Vehicle Routing: Methods and Studies
B.L. Golden and A.A. Assad (Editors)

DYNAMIC VEHICLE ROUTING PROBLEMS*

Harilaos N. PSARAFTIS

Massachusetts Institute of Technology, Room 5-211, Cambridge, MA.02139, U.S.A.

The purpose of this paper is to put dynamic vehicle routing into perspective within the broader area of vehicle routing, as well as provide a flavor of recent progress in this area. We identify the important issues that delineate the dynamic case vis-a-vis the static one, comment on methodological issues, review generic design features that a dynamic vehicle routing procedure should possess, discuss the adaptation of static approaches to a dynamic setting, and describe an algorithm for the dynamic routing of cargo ships in an emergency situation. We conclude by recommending directions for further research in this area.

1. INTRODUCTION

By "dynamic vehicle routing" one traditionally means the dispatching of vehicles to satisfy multiple demands for service that evolve in a real-time ("dynamic") fashion. The vehicles may be taxicabs, trucks, ships, aircraft, etc. The service provided may consist of dropping off a passenger to the airport, picking up and/or delivering small parcels, delivering gases to industrial customers, shipping troops and materiel in case of a mobilization situation, or, in general, satisfying a wide variety of other transportation or distribution requirements in a broad spectrum of settings.

For all the explosive growth in the vehicle routing literature over the past several years (see Bodin et al. (1983), and, more recently, Golden and Assad (1987)), in a strict sense (see definition in Section 2) very little has been published on dynamic variants of vehicle routing problems. Of the 62 references cited in Golden and Assad (1987), only three include phrases such as "dynamic", "real-time", or "on line" in their titles. This state of affairs is to be contrasted with the real-world picture, in which a significant proportion of applications are dynamic rather than static. Among publications that have explicitly addressed a dynamic vehicle routing situation we may mention several papers or reports in the paratransit (or "demand responsive" transportation) area, such as Wilson et al. (1971, 1976, 1977) and Psaraftis (1980), a paper by Brown and Graves (1981) on the real-time dispatching of

*Research supported in part through Contract No. N00016-83-K-0220 of the Office of Naval Research and through a UPS Foundation grant to the MIT Center for Transportation Studies.

petroleum tank trucks, the award-winning work of Bell et al. (1983) on the bulk delivery of industrial gases, a report by Powell (1985) on the dynamic allocation of trucks under uncertain demand, and a report by Psaraftis et al. (1985) on the problem of cargo ship routing in a mobilization situation. This situation reflects the rather scant methodological base in dynamic vehicle routing as compared to static; indeed, most real-time implementations of vehicle routing problems are straightforward adaptations of static approaches. By contrast, the state-of-the-art in other areas that can be conceivably considered "close relatives" to dynamic vehicle routing, such as the dynamic dispatching of mobile servers (e.g., ambulances, fire engines, even tugboats - see Larson and Odoni (1980), Minkoff (1985), etc.), or the dynamic routing in communications networks (see Bertsekas and Gallager (1987), among others), is relatively rich in specialized methodologies explicitly developed for these problems.

The purpose of this paper is to put dynamic vehicle routing into perspective within the broader area of vehicle routing, as well as provide a flavor of recent progress in this area. It is not the intention of the paper to be encyclopaedic. Rather, the scope of the paper is to identify the important issues that delineate the dynamic case vis-a-vis the static one, comment on methodological issues, and describe one specific context and algorithm in the dynamic vehicle routing area.

In Section 2 of this paper we explore the relationship between static and dynamic routing, by identifying factors that make these two problems drastically different, and by commenting on the methodological implications of these differences. In Section 3 we review generic design features that a dynamic vehicle routing procedure should possess, discuss the transferrability of static approaches to a dynamic setting, and end by describing an algorithm for the dynamic routing of cargo ships. Finally, Section 4 recommends directions for further research in this area by introducing and briefly discussing the Dynamic Traveling Salesman Problem.

2. DIFFERENCES BETWEEN STATIC AND DYNAMIC VEHICLE ROUTING

In this section we explore the relationship between static and dynamic vehicle routing problems by focusing on those elements that make dynamic routing different from static, and hence generally necessitate specialized solution procedures for dynamic vehicle routing problems.

