problem, the associated integer programming problem cannot be solved, but testing on small problems suggests the heuristic is a promising approach for finding good solutions.

1 Introduction

We consider a problem with sets of sources and destinations and distinct items which must be delivered from a specific source to a specific destination, using any of a variety of vehicles (potentially with different speeds and capacities). The vehicles must travel from a source to a destination and then to another source — strings of destinations are not allowed. Each vehicle may be able to carry several items at one time, but the total weight of the load affects the speed of the vehicle. Each destination has associated with it a series of time windows — delivery may not take place outside these times. Each item has a due time, and penalties occur if that item is delivered either before or after that due time. The main aim is to minimise the maximum earliness or lateness of deliveries. If several possible schedules are equal under this objective, then it is desirable to minimize the total expense of travel for the vehicles across these solutions.

This problem arises in a defence application, and was originally suggested by the Defence Science and Technology Organization. The exact details of that application are classified. However, it could have applications and extensions in a number of logistics areas, for example, where early delivery leads to spoilage, and storage and/or security costs.

In Figure 1, we show a possible sequence of trips for one vehicle. A full solution would also give arrival times at each of the nodes. Note that it is possible for a vehicle to make several deliveries between two nodes (if the due times are sufficiently far apart), and that each item is unique and has a specified source and destination point.

In Section 2 of this paper, we compare this problem to other problems studied in the literature, and then introduce notation and develop an integer programming formulation in Section 3. In Section 4, we develop a heuristic solution approach for this problem based on tabu search. We then present some preliminary numerical results in Section 5 before making brief concluding remarks in Section 6.

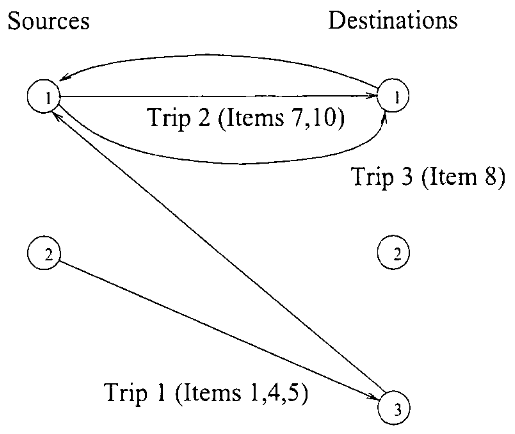

Figure 1: A possible schedule of trips for one of the vehicles.

2 Literature Review

The problem we consider has some similarities to pick-up and delivery problems (see e.g. Bent and Van Hentenryck [1]) although it appears that the objective in this paper is novel. In particular, whilst it is common to penalize lateness, it appears that delivery problems with a penalty for early or late deliveries have not been studied previously. A similar penalty has been used in machine scheduling (see e.g. Chrétienne [3]), but only on a single machine. Another feature of our problem is that delivering to several destinations in succession is prohibited, which is unusual. Hence, we refer to the problem considered as a Single-Destination Due-Time-Related Pickup and Delivery Problem with Time Windows (SDPDPTW).

Many similar problems have been approached with heuristic techniques. For example, Lau et al. [8] looked at the vehicle routing problem with time windows and a limited number of vehicles; Psarras et al. [10] used local search (along with constraint programming) to find good solutions to a fleet scheduling problem; and Tuzun and Burke [12] used tabu search on a location routing problem.

We have selected to apply a tabu search heuristic to the problem. Tabu search is a meta-heuristic which was introduced by Glover [6], and is often used to direct the operations of other heuristic procedures. Using local search methods within each iteration of tabu search, the current solution is updated, even if it is moving to a solution with an increased objective. This forces the solution to move away from a local optimum if necessary. Incorporated in the use of memory is a "tabu list" which records the inverse transformations or reverse moves associated with the moves actually performed in the search process. This prevents the solution from moving back to a previously visited solution for a certain number of iterations and also helps to avoid solution cycling. The notions of forcing a solution update at each iteration and the use of a tabu list strategy guide the search away from local optimal solutions and help explore new parts of the solution space.

There have also been attempts to find exact solutions to some delivery problems, for example, Desaulniers et al. [4] solve a multi-depot vehicle scheduling problems with time windows and waiting costs using integer programming (in particular, column generation). Note, though, that this problem is still substantially simpler than the SDPDPTW since each vehicle transports a single item for each trip.

