Parallel Tabu Search Heuristics for the Dynamic Multi-Vehicle Dial-a-Ride Problem

Andrea ATTANASIO[1], Jean-François CORDEAU[2],
Gianpaolo GHIANI[3], Gilbert LAPORTE[2]

[1] *Centro di Eccellenza sul Calcolo ad Alte Prestazioni*
Università degli Studi della Calabria, 87030 Rende (CS), Italy
attanasio@unical.it

[2] *Canada Research Chair in Distribution Management, HEC Montréal*
3000 Chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7
{cordeau, gilbert}@crt.umontreal.ca

[3] *Dipartimento di Ingegneria dell'Innovazione*
Università degli Studi di Lecce, 73100 Lecce, Italy
gianpaolo.ghiani@unile.it

December 18, 2003

Abstract

In the *Dial-a-Ride Problem* (DARP) users specify transportation requests between origins and destinations to be served by vehicles. In the Dynamic DARP, requests are received throughout the day and the primary objective is to accept as many requests as possible while satisfying operational constraints. This article describes and compares a number of parallel implementations of a tabu search heuristic previously developed for the static DARP, i.e., the variant of the problem where all requests are known in advance. Computational results show that the proposed algorithms are able to satisfy a high percentage of user requests.

Keywords: Dial-a-Ride problem, parallel computing, metaheuristics.

1 Introduction

The *Dial-a-Ride Problem* (DARP) arises in contexts where n users specify transportation requests between origins and destinations, subject to scheduling constraints. These requests must be satisfied in a cost effective fashion by a fleet of m vehicles, subject to a number of operational constraints. The most common application of the DARP is encountered in door-to-door transportation services for the elderly and the handicapped (see, e.g., Madsen, Ravn and Rygaard (8), Toth and Vigo (10; 11), and Borndörfer et al. (1)). With the ageing of the population and the need to reduce public expenditures in most western societies, we believe dial-a-ride services will gain in popularity in coming years and the need to run these efficiently will also increase.

Dial-a-ride services operate according to one of two modes. In the *static* mode all requests are known in advance, typically one day before transportation actually takes place. In the *dynamic* mode requests are gradually revealed and assigned to existing vehicle routes in real-time so that *a priori* planning is impossible. In practice, the distinction between these two modes is not always clear cut. Even if planned routes are available, changes may be necessary due to last minute cancellations. Also, in dynamic environments, a large number of requests are often known at the start of the planning horizon. There also exist several versions of the DARP for both modes since the objective, constraints and operating rules vary from one context to another.

In the last twenty years or so, several models and algorithms have been devoted to the DARP. For a recent overview, see Cordeau and Laporte (3). With a few exceptions, most available algorithms are heuristics. Tabu search (TS) – see, e.g., Cordeau and Laporte (4) – stands out as a very powerful tool for the DARP since it is at the same time highly flexible and efficient. Flexibility stems from the capacity of handling a large number of variants within the same search framework. Efficiency is associated with solution quality. It is now clear that TS is capable of consistently generating high quality solutions on a large variety of routing problems (see, e.g., Cordeau et al. (2)).

On the negative side the running time of TS algorithms can be rather high. Thus, using real data from a Danish transporter (with $n = 200$ and 295), Cordeau and Laporte have run three versions of their TS algorithm. These versions vary in the the thoroughness of the exploration of the solution space and yield highly varying running times. The simplest version yielded computing times varying between 13.21 and 28.40 minutes on a Pentium 4, 2 GHz computer, while the most thorough version required between 104.48 and 267.82 minutes. The improvement in solution quality between the two versions is almost 6%. There are two reasons why it is important to devise faster algorithms. First, there exist several contexts

where the problem size is much larger (the number of requests per day in some European and North-American cities often exceeds 2000). Second, while it makes sense to run an algorithm for a few hours in a static context, much faster response times are required in a dynamic environment (see, e.g., (6) and (7)). In fact whenever a user makes a transportation request by telephone or on the Internet, the operating system should be able to tell, within a few seconds, whether the request can be accommodated. In addition, if the request is accepted, it should be embedded in the existing vehicle routes within a relatively short time (no more than ten minutes, say).