To make our discussion more clear, we define a vehicle routing problem as "static" if the assumed inputs to this problem do not change, either during the execution of the algorithm that solves it, or during the eventual execution of the route. By contrast, in a "dynamic" vehicle routing problem,

inputs may (and, generally, will) change (or be updated) during the execution of the algorithm and the eventual execution of the route. Actually, algorithm execution and route execution are processes that evolve concurrently in a dynamic situation, in contrast to a static situation in which the former process clearly precedes (and has no overlap with) the latter.

Dynamic vehicle routing differs from static in several ways, some of them obvious, some less obvious. The main differences are listed below. For dynamic vehicle routing:

(1) Time dimension is essential;
(2) Problem may be open-ended;
(3) Future information may be imprecise or unknown;
(4) Near-term events are more important;
(5) Information update mechanisms are essential;
(6) Resequencing and reassignment decisions may be warranted;
(7) Faster computation times are necessary;
(8) Indefinite deferment mechanisms are essential;
(9) Objective function may be different;
(10) Time constraints may be different;
(11) Flexibility to vary vehicle fleet size is lower;
(12) Queueing considerations may become important.

We now discuss each of these points and their implications in some detail.

(1) Time dimension is essential

In static vehicle routing, the time dimension may or may not be an important factor in the problem. If it is, the problem is usually termed a routing and scheduling problem. However, not all static situations have a scheduling component. Actually, most classical generic routing problems such as the Traveling Salesman Problem (single or multiple TSP), and the Vehicle Routing Problem (VRP) do not have a scheduling component. In all of these problems, times are assumed proportional to distances traveled, and therefore do not have to be considered explicitly and separately in the formulation and solution of the problem.

By contrast, in every dynamic vehicle routing situation, whether it be time-constrained or not, the time dimension is essential. At a minimum, we need to know the spatial location of all vehicles within our fleet at any given point in time during their schedule, and particularly when new customer or cargo requests or other information are made known. A fortiori, and in more common situations, we need to keep track of how vehicle schedules and scheduling options dynamically evolve in time.

(2) Problem may be open-ended

In contrast to a static situation, in which the duration of the routing

process is more or less bounded or known in advance, the duration of such process in a dynamic situation may neither be bounded, nor known. In fact, a typical dynamic vehicle routing scenario is that of an open-ended process, going on for an indefinite period of time. An implication of this is that whereas in a static problem one usually considers tours (vehicles return to their depot), in a dynamic problem one considers (open) paths. Other implications regard the types of objective functions that are relevant in a dynamic routing problem (see also (9) below).

(3) Future information may be imprecise or unknown

In a static case there may be no "past", "present" or "future", particularly if the problem has no scheduling component. But even if it has, information about all problem inputs is assumed to be of the same quality, irrespective of where within the schedule this input happens to be (beginning, middle, or end). This is not the case in a dynamic problem, in which information on any input is usually precise for events that happen in real time, but more tentative for events that may occur in the future. As in any real life situation, the future is almost never known with certainty in a dynamic vehicle routing problem. Probabilistic information about the future may be available (e.g., we may know the probability that a certain customer will request service on a particular day), but in many cases even that type of information may not exist (a taxicab company waiting for customers is a typical example).

(4) Near-term events are more important

An implication of the previous point is that in terms of making decisions in a dynamic vehicle routing situation, near-term events are more important than longer-term ones. This is not the case in a static setting, where because of uniformity of information quality and lack of input updates all events (whether in the beginning, in the middle, or at the end of a vehicle's route) carry the same "weight". In dynamic routing, it would be unwise to immediately commit vehicle resources (i.e., decide to assign a vehicle, or make routing decisions) to requirements that will have to be satisfied way into the future, because other intermediate events may make such decisions suboptimal, and because such future information may change anyway. Focusing more on near-term events (of course without adopting a totally myopic policy) is therefore an essential aspect of a dynamic vehicle routing problem.

