Available online at www.sciencedirect.com
ScienceDirect
Transportation Research Part C 14 (2006) 157–174
TRANSPORTATION RESEARCH PART C
www.elsevier.com/locate/trc

Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries

Michel Gendreau [a,b], François Guertin [a], Jean-Yves Potvin [a,b,*], René Séguin [c]

[a] *Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Québec, Canada H3C 3J7*
[b] *Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Québec, Canada H3C 3J7*
[c] *Defence Research and Development Canada, Centre for Operational Research and Analysis, Ottawa, Ont., Canada K1A 0K2*

Received 5 December 2002; received in revised form 23 March 2006; accepted 25 March 2006

Abstract

This paper proposes neighborhood search heuristics to optimize the planned routes of vehicles in a context where new requests, with a pick-up and a delivery location, occur in real-time. Within this framework, new solutions are explored through a neighborhood structure based on ejection chains. Numerical results show the benefits of these procedures in a real-time context. The impact of a master–slave parallelization scheme, using an increasing number of processors, is also investigated.

Keywords: Vehicle dispatching; Real-time; Tabu search; Neighborhood; Ejection chains; Parallel computing

1. Introduction

Dynamic vehicle routing and dispatching problems have emerged as an area of intense investigations, due to recent advances in communication and information technologies that now allow information to be obtained and processed in real-time (Dror and Powell, 1993; Gendreau and Potvin, 1998; Powell et al., 1995). As compared to their static counterpart, these problems exhibit distinctive features (Psaraftis, 1995). In particular, the data (e.g., customers to be serviced) are not completely known before solving the problem, but are dynamically revealed as the current solution, based on incomplete and uncertain information, is executed. Thus, it is not possible for the decision maker to solve the entire problem at once. Such problems are found in many different application domains, like delivery of petroleum products and industrial gases (Bausch et al., 1995; Bell et al., 1983), truckload and less-than-truckload trucking (Powell et al., 1995; Powell, 1996), dial-a-ride

* Corresponding author. Address: Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Québec, Canada H3C 3J7.
E-mail address: potvin@iro.umontreal.ca (J.-Y. Potvin).

0968-090X/$ - see front matter
doi:10.1016/j.trc.2006.03.002

systems (Wilson and Colvin, 1977) and emergency services (Gendreau et al., 1997). In this paper, we focus on a problem typically found in courier services for the same-day local pick-up and delivery of small sized items (letters, parcels). As opposed to dispatching systems where a vehicle is dedicated to a single customer, like those found in emergency services or truckload trucking applications, planned routes are associated with each vehicle to specify the order of visit of the previously assigned, but not yet serviced, customer requests. Apart from exhibiting a strong routing component, the problem is further compounded by the presence of a pick-up and a delivery location associated with each request, as well as soft time windows at both locations.

In this context, many different factors must be considered when a decision about the allocation and scheduling of a new request is taken: the current location of each vehicle, their current planned route and schedule, characteristics of the new request, travel times between the service points, characteristics of the underlying road network, service policy of the company and other related constraints. It is thus a complex decision problem where the decision must typically be taken under considerable time pressure. Here, neighborhood search heuristics, in particular tabu search, are proposed as a means to effectively and efficiently tackle this dynamic problem and optimize the planned routes between the occurrence of new events. Such heuristics are shown to be more effective than simple dispatching rules based on insertion methods, even under stringent time pressure conditions. This result is achieved through a powerful neighborhood structure based on ejection chains. Conceptually introduced by Glover (1996), ejection chains have seldom been implemented in practice for solving vehicle routing problems, with notable exceptions for the Traveling Salesman Problem (Rego, 1998a), the Vehicle Routing Problem (Rego and Roucairol, 1996; Rego, 1998b) and the Vehicle Routing Problem with Time Windows (Bräysy, 2003; Caseau and Laburthe, 1999; Rousseau et al., 2002; Sontrop et al., 2005). This paper now shows that ejection chains can be made operational for a complex real-time pick-up and delivery problem with time windows, where response time is a crucial issue.

The remainder of the paper is divided along the following lines. First, the specific problem to be addressed is introduced in Section 2. Solution procedures based on neighborhood search heuristics are then proposed in Section 3 and their adaptation to a dynamic environment is presented in Section 4. A simulator that generates different operating scenarios is introduced in Section 5. Finally, computational results are reported in Section 6 and the conclusion follows.

2. Problem definition

2.1. Static problem

Our problem is motivated by local area (e.g. intra-city) courier services, where parcels or letters are picked up and delivered during the same day by the same vehicle. This problem falls in the general class of pick-up and delivery problems. Following the notation used in Savelsbergh and Sol (1995), the static version of the problem, where all requests are known in advance, can be formally stated as follows. Let N be a set of transportation requests of cardinality n, where the load of each request i must be carried from a pick-up location o_i to a delivery location d_i. Let N^+ and N^- be the set of all pick-up and delivery locations, respectively. That is, $N^+ = \cup_{i \in N} o_i$ and $N^- = \cup_{i \in N} d_i$. We define the set of locations V as $N^+ \cup N^-$ and V_0 as $V \cup \{0\}$, where 0 is a central depot. Let also M be a set of vehicles of cardinality m, with each vehicle $k \in M$ starting and ending its route at the central depot (at time $\underline{t}_k$ and $\overline{t}_k$, respectively). At each location $v \in V$, there is a time window $[e_v, l_v]$ for the start of service, where e_v and l_v are the lower and upper bounds, respectively. It is assumed that the service cannot start before e_v; thus, a vehicle must wait if it arrives before the lower bound. On the other hand, the upper bound is a soft constraint which may be violated. There is also a time window at the depot 0, where e_0 and l_0 are the earliest start time and latest end time of each route, respectively; a vehicle is allowed to end its route after l_0. For each pair of locations v and $v' \in V_0$, $t_{v,v'}$ is the travel time between v and v'. Dwell times at pick-up and delivery locations can be easily incorporated into the travel times and will not be explicitly considered in the following.

Given the above notation, a pick-up and delivery route R_k for vehicle k is a directed route through a subset $V_k \subset V$ such that

- R_k starts and ends in 0;
- $\{o_i, d_i\} \cap V_k = \emptyset$ or $\{o_i, d_i\} \cap V_k = \{o_i, d_i\}$ for all $i \in N$;

- If $\{o_i, d_i\} \cap V_k = \{o_i, d_i\}$ then o_i is visited before d_i;
- vehicle k visits each location in V_k exactly once;
- if v' is the predecessor of v in V_k and $v' \neq 0$, the arrival time of the vehicle at location v is $t_v = \max(t_{v'}, e_{v'}) + t_{v',v}$; otherwise $t_v = \underline{t}_k + t_{0,v}$.

Note that there is no capacity constraint in this problem because the items to be transported are rather small as compared to the overall vehicle capacity. A solution to this problem is a set of routes such that:

- R_k is a pick-up and delivery route for every $k \in M$;
- $\{V_k \mid k \in M\}$ is a partition of V.