One natural way to speed up computation time is through the use of parallel computing. We report on the use of a number of parallel implementations of the Cordeau and Laporte (4) static DARP heuristic in a real-time context. The remainder of this article is organized as follows. In Section 2, we describe the main features of our version of the DARP. This is followed by a summary of the single processor static TS heuristic in Section 3, and by its parallel and dynamic implementations in Section 4. A computational assessment of the parallel and dynamic implementations is presented in Section 5 and conclusions follow in Section 6.

2 Problem definition

As is often the case in dial-a-ride contexts, users specify an *outbound* request from their home to a destination (e.g., a hospital) and an *inbound* request for the return trip. To achieve a fair balance between user convenience and the transporter's cost, we believe users should be able to specify windows on the arrival time of their outbound journey and on the departure time of their inbound journey. The transporter guarantees each user a *maximal ride time*, i.e., an upper bound on the time spent by a user in the vehicle. This can either be a constant, or a value obtained by multiplying the minimal achievable travel time by a preset constant.

Formally, the static DARP is defined as follows. Let $G = (V, A)$ be a complete graph where $V = \{v_0, v_1, \ldots, v_{2n}\}$ is the vertex set and $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is the arc set. Vertex v_0 represents a depot at which is based a fleet of m vehicles, and the remaining $2n$ vertices represent origins and destinations for the transportation requests. Each vertex pair (v_i, v_{i+n}) represents a request for transportation from origin v_i to destination v_{i+n}. With each vertex $v_i \in V$ are associated a load q_i (with $q_0 = 0$), a non-negative service duration d_i (with $d_0 = 0$) and a time window $[e_i, l_i]$, where e_i and l_i are non-negative integers. The load is equal to 1 for vertices $v_1, \ldots, v_n$ and to -1 for vertices $v_{n+1}, \ldots, v_{2n}$. Service duration

corresponds to the time needed to let the client get on or off the vehicle. Let T denote the end of the planning horizon. In the case of an outbound request, it is assumed that $e_i = 0$ and $l_i = T$. Similarly, $e_{i+n} = 0$ and $l_{i+n} = T$ for an inbound request. Each arc (v_i, v_j) has an associated non-negative cost c_{ij} and a non-negative travel time t_{ij}. Finally, let L denote the maximum ride time of a client. The DARP consists of designing m vehicle routes on G such that

(i) every route starts and ends at the depot;

(ii) for every request i, vertices v_i and v_{i+n} belong to exactly one route and vertex v_{i+n} is visited later than vertex v_i;

(iii) the load of vehicle k does not exceed at any time a preset bound Q_k;

(iv) the total duration of route k does not exceed a preset bound T_k;

(v) the service at vertex v_i begins in the interval $[e_i, l_i]$, and every vehicle leaves the depot and returns to the depot in the interval $[e_0, l_0]$;

(vi) the ride time of any client does not exceed L;

(vii) the total routing cost of all vehicles is minimized.

We denote by A_i the arrival time of a vehicle at vertex v_i, by $B_i \geq \max\{e_i, A_i\}$ the beginning of service at vertex v_i, and by $D_i = B_i + d_i$ the departure time from vertex v_i. The time window constraint at vertex v_i is violated if $B_i > l_i$. However, arrival before e_i is allowed and the vehicle then incurs a waiting time $W_i = B_i - A_i$. The ride time associated with request i is computed as $L_i = B_{i+n} - D_i$. If there were no ride time constraints, it would always be optimal to set $B_i = \max\{e_i, A_i\}$. However, it may sometimes be profitable to delay the beginning of service at vertex v_i so as to reduce the unnecessary waiting time at vertex v_{i+n} (or at any other vertex visited between v_i and v_{i+n}) and thus, the ride time associated with request i.

In the *Dynamic* DARP, operational constraints are the same as in the static problem. However, the primary goal is to satisfy as many requests as possible with the available fleet of vehicles. Requests are dealt with one at a time in a first come, first served fashion. Whenever a request can be served without violating any of the constraints, it is accepted and becomes a part of the problem. Of course, as the day goes on, the degree of flexibility decreases and the last requests to be formulated are likely to be rejected.