(5) Information update mechanisms are essential

Virtually all inputs to a dynamic routing problem are subject to revision at any moment during the execution of the route. For instance, a vehicle may break down. A customer requesting service may change the time he or she wishes to be picked up. "No-show" situations may occur. Due to unpredictable weather conditions, a ship may not be able to arrive at a certain port as scheduled. And so on. It is therefore imperative that information update mechanisms be an integral part of the algorithm structure and input/output interface in a dynamic situation. Data structures and database management techniques that help efficiently revise problem inputs as well as efficiently figure out the consequences of such revisions (see also (6) below) are central to a dynamic routing scheme. By contrast, in a static scenario, the scope of such mechanisms is either nonexistent, or, at best, tangential to the core of the problem (e.g., perform sensitivity analysis, play "what if" games, etc.).

(6) Resequencing and reassignment decisions may be warranted

In a dynamic vehicle routing situation, the appearance of a new input may render decisions already made before that input's appearance suboptimal (with respect to a certain objective). This fact concerns both sequencing decisions (decide sequence of stops to serve a given set of points) and assignment decisions (allocate vehicles to demand points). Thus, the appearance of a new input (such as a new customer request) may necessitate either the resequencing of the stops of one (or more) vehicle(s), or the reassignment of those vehicles to demands requesting service (or both).

Figures 1 and 2 help further illustrate this point. Euclidean space is assumed in both cases. In Figure 1, a dial-a-ride vehicle starts from point A to service customers 1 and 2 (a pickup point is denoted by a "+" and a delivery point by a "-"). The objective is to minimize the total distance traveled by the vehicle until the last customer is delivered. Figure 1(a) shows the optimal route. If now customer 3 requests service - while the vehicle is still at A (Fig. 1(b)), the new optimal route is shown in Figure 1(c). Notice that under the presence of customer 3, it is no longer optimal to adhere to the same sequence of pick ups and deliveries deemed optimal for customers 1 and 2 alone. Put another way, if we were to keep the same sequence and were simply to find the best insertion of customer 3 into the previously optimal route, we would arrive at a suboptimal solution, shown in Figure 1(d).

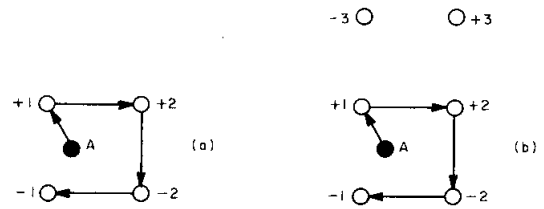

FIGURE 1 (a) & (b)

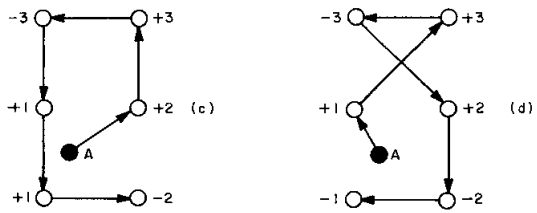

FIGURE 1 (c) & (d)

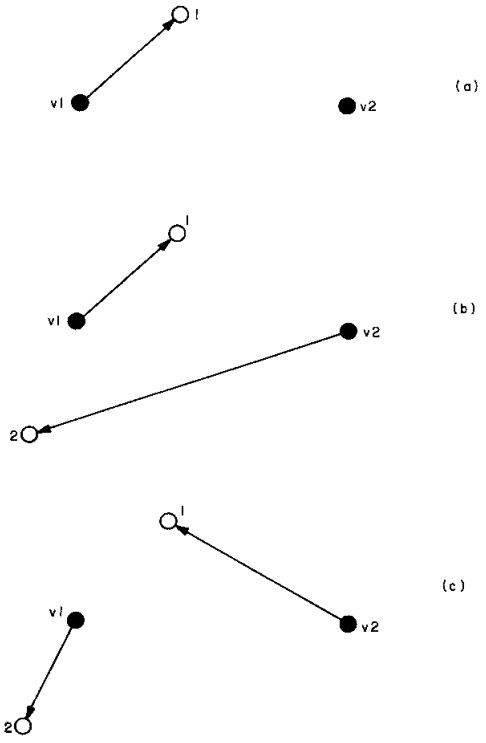

FIGURE 2

Figure 2 illustrates our point in a reassignment situation. Figure 2(a) shows the optimal allocation of two vehicles, v1 and v2 to satisfy the demand originating at point 1 (imagine that the vehicles are fire engines and point 1 is a fire). "Optimal" here means minimizing the maximum distance traveled by a vehicle. Under this assignment, v1 is dispatched to 1 and v2 idles. If now there is a second demand at point 2, the previous allocation is clearly suboptimal, for if we were to adhere to it we would have to have v2 travel a very long distance to go to point 2 (see Figure 2(b)). The optimal allocation in this case is the one shown in Figure 2(c). (Note that exactly the same observations are true if the objective is to minimize the total distance traveled by all vehicles).