2.2. Dynamic problem

In a typical courier service application, the problem is dynamic because the company continuously receives new calls for the pick-up and delivery of small items. Consequently, for each new request, a decision must be quickly taken about its allocation to a particular vehicle and its scheduling within the vehicle's planned route. Fig. 1 depicts a typical situation for one vehicle, where $+i$ and $-i$ represent the pick-up and delivery location of customer i, respectively. At the current time, the vehicle is located at point X and is moving towards its next service location. Its current planned route is the following: pick-up customer 1, pick-up customer 2, deliver customer 2, pick-up customer 3, deliver customer 3 and deliver customer 1. Assuming that a call from customer 4 comes in, the problem is to include the new pick-up and delivery locations in the route in order to minimize some cost function (which may imply resequencing the planned route).

Within the current simulated environment, the following assumptions are also made:

- Requests must be received before a fixed time deadline to be serviced the same day. Those that are received after the deadline, however, may be kept for the next day. Thus, there may be a number of pending or "static" service requests (for which a solution may have been constructed beforehand).
- Uncertainty comes from a single source, namely the occurrence of new service requests. In particular, there is no other source of uncertainty associated with customer requests, like cancellations or erroneous information. Furthermore, the travel times are known with certainty; no unexpected perturbation is assumed to come from the external world, like sudden congestion on the network caused by an accident, vehicle breakdown, etc.
- Communication between the central dispatch office and the drivers take place at each service point and is aimed at identifying their next destination. Consequently, the drivers do not know the global "picture" represented by their current planned route (which may be modified frequently by the optimization procedure).

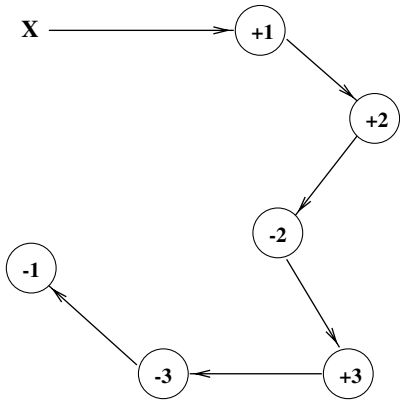

Fig. 1. A planned route.

- If some waiting time is expected at the driver's next destination, he is asked to wait at his current location. This is a form of least commitment strategy, as a movement is performed at the latest possible time to allow for "last minute" changes to the planned route due to the arrival of new service requests. Once a driver is "en route" to his next location, however, he must necessarily service that location (i.e., the vehicle cannot be diverted away from its current destination).

2.3. Objective

The objective can take many forms, but is typically aimed at achieving a good trade-off between operations costs (total travel time, overtime) and customer satisfaction (time window compliance). The cost function used throughout this work is to minimize a weighted sum of three different criteria: total travel time, sum of lateness over all pick-up and delivery locations and sum of overtime over all vehicles. The resulting function is thus

$$\sum_{k\in M} T_k + \alpha \sum_{v\in V} \max\{0, t_v - l_v\} + \beta \sum_{k\in M} \max\{0, \bar{t}_k - l_0\}$$

where T_k is the total travel time on route R_k and α and β are weighting parameters. In the experiments reported in Section 6, α and β were set to 1.

It is worth noting that this objective function does not include any probabilistic information about future incoming requests. In fact, the optimization procedure is applied only to requests known with certainty at a given time (see Powell (1996), for a model that incorporates uncertainty, although in a simpler setting where no consolidation of customer requests takes place).

3. Solution procedure

In recent years, neighborhood search heuristics have been applied with success to a variety of hard problems. For vehicle routing and dispatching problems, tabu search, in particular, has led to the design of heuristics that are among the most effective for pick-up or delivery-only problems (Gendreau et al., 1994; Gendreau et al., 1999; Potvin et al., 1996; Taillard et al., 1997). The presence of two-point requests, however, with a precedence constraint between the pick-up and the delivery makes the courier service problem much harder.

Since the optimization takes place over known requests only, a series of static multiple vehicle pick-up and delivery problems with time windows are solved over time, using appropriate information update mechanisms to keep track of the planned routes. Static problems of this kind are typically addressed through heuristic means. Many problem-solving approaches first cluster customers into smaller groups to reduce the main problem into different single vehicle problems, which are then solved exactly or heuristically (Bodin and Sexton, 1986; Dumas et al., 1989; Ioachim et al., 1995). Other approaches are based on pure insertion and exchange methods (Jaw et al., 1986; Madsen et al., 1995; Roy et al., 1984; Wilson and Colvin, 1977). In Van der Bruggen et al. (1993), a simulated annealing metaheuristic using a neighborhood structure based on arc exchange has produced solutions of good quality for single vehicle problems, but in relatively large CPU times. An exact algorithm is also reported in Dumas et al. (1991).

Solving pick-up and delivery problems in a dynamic setting introduces another level of complexity, as the response time becomes a crucial issue. In fact, most of the problem-solving methods mentioned above are too computationally expensive to be applied in a dynamic setting. The implementation of a sophisticated neighborhood structure based on ejection chains in such an environment thus represents a considerable challenge.

In the following, the ejection chain neighborhood developed for our dynamic pick-up and delivery application will be described in detail. Then, the iterative search framework will be presented.

3.1. Ejection chain neighborhood

Ejection chains (Glover, 1996; Rego and Roucairol, 1996; Thompson and Psaraftis, 1993; Xu and Kelly, 1996) are at the core of our problem-solving methodology. In our application, a request (with both its

pick-up and delivery location) is taken from one route and moved to another route, thus forcing a request from that route to move to yet another route, and so on. The chain may be of any length and may be cyclic or not. That is, it may end with the insertion of the last request in the route that started the process, thus producing a cycle, or in some other route not yet included in the ejection chain (note that a chain of length 0 corresponds to a request being rescheduled in its own route).

3.1.1. Ejection and insertion moves

An ejection move can be described as follows (where r_i and r_j denote the current routes of requests i and j, respectively)

1. select two requests i and j;
2. remove (eject) j from r_j;
3. move i from r_i to r_j.

In step 1, there are $O(n^2)$ ejection moves for a problem defined over n requests. Step 2 is performed in constant time, since the impact of removing each request from its own route is computed only once at the start of the neighborhood evaluation. This is done in $O(n^2)$ and does not add to the overall complexity ($O(n)$ calculations are required for each request, as the removal or addition of a request must be propagated along the route to get the new service times at each location and the exact objective value). Step 3 is performed in $O(n)$ through the following:

(a) choose the best insertion place for the pick-up location of request i in route r_j (without j), using an approximation function;
(b) propagate the impact of the chosen insertion place along the route;
(c) choose the best insertion place for the delivery location of request i in route r_j (with pick-up location of request i, but without j) using an approximation function;
(d) propagate the impact of the chosen insertion place along the route.