The SDPDPTW was not found to have been studied before. The most similar problem found with a practical application developed was by Horn [7]. This paper describes a software system developed to manage the deployment of a fleet of demand responsive passenger vehicles. It is a pick-up and delivery problem with time windows, but the objective is to minimise costs or total distance travelled. However, the demand is not necessarily predetermined in Horn; requests are made during the operation and frequent periodic updates are made the schedule. Other differences include the different use of time windows, different load (or passenger) requirements, and different periodic demand requests. In the approximation algorithm Horn uses a steepest descent method with periodic local search to overcome local minima from steepest descent. Horn mentions the possible use of more advanced algorithms such as tabu search and simulated annealing but for a constantly changing schedule the time taken is not practical.

In Horn (as with most applications with time windows) each job has a single time window in which a job or part of a job should or must take place. Typically, a penalty function for violation of time window constraints is used. In the SDPDPTW each job has multiple time windows, determined by their destination points, and the delivery of goods can take place within any of these time windows. It is impossible to use a simple penalty function in this case, as the penalty may guide the solution towards a time window which is not the best to use.

Tan et al. [11] compare three metaheuristics (tabu search, simulated annealing and genetic algorithms) for the vehicle routing problem with time windows, which has some similarities to the SDPDPTW. The genetic algorithm approach requires the solution to be of a specific chromosome structure whereas simulated annealing and tabu search are much more flexible with the solution structure. Tan et al. found that the simulated annealing process can take a very long time to converge for problems with a complex objective function. This suggests that tabu search would be easier to implement and quicker in obtaining good solutions to the SDPDPTW and thus an appropriate heuristic to use.

There were no findings of heuristic methods developed to solve any variants of the vehicle routing problem, fleet scheduling problem, or pick-up and delivery problem with due time related objectives.

An advantage of the adversity tabu search is that a well structured heuristic is effective in finding good solutions to varying types and size of problems. Implementations of extensions and variations to the tabu search heuristic have also shown to be successful such as in Tuzun and Burke [12] where a two-phase tabu search approach is used to solve the location routing problem. Therefore we can be fairly confident that tabu search is a good first approach to take for the SDPDPTW.

The SDPDPTW is NP-complete, since we can reduce the 3-Partition problem to an instance of the SDPDPTW . In particular, suppose we have one source point, one destination point, one vehicle (of capacity b), no time windows and $3t$ items with weights a_j to deliver. Furthermore, suppose the time to deliver a load of items is 1 unit (regardless of the size of the load) and all items are due at time 0. Finally, suppose

$$\frac{b}{4} \leq a_j \leq \frac{b}{2} \text{ for all } i = 1, \ldots, 3t, \text{ and}$$
$$\sum_{j=1}^{3t} a_j = bt.$$

Then the question "is there an optimal solution with value t?" is equivalent to the question "can we partition the integers a_j into subsets of three elements adding to b?" This is the 3-Partition problem, which is known to be NP-complete (see e.g. Papadimitriou and Steiglitz [9])

Some problems, whilst NP-complete, have been successfully solved to a provably optimal solution for large problem instances (for example, the travelling salesman problem with 24 978 cities as outlined in [14]). However, the SDPDPTW is very complicated, and an initial attempt at solving this problem with integer programming can find optimal solutions only for small instances. Thus, the focus of this paper is on finding good approximate solutions. This will be done by using tabu search.

3 An Integer Programming Model for the SDPDPTW

To provide some comparisons for our heuristic, we develop an integer programming model for the SDPDPTW. For some small instances, this can be solved to get an exact solution which can be compared with the best solution provided by the heuristic. Note that to obtain a linear model, we assume that the speed of a vehicle is unaffected by the total weight of the load it is carrying — this is a simplification of the actual problem, and is another reason why we choose to use a heuristic for the SDPDPTW.

Firstly, we define notation for some of the elements of the problem. We consider an instance of the problem with n items and V vehicles with capacity C_v and speed s_v. For each item, we define

$$\begin{aligned} d_i &= \text{ due time of item } i, \text{ and} \\ c_i &= \text{ actual delivery time of item } i. \end{aligned}$$

We define a set $\mathcal{D}$ of delivery arcs, going from a source to a destination. Note that an arc is only included in $\mathcal{D}$ if there is an item to be delivered from that source to that destination node. We also define a set $\mathcal{R}$ of return arcs, which includes all possible arcs from destination nodes to source nodes. We define a trip taken by a vehicle to traverse one delivery arc. We assume each vehicle can take up to T trips.