A similar argument can be made whenever an input disappears (for instance a request is cancelled, a customer is a no-show, etc). As before, the deletion of an input will generally warrant a resequencing or a reassignment consideration.

(7) Faster computation times are necessary

The need to reoptimize routes and/or vehicle assignments on a continual basis in real-time necessitates computation times faster than those necessary in a static situation. In a static routing setting one may indeed afford the luxury of waiting for a few hours in order to get the output of the code solving the problem at hand. In such a setting, the problem may be solved exactly, and the code run in batch mode, perhaps overnight. This is not the case in a dynamic routing situation, in which the dispatcher wishes to know as soon as possible (i.e., in a matter of minutes, not hours) what the solution to a particular problem is in the presence of new information. The dispatcher may also want to run (again, in real-time) a few "what if" scenarios before deciding on the final action to take. The usual implication of this "running-time" constraint is that rerouting and reassignment decisions tend (by necessity) to be made on a heuristic and "local" fashion. Fast heuristics such as insertion, k-interchange and other improvement routines lend themselves to such a scheme (see also Section 3).

(8) Indefinite deferment mechanisms are essential

By indefinite deferment we mean the eventuality that the service of a particular demand be postponed indefinitely because of that demand's unfavorable geographical characteristics relative to other demands. An example of indefinite deferment for the single-vehicle case is depicted in Figure 3. As long as there are no time or priority constraints, and as long as there are unserviced requests near the current location of the vehicle, customer 1 (located far away from the central area) will always be scheduled to be serviced last (objective is to minimize total distance traveled until last customer is serviced).

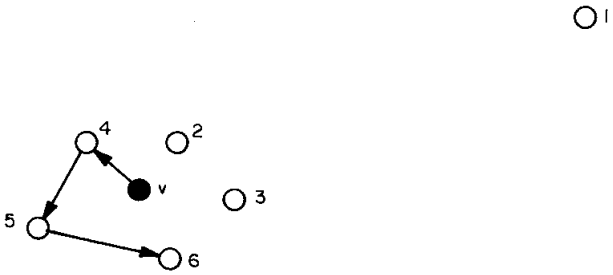

FIGURE 3

There are a variety of ways to alleviate this problem. Time constraints (in the form of time windows for instance) can generally force the vehicle to service a particular demand point on time, irrespective of that point's geographical location. A nonlinear objective function that penalizes excessive wait may also achieve the same goal (see also (9) below). Finally, priority constraints such as for instance limiting the number of positions each demand point can be shifted (up or down) away from its First-Come, First-Served (FCFS) position, can do the job as well (see Psaraftis (1980)).

(9) Objective function may be different

In a strict sense, the traditional "static" objectives of minimizing total distance traveled, or maximum distance traveled, or the overall duration of the schedule, may be meaningless in a dynamic setting. After all, if the process is open-ended, the overall duration of the schedule will be unbounded too. Measures of performance that have more meaning in a dynamic situation are more "throughput" or "productivity" - related. For instance, in a share-a-cab system we may wish to maximize the long-term average number of serviced customers per vehicle hour. Unfortunately, such an objective function does not always lend itself to algorithmic implementation, and one typically ends up replacing it with a set of surrogate objectives, sometimes identical, or closely related to the traditional static objectives, sometimes more complicated. Such criteria are typically applied to parts of the overall problem (decomposition by time or by space).

Optimizing only over known inputs might be a reasonable way to proceed if no information about future inputs is available. However, if some information about future inputs is available, it would make sense if such information is explicitly considered by the objective function. Since such information is usually vague, algorithms typically devise surrogate criteria

(sometimes nonlinear) which attempt to "predict" future system workload (see also Wilson and Weissberg (1976), Jaw et al. (1986)).

