Annals of Operations Research 60(1995)193–208

A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives

Oli B.G. Madsen, Hans F. Ravn and Jens Moberg Rygaard
IMM, Institute of Mathematical Modelling, Building 321,
The Technical University of Denmark, DK-2800 Lyngby, Denmark
E-mail: ogm@imm.dtu.dk

The paper describes a system for the solution of a static dial-a-ride routing and scheduling problem with time windows (DARPTW). The problem statement and initialization of the development project was made by the Copenhagen Fire-Fighting Service (CFFS). The CFFS needed a new system for scheduling elderly and disabled persons, involving about 50.000 requests per year. The problem is characterized by, among other things, multiple capacities and multiple objectives. The capacities refer to the fact that a vehicle may be equipped with e.g. normal seats, children seats or wheel chair places. The objectives relate to a number of concerns such as e.g. short driving time, high vehicle utilization or low costs. A solution algorithm REBUS based on an insertion heuristics was developed. The algorithm permits in a flexible way weighting of the various goals such that the solution reflects the user's preferences. The algorithm is implemented in a dynamic environment intended for on-line scheduling. Thus, a new request for service is treated in less than 1 second, permitting an interactive user interface.

Keywords: Dial-a-ride, dynamic routing.

1. Introduction

We will denote a static dial-a-ride routing and scheduling problem by DARP and if time windows are included by DARPTW. In DARPTW we have n customers known in advance, each of which possesses an origin location (pick-up location), a destination location (delivery location), a desired time window for delivery or for pick-up, and a specific type of transportation requirement. A given number of vehicles with the same capacity are available. We seek a route and a schedule for the vehicles such that a specified objective is optimized. The objective can vary but is usually dependent on the number of vehicles, the mileage driven, and some

service measure of the customers. The service measure often involves the excess ride time of a customer (the actual ride time minus the direct ride time), and the delivery time (pick-up time) deviation of a customer (the deviation from the desired delivery or pick-up time).

The above mentioned problem type can be found in many practical applications, for example:

- Transportation of elderly and/or disabled persons.
- Telebuses.
- Taxis.
- Ambulances.
- Certain courier services where the time necessary for service is relatively small compared to the driving time.

This kind of transportation is very costly and labour intensive. It is therefore important to develop effective methods to solve DARPTW.

Surprisingly few papers have been published in the open literature on DARP(TW). Stein [16,17] has presented a probabilistic analysis of DARP. Daganzo [3] has developed approximate models to evaluate the performance of DARP. Psaraftis [7] has described an exact approach to DARPTW based on dynamic programming. Psaraftis [8] has further developed a dynamic programming algorithm for solving the single vehicle DARPTW optimally for a small number of customers, and later [9,10] described several polynomial-time heuristics for solving the single vehicle DARP. In Bodin et al. [1], a survey on DARP and DARPTW is presented. Jaw et al. [6] present a parallel insertion heuristic, ADARTW, for DARPTW. A flexible objective function balances the cost of providing service and short ride times. Desrosiers et al. [4] describe a dynamic programming solution of a large-scale single vehicle DARPTW. Other work on pick-up and delivery problems specific to the DARP context may be found in Sexton and Bodin [13,14], and Sexton and Choi [15]. Psaraftis [11] presents a survey on two heuristic methods to DARPTW – a grouping, clustering, routing algorithm, and ADARTW. Desrosiers et al. (1988) discuss the multiple vehicle DARP. To the best of the authors' knowledge nothing has been published on DARP in the open literature since 1988.

Concerning internal reports and thesis work, groups from the following universities have worked on DARP or DARPTW:

- MIT, Cambridge, MA, USA (since 1971).
- CRT and GERAD, Montreal, Canada (since 1978).
- State University of New York at Stony Brook, NY, USA (around 1979).
- University of Maryland at College Park, Maryland, USA (since 1980).
- IMM, Technical University of Denmark, Lyngby, Denmark (since 1990).

Of these reports, Roy et al. [12] describe a combined clustering method and insertion heuristic for an almost static DARP with dynamic features. Ioachim et al. [5] present a near optimal method that uses mathematical optimization techniques to globally define a set of mini-clusters. Multi-dimensional capacities are allowed.