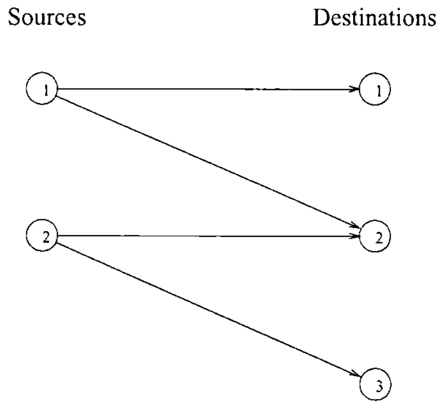

Figure 2: Delivery Arcs

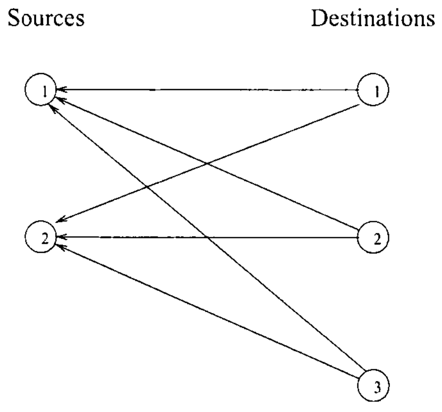

Figure 3: Return Arcs

Then we define the variables:

$$\begin{aligned} x_{ivt} &= \begin{cases} 1, & \text{if item } i \text{ is delivered on trip } t \text{ of vehicle } v, \\ 0, & \text{otherwise.} \end{cases} \\ a_{vt} &= \text{the arrival time of trip } t \text{ of vehicle } v \text{ at its destination.} \end{aligned}$$

Furthermore, we need to ensure that all items on a given trip are going between the same source and destination pair, and also calculate travel times not only on delivery arcs, but also on return arcs. If we label the set of delivery arcs by $j = 1,\ldots,J$, we can partition the set of items into J subsets $I_1,\ldots,I_J$, where I_j is made up of all items to be delivered along delivery arc j.

For each delivery arc, we record the distance between its source and destination as D_j. We also need to record the length of all return arcs, which we do by calculating $\hat{D}_{jk}$ as the distance from the destination of delivery arc j to the source of delivery arc k, for any combination of delivery arcs j and k. These calculations are performed before solution of the integer program begins.

Each trip must be assigned to exactly one delivery arc — the assignment is controlled by the variable y_{vtj}, where:

$$y_{vtj} = \begin{cases} 1, & \text{if trip } t \text{ of vehicle } v \text{ is on delivery arc } j, \\ 0, & \text{otherwise.} \end{cases}$$

To calculate the correct time taken to return from a destination before the next trip, we consider another binary variable to link consecutive trips on each vehicle:

$$z_{vtjk} = \begin{cases} 1, & \text{if vehicle } v \text{ has trip } t \text{ on delivery arc } j \text{ and trip } t+1 \text{ on delivery arc } k, \\ 0, & \text{otherwise.} \end{cases}$$

Note that if trip t is the last one for vehicle v then z_{vtjk} is not meaningful.

Finally, we take M to be a sufficiently large number.

For ease of exposition, we have omitted the time windows — since there are multiple time windows for each destination point, this would require binary variables for each trip and each time window, further complicating the formulation.

Thus an integer programming formulation for a simplified version of the SDPDPTW (with vehicle speed independent of the weight of the load, and omitting time windows) is:

$$\min \quad \eta \tag{1}$$

subject to:

$$\eta \geq \alpha_i + \beta_i \text{ for } i = 1,\ldots,n \tag{2}$$

$$\alpha_i - \beta_i = c_i - d_i \text{ for } i = 1,\ldots,n \tag{3}$$

$$\sum_{v=1}^{V}\sum_{t=1}^{T} x_{ivt} = 1 \text{ for } i = 1,\ldots,n \tag{4}$$

$$\sum_{i=1}^{n} x_{ivt} \leq C_v \text{ for } v = 1,\ldots,V,\ t = 1,\ldots,T \tag{5}$$

$$\sum_{j=1}^{J} y_{vtj} \leq 1 \text{ for } v = 1,\ldots,V,\ t = 1,\ldots,T \tag{6}$$

$$\sum_{i\in I_j} x_{ivt} \leq My_{vtj} \text{ for } j = 1,\ldots,J, v = 1,\ldots,V, t = 1,\ldots,T \tag{7}$$