3 Sequential tabu search heuristic

The TS algorithm developed by Cordeau and Laporte (4) for the static DARP works as follows. Starting from an initial solution s_0, the algorithm moves at iteration t from s_t to the best solution in a neighbourhood $N(s_t)$ of s_t. To avoid cycling, solutions possessing some attributes of recently visited solutions are declared forbidden, or *tabu*, for a number of iterations, unless they constitute a new incumbent. As is common in such algorithms, a continuous diversification mechanism is put in place in order to reduce the likelihood of being trapped in a local optimum.

As is now often the case in TS implementations, the search mechanism explores infeasible solutions through the use of a penalized objective of the form

$$f(s) = c(s) + \alpha q(s) + \beta d(s) + \gamma w(s) + \tau t(s),$$

where $c(s)$ is the routing cost of solution s and $q(s), d(s), w(s)$ and $t(s)$ represent violations of vehicle capacity, route duration, time window and user ride time constraints, respectively. Initially set equal to 1, the coefficients α, β, γ and τ are multiplied or divided by $1 + \delta$ at each iteration, where $\delta > 0$ is a user-defined parameter. If s is infeasible with respect to a constraint, the associated parameter is multiplied by $1 + \delta$; otherwise it is divided by $1 + \delta$.

An initial solution is obtained by randomly assigning requests to routes while satisfying constraints (i) and (ii). At each iteration t, the search proceeds to the best non-tabu solution in the neighbourhood $N(s_t)$ of the current solution s_t, with respect to the objective function $f(s) + p(s)$, and the coefficients α, β, γ and τ are updated. Every κ iterations, and whenever a new best solution is identified during the search, intra-route exchanges are performed to improve $f(s)$.

To define neighbourhoods, an attribute set $U(s) = \{(i, k) : \text{request } i \text{ is assigned to vehicle } k\}$ is associated with each solution s. The neighbourhood $N(s)$ of s is made up of all solutions reachable from s by removing an attribute (i, k) from $U(s)$ and replacing it by (i, k'), where $k' \neq k$. This means that v_i and v_{i+n} are removed from route k which is then reconnected by linking the predecessors and successors of these vertices. Vertices v_i and v_{i+n} are inserted into route k' so as to minimize the total increase in $f(s)$ by performing simple insertions (i.e., the ordering of the vertices already in route k' remains unchanged). To avoid cycling, reinserting v_i and v_{i+n} into route k is tabu for θ iterations, unless such a move would result in a new best solution among all those possessing attribute (i, k).

A continuous diversification scheme is also applied. To discourage frequently moving the same attribute, solutions $\bar{s}$ such that $f(\bar{s}) > f(s)$ are penalized by a factor proportional to the

frequency of addition of its distinguishing attributes and a scaling factor. More precisely, let ρ_{ik} be the number of times attribute (i,k) has been added to the solution during the search, divided by the number of iterations performed. If (i,k) denotes the attribute added to s to obtain $\bar{s}$, a penalty

$$p(\bar{s}) = \lambda\, c(\bar{s})\, \sqrt{nm}\, \rho_{ik}$$

is thus added to $f(\bar{s})$ when evaluating the cost of $\bar{s}$. The scaling factor $c(\bar{s})\,\sqrt{nm}$ introduces a correction to adjust the penalties with respect to the total solution cost and the size of the problem as measured by the number of possible attributes. Finally, the parameter λ is used to control the intensity of the diversification. These penalties have the effect of driving the search process toward less explored regions of the solution space whenever a local optimum is reached.

When evaluating a neighbour solution, route schedules must be determined. This is critical when maximum route durations or user ride time constraints are imposed since route duration and ride times can sometimes be reduced, without changing the sequence of vertices, by acting on departure times from each location. Consider a route $k = (v_0, \ldots, v_i, \ldots, v_q)$ where both v_0 and v_q correspond to the depot. If time windows can be satisfied, a solution can be identified by sequentially setting $B_0 = e_0$ and $B_i = \max\{e_i, A_i\}$ for $i = 1, \ldots, q$. To reduce route duration and ride times, it may be advantageous to delay departure from the depot and the beginning of service at origin vertices. For this, one must compute for each v_i the maximum delay F_i that can be incurred before service starts so that no time window in route k will be violated. Savelsbergh (9) calls F_i the *forward time slack* of v_i. It is computed as