In this paper, we will present a new DARPTW algorithm, some computational results, and the relation to a real-life case. Section 2 presents the main problems in the real-life case which originated this paper. Section 3 describes the different objectives implemented in REBUS. Section 4 then describes REBUS – an improved and generalized version of the earlier mentioned ADARTW heuristic. Implementational issues are discussed in section 5. Section 6 summarizes computational experience with REBUS, and section 7 contains some brief concluding remarks.

2. The problem and its special requirements

In 1992 the authors were contacted by The Copenhagen Fire-Fighting Service (CFFS), which also takes care of the transportation of elderly and disabled persons, about 50.000 requests for transportation per year. The purpose of the contact was to establish cooperation and development of a system for on-line planning of this transportation.

In this section, we describe the problem as specified by the CFFS. At the time of the planning the following information is available:

A number of requests for transportation service from the customers. For each customer the following is known:

- the locations for the pick-up and the delivery;
- the time for picking up the customer (from the time the vehicle stops at the location until it starts again (without waiting time));
- the time for delivering the customer;
- either a desired time window for pick-up or a desired time window for delivery;
- a customer priority level;
- a multi-dimensional capacity requirement:
  - number of ordinary seats,
  - number of lying seats,
  - number of children seats,
  - number of wheel chair places,
  - number of bed places.

For each vehicle in the CFSS fleet, the following is known:

- average speed;
- the time periods where the vehicle is available;
- the multi-dimensional capacity and possibilities for substituting one capacity type for another;
- the cost of operation of the vehicle.

Furthermore, the driving times between all the above mentioned locations are given, as well as time intervals for drivers' lunch time, other breaks, and times for vehicle service or repair.

The CFFS wants to choose among several objectives or a mixture of objectives such as:

- minimize the total driving time,
- minimize the number of vehicles,
- minimize the total waiting time,
- minimize deviation from promised service,
- minimize the cost,

subject to capacity and time constraints as well as an upper limit on the excess ride time of a customer (the actual ride time minus the direct ride time).

If the load is very heavy it is possible to allocate some requests to taxis. The taxis are given a special cost and are added to the list of vehicles (and, for example, no lunch time).

In reality, at the time for starting the planning the CFFS does not know all the requests for transportation. Therefore a dynamic updating capability is required so that customers could be added to the vehicle schedules on a real-time basis.

The planning system should be able to handle up to 300 requests and 24 vehicles at the same time, and a new request should be added and scheduled in a maximum of 2 seconds.

Relative to the problems described in the referred literature the problem as specified by the CFFS has at least the following complicating features:

- the capacity requirements has several (viz. five) dimensions;
- the objective function should reflect a variety of subgoals;
- the solution algorithm should work in a dynamic environment with strict limits on computation time.

Based on the above mentioned requirements, it was decided to develop REBUS, an improved and generalized version of ADARTW (Jaw et al. [6]) which will be discussed in the next sections.

3. Objective functions

In the problem statement given by the CFFS, it was obvious that it would not be possible to state a criterion such that a "best" solution could be indentified. Rather, a "good" solution could be distinguished from a "bad" solution only by relating the solution to a number of goals, partially conflicting. It was decided to model this by the inclusion of a number of parameters, which would indicate the relative weights of the different goals.

The user therefore has to specify a set of parameter values that guide the construction of tours. In the sequel, we list these parameters and give a brief description of their meaning. At the end of this section, we illustrate how the selection of specific values for the parameters influences the characteristics of the tours.

The parameters fall into two groups, viz., those that relate to the jobs, and those that relate to the job-insertion.

3.1. PARAMETERS RELATED TO JOBS

In connection with the insertion procedure the jobs are sorted, such that jobs that apparently will be "difficult" to insert are inserted first. Such jobs are therefore given higher priority.

Time window A narrow time window seems more difficult to satisfy than a wider one. As a quantititative measure of this, the following is used:

$$M_{win} = \begin{cases} c2_{win} \cdot (\delta t_{win})^{-1} + c1_{win} & \text{if } t_{win} > 0, \\ \max_{win} & \text{if } \delta t_{win} = 0. \end{cases}$$

Here, δt_{win} is the window width and $c1_{win}$, $c2_{win}$ and $\max_{win}$ are non-negative constants, with $\max_{win}$ being a suitably large value.