$$y_{vtj} \leq \sum_{i\in I_j} x_{ivt} \text{ for } v = 1,\ldots,V,\ t = 1,\ldots,T,\ j = 1,\ldots,J \tag{8}$$

$$\sum_{j=1}^{J} y_{vtj} \geq \sum_{j=1}^{J} y_{v(t+1)j} \text{ for } v = 1,\ldots,V,\ t = 1,\ldots,T \tag{9}$$

$$\begin{aligned}
\sum_{k=1}^{J} z_{vtjk} &= y_{vtj} \text{ for } v=1,\dots,V, t=1,\dots,T, j=1,\dots,J && (10)\\
\sum_{j=1}^{J} z_{vtjk} &= y_{v(t+1)k} \text{ for } v=1,\dots,V, t=1,\dots,T-1, k=1,\dots,J && (11)\\
a_{v1} &\geq \frac{1}{s_v}\left(\sum_{j=1}^{J} D_j y_{v1j}\right) \text{ for } v=1,\dots,V && (12)\\
a_{vt} &\geq a_{v(t-1)} + \frac{1}{s_v}\left(\sum_{j=1}^{J} D_j y_{vtj} + \sum_{j=1}^{J}\sum_{k=1}^{J} \hat{D}_{jk} z_{v(t-1)jk}\right) && (13)\\
& \qquad \text{for } v=1,\dots,V,\ t=1,\dots,T &&\\
a_{vt} - c_i &\leq M(1-x_{ivt}) \text{ for } i=1,\dots,n,\ v=1,\dots,V,\ t=1,\dots,T && (14)\\
c_i - a_{vt} &\leq M(1-x_{ivt}) \text{ for } i=1,\dots,n,\ v=1,\dots,V,\ t=1,\dots,T && (15)\\
\alpha_i, \beta_i, c_i, a_{vt} &\geq 0 &&\\
x_{ivt}, y_{vtj}, z_{vtjk} & \quad \text{binary.} &&
\end{aligned}$$

The objective is to minimize $\max_{i=1,\dots,n} |c_i - d_i|$, which is expressed through (1) – (3). Constraint (4) ensures that all items must be assigned to a trip on a vehicle and constraint (5) ensures that a vehicle never carries more than its capacity.

Constraint (6) ensures that a trip is assigned to exactly one delivery arc, whilst constraint (7) ensures that only items to be delivered on that arc can be assigned to that trip. Constraint (8) requires y_{vtj} to be zero if no items are delivered on that trip and arc. Constraint (9) is not necessary, but does reduce the symmetry in the problem, thus making it easier to solve with branch and bound — it ensures that if a vehicle v takes $q < T$ trips, then they are recorded as trips $1,\dots,q$.

Constraints (10) and (11) force z_{vtjk} to reflect that delivery arc k really does follow delivery arc j in the list of trips that vehicle v makes.

Constraints (12) make sure that items on the first trip of a vehicle cannot be delivered quicker than a vehicle can travel. The most complicated set of constraints (13) relate the fact that, after the first trip and all subsequent trips that a vehicle makes, it must return to the next source and then travel to the destination. Finally, (14) and (15) ensure that the arrival time of each trip is the same as the delivery times of the items on that trip.

Obviously, the size of this integer programming formulation grows very rapidly as the number of items, source and destination points, and/or vehicles grows. Thus, even relatively small instances of the SDPDPTW can lead to a very large integer programming problem.

It may very well be possible to produce a formulation for this problem using fewer variables and constraints, although it should be noted that this does not necessarily mean such formulations would lead to improved solution times (see Wolsey [13]). Improved formulations are one source of future research.

It may also be possible to decompose the problem in an intelligent manner — for example, with a column generation algorithm, it might be possible to have a column consisting of a set of trips (and items carried) for a given vehicle. Then the master problem would be to ensure that all items are delivered in such a way as to minimize the deviation from due time. This is somewhat reminiscent of the aircrew scheduling problem, where the column generation subproblem assigns aircrew to a sequence of trips (flight legs). This is another direction to be explored in future research.

4 A Tabu Search Algorithm for the SDPDPTW

It may eventually be possible to solve reasonable-sized instances of the SDPDPTW exactly, but as an initial approach, we developed a heuristic based on Tabu Search. Further details of this heuristic (including pseudocode) can be found in [2].