Approximations are thus used in Step 3 to achieve a total complexity of $O(n^3)$. These approximations are the followings:

- The insertion of the pick-up and delivery locations are not done simultaneously but rather in a sequential fashion. That is, the best insertion place for the pick-up is first identified. Then, the best insertion place for the delivery is chosen (after the insertion of the pick-up location).
- When evaluating the insertion places of the pick-up or delivery location, an approximation of the true objective function is used (with a propagation being performed only when the insertion place for the pick-up or the delivery has definitively been chosen). An approximation function is maintained at each service location to evaluate in constant time how a modification to the service time at this location impacts the cost function. For example, when a new location i is inserted between locations $k-1$ and k, the approximation function associated with location k is used to evaluate the impact of the time shift at k on the total cost. The approximation functions are constructed at each location at the start of the neighborhood evaluation: an exact evaluation with propagation is done for a few time shift values; then, a linear interpolation between these values provides the approximation function. Further details may be found in Taillard et al. (1997).

Insertion moves, where a request is moved to another route without ejection are evaluated in a similar way. Since there are $O(n)$ ways to select the request to be moved and only $O(m)$ ways to select the target route, the complexity in this case is $O(n^2m)$. The total complexity for both insertion and ejection moves is thus $O(n^2(n+m))$.

Note that the adverse effects associated with the use of approximation functions are alleviated by also considering the location just before and after the best insertion place (according to the approximation function). Hence, $3^2 = 9$ different insertions for the pick-up and delivery are considered and evaluated exactly, and the best one is selected at the end. The increase in computation time is offset by a better quality and stability of the

solutions produced. Further tests, using $5^2 = 25$ insertion places, did not produce any substantial improvement.

3.1.2. Ejection chains

The task of finding the best chain or cycle of ejection/insertion moves over the current set of routes is modeled as a constrained shortest path problem (which is then solved heuristically). In the associated graph, a "layer" of vertices is associated with each route. The vertices in a given layer correspond to the customer requests serviced on a particular route, plus a dummy vertex (see below). Most arcs connect pairs of vertices in different layers and are of two different types, depending on the move:

- Ejection arcs (i,j) model the ejection of request j by request i. The cost on these arcs corresponds to the insertion cost of request i in the new route (without j) minus the savings obtained by removing i from its current route. The best insertion place for request i is found using the procedure reported in Section 3.1.1, namely, a sequential insertion of the pick-up and delivery locations based on approximation functions constructed at the start of the neighborhood evaluation.
- Insertion arcs (i, dummy_k), $k \in M$, connect request i to the dummy vertex associated with route k. These arcs model the insertion of request i in route k (without ejection). The cost on these arcs corresponds to the insertion cost of request i in its new route minus the savings obtained by removing i from its current route.

Apart from these inter-layer arcs, intra-layer insertion arcs connect each request to its corresponding dummy route vertex. In this case, the request is simply rescheduled in the same route. This alternative is mandatory when a delivery location cannot be moved to another route because the corresponding pick-up location has already been serviced by the vehicle.

The shortest path calculated over this graph is constrained as the ejection chain is not allowed to eject more than one request per route. This restriction is necessary, otherwise the ejection costs would not be accurate for a second or any subsequent visit, due to previous ejections in the routes. This constraint also implies that negative cycles are not a problem since a layer cannot be visited twice (except to close a path to evaluate an ejection cycle).

The resulting constrained shortest path problem is NP-hard. In particular, the Generalized Traveling Salesman Problem (GTSP) is a special case of this problem as it corresponds to finding a shortest cycle which visits every route exactly once (Noon and Bean, 1991). The problem is thus solved in a heuristic way using an adaptation of the all-pairs Floyd–Warshall shortest path algorithm (Ahuja et al., 1993). This algorithm is modified to constrain the shortest paths to visit each layer or route at most once. Although the paths produced do respect the constraint, they are not optimal. The algorithm constructs a path, at step j, by considering the addition of a vertex in paths that cover (at most) route *1* to route $j-1$, where route k contains the kth selected vertex, $1 \leqslant k \leqslant j-1$. Clearly, the examination order of the vertices directly impacts the solution produced at the end. An approximate solution is obtained by a priori fixing the ordering strategy. Different strategies have been considered and tested: ordering based on the index of the vertices, hierarchical ordering based on the index of the routes followed by the vertex which provides the first improvement, and hierarchical ordering based on the index of the routes followed by the vertex which provides the best improvement. The second strategy proved to be a good compromise between efficiency and effectiveness and was finally selected. A greedy heuristic, which simply extends the current path with the best available ejection/insertion move, as long as an improvement is obtained, was also tried in place of the Floyd–Warshall algorithm. The heuristic was very fast but the results were poor.

3.2. Tabu search

Our tabu search heuristic follows the general guidelines provided in Glover (1989) to exploit the neighborhood structure presented in Section 3.1. In its simplest form, it can be summarized as follows:

1. start from a solution s;
2. $s_{\text{best}} \leftarrow s$;

3. set the tabu list to the empty list;
4. while a stopping criterion is not met do:
   (a) generate the neighborhood of s through non tabu moves (or tabu moves that lead to solutions that improve s_{best}) and select the best solution s';
   (b) if s' is better than s_{best} then $s_{best} \leftarrow s'$;
   (c) $s \leftarrow s'$;
   (d) update the tabu list.
5. output s_{best}.

Typically, the stopping criterion is based on a number of iterations (fixed number of iterations or maximum number of consecutive iterations without improving the best solution s_{best}).

An adaptive memory (Rochat and Taillard, 1995) and a decomposition procedure (Taillard, 1993) are added to this basic scheme to diversify and intensify the search. In the first case, a pool of routes taken from the best solutions visited thus far is exploited to restart the search in a new unexplored region of the search space. In the second case, the problem is decomposed to focus the search on smaller subproblems. These two mechanisms are described in the following. Additional details about the tabu list and its management are also provided.

3.2.1. Adaptive memory

The adaptive memory serves as a repository of routes associated with the best visited solutions. These routes are used after a fixed number of iterations to provide a new starting solution for the search. A solution is created by randomly selecting routes from the solutions found in memory, one at a time. The selection mechanism is designed in such a way that routes issued from better solutions have a higher probability of being selected. After each selection, routes with one or more requests in common with those found in the solution being constructed are discarded from the selection process. Consequently, some requests may need to be independently inserted at the end, due to the lack of admissible routes. The insertion is done in the same way as in Section 3.1.1 (i.e., pick-up first, delivery second), but an exact evaluation with propagation is used. In this case, it is less expensive to propagate than to construct approximation functions, as only a few insertions are typically needed to produce a complete solution. Through this approach, the new solution is typically made of routes taken from different solutions in memory, in a manner reminiscent of genetic algorithms.