Maximal travel time Let δt_{trt} be the difference between the maximal allowable travel time and the shortest possible travel time. Then, with $c1_{trt}$, $c2_{trt}$ and $\max_{trt}$ being non-negative constants, with $\max_{trt}$ being a suitably large value, the following is used

$$M_{trt} = \begin{cases} c2_{trt} \cdot (\delta t_{trt})^{-1} + c1_{trt} & \text{if } t_{trt} > 0, \\ \max_{trt} & \text{if } \delta t_{trt} = 0. \end{cases}$$

Job capacity Certain jobs may be difficult to assign because there are few vehicles that are able to perform these jobs. Thus, c_{seat}, c_{aseat}, c_{cseat}, c_{wchair} and c_{bed} are constants corresponding to the number of normal seat, ambulance beds,

childrens' seats, wheelchairs and beds, respectively, and d_{seat}, d_{aseat}, d_{cseat}, d_{wchair} and d_{bed} are the corresponding capacity requirements. Then the following measure is used:

$$\sum c_i \cdot d_i,$$

where the summation is over all the relevant capacity types.

The priority assigned to a job is then the sum of the above expressions. A higher number implies the job is inserted earlier in the procedure.

3.2. PARAMETERS RELATED TO JOB-INSERTION

In order to rank the different possible insertions of a job into a given preliminary plan, the "load" of the insertion is calculated.

The following aspects are covered:

Driving time Let the minimum driving time be denoted by t_{dr_time}, the waiting time by t_{wait} and the handling time by t_{handl}. Then with $c_{variable}$ and c_{const} being non-negative constants, the following expression is used:

$$c_{variable} \cdot t_{dr\_time} + c_{constant} \cdot (t_{wait} + c_{handl}).$$

Waiting time With $c1_{wait}$ and $c2_{wait}$ as non-negative constants, and t_{wait} the waiting time, the following is used:

$$c2_{wait} \cdot (t_{wait})^2 + c1_{wait} \cdot t_{wait}.$$

Deviation from desired service time With c_{dev} a non-negative constant and δt_{dev}, the deviation between desired and planned service time, the following is used:

$$c_{dev}(\delta t_{dev})^2.$$

Capacity utilization Vehicles with unutilized capacity may be considered undesirable. With c_i being a non-negative constant related to capacity type i and $vfree_i$ being the amount of untilized capacity of type i, the following is used:

$$\sum c_i (vfree_i)^2.$$

These expressions are for each stop, so therefore summation is performed over all stops. The indicated parameters may be given individual values for each vehicle.

3.3. ILLUSTRATION

We now by an example indicate the functioning of the described parameters. We present a small case with only four vehicles, each one with four seats. On each seat there may be either a child seat or an ordinary seat. There are 30 jobs.

The parameter c_{win} has the value 1, all other parameters related to jobs are zero. The parameters related to job insertion are given in the following four versions:

Case 1: $c_{var} = 1$ and all other parameters are zero.

Case 2: $c2_{wait} = c1_{wait} = 1$ and all other parameters are zero.

Case 3: $c_{dev} = 1$ and all other parameters are zero.

Case 4: $c_i = 1$ for all i and all other parameters are zero.

In table 1, the results of runs of these four cases are indicated.

Table 1

Result of the experiments with the four cases. Column 1: case number; column 2: average number of jobs inserted out of the 30 requested; column 3: average drive time for the jobs (minutes); column 4: average total time per job (minutes); column 5: average waiting time per job (minutes); column 6: percentage of the time, where more than one job is allocated to a vehicle; column 7: average total time per job where a vehicle is driving empty (minutes); column 8: average deviation per job from desired service time (minutes); column 9: average number of seats occupied, when there is at least one job in the vehicle. A star indicates which of the objectives the parameter setting in the experiment tries to minimize.