The key feature of our approach is a decomposition which allows an efficient search of the feasible space. The tabu search aims to find a feasible assignment of items to vehicles and trips (see Figure 4 for an example assignment with 3 vehicles and 11 items). It does not consider the times at which each trip should depart. In order to calculate the objective for a given assignment, we need to know departure and arrival times, so these are calculated using an iterative approach that determines an optimal schedule given the assignments.

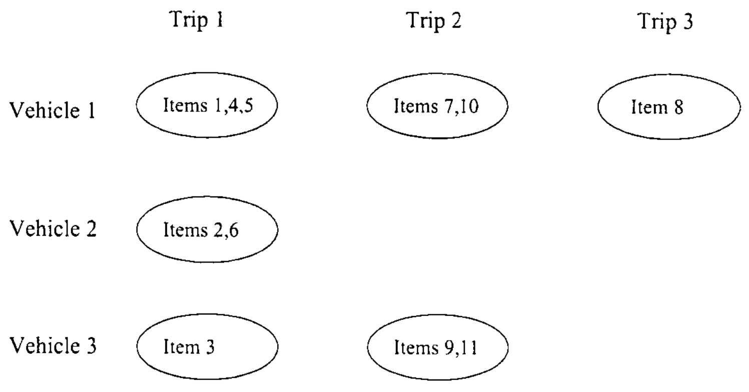

Figure 4: A sample assignment of items to vehicles and trips.

4.1 Calculating Departure and Arrival Times

An important part of the algorithm is to calculate optimal departure and arrival times for each trip of each vehicle, given an assignment of items to all vehicles and trips. This part of the algorithm arises from the particular objective function considered here but could be altered to consider a different objective.

Each vehicle can be considered separately for this calculation (since the items have already been assigned to vehicles). We thus consider a *vehicle objective* η_v for vehicle v, which is the maximum deviation from due time of any item assigned to vehicle v. Then the overall objective is $\max_v \eta_v$.

Due to the existence of time windows, it may not be obvious as to when the trips should be scheduled. Thus it is necessary to use an iterative approach to find an optimal schedule.

The first step of the process is (for each vehicle) to calculate the earliest possible delivery time for each trip, allowing for time windows. If any item is late, then the lateness is a lower bound on the optimal vehicle objective, so set z_L to the maximum lateness. If no item is late, $z_L = 0$. Set z_U to the maximum deviation from due time, which is an upper bound on the vehicle objective (since this is a feasible solution).

Then the latest possible arrival times for each trip are calculated, allowing for time windows. If any item is early, then the earliness e is a lower bound on the optimal vehicle objective, so set $z_L = \max\{e, z_L\}$. If there is a feasible solution with maximum deviation from due time for any time less than z_U, then set z_U to the value of this solution.

Then in an iterative manner, attempt to find a solution with vehicle objective equal to $z_A = \frac{z_L+z_U}{2}$. This is done by trying a forward pass — attempting to make trips as early as possible, such that no trip is more than z_A units early, then trying a backward pass — attempting to make trips as late as possible, such that no trip is more than z_A units late. Define l as the maximum lateness of any item on the forward pass, and e as the maximum earliness of any item on the backward pass. Each pass gives us a feasible solution (and hence the objective is an upper bound on the optimal vehicle objective), and also gives a lower bound, as summarized in Table 4.1.

pass	max earliness	max lateness	upper bound	lower bound
forward	z_A	l	$\max\{z_A, l\}$	$\min\{z_a, l\}$
backward	e	z_A	$\max\{e, z_A\}$	$\min\{e, z_a\}$

Table 1: Bounds on Solution

Thus, the best upper bound obtained is

$$\begin{aligned} z_U &= \min\{\max\{z_A, l\}, \max\{e, z_A\}\} \\ &= \max\{z_A, \min\{e, l\}\}, \end{aligned}$$

and the best lower bound obtained is

$$\begin{aligned} z_L &= \max\{\min\{z_A, l\}, \min\{e, z_A\}\} \\ &= \min\{z_A, \max\{e, l\}\}. \end{aligned}$$

This method continues until the upper and lower bounds are within some given value of each other, i.e. $z_U - z_L \leq \varepsilon$. Hence η_v is obtained.

This is a finite algorithm — in the worst case, this algorithm could converge like binary search, but on the small problems we tested, it seldom took more than a few iterations to converge.