The adaptive memory being of limited length, a solution produced by the search can be stored only if it is not already found in this memory. Forbidding duplicates is important, since the memory would tend, otherwise, to be filled up with identical or nearly identical solutions. To obtain a diversified search, a variety of solutions is thus to be preferred over a few identical solutions, even if some of these solutions are not of very high quality. Assuming that the new solution is not a duplicate, its inclusion will take place if the memory is not filled yet or if the solution is better than the worst solution in memory, in which case the worst solution is dropped.

In Section 6, another problem-solving scheme, also based on this adaptive memory, is investigated. In this case, the tabu search is restricted to a descent to the first local minimum. This approach thus corresponds to a multi-start local search descent with starting points provided by the adaptive memory. It is referred to as the adaptive descent heuristic in the following.

3.2.2. Decomposition

To intensify the search, the starting solution is decomposed into disjoint subsets of adjacent routes, with each subset or subproblem being processed by a different tabu search. When every subproblem is solved, the new routes are simply merged back to form a complete solution. The decomposition uses the polar angle associated with the center of gravity of each route. Through a sweep procedure, the domain is partitioned into sectors that approximately contain the same number of routes. A certain number of decomposition and reconstruction cycles take place before the final solution is sent to the adaptive memory, with each decomposition being applied to the solution resulting from the previous decomposition. At each cycle, the decomposition (i.e., the subset of routes in each subproblem) changes by choosing a different starting angle to construct

the sectors. Also, the number of tabu search iterations is increased from one decomposition to the next, as the solutions are increasingly difficult to improve (see Taillard (1993) and Taillard et al. (1997) for further details).

3.2.3. Tabu list

For efficiency purposes, the tabu list exploits the objective value of the new solution produced through a particular move. Namely, each potential solution is associated with a position in the list, which is the objective value of the solution (multiplied by 100 and rounded to the nearest integer) modulo the size of the list. The value stored at this position is an iteration number which corresponds to the end of the tabu status of the associated solution. Consequently, if the value found at this position is larger than the current iteration number, the move leading to the new solution is tabu. This approach can filter out legitimate solutions when they collide at the same position (i.e., if their objective value differs by a multiple of the tabu list size). However, this eventuality is unlikely, as the size of the tabu list is set to a large value.

4. A dynamic environment

Optimization takes place in a context where the dispatching situation is evolving dynamically due to the occurrence of new events. This has a number of implications on the algorithmic design presented in Section 3.2. First, appropriate update mechanisms must be implemented to keep track of the planned routes and to ensure that the adaptive memory is consistent with the current state of the world. Second, response times must be fast enough to cope with the dynamic environment. These issues are addressed in the following.

4.1. Handling new events

Basically, the procedure runs between the occurrence of new events to optimize the current planned routes. It interacts with its environment as follows:

1. while "no event" run the optimization procedure;
2. if "event" then
   (a) stop the optimization procedure;
   (b) if the event is "occurrence of a new request" then
      (i) update the solution found by the optimization procedure, as well as all solutions in the adaptive memory, through the insertion of the new request;
      (ii) add the updated solution found by the optimization procedure to the adaptive memory, if indicated;
      (iii) apply a local descent to the best solution in memory;
      otherwise ("end of service at a location")
      (i) add the solution found by the optimization procedure to the adaptive memory, if indicated;
      (ii) identify the driver's next destination, using the best solution stored in the adaptive memory;
      (iii) update all solutions in memory (see below);
   (c) restart the optimization procedure with a new solution generated from the adaptive memory.

Thus, each new request is inserted in every solution in adaptive memory. To quickly produce at least one solution of very high quality with the new request, a local descent is then applied to the best solution in memory. The descent stops at the first local minimum using the neighborhood structure based on ejection chains.

In the case of an "end of service", the serviced location is first removed from all solutions in memory. Then, the best solution in memory is used to identify the driver's next destination. The other solutions in memory are updated accordingly, that is, the driver's next destination is moved in first place if it is not already there. Given that a vehicle must necessarily service a location once it is "en route" to this location, the driver's next destination is then fixed in all solutions (and cannot be rescheduled by the optimization procedure).

4.2. Parallelization

In this application, it is important to react promptly to new occurring events. Fortunately, the nature of our algorithm (neighborhood search coupled with an adaptive memory) makes it a good candidate for parallelization. A parallel implementation was thus developed to increase the amount of computational work performed between the occurrence of events. Due to the coarse-grained, loosely connected parallel environment available to us (i.e., a network of SUN UltraSparc workstations), an implementation based on a master–slave scheme was developed. Basically, the master manages the adaptive memory and generates starting solutions from it; these solutions are sent to slave processes that improve them by performing tabu search and return the best solution found to the master. Apart from its low communication overhead, this scheme has one main advantage: it induces a multi-thread search strategy in which several distinct paths are followed in parallel (Badeau et al., 1997; Crainic et al., 1997). Another level of parallelization is provided within each search thread through the decomposition procedure introduced in Section 3.2.2. Namely, the subproblems created by the decomposition are allocated to different processors and are solved in parallel. Our approach thus combines the master–slave scheme presented above with this decomposition procedure to yield a two-level parallel organization.

To cope with environments with a high request arrival rate, one processor is dedicated to the management of the adaptive memory, decomposition of each problem into subproblems and dispatch of the workload among slaves. This processor does not perform tabu search, but applies a local descent on the best solution in memory after the insertion of a new request (cf., Section 4.1). This descent is performed by a specific process to prevent the master from optimizing for too much time while other tasks are accumulating (e.g., new requests coming in). Time sharing on the master processor allows for all tasks to be taken care of simultaneously, although more slowly.

Fig. 2 illustrates the parallelization approach on five processors, using two search threads and a decomposition of the main problem into two subproblems within each search thread. In the figure, processor 1 is the master, and processors 2–5 are slaves running the tabu search on their particular subproblem.

5. Simulator

A simulator was developed to produce different operating environments that reflect as closely as possible what is observed in the real world. This simulator is responsible for

- generating new requests according to different time–space distributions;
- updating the current dispatching situation (e.g., monitoring the movement of vehicles and the actual pick-up and delivery times);
- generating unexpected stochastic events, such as vehicle breakdowns and accidents.

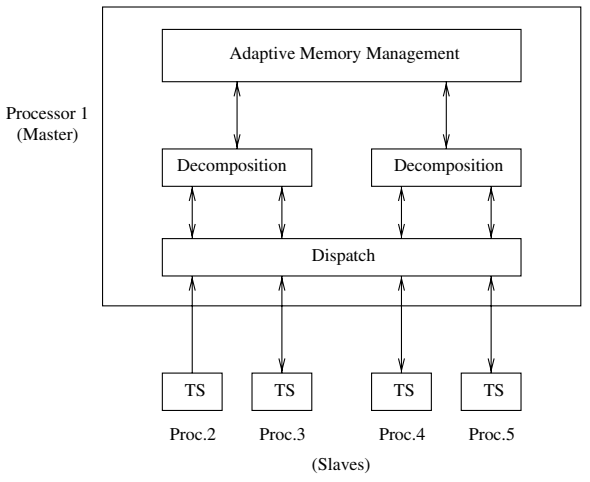

Fig. 2. Two-level parallelization scheme.