1	2	3	4	5	6	7	8	9
1	27	24.59*m./j	23.63m./j	5.56m./j	41.6%	9.56m./j	6.69m./j	2.48
2	27	28.15m./j	23.70m./j	3.44*m./j	33.63%	10.85m./j	6.41m./j	2.09
3	26	28.27m./j	23.63m./j	4.27m./j	41.37%	10.92m./j	3.88*m./j	2.22
4	26	27.04m./j	25.77m./j	5.15m./j	43.10%	9.42*m./j	5.46m./j	2.22

As seen in cases 1 and 2, three out of the 30 jobs have not been placed in any vehicle; the corresponding number for cases 3 and 4 are four. These jobs will usually be allocated to taxis. The taxis are given a special cost and they are added to the list of vehicles.

In case 1, the parameter setting reflects the desire to give priority to low driving time. From column 3 in table 1, it is seen that indeed the driving time (24.59 minutes) for case 1 is lower than the driving time for the other cases. From columns 5 and 8, it is seen that this desirable result is obtained partly at the expense of

(relative to the other cases) high waiting time and deviation from desired service time.

By studying the table, we will observe that for all four cases the setting of the parameter values implies that the main objective of any one of the cases is fulfilled well. Thus, cases 2, 3 and 4 have the best waiting time, capacity utilization and deviation from desired service time, respectively.

We may consider the parameter setting as the assignment of relative weights to objectives in a multi objective optimization problem. Therefore, we may interpret the result of these experiments to mean that it is possible to balance the objectives such that they reflect the user's preferences.

4. The insertion algorithm REBUS

REBUS is based on a modified insertion algorithm, originally developed by Jaw et al. [6]. The name REBUS is an abbreviation of a Danish sentence. This algorithm is a heuristic method, which may be briefly explained as follows. A set of jobs J that are not yet assigned to a schedule, and a set of vehicles V are given. Each job can either represent a customer (with a pick-up and a delivery stop) or a break (with only one stop). Furthermore, an objective function is assumed. The way the jobs are ordered in J is described in section 3. The insertion algorithm then works as follows:

Step 1. Consider the next job in J that is not allocated in one of the schedules for the vehicles.

Step 2. For each vehicle in V:

Generate all feasible insertions of the job in the schedule and calculate the change in the objective function.

Step 3. If there exists a feasible insertion of the job, then the insertion with the minimum change in the objective function is selected, the job is inserted in the schedule and the job is removed from J.
If a feasible insertion does not exist for the job, this is returned to a list of jobs NS that cannot be served.

Step 4. If J is not empty, then go to step 1, otherwise stop.

It is possible either to let V be the vehicles in the fleet or to let V be the fleet of vehicles plus a number of fictitious vehicles, i.e. taxis, in which it is possible to insert jobs, which cannot be assigned to the ordinary vehicles. The cost for assigning a job to a fictitious vehicle is higher compared to the costs for assigning a job to an ordinary vehicle.

Before the insertion algorithm is executed, the jobs in the set J are ordered by a measure that indicates how difficult a job might be to insert into the timetable (see section 3.1).

Our insertion algorithm differs from ADARTW described in Jaw et al. [6] in the following ways:

(1) The unassigned jobs in J are sorted according to costs (see section 3.1).

(2) Several parameters related to job insertion are applied (see section 3.2). (REBUS mainly differs from ADARTW in that the waiting times for a vehicle (see section 3.2) also can be minimized, and for each customer an upper limit on the total transportation time is given.) The user has the possibility of adjusting the solution according to the user's preferences.

(3) The algorithm that makes the insertion of jobs in a schedule has been improved applying certain properties (see section 4.1) in order to reduce the number of insertions to evaluate. REBUS does not (as ADARTW does) operate a schedule consisting of blocks as this increases the running times for the insertion algorithm.

(4) Multiple capacity types can be handled as well in REBUS as in ADARTW. Furthermore, REBUS allows vehicles, which can serve different types of capacity requirements, that might exclude one another. For example, a large ambulance can either transport a bed or at maximum four wheelchairs.

(5) In REBUS, two types of service requirements can be scheduled. Either a job, which is characterized by a pickup stop and a delivery stop, or a break, which is characterized by only one stop (ADARTW only considers jobs).

(6) REBUS is designed to solve dynamic route problems, i.e. that new jobs enter the system at random times, driving times between two stops might change during a day, and vehicles might break down or be reallocated in emergency situations.