4.2 Generating an Initial Solution

The initial assignment of items to vehicles and trips is done by a greedy algorithm. The items are sorted into increasing order of due time, and then assigned to a vehicle and trip in order. It is also possible to sort the items into decreasing order of weight, which seemed to perform better on some problems where vehicle speed is greatly affected by the weight of the load it is carrying.

To begin with, there are no trips. For each item i, the algorithm considers adding it to an already existing trip, or adding a new trip. To check if item i can be added to an existing trip we first check that the trip has the correct source point and destination point, and that item i can be added without violating the weight constraint for that vehicle. If this is satisfied and the item can be added without changing the vehicle's objective η_v, then the item is added and we proceed to consider the next item. Otherwise, we consider this as a possibility, but check to see whether a better objective can be obtained by creating a new trip carrying just item i, or by adding the item to other vehicles.

Obviously, the greedy algorithm does not always perform well — it often has trouble placing the last few items (since no planning has been made for these items). In order to ensure that a "feasible" solution is generated, we consider an extra time window for each destination point, opening a long time after the final due time and staying open forever. If this time window is used, then the cost of such a solution will be very large (this is equivalent to adding a penalty for violation of the time window constraints).

4.3 Defining Neighbourhoods

An important factor in tabu search is determining an appropriate and effective neighbourhood for the search of improving solutions.

With the SDPDPTW, we require different neighbourhood to the standard travelling salesman problem, or vehicle routing problems, as each node (source/destination points) may have to be visited more than once. There are multiple vehicle types with different possible vehicle delivery combinations. Also there is the extra decision of how many and/or which items to carry on each trip. Therefore, based on the feasible solution structure, the following six "neighbouring moves" are used:

1. **Two Trip Swap:** Swap the vehicle and trip sequence positions of two trips, from either the same or different vehicles.

2. **Reinsertion of a Trip:** Remove a trip from the schedule of one vehicle and insert it between two consecutive trips of a different vehicle.

3. **Joining Two Trips:** Join together the items of two trips originating from the same source point which have the same destination point to form one trip in the position of either of the original trips.

4. **Separation of a Trip:** Split all the items of one trip into two separate trips, with items separated subject to due date proximity. Keep one of the new trips in the position of the original trip and insert the other trip in a feasible insertion position of either the same or some other vehicle.

5. **Item Transfer:** Remove one item from the load of one trip (containing more than one item) and add it into the load of some other trip, originating from the same source point and having the same destination point.

6. **Vehicle Swap:** Reallocate all of the trips from one vehicle to another vehicle of different type and vice versa, keeping the order of all trips the same. (This is not valid for a data set with only one vehicle type.)

4.4 The Tabu Lists

In order to escape from local minima, the tabu search heuristic accepts neighbouring solutions with higher objective values, and attempts to make the reverse move "tabu" (or disallowed). This is complicated by the fact that it may be possible to return to the same solution via a sequence of moves (see Glover [6] for more details).

The tabu search stores a list of moves that are not allowed in a "tabu list". The tabu list has a limited length, so moves remain tabu for a limited period of time. We chose to implement several tabu lists — one for each type of move (this allows great flexibility as each list can be of a different length). The trips can not be characterized by a unique identification number since the load of a trip may change throughout the search process due to other swap types. Therefore to keep track of which trip has been moved we keep a record of the first item on the trip.

We prevent the direct reversal of any move. For example, if a trip is separated, the relevant tabu list stores an item number from each of the two new trips and prevents those two items being combined by a join trip move (whilst this restriction remains on the tabu list).

4.5 Strategy for the Tabu Search

The success of a tabu search algorithm depends largely on the structure of the heuristic. The aim is to efficiently search the solution space. Some aspects of the tabu search strategy implemented include:

- Control of the neighbourhood search — which neighbouring moves to try and when.
- Aspiration criteria — when to accept a move to a neighbouring solution.
- Limiting the search space within each type of neighbouring move.
- Diversification scheme — jumping to other parts of the solution space.

The main strategy used for the control of the neighbourhood search starts with an initial feasible solution. We automatically accept moves that improve a vehicle objective, even if they do not improve the overall objective.

We begin by using neighbouring move types 1 and 2, swapping and reinserting trips, until no further improvement can be obtained. Then an item transfer (move type 5) is considered. If the best item transfer reduces a vehicle objective (without increasing any other vehicle's objective) then the solution is updated. Then we consider move types 1 and 2 again until no further improvement can be obtained. Then the joining and separation of trips (move types 3 and 4) are considered. Similarly, if a vehicle objective is reduced, then the solution is updated and we return to move types 1 and 2.