It is worth noting that the third capability of the simulator has not been exploited yet. The simulator first sets up a time horizon that spans the entire day. For example, an operations day which runs from 9:00 AM to 4:30 PM, has a time horizon of 450 min (or 7.5 h). Within this horizon, discrete-time events are produced by the simulator to account for the occurrence of new service requests and the execution of the planned routes (i.e., end of service at a pick-up or delivery location). We will now look more closely at these events.

5.1. Occurrence of a new request

The simulator generates the characteristics of each new request, namely, its time of occurrence as well as the pick-up and delivery locations with their time windows. These values must be feasible. For example, the time windows must leave enough time for the vehicle to reach the pick-up point, service this point, and go directly to the delivery point. The values must also be "realistic", as the occurrence of requests varies over time (e.g., peak hours) and space (e.g., densely populated area). To this end, both time and space are discretized: the dispatching area is divided into a set P of smaller rectangular zones and the time horizon is divided into a set L of smaller time intervals of, possibly, different lengths.

An activity matrix A is given where each element a_p^l, $p \in P$, $l \in L$, stands for the probability of any new activity request (either a pick-up or a delivery) to appear in zone p in the corresponding time interval l. Using this information, the simulator generates origin-destination matrices Π^l, $l \in L$, where π_{pq}^l is the probability that a request with pick-up location in zone p and delivery location in zone q will appear in time interval l.

The generation of a new request then proceeds as follows: given the current time interval l, a Poisson law of intensity λ^l is applied to determine the time of occurrence of the next request. Then, the Π^l matrix is used to determine the zones within which the pick-up and delivery locations will appear. Within each zone, the exact pick-up and delivery locations i and j are generated with a uniform distribution.

With regard to the time windows, the straight line distance between the pick-up and delivery locations i and j is used to compute the minimum travel time $\underline{t}(i,j)$ (using some a priori average speed). The time window interval at the pick-up location is then produced by first determining the earliest service time e_i based on the probability distribution shown in Fig. 3.

In this figure, t is the current time and $\tilde{t}$ is the latest feasible time to begin the service, namely:

$$\tilde{t} = l_0 - [\underline{t}(i,j) + \underline{t}(j,0)].$$

Hence, it is not possible to come back to the depot before the end of the day, l_0, if the service starts after $\tilde{t}$. Given these, the earliest pick-up time e_i is generated in the interval $[t, (t+\tilde{t})/2]$ with probability β and in the interval $[(t+\tilde{t})/2, \tilde{t}]$ with probability $(1-\beta)$, where β is drawn from a uniform distribution $U[0.6,1.0]$. Note that by using β values greater than 0.5, the earliest pick-up time distribution is biased towards the current time. The time window at the pick-up location is then set to $[e_i, e_i + \delta(l_0 - t)]$ where $\delta(l_0 - t)$ is a fraction of the time remaining until the end of the day. In this formula, δ is drawn from a uniform distribution $U[\underline{\delta}, \overline{\delta}]$ where $\underline{\delta}$ and $\overline{\delta}$ are user-defined parameters which may vary with location and time.

A time window is generated in the same way for the delivery location. In this case, the probability distribution for the generation of the earliest service time, as shown in Fig. 3, is defined over the interval

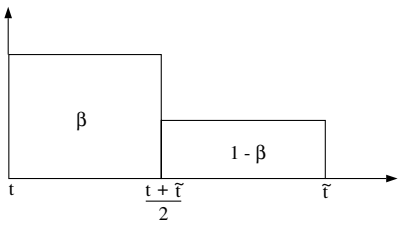

Fig. 3. Earliest pick-up time distribution.

$[e_i + \underline{t}(i,j), \tilde{t} + \underline{t}(i,j)]$ rather than $[t, \tilde{t}]$. As for the pick-up location, the width of the time window is a fraction of the remaining time.

5.2. End of service

Currently, this event is not random because no stochastic event can actually perturb the travel times between service locations. The arrival and end of service times at the next location are calculated for each planned route from the preceding location and the average speed of the vehicle.

6. Numerical experiments

The aim of this section is to compare different heuristic methods for handling new requests in dynamic environments when the arrival rate of these requests is of increasing intensity. The methods range from sophisticated ones, like tabu search, to simple ones, like insertion. It is expected that the relative performance of these methods will change depending on the environment. In the following, the operating scenarios considered for the computational experiments are first introduced. Numerical results follow.

6.1. Operating scenarios

Problems with 10 and 20 vehicles were generated with all vehicles moving at a constant average speed of 30 km/h. The area is a $5\text{ km} \times 5\text{ km}^2$ and the depot is located at (2.0 km, 2.5 km). The area is divided into 4×5 rectangular zones. Only one activity matrix A is defined; that is, the probability distribution over the zones do not change over time. These probabilities range from 0.01 to 0.13, depending on the zone; they are selected to create a high activity region around which the activity diminishes. Note that a uniform spatial distribution was also tested: although the solutions produced were somewhat different (e.g., the travel times were typically larger), the observations made in the following still hold in this case.

The day is divided into five time periods: early morning, late morning, lunch time, early afternoon and late afternoon. The lunch time period is half the length of the other ones (which are of equal length). Two sets of Poisson intensity parameters, in requests per minute, were used for these periods: (0.75, 1.10, 0.25, 0.40, 0.10) and (0.55, 0.70, 0.10, 0.40, 0.10). This leads to 33 and 24 requests per hour on average. The service time is equal to 5 min at each service location and a call is accepted only if there is at least 30 min between the call and the latest pick-up time. The variables δ for generating the pick-up and delivery time windows are taken from the uniform distributions $U[0.1, 0.8]$ and $U[0.3, 1.0]$, respectively. Simulations have been done for "days" of 4 and 7.5 h, using five different problems in each case.

6.2. Parameter settings

Preliminary experiments were performed to determine the parameter values for our optimization procedure. These are the followings:

6.2.1. Tabu search

The length of the tabu list is set to 100,000, with each entry in the list being associated with a particular objective value (objective values are multiplied by 100 and then rounded to the nearest integer to obtain the corresponding entry in the list). With respect to the neighborhood, exact evaluations are performed for three different time shift values at each service location to construct the approximation functions.

The number of iterations, as mentioned in Section 3.2.2, depends on the decomposition cycle. This number is set to 10 for the first decomposition and is incremented by 5 at each new decomposition (it is automatically reset at 10 after a new restart from the adaptive memory). The tabu tenure is set to half the number of iterations of the current decomposition cycle.

The minimum number of decomposition cycles is set to 3. However, this number is dynamically adjusted: when the number of restarts from the adaptive memory increases between two consecutive events (i.e., the time

interval between two events is longer, thus meaning that the current period is less intense), the number of decomposition cycles is also increased to allow the tabu search to work longer before returning a solution to the adaptive memory. Conversely, the number of decomposition cycles is decreased when the number of restarts between two consecutive events decreases.