These improvements reduce the running times of REBUS compared to ADARTW and makes REBUS more flexible, but the quality of the schedules can not be expected to be significantly better than schedules produced by an ADARTW algorithm. Furthermore, REBUS has been extended in relation to ADARTW in order to operate in a dynamic environment considering uncertainties in the information the system receives. For example, driving times between stops may change, service requirements enter the system at random times, and jobs might be cancelled.

4.1. THE INSERTION OF JOBS

The time-consuming part in the insertion algorithm is step 2. If there are n stops in a schedule related to a vehicle, then the number of possible insertions for

a job with two stops is $(n + 2)(n + 1)/2$. For each insertion the following calculations have to be done:

1. **Feasibility**:

   For each stop that is affected by the insertion of the job, the arrival time for the vehicle has to be updated. Then it is checked if a time window at a stop is violated, or if the maximum travel time for one of the jobs is not satisfied. Furthermore, it is necessary to check if the capacity constraints are fulfilled, i.e. that the capacity required by the job is available all way from the pickup stop to the delivery stop.

2. **Objective function**:

   The change in the objective function has to be calculated for each new insertion.

In the application at the CFFS, it is important to reduce the computer time required for determining the schedules for each of the vehicles, since the algorithm is called several times during a day because the set of jobs J is changed. The number of different insertions that are evaluated is reduced by applying the following properties:

1. **Capacity property**:

   If a job can be inserted into a schedule, then the capacity required by the job can be obtained at any of the stops between the pickup and the delivery stop. This means that if there exists a stop where the capacity demanded by a job cannot be satisfied, then if the job can be inserted into the schedule both the pickup stop and the delivery stop will either be before or after this stop.

2. **Maximum travel time property**:

   If a job is inserted into the schedule and the maximum travel time is exceeded for the job, then either the pickup stop has to be placed later in the schedule, or the delivery stop has to be placed earlier in the schedule.

3. **Time windows property**:

   If the upper time limit on a time window related to a stop is violated, then the time window will still be violated if the stop is moved to the right on the time-axis, i.e. the stop is effected at a later time (see figure 1).

In figure 1, a schedule related to a vehicle is shown before and after the stop s_1 is moved one stop to the right. It is assumed that the schedule always has at least two stops T_0, T_1, where the first stop T_0 represents the depot from which the vehicle starts and the last stop T_1 represents the depot where the vehicle ends.

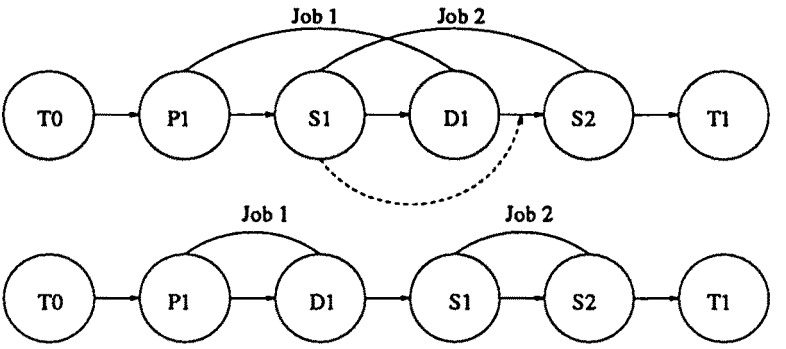

Figure 1. A schedule before and after the stop s_1 is moved one stop to the right. After s_1 is moved, s_2 is just after s_1. T_0, T_1 are depots, stops p_1, d_1 are pick-up and delivery stops related to job 1, and s_1, s_2 are stops related to job 2.

If a stop can be inserted into a schedule related to a vehicle, there might be several ways this can be done, viz. the stop can be inserted, so that service will either begin as soon as possible, as late as possible or as close as possible to the desired time for service. In the application of the CFFS, it was chosen that the service should take place as soon as possible, since the customers in a vehicle otherwise might have to wait until the desired time for service has been reached for the next job, when the vehicle arrives at a stop.

The algorithm, that for a given vehicle investigates all feasible insertions of the job, may be described as follows for a job with two stops s_1, s_2:

Step 1. Place s_1, s_2 just after the first stop T_0 in the schedule, and update the schedule.