Eventually, we will reach a local minimum with no further improvement possible. When this occurs a non-improving reinsertion or separation move is performed on the vehicle with the maximum objective. Once out of the local minimum, the whole search process is continually repeated until a stopping criterion is met. The stopping criteria used were a maximum number of total tabu search iterations, and a maximum number of iterations since the best solution has been updated.

Another aspect of aspiration criteria used, as in most tabu search applications, is that at any iteration if a move listed as tabu results in a new best solution, then the tabu status is removed for that iteration and the solution updated.

It is undesirable to unnecessarily search bad solution spaces as this is costly in terms of computational time. Thus by limiting the search space to a "good" subset of neighbouring moves, the efficiency of the heuristic is improved. This is done by performing simple due time related tests to determine if the objective of a new solution is worth calculating. For example, we do not consider a move which would swap two items with very different due times.

To avoid cycling of solutions in the search, a diversification scheme is implemented. If any local minimum is visited more than once, then a move or series of moves are forced in order to jump to a new solution space and diversify the search. The implementation of this scheme uses a vehicle swap (move type 6), if any feasible swaps exist. Otherwise two or more (depending on the size of the problem) item transfers are forced.

5 Results and Future Work

Limited testing has been done on the algorithm presented in this paper, and much more would be required to suggest that it is a good approach for solving the SDPDPTW. In particular, it is necessary to test it on realistic problems, which the DSTO will do shortly. Unfortunately, the details of these instances are confidential.

The approach has been tested on several small problems. In particular, we attempted to compare the heuristic's output with a provably optimal solution on small problems. The integer programming formulation from Section 3 was implemented in AMPL (see [5]), using CPLEX 8.1 as the solver. However, due to the large number of variables used, even relatively small problems caused problems for the solver. For example, an invented problem with just 14 items, 2 vehicles, 2 source points and 4 destination points was unable to be solved within 12 hours on a 1.6 GHz Pentium 4 with 512MB memory. In particular, CPLEX struggled to find good lower bounds. Further testing using the full range of options with CPLEX, and especially using improved formulations may improve this. However, it seems unlikely that problems with the order of hundreds of items can be solved exactly at this stage.

As the integer programming model could not be solved for even very small problems, we tested the code on a few small data sets where the optimal solution could be obtained by complete enumeration of all possible assignments of items to vehicles and trips. The heuristic found the optimal solution in all such cases (often with the initial feasible solution heuristic described in Section 4.2). However, the largest such problem tested had only 10 items. We also tested a constructed problem instance with 22 items, 3 vehicles, 3 source points and 3 destination points and found that the heuristic found several good solutions, and effectively escaped local minima. As a stopping criterion, we used exactly 500 iterations, which took less than 5 seconds in total.

Obviously, much more testing should be done on this algorithm. In particular, comparisons against other heuristics would be enlightening. However, no other studies of the SDPDPTW are known, so comparing heuristics remains for future work.

Other extensions of this work include using different objectives (for example $\min \sum_i |c_i - d_i|$ instead of $\min \max_i |c_i - d_i|$) and using a secondary objective (for example, if there are many feasible solutions with the same maximum deviation from due time, choose the solution with minimum fuel cost for the vehicles). It would also be interesting to remove the restriction that each vehicle returns to a source after visiting one destination, and compare the heuristic against other approaches for the pick-up and delivery problem with time windows (see e.g. Bent and Van Hentenryck [1]).

6 Conclusions

This paper introduces a novel variation on the pick-up and delivery problem with time windows, where the objective is to minimize the maximum lateness or earliness of any delivery. This problem arose in a defence context, but may have wider application in the delivery of perishable goods. We describe a heuristic approach for solving this problem, based on the tabu search metaheuristic. The key feature of this approach is to decompose the problem into a higher level problem assigning items to be delivered to trips of vehicles (which we solve using tabu search) and a lower level problem choosing optimal delivery times for these trips (which we solve by an exact iterative approach). Whilst much further testing is necessary, testing on very small problems suggests that this algorithm is effective, and worthy of further study.

Acknowledgements

The authors acknowledge the support of the DSTO for this research, and would like to particularly thank Jamie Watson and Vicky Alekson for their assistance. We also thank Rosemary Hooke of Unibusiness Consulting for arranging the project, and Yakov Zinder at UTS for his help in the early stages of the project.