6.2.2. Decomposition

For problems involving 10 and 20 vehicles, the solution is decomposed into 2 and 4 subsets of routes, respectively. As the problems are dynamic and may contain empty routes at the start, a solution is decomposed only if the resulting subsets contain at least 3 non empty routes.

6.2.3. Adaptive memory

The size of the adaptive memory is set to 32 solutions.

6.3. Numerical results

In this section, the tabu search is compared with other heuristic ways of handling new requests. The *insert* heuristic simply inserts a new request at its best insertion place, using the insertion procedure reported in Section 3.1.1. The *construct* heuristic rebuilds a new solution each time a request comes in. That is, all service locations are removed from the planned routes and a new set of routes is constructed by sequentially inserting each request at its best insertion place, using again the insertion procedure reported in Section 3.1.1. The inser-

Table 1
Simulation of 7.5 h with 20 vehicles and 24 requests per hour

Instance number	Insert	Construct	Insert+	Construct+	Adaptive decent	Tabu
1	785[a]	722	599	650	563	539
	50[b]	37	1	4	3	1
	21[c]	19	1	4	3	0
	856[d]	778	601	658	569	540
2	832	790	608	664	567	614
	47	43	1	0	3	3
	28	28	1	0	2	1
	907	861	610	664	572	618
3	924	823	702	773	676	629
	90	131	2	10	0	2
	69	92	2	8	0	1
	1083	1046	706	791	676	632
4	1080	880	748	808	695	700
	75	98	0	26	2	6
	54	74	0	23	1	5
	1209	1052	748	857	698	711
5	997	849	747	776	696	694
	84	109	0	2	3	0
	59	74	0	1	1	0
	1140	1032	747	779	700	694
Average	923	813	681	734	639	635
	69	84	1	8	2	2
	46	57	1	7	1	1
	1038	954	683	749	642	638

[a] Travel time.
[b] Lateness.
[c] Overtime.
[d] Total.

tion order is based on the time of occurrence of each request. The *insert*+ heuristic is similar to the *insert* heuristic, except that a local descent, using the neighborhood based on ejection chains, is applied to the solution obtained after the insertion. The descent stops at the first local minimum. The *construct*+ heuristic is similar to the *construct* heuristic, except that a local descent is applied to the solution obtained at the end of the construction. The last two heuristics, *tabu search* and *adaptive descent* both exploit the adaptive memory. The *adaptive descent* heuristic, as previously mentioned, is a variant of the former where the tabu search is restricted to a local descent to the first minimum. It is thus a multi-start local descent with starting points provided by the adaptive memory.

In Table 1, the results obtained on five different problem instances generated under scenario 1 are reported. This scenario corresponds to simulations of 7.5 h executed on a single processor, using 20 vehicles and the less intense request arrival rate (i.e., 24 requests per hour, on average), with the computer simulating and solving events at the same time as the real clock. Under scenario 1, each vehicle services 1.2 request per hour, on average. Given that a service time of 5 min is associated with each service point, 12 min are used for this purpose, thus leaving 48 min for traveling. Each entry in the table contains four different numbers: travel time, lateness, overtime and total (sum of the three first numbers), in this order. All results are in minutes, rounded to the nearest integer. Clearly, the methods using a local descent are much better than the others, with *adaptive descent* and *tabu* being the best. The poor results produced by the *construct*+ heuristic should be noted. At first sight, this heuristic should perform better than *insert*+, since a construction looks as a better way to reoptimize the current solution than a simple insertion. The explanation of this unexpected behavior is that each time a new request comes in, the *construct*+ heuristic loses the gains previously obtained by the local descent: the

Table 2
Simulation of 4 h with 10 vehicles and 24 requests per hour

Instance number	Insert+	Adaptive descent	Tabu
1	357[a]	345	336
	118[b]	83	65
	84[c]	51	55
	559[d]	480	456
2	370	369	386
	86	85	68
	68	58	53
	524	512	506
3	358	355	352
	129	120	139
	87	86	98
	574	562	589
4	359	350	359
	92	63	38
	71	48	31
	521	461	428
5	372	350	348
	153	88	75
	90	64	52
	615	502	476
Average	363	354	357
	115	88	78
	80	61	58
	559	503	493

[a] Travel time.
[b] Lateness.
[c] Overtime.
[d] Total.

occurrence of each new request triggers the reconstruction of the solution, without any consideration for the previous solution.

Given that scenario 1 is not very intense, most of the total cost comes from the traveling time, with only a few minutes of lateness or overtime. The results of *tabu* and *adaptive descent* are very close, with a difference of 0.6% in favor of *tabu*. The method *insert+* is the best among the other methods, but it is still 7% over *tabu*. It is worth noting that *tabu* and *adaptive descent* contain stochastic features and do not necessarily produce the same solution when applied two or more times on the same problem. We ran the tabu search heuristic 20 times on a particular problem instance and obtained a standard deviation of 2.7%. Since five different instances are solved by each heuristic, the standard deviation for the average is approximately 1.2%. Discrepancies are also due to the real-time nature of the problem. Since the times at which calculations and actions are performed by the algorithm can slightly vary from one execution to the next, the "synchronization" with external events is modified, with an obvious impact on the final solution. Hence, even simpler heuristics that use a descent phase may also exhibit some variance in the results, although to a much lesser extent.

Given the fact that *insert*, *construct* and *construct+* were not competitive in any of our experiments, we will restrict ourselves to the *adaptive descent*, *tabu* and *insert+* heuristics in the following. Scenario 2 was obtained by setting the horizon to 4 h and by reducing the fleet size from 20 to only 10 vehicles. The same request arrival rate of 24 requests per hour was kept, and each vehicle now services 2.4 requests per hour, on average. Table 2 shows the results. We can see that the lateness and overtime values have increased with regard to scenario 1. This is an indication that the reduced number of vehicles leads to a more challenging environment. The tabu search heuristic has now increased the gap with both *adaptive descent* (2%) and *insert+* (11.8%).

Table 3
Simulation of 4 h with 10 vehicles and 33 requests per hour

Instance number	Insert+	Adaptive descent	Tabu
1	486[a]	458	473
	4580[b]	4417	4392
	709[c]	696	699
	5775[d]	5571	5564
2	441	412	402
	973	944	867
	316	311	297
	1730	1667	1566
3	437	426	455
	1092	1009	853
	336	333	303
	1865	1768	1611
4	446	405	434
	1210	1031	1104
	344	317	348
	2000	1753	1886
5	524	525	495
	4325	4206	4121
	695	701	687
	5544	5432	5303
Average	467	445	452
	2436	2322	2267
	480	472	467
	3383	3239	3186

[a] Travel time.
[b] Lateness.
[c] Overtime.
[d] Total.