$$F_i = \min_{i \le j \le q} \left\{ l_j - \left(B_i + \sum_{i \le p < j} t_{p,p+1} \right) \right\}, \tag{1}$$

which can be rewritten as

$$F_i = \min_{i \le j \le q} \left\{ \sum_{i < p \le j} W_p + (l_j - B_j) \right\} \tag{2}$$

since

$$B_j = B_i + \sum_{i \le p < j} t_{p,p+1} + \sum_{i < p \le j} W_p.$$

When time windows must be satisfied, the forward time slack is the largest increase in the beginning of service at vertex v_i that will not cause any time window to become violated. In our case, since infeasible solutions are allowed during the search, the notion of forward

time slack must be slightly modified to represent the largest increase in the beginning of service at vertex v_i that will not cause any increase in time window violations. Hence, the term $(l_j - B_j)$ should be replaced with $(l_j - B_j)^+$ in (2) because even if the time window for vertex v_j cannot be satisfied in the current route, one can nevertheless increase the beginning of service at vertex v_i by as much as $\sum_{i<p\leq j} W_p$ without increasing the violation of the time window constraint at vertex v_j.

When computing the forward time slack of a vertex $v_i \neq v_0$, care must also be taken not to increase the violation of ride time constraints. Indeed, by delaying the beginning of service at vertex v_i, one may increase the ride time for a request whose origin vertex is before v_i and whose destination vertex is at or after v_i. As a result, equation (2) becomes

$$F_i = \min_{i\leq j\leq q} \left\{ \sum_{i<p\leq j} W_p + \left(\min\left\{l_j - B_j, L - P_j\right\}\right)^+ \right\}, \tag{3}$$

where P_j denotes the ride time of the user whose destination vertex is v_j if $n+1 \leq j \leq 2n$ and $P_j = -\infty$, otherwise.

The impact of deleting vertices v_i and v_{i+n} from route k and inserting them at pre-specified locations in route k' can thus be assessed by performing the desired insertions and deletions and then applying the following procedure to each of the routes involved in the exchange :

1. Set $D_0 := e_0$.
2. Compute A_i, W_i, B_i and D_i and for each vertex v_i in the route.
3. Compute F_0.
4. Set $D_0 := e_0 + \min\left\{F_0, \sum_{0<p<q} W_p\right\}$.
5. Update A_i, W_i, B_i and D_i for each vertex v_i in the route.
6. Compute L_i for each request assigned to the route.
7. For every vertex v_j that corresponds to the origin of a request j
   (a) Compute F_j.
   (b) Set $B_j := B_j + \min\{F_j, \sum_{j<p<q} W_p\}$; $D_j := B_j + d_j$.
   (c) Update A_i, W_i, B_i and D_i, for each vertex v_i that comes after v_j in the route.
   (d) Update the ride time L_i for each request i whose destination vertex is after v_j.

This procedure first minimizes time window constraints violations in steps (1) and (2). It then minimizes route duration without increasing time window constraints violations in steps (3)-(6). Finally, in step (7), it sequentially minimizes ride times by delaying the beginning of service at each origin node as much as possible without increasing route duration, time window or ride time constraints violations. When applied to a route for which time windows can be satisfied, the procedure will yield departure and arrival times that minimize route duration and then minimize the total violations of ride time constraints. Since minimizing route duration can only help reduce ride times, it is not suboptimal to perform these two steps sequentially. In addition, treating requests sequentially in step (7) is optimal for minimizing the violation of ride time constraints because delaying the beginning of service at a given vertex will never increase the violation of ride time constraints. It could, however, increase the ride time of a request for which the constraint is satisfied.