Step 2. While all insertions have not been evaluated do

(a) If s_2 is before the last stop T_1 in the schedule,
then move s_1 one stop to the right and place s_2 just after s_1 (see figure 1) and update the schedule
else move s_2 one stop to the right.

(b) Check for feasibility.

i. If the insertion is feasible, then calculate the change in the objective, and compare to the previously found insertions.

ii. If the insertion is not feasible, check for the following situations:

A. if the capacity constraints, the maximum travel time or the time window related to s_2 have been violated
then move s_1 one stop to the right and place s_2 just after s_1 and update the schedule. Go to 2(b).

B. if the time window related to s_1 is violated then stop.

C. else go to 2.

When updating the schedule only the stops that are affected by the insertion are considered. In step 2(b), only the basic ideas are illustrated. Other situations, not to be described here, are considered in order to reduce the number of combinations that have to be searched.

Checking the feasibility of an insertion includes verification that the capacity constraints and the time constraints for all of the stops affected by the insertion are satisfied.

A similar algorithm has been developed to handle jobs that only consist of one stop, e.g. breaks.

5. Implementation

The program REBUS consists of a shell, through which the communication with the surrounding system takes place. REBUS is designed to operate in a dynamic environment. The shell consists of a number of functions that handle the following tasks:

1. Creation of a shell and initialization of related data.
2. Creation or deletion of vehicles related to the shell.
3. Creation or deletion of jobs, i.e. customers or breaks.
4. Making schedules for each of the vehicles, given a set of customers.
5. Updating schedules.
6. Insertion of breaks.
7. Extraction of data from the database about the schedules.
8. Generation of statistics.

As the requests for service, i.e. jobs, enter the CFFS, they are transferred to REBUS, which stores the jobs in the set of jobs J that are not yet assigned to a schedule. After a certain amount of time, a new schedule is generated for each of the vehicles, for the jobs which are not under service, using the insertion algorithm. The jobs that could not be assigned in a schedule are then placed in a set, from which the surrounding system can extract the jobs for further handling.

The breaks are inserted into schedules related to a particular vehicle. Such a job will always be inserted, which means that if it is not possible to insert a break, then the jobs under service will be removed and the break will be inserted. Afterwards, new schedules for each of the vehicles will be generated. The breaks can therefore also be used in case a vehicle breaks down, or has to be assigned for other tasks in a shorter period.

When a vehicle reaches a stop, the arrival time and the vehicle number are transmitted to REBUS, which updates the arrival time for the following stops in the schedule. If some of the time windows for the future jobs or maximum travel times

are violated, then a message is passed to the surrounding system. Furthermore, REBUS will delete jobs or breaks that are finished.

It is possible for a user to manually switch jobs between the vehicles that are not under service. This means that the user has the possibility to change a solution. REBUS also collects statistics about the schedules, which can be used to evaluate the utilization of the set of vehicles during a day.