References

[1] Bent R and Van Hentenryck P, "A Two-Stage Hybrid Algorithm for Pickup and Delivery Vehicle Routing Problems with Time Windows", in *Lecture Notes in Computer Science* 2833 (2003) pp.123–137.

[2] Carmody G, "An Approximation Algorithm for a Due Time Related Pick-up and Delivery Problem with Time Windows", Honours Thesis, Department of Mathematical Sciences, University of Technology, Sydney, Nov 2003.

[3] Chrétienne P, "Minimizing the Earliness and Tardiness Cost of a Sequence of Tasks on a Single Machine", *RAIRO Operations Research* 35 (2001) pp.165–187.

[4] Desaulniers G, Lavigne J, Soumis F, "Multi-depot vehicle scheduling problem with time windows and waiting costs", *European Journal of Operations Reasearch* 111 (1998) pp.479–494.

[5] Fourer D, Gay DM, Kernighan BW, *AMPL: A Modelling Language for Mathematical Programming*, Scientific Press, 1993.

[6] Glover F, "A user's guide to tabu search", *Annals of Operations Research* 41 (1993) pp.3–28.

[7] Horn M, "Fleet scheduling and dispatching for demand-responsive passenger services", *Transportation Research Part C* 10 (2002) pp.35–63.

[8] Lau HC, Sim M, Teo KM, "Vehicle routing problem with time windows and a limited number of vehicles", *European Journal of Operations Research* 148 (2003) pp.559-569.

[9] Papadimitriou CH, Steiglitz K, *Combinatorial Optimization: Algorithms and Complexity* Prentice-Hall, 1982.

[10] Psarras J, Stefanitsis E and Christodoulou N, "Combination of local search and CLP in the vehicle-fleet scheduling problem", *European Journal of Operations Research* 98 (1997) pp.512–521.

[11] Tan KC, Lee LH, Zhu QL and Ou K, "Heuristic methods for vehicle routing problem with time windows", *Artificial Intelligence in Engineering* 15 (2001) pp.281–295.

[12] Tuzun D and Burke LI, "A two-phase tabu search approach to the location routing problem", *European Journal of Operations Research* 116 (1999) pp.87–99.

[13] Wolsey LA, *Integer Programming*, Wiley, 1998.

[14] "Solving Travelling Salesman Problems", www.tsp.gatech.edu.

Welcome to ICOTA 6!

22 November, 2004

On behalf of the ICOTA 6 Organizing Committee it is our pleasure to welcome you to the University of Ballarat and the Conference.

The International Conference on Optimization: Techniques and Applications (ICOTA) is the official conference series of the Pacific Optimization Research Activity Group (POP). The group was established in October, 2000 and it currently serves more than 470 members from more than 40 countries.

ICOTA's main goal is to provide an international forum for scientists, researchers, software developers, and practitioners to exchange ideas, present research findings and state-of-the-art solutions, share experiences and open new avenues of research on all issues and topics related to optimization theory and its applications.

The first five conferences in this series were held in Singapore (1987 and 1992), Chengdu (1995), Perth (1998), and Hong Kong (2001).

ICOTA 6 is the largest ICOTA so far: 254 delegates from 31 countries have already registered.

The technical program features 2 keynote lectures, 8 invited lectures and 235 presentations. The proceedings volume contains 120 refereed papers and is distributed to delegates on a CD-ROM.

Four pre-conference workshops/tutorials are scheduled for December 8 and an ILOG workshop is scheduled for December 9.

Selected papers presented at the conference will be published in special issues of the following journals:

- Computational Optimization and Applications
  Guest editors: Liqun Qi and Masao Fukushima

- Optimization Methods and Software
  Guest editors: Alex Rubinov and Adil Bagirov

- Pacific Journal of Optimization
  Guest editors: Lou Cacceta and Kok Lay Teo

The conference is sponsored by the University of Ballarat through the School of Information Technology and Mathematical Sciences and the Centre for Informatics and Applied Optimization, ILOG, Australian Mathematical Science Institute, Western Australia Centre of Excellence in Industrial Optimization (Curtin University), IBM Global service Australia, and Pinter Consulting Services.

We wish you an interesting and stimulating meeting!

Alex Rubinov (Conference Co-Chair)
Moshe Sniedovich (Conference Co-Chair)