Finally, to reduce computation time, the following insertion rule is applied to evaluate the impact of inserting vertices v_i and v_{i+n} in another route. It uses the notion of *critical vertex*. A vertex v_i is critical if it has a non-trivial time window, i.e., $e_i \neq 0$ or $l_i \neq T$. Given that each user specifies either a desired departure or arrival time but not both, there is always one such vertex per request i. First, the best insertion position is determined for the critical vertex. Then, holding the critical vertex in its best position, the best insertion position is determined for the non-critical vertex. This rule has a dramatic effect on computing times as it reduces the maximum number of possible exchanges involving request i from $O(r^2)$ to $O(r)$, where r is the number of vertices in route k.

4 Dynamic and Parallel Implementations

Our dynamic and parallel DARP algorithms work as follows. A *static solution* is constructed on the basis of the requests known at the start of the planning horizon. When a new request arrives, the algorithm performs a *feasibility check*, i.e., it searches for a feasible solution including the new service request. Once it has been decided whether the new request can be accepted or not, the algorithm performs a *post-optimization*, i.e., it tries to improve the current solution. In the following the main ingredients of our procedure are described.

Parallelization strategy. The parallelization of a TS heuristic can be accomplished in a number of ways depending on problem structure and hardware at hand. Crainic, Toulouse and Gendreau (5) classify parallel TS procedures according to three criteria: *Search Control Cardinality*, *Search Control Type* and *Search Differentiation*. Let p be the number of threads.

The first criterion indicates whether the control of the parallel search is performed by a single processor (*1–control*, 1-C), or distributed among several processors (*p–control*, p–C). The second measure denotes the way (*synchronous* or *asynchronous*) communication is performed among processors. In asynchronous communication, a processor that finds a new best solution broadcasts a message to the other processors. In the simplest case, the single solution is sent (*collegial* communication, C) while in *knowledge-based collegial* communication (KC) additional information are transmitted to the receivers. Finally, the third criterion accounts for the way different searches are carried on. Four alternatives are available: *Single Initial Point - Single Strategy* (SPSS) if a single search is performed; *Single Initial Point - Multiple Strategies* (SPMS) when each processor carries on a different search starting from the same initial solution; *Multiple Initial Points - Single Strategy* (MPSS) if each processor performs the same search starting from different initial solutions; *Multiple Initial Points - Multiple Strategies* (MPMS) if each processor carries on a different search starting from a different initial solution. In SPMS and MPMS, the searches may be performed by different algorithms or, more commonly, by the same algorithm with different parameters.

We have implemented a p–C/C/SPMS strategy and a p–C/C/MPSS strategy for solving the initial static problem and for the post-optimization phase. In the p–C/C/SPMS strategy each processor performs a different search from the same initial solution. Once a processor finds a new best solution, it sends it to the other processors that re-initialize their searches. As in the sequential procedure (see Section 3), every μ iterations, parameters δ, λ, θ are randomly generated in the intervals $[0, \delta_{max}]$, $[0, \lambda_{max}]$ and $[0,\theta_{max}]$, respectively. In our implementation, δ_{max}, λ_{max} and θ_{max} were chosen in such a way that $p/2$ processes perform an intensification and the remaining $p/2$ processes carry on a diversification.

In the p–C/C/MPSS strategy each processor performs the same search from different solutions using the best parameter settings identified for the sequential heuristic. Then, in order to obtain the best diversification effect, every ω iterations each thread broadcasts the ρ_{ik} values obtained since the last update. Each other processor then aggregates all values received and continues its exploration with this information. A slight adjustment has to be made to the computation of the diversification penalties: instead of dividing by the number of iterations performed since the beginning of the search, one must divide by the total number of iterations performed by all searches (this can be approximated by multiplying the number of iterations by the number of processors).

Static solution construction. We have implemented two different procedures for generating a static solution. In the first approach (*SS1*), the static TS of Cordeau and Laporte (4) is invoked while leaving vehicle capacities unchanged. In the second approach (*SS2*), a static problem with reduced vehicle capacities $Q'_k := 0.5Q_k$ $(k = 1, \cdots, m)$ is defined

and the static TS of Cordeau and Laporte (4) is run. If a feasible solution is obtained, we stop. Otherwise, we reset $Q'_k := 0.75Q_k$ $(k = 1, \cdots, m)$ and run the TS again. If a feasible solution is obtained, we stop. Otherwise, we restore the original vehicle capacities $Q'_k := Q_k$ $(k = 1, \cdots, m)$ and run the TS again. This procedure is designed in the hope of equally distributing requests among vehicle routes.