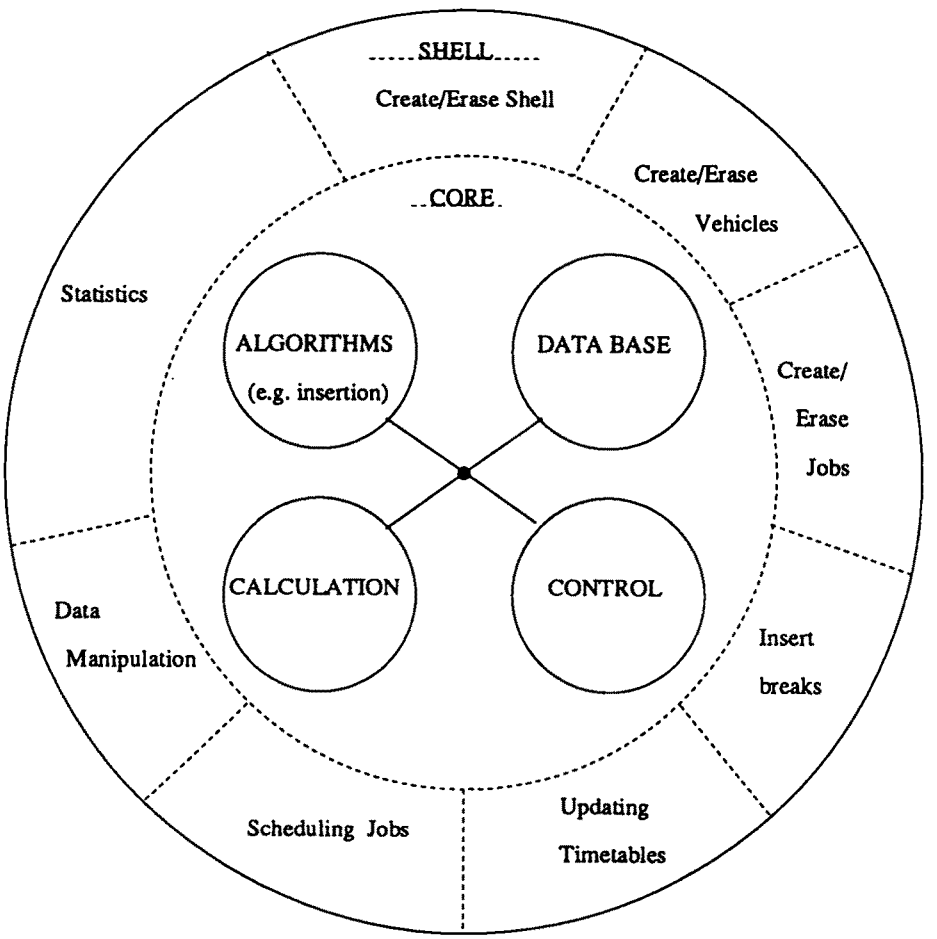

Figure 2. Diagram that illustrates the structure of REBUS:

In figure 2, the structure of REBUS is illustrated. Behind the shell, there is a core, that consists of 4 modules, which contain algorithms, objective functions (calculation), a data base and procedures that manipulate data. REBUS is implemented in C++ using an object-orientated approach. This means that the system can handle several groups of vehicles and jobs that are independent of each other. This

is relevant in the case of the CFFS, since the transportation of prone passengers is done in ambulances and the transportation of passengers that use normal seats or wheelchairs is done in cars and vans.

In figure 3, it is shown how REBUS is integrated in a final system. REBUS requires a data base containing shortest distances between selected geographical points. An interface system, that presents solutions and performs the communication with the user, is required, and a control system that updates REBUS with information about new customers and activates the planning function is required.

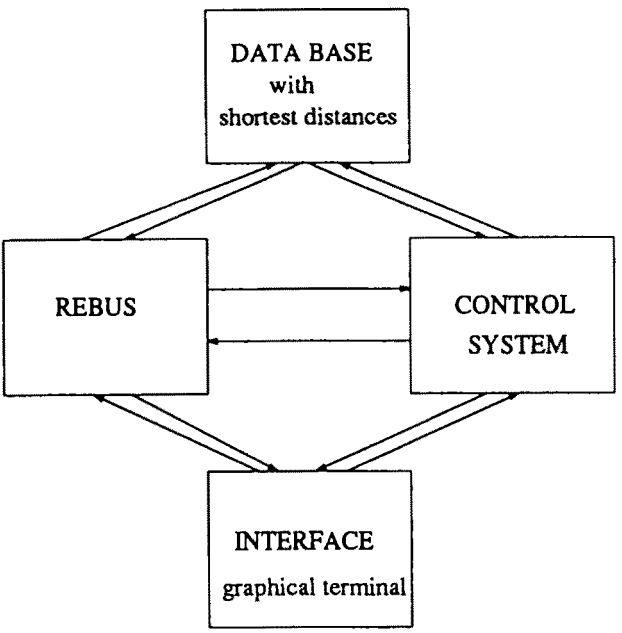

Figure 3. Diagram that illustrates REBUS and the surrounding system.

6. Computational characteristics

REBUS has been implemented in C++ and tested on a HP-735/9000 computer with the UNIX operating system. The library file that contains the REBUS system required around 150 KB RAM.