Scenario 3 is more dynamic than scenario 2 and was obtained by increasing the request arrival rate from 24 requests to 33 requests per hour, while keeping the same horizon length and number of vehicles. Under scenario 3, each vehicle services 3.3 requests per hour, on average. Hence, only 27 min are left for traveling during each hour, once the service times are subtracted. This increase in the number of requests has a dramatic impact on solution quality, as observed in Table 3. Almost 90% of the total cost is now due to overtime and lateness at customer locations. This scenario is also a ''turning point'' in the sense that the gap between *tabu* and both *adaptive descent* and *insert*+ has dropped from 2% to 1.5% and from 11.8% to 5.8%, respectively. When the request arrival rate is increased even more, we observed that this tendency keeps on. That is, the *adaptive descent* and *insertion*+ methods both get progressively closer to *tabu*, until no distinction can be made between

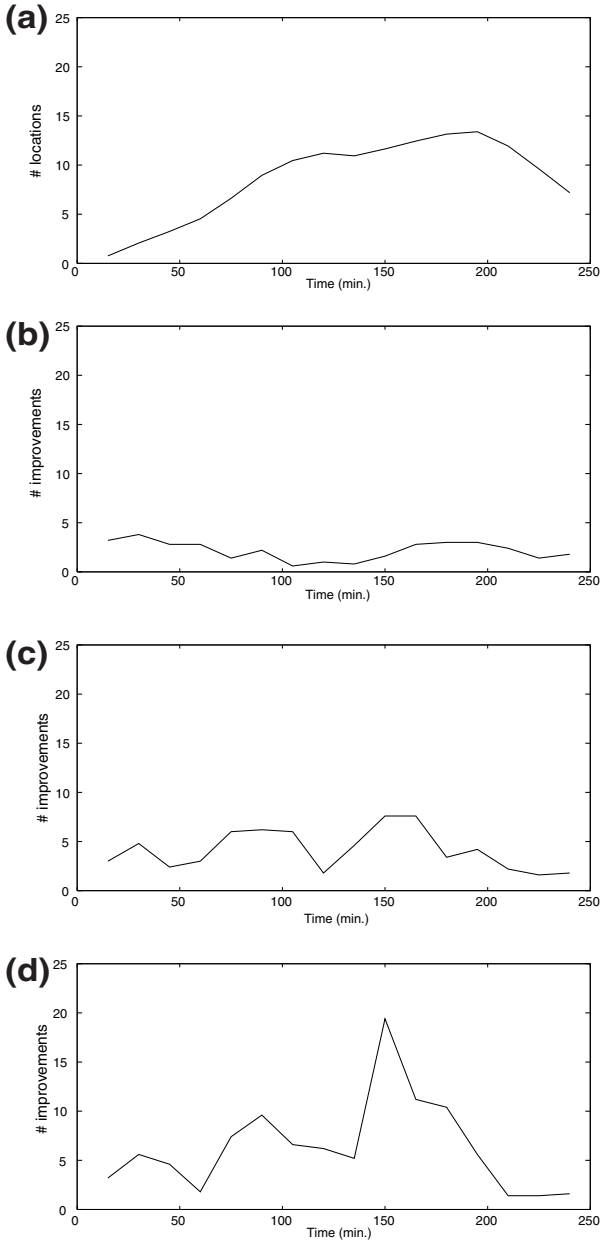

Fig. 4. Behavior of tabu search with increasing computation time: (a) average number of locations per route, (b) tabu search activity with normal clock, (c) slowing down the clock by a factor of 2.5 and (d) slowing down the clock by a factor of 5.

the various methods. At this point, the benefits associated with a sophisticated search framework vanish. One might say that with more requests per route, more opportunities for optimization are available; however, these cannot be exploited due to a lack of computation time between the occurrence of events. Fig. 4 illustrates this point on one instance taken from Table 3 (produced under scenario 3, a simulation of 4 h with 10 vehicles and 33 requests per hour, on average).

Curve (a) shows the average number of service locations per vehicle on the planned routes over time (i.e., locations that have been assigned to a vehicle but not serviced yet). The next three curves show the number of improvements to the best known solution over time slices of 15 min. Curve (b) is the "normal" one when the simulator generates events at the same time as the real clock. Curves (c) and (d) were produced by simulating a faster computer: in this case, the time was artificially slowed down by factors of 2.5 and 5, respectively. Clearly, the tabu search is more active when more computation time is available. In particular, the "peak" observed on curve (d), which happens when there is a fairly large number of service locations on each route, shows that the tabu search can now take advantage of optimization opportunities.

6.3.1. Parallelization

The benefits of the parallel implementation are illustrated for scenario 3, using 8 and 16 processors (plus the master). Table 4 shows the average result obtained with *tabu* over the five instances of Table 3. Improvements of 4.3% and 5.7% are obtained over the sequential implementation with 8 and 16 processors, respectively. The same trend is observed for the two other scenarios, but the gains are smaller. Under scenario 1, for example, an improvement of 2.2% is observed with 16 processors.

6.3.2. Computation times

In a dynamic setting, the number of requests on the planned routes varies over time. To provide a meaningful comparison, each method was evaluated when there is a "peak" of requests on the planned routes, using again the five instances of Table 3. This peak is reached after two to three hours of simulation: at this point, about 10 to 15 locations are found on each planned route. Table 5 shows the average computation time in seconds for a simple insertion, a reconstruction, one iteration of a neighborhood search based on ejection chains and a descent to the first local minimum.

Insertion and reconstruction procedures are faster, but they are likely to exploit only a small fraction of the time available between two events. When their task is over, they just sit and wait for the next event. The adaptive descent and tabu search heuristics, on the contrary, exploit all the time available between two events to optimize the current solution. They run as long as they are not interrupted by a new event, and are thus likely to find a better solution.

Table 4
Parallel implementation of tabu search

	1 processor	8 processors	16 processors
Average	452[a]	439	433
	2267[b]	2160	2125
	467[c]	449	447
	3186[d]	3048	3005

[a] Travel time.
[b] Lateness.
[c] Overtime.
[d] Total.

Table 5
Computation times (in seconds)

Insertion	Reconstruction	1 iteration (local search)	Descent (local search)
.02 s	.15 s	.23 s	7.7 s

7. Conclusion

This study has shown that both the adaptive descent and tabu search heuristics can cope with complex dynamic environments found, for example, in local pick-up and delivery services. When enough computing power is available, they produce improved results over simpler heuristics (even if the optimization takes place over known requests only, with no consideration for the future).