Feasibility check. The *feasibility check* must be performed in a short amount of time since users requesting transportation by telephone or on the Internet are usually willing to wait just a few seconds. All threads are re-initialized using new solutions obtained by randomly inserting the new request in the current solution. If at least one of the solutions is feasible, this phase terminates. Otherwise, a parallel TS, in which communication among threads is suspended, is run. Hence, the feasibility check is made up of p independent threads. We have implemented two different procedures. In the first approach (*Finite Penalty*, FP), we update penalty parameters $\alpha = \beta = \gamma = \tau$ as in the static TS (4). In the second approach (*Infinite Penalty*, IP), we set $\alpha = \beta = \gamma = \tau = \infty$ in the hope of reaching feasibility more quickly.

Post-optimization phase. The aim of this phase if to reduce routing cost as much as possible. To this purpose, the parallel implementation of the static TS of Cordeau and Laporte (4)is used.

5 Computational results

The proposed parallel heuristics were implemented on a cluster of eight PCs, each of which has a Pentium III processor clocked at 700 MHz. Each process was coded in C++ and process communications was handled by the *Message Passing Interface* (MPI) software (12).

Except for the IP feasibility check, penalty parameters have been set as follows. In the SPMS procedures, parameters δ_{max}, λ_{max} and θ_{max} were set equal to 0.01, $0.005\sqrt{nm}$ and $5\log_{10} n$, respectively, for the $p/2$ processors performing intensification, and equal to 1, $0.05\sqrt{nm}$ and $15\log_{10} n$, respectively, for the $p/2$ processors performing diversification. In all cases, the values of μ and κ were both set equal to 10. In the MPSS procedures, $\omega = 100$ while the remaining parameters were set as in the sequential procedure (3). Moreover, after a preliminary test, we have set $\mu = 10$ and $\eta = 10000$.

The heuristics were tested on a set of 26 instances introduced by Cordeau and Laporte (4). Twenty of them (instances 1 to 20) were randomly generated on the basis of information provided by the Montreal Transit Authority, while the remaining six instances are real-life

large-scale problems provided by a Danish company. For each instance, half of the requests, randomly chosen, were considered as static. Each dynamic request i is supposed to become known T_i instants ahead, where T_i is uniformly distributed in [90,150].

The main goal of our computational experiments was to compare the eight procedures SPMS-SS1-IP, SPMS-SS1-FP, SPMS-SS2-IP, SPMS-SS2-FP, MPSS-SS1-IP, MPSS-SS1-FP, MPSS-SS2-IP, MPSS-SS2-FP with $p = 1$, 4 and 8 threads. In all cases, 30 seconds were allowed for each feasibility check. Average results on the randomly generated instances are reported in Table 1 while average results on the real-world instances are reported in Table 2. The meanings of the column headings are as follows:
- INSTANCE: instance number;
- SATREQ: percentage of requests satisfied;
- COST: routing cost of the solution.

Computational results show that the $SS1$ and $SS2$ procedures are almost equivalent, as they were able to satisfy 80.16% and 79.60% of the requests, respectively. The IP procedures are slightly better than the FP algorithms as they have satisfied 81.59% and 79.17% of the requests, respectively. Parallel computing can be beneficial in solving real-time vehicle routing problems as shown by the average percentage of requests satisfied for p=1, 4, 8 (76.84%, 80.79%, 83.51%, respectively). Finally, it is worth noting that randomly generated instances are much harder to solve than real-world instances (see Tables 1 and 2).