REBUS has been tested on a variety of problems that reflect the traffic situation in the Copenhagen area and typical request patterns of the CFFS. Aspects of the multi-objective decision problem have been discussed in section 3. The following observations on computation refer to a problem with 24 vehicles and 300 customers. The planning problems are around the same size in the CFFS. REBUS determined a solution in less than 10 seconds, using less than 1MB RAM. The

geographical information was stored in the RAM. This storage requirement might be expected to increase linearly with the number of vehicles and jobs. A new request added can be scheduled in less than 1 second. The time required for determining a new set of schedules indicates that REBUS is suitable for solving dynamic problems. In the case of the CFFS, new schedules might have to be generated every five minutes, due to the fact that new jobs are sent to REBUS and vehicles arrive at their destinations. The computation time required increases with the square of the number of jobs and linearly with the number of vehicles.

In section 4, a verbal comparison was made between REBUS and ADARTW. Since we do not have the ADARTW code available, it is very difficult to make direct computational comparisons. However, in an earlier study (Christensen and Jensen [2]), a code PDAR, extremely similar to ADARTW, was implemented and tested on real data. These data were very similar to our data with respect to problem area, customer locations, time windows, number of customers and vehicles, vehicle capacities, and the computer used. The results seem to indicate that PDAR (and therefore ADARTW) is 6–10 times slower than REBUS. The quality of the solutions seems to be very similar as far as it is possible to compare multi-objective solutions.

7. Conclusion

We have presented a new algorithm for the dial-a-ride problem with time windows, multiple capacities, and multiple objectives. The algorithm, which may be considered as a generalized, revised and computationally improved parallel to the ADARTW algorithm, is an insertion heuristic which is fast and flexible, such that the algorithm can be applied on-line in a dynamic environment. The algorithm has been tested on real-life cases.

References

[1] L.D. Bodin, B.L. Golden, A. Assad and M.O. Ball, Routing and scheduling of vehicles and crews: The state of the art, Computers and Operations Research 10(1983)63–211.

[2] L.L. Christensen and J.K. Jensen, Transportation of patients – a dynamic dial-a-ride system (in Danish), Report EP 25/91, IMSOR, The Institute of Mathematical Statistics and Operations Research, The Technical University of Denmark, DK-2800 Lyngby, Denmark (1991).

[3] C. Daganzo, An approximate analytic model of many-to-many demand responsive transportation systems, Transportation Research 12(1978)325–333.

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

[5] I. Ioachim, J. Desrosiers, Y. Dumas and M.M. Solomon, A request clustering algorithm for door-to-door transportation, GERAD Report-91-50, 5255 Avenue Decelles, Montréal, Canada H3T 1V6 (1991).

[6] J.-J. Jaw, A.R. Odoni, H.N. Psaraftis and N.H.M. Wilson, A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows, Transportation Research 20B (1986)243–257.

[7] H.N. Psaraftis, A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem, Transportation Science 14(1980)130–154.
[8] H.N. Psaraftis, An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows, Transportation Science 17(1983)351–357.
[9] H.N. Psaraftis, k-interchange procedures for local search in a precedence-constrained routing problem, European Journal of Operational Research 13(1983)391–402.
[10] H.N. Psaraftis, Analysis of an $O(N^2)$ heuristic for the single vehicle many-to-many Euclidean dial-a-ride problem, Transportation Research 17B(1983)133–145.
[11] H.N. Psaraftis, Scheduling large-scale advance request dial-a-ride systems, The American Journal of Mathematical and Management Sciences 6(1986)327–368.
[12] S. Roy, J.M. Rousseau, G. Lapalme and J.A. Ferland, Routing and scheduling for the transportation of disabled persons – the algorithm, CRT Report 412A, Université de Montréal, Montréal, Québec, Canada H3C 3J7 (1985).
[13] T. Sexton and L. Bodin, Optimizing single vehicle many-to-many operations with desired delivery times: I. Scheduling, Transportation Science 19(1986)378–410.
[14] T. Sexton and L. Bodin, Optimizing single-vehicle many-to-many operations with desired delivery times: II. Routing, Transportation Science 19(1986)411–435.
[15] T. Sexton and Y. Choi, Pick-up and delivery with partial loads with "soft" time windows, The American Journal of Mathematical and Management Sciences 6(1986)369–388.
[16] D.M. Stein, An asymptotic, probabilistic analysis of a routing problem, Mathematics of Operations Research 3(1978)89–101.
[17] D.M. Stein, Scheduling dial-a-ride transportation systems, Transportation Science 12(1978) 232–249.