This work is based on some simplifying assumptions. Future work will thus be aimed at closing the gap with real-world courier service applications. In particular, we will look more closely at the following issues:

- Ability to divert a vehicle away from its current destination to service a new request in the vicinity of the vehicle's current position. Such opportunities may be exploited when communication between the dispatch office and the drivers can take place at any time (not just at service locations). In this case, a good trade-off must be found between computation time and solution quality, since vehicles are moving fast: if the evaluation takes too long, diversion opportunities may well be lost.
- Consideration of additional stochastic elements such as stochastic service times and travel times. Recourse actions, in particular, may be developed in the case of sudden unexpected events, like vehicle breakdown or congestion due to an accident.
- Ability to exploit probabilistic information about the future, like time-space distribution of service requests (using, for example, historical data). This ability could definitively improve the decision-making process at the current time.
- Consideration of complicating factors often found in real-world operations (e.g., drop-off sites for the transfer of packages between vehicles, hierarchical organization of the distribution system with a central hub and mini-hubs, etc.).
- Cooperation between dispatchers, either in a self-interested context or in a corporate context where all dispatchers work for the same company.

Acknowledgements

This work was partly supported by the Canadian Natural Sciences and Engineering Research Council under grant CRD 177440. Financial support was also provided by Lockheed Martin Electronic Systems Canada, and by the Defense Research Establishment Valcartier.

References

Ahuja, R.K., Magnanti, T.L., Orlin, J.B., 1993. Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Englewood Cliffs, New Jersey.

Badeau, P., Guertin, F., Gendreau, M., Potvin, J.-Y., Taillard, É., 1997. A parallel tabu search for the vehicle routing problem with time windows. Transportation Research 5C, 109–122.

Bausch, D.O., Brown, G.G., Ronen, D., 1995. Consolidating and dispatching truck shipments of Mobil heavy petroleum products. Interfaces 25, 1–17.

Bell, W., Dalberto, L.M., Fisher, M.L., Greenfield, A.J., Jaikumar, R., Kedia, P., Mack, R.G., Prutzman, P.J., 1983. Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer. Interfaces 13, 4–23.

Bodin, L., Sexton, T., 1986. The multi-vehicle subscriber dial-a-ride problem. TIMS Studies in the Management Sciences 26, 73–86.

Bräysy, O., 2003. A reactive variable neighborhood search for the vehicle routing problem with time windows. INFORMS Journal on Computing 15, 347–368.

Caseau, Y., Laburthe, F., 1999. Heuristics for large constrained vehicle routing problems. Journal of Heuristics 5, 281–303.

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

Dror, M., Powell, W.B. (Eds.), 1993. Special Issue on Stochastic and Dynamic Models in Transportation. Operations Research 41, 1–235.

Dumas, Y., Desrosiers, J., Soumis, F., 1989. Large scale multi-vehicle dial-a-ride problems. Cahiers du GERAD G-89-30, École des Hautes Études Commerciales, Montréal, Canada.

Dumas, Y., Desrosiers, J., Soumis, F., 1991. The pick-up and delivery problem with time windows. European Journal of Operational Research 54, 7–22.

Gendreau, M., Potvin, J.-Y., 1998. Dynamic vehicle routing and dispatching. In: Crainic, T.G., Laporte, G. (Eds.), Fleet Management and Logistics. Kluwer, Boston, pp. 115–226.

Gendreau, M., Hertz, A., Laporte, G., 1994. A tabu search heuristic for the vehicle routing problem. Management Science 40, 1276–1290.

Gendreau, M., Laporte, G., Semet, F., 1997. Solving an ambulance location problem by tabu search. Location Science 2, 75–88.

Gendreau, M., Guertin, F., Potvin, J.-Y., Taillard, É., 1999. Tabu search for real-time vehicle routing and dispatching. Transportation Science 33, 381–390.

Glover, F., 1989. Tabu search – Part I. ORSA Journal on Computing 1, 190–206.

Glover, F., 1996. Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Applied Mathematics 65, 223–253.

Ioachim, I., Desrosiers, J., Dumas, Y., Solomon, M.M., Villeneuve, D., 1995. A request clustering algorithm for door-to-door handicapped transportation. Transportation Science 29, 63–78.

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

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

Noon, C.E., Bean, J.C., 1991. A lagrangian based approach for the asymmetric generalized traveling salesman problem. Operations Research 39, 623–632.

Potvin, J.-Y., Kervahut, T., Garcia, B.-L., Rousseau, J.-M., 1996. The vehicle routing problem with time windows – Part I: Tabu search. INFORMS Journal on Computing 8, 158–164.

Powell, W.B., 1996. A stochastic formulation of the dynamic assignment problem, with an application to truckload motor carriers. Transportation Science 30, 195–219.

Powell, W.B., Jaillet, P., Odoni, A., 1995. Stochastic and dynamic networks and routing. In: Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L. (Eds.), Network Routing. North-Holland, Amsterdam, pp. 141–295.

Psaraftis, H.N., 1995. Dynamic vehicle routing: status and prospects. Annals of Operations Research 61, 143–164.

Rego, C., 1998a. Relaxed tours and path ejections for the traveling salesman problem. European Journal of Operational Research 106, 522–538.

Rego, C., 1998b. A subpath ejection method for the vehicle routing problem. Management Science 44, 1447–1459.

Rego, C., Roucairol, C., 1996. A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In: Osman, I.H., Kelly, J.P. (Eds.), Metaheuristics: Theory and Applications. Kluwer, Boston, pp. 661–675.

Rochat, Y., Taillard, É., 1995. Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics 1, 147–167.

Rousseau, L.-M., Gendreau, M., Pesant, G., 2002. Using constraint-based operators to solve the vehicle routing problem with time windows. Journal of Heuristics 8, 43–58.

Roy, S., Rousseau, J.-M., Lapalme G., Ferland, J., 1984. Routing and scheduling of transportation services for the disabled: summary report. Technical Report CRT-473A, Centre de recherche sur les transports, Montréal, Canada.

Savelsbergh, M.W.P., Sol, M., 1995. The general pickup and delivery problem. Transportation Science 29, 17–29.

Sontrop, H.M.J., van der Horn, P., Uetz, M., 2005. Fast ejection chain algorithms for vehicle routing with time windows. In: Blesa, M.J., Blum, C., Roli, A., Sampels, M. (Eds.), Lecture Notes in Computer Science 3636. Springer, Berlin, pp. 78–89.

Taillard, É., 1993. Parallel iterative search methods for vehicle routing problems. Networks 23, 661–673.

Taillard, É., Badeau, P., Gendreau, M., Guertin, F., Potvin, J.-Y., 1997. A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science 31, 170–186.

Thompson, P., Psaraftis, H.N., 1993. Cyclic transfer algorithms for multi-vehicle routing and scheduling problems. Operations Research 41, 935–946.

Van der Bruggen, L.J.J., Lenstra, J.K., Schuur, P.C., 1993. Variable-depth search for a single-vehicle pickup and delivery problem with time windows. Transportation Science 27, 298–311.

Wilson, N., Colvin, N.H., 1977. Computer control of the Rochester dial-a-ride system. Technical Report R-77-30, Department of Civil Engineering, MIT, Cambridge, USA.

Xu, J., Kelly, J.P., 1996. A network flow-based tabu search heuristic for the vehicle routing problem. Transportation Science 30, 379–393.