Table 1: Comparison among dynamic DARP procedures - randomly generated instances

Procedure	SS1-IP SATREQ	SS1-IP COST	SS2-IP SATREQ	SS2-IP COST	SS1-FP SATREQ	SS1-FP COST	SS2-IP SATREQ	SS2-IP COST
SPMS - 1 thread	66.80	4494.68	61.58	3693.72	71.76	3531.08	56.32	3811.08
SPMS - 4 threads	73.36	4171.36	73.19	4086.97	61.68	4087.46	68.55	3853.25
SPMS - 8 threads	64.96	4237.70	74.39	4703.04	70.63	4068.31	68.09	3995.17
MPSS - 1 thread	62.85	4055.36	57.35	3628.76	65.00	3321.01	55.87	3379.43
MPSS - 4 thread	72.86	4201.69	69.93	3872.07	65.88	4012.68	67.43	3915.53
MPSS - 8 thread	72.64	4268.71	71.53	4424.45	73.44	4249.16	65.49	3973.79

6 Conclusion

We have developed a number of parallel heuristics for the dynamic DARP, based on a tabu search previously proposed for the static case. Computational experiments indicate that

Table 2: Comparison among dynamic DARP procedures - real-world instances

Procedure	SS1-IP SATREQ	SS1-IP COST	SS2-IP SATREQ	SS2-IP COST	SS1-FP SATREQ	SS1-FP COST	SS2-IP SATREQ	SS2-IP COST
SPMS - 1 thread	95.69	5086.97	94.32	5084.80	91.65	4941.25	92.54	4212.53
SPMS - 4 threads	94.06	5219.90	95.29	5823.37	89.57	4890.97	90.83	4908.81
SPMS - 8 threads	97.39	5333.58	99.39	5789.58	95.66	5192.30	92.16	5075.51
MPSS - 1 thread	89.35	4776.41	91.66	4797.92	87.21	3700.72	89.46	4060.24
MPSS - 4 threads	92.18	5472.32	91.41	5731.15	95.48	5322.58	91.04	5041.70
MPSS - 8 threads	98.95	5735.35	97.14	5978.11	98.90	4713.78	95.36	5022.77

parallel computing can be beneficial in solving real-time vehicle routing problems. Moreover, the IP mechanism turns out to provide the best results while the choice of the initial static solution seems to be irrelevant. As far as future research is concerned, efforts should be concentrated on incorporating some look-ahead features into route building.

Acknowledgment

This research was partially supported by the Italian National Research Council, by the Center of Excellence on High-Performance Computing, University of Calabria, Italy, by a Strategic research grant provided by HEC Montréal, and by the Canadian Natural Sciences and Engineering Research Council under grants 227837-00 and OGP0039682. This support is gratefully acknowledged.

References

[1] R. Borndörfer, M. Grötschel, F. Klostermeier, and C. Küttner. Telebus Berlin: Vehicle scheduling in a dial-a-ride system. Technical Report SC 97-23, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1997.

[2] J.-F. Cordeau, M. Gendreau, G. Laporte, J.-Y. Potvin, and F. Semet. A guide to vehicle routing heuristics. *Journal of the Operational Research Society*, 53:512–522, 2002.

[3] J.-F. Cordeau and G. Laporte. The dial-a-ride problem: Variants, modeling issues and algorithms. *4OR - Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1: 89-101, 2003. Forthcoming.

[4] J.-F. Cordeau and G. Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research B*, 37:579-594, 2003.

[5] T.G. Crainic, M. Toulouse, and M. Gendreau. Towards a taxonomy of parallel tabu search algorithms. *INFORMS Journal on Computing*, 9:61–72, 1997.

[6] G. Ghiani, F. Guerriero, G. Laporte, R. Musmanno. Real-Time Vehicle Routing: Solution Concepts, Algorithms and Parallel Computing Strategies. *European Journal of Operational Research*, forthcoming, 2003.

[7] H. N. Psaraftis. Dynamic vehicle routing: status and prospects. *Annals of Operations Research* 61, 143-164, 1995.

[8] O.B.G. Madsen, H.F. Ravn, and J.M. Rygaard. A heuristic algorithm for the a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60:193–208, 1995.

[9] M. W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4:146–154, 1992.

[10] P. Toth and D. Vigo. Fast local search algorithms for the handicapped persons transportation problem. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory and Applications*, pages 677–690. Kluwer, Boston, 1996.

[11] P. Toth and D. Vigo. Heuristic algorithms for the handicapped persons transportation problem. *Transportation Science*, 31:60–71, 1997.

[12] G. William, M. Snir, W. Gropp, B. Nitzberg, and E. Lusk. *MPI: The Complete Reference*. MIT Press, Boston, Massachussetts, 1998.