Nonlinear objective functions are also used to induce the algorithm to avoid certain undesirable phenomena such as indefinite deferment in the absence of "hard" time constraints (see Tharakan and Psaraftis (1980)). Queueing considerations may also warrant a nonlinear objective treatment (see Psaraftis et al. (1985) and (12) below).

(10) Time constraints may be different

The main difference between static and dynamic vehicle routing as far as time constraints are concerned is that in dynamic routing inputs such as earliest pickup (or delivery) times or (especially) latest pickup (or delivery) times tend to be softer than in a static situation. A time constraint is "soft" (as opposed to "hard") not only if it can be violated at a penalty - and this can happen in a static problem too - but also if it is subject to update and eventual revision. In the eyes of the dispatcher, a customer-imposed latest delivery time is essentially a soft constraint (even if in the eyes of the customer it is a hard one). This is so because denying service to that customer if that constraint cannot be met is usually a less viable alternative. If a "hard" deadline makes a routing problem infeasible, it is far better to renegotiate that deadline so as to make it feasible, than to declare infeasibility and quit. However, a deadline that is relaxed is, by definition, a "soft" constraint. Of course, some of these constraints may indeed be hard (or harder than others). However, in dynamic routing we would expect at least some of these constraints to be soft.

(11) Flexibility to vary vehicle fleet size is lower

In theory, another alternative to denying service to a customer if a time constraint cannot be met is to add an additional vehicle to serve that customer, at a cost. However, this proposition may not necessarily be viable in dynamic vehicle routing, because it may not be possible to have access to backup vehicle resources in real time. In a static situation, the time gap between execution of the algorithm and execution of the route is usually long enough to allow such determination to be made. Such a high flexibility does not generally exist in dynamic routing. Implications: if vehicle resources are scarce, some customers will receive lower quality of service (their due dates will be shifted, etc.) This may also result in queueing phenomena, which we discuss next.

(12) Queueing considerations may become important

A dynamic vehicle routing system may at times become saturated (or congested). This will happen if the rate of customer demand exceeds a certain threshold, beyond which the system simply cannot handle all of the

requests without creating excessive delays. In this case, any algorithm which tries to make assignment and routing decisions according to classical static surrogate criteria is bound to produce meaningless results. Unfortunately, whereas queueing theory and vehicle routing are two especially rich disciplines, very little is known about their interface. Under the current state of the art, including queueing considerations in vehicle routing is limited to empirical modeling (see also the MORSS algorithm described in Section 3 and the discussion of the Dynamic Traveling Salesman Problem introduced in Section 4).

It is clear that many of the above points are interrelated. For instance, (4) is an implication of (3), (6) and (7) are generally in conflict with each other, (9) and (10) may be used to take care of (8), (10) is due, in part, to (11), and (9) is true in part because of (12). Of course, specific dynamic environments vary, in general, with respect to each of these 12 points.

3. DYNAMIC VEHICLE ROUTING: SOLUTION METHODS

In this section we first review some generic design features that a dynamic vehicle routing procedure should possess in order to be useful in practice. We then discuss the adaptability of static approaches to a dynamic setting. We conclude by presenting the MORSS procedure developed by the author and his colleagues for the dynamic routing of cargo ships in a mobilization situation. Some of the concepts below draw from Psaraftis et al. (1985).

Design Features

In a generic sense, a dynamic vehicle routing procedure should possess, by design, the following features:

(1) It is obvious that such a procedure should be interactive. One should always have the "human in the loop" and enable him/her to override the computer at will. Various options should be designed, ranging from a completely "manual" approach where all major allocation (and possibly routing) decisions are made by the human operator, to more sophisticated modes where the computer deals with more difficult problems (e.g., routing) but still allows user discretion for "key" decisions. A fully automated mode, in which incoming data (e.g., new customer requests) are directly fed into the computer (say, by telephone, or by entering data in "checkpoint stations" in the area of service) can also be considered, so long as the human has the capability to override the machine in case an "unpredictable" situation occurs.

(2) The procedure should have a "restart" capability, that is, should be able to efficiently update routes and schedules at any time within the execution of a plan, without compromising "key decisions" already made. For

instance, if a certain cargo is en route from its origin to its destination at the time of the update, it would be nonsensical to have the procedure recommend that the cargo be reassigned on a different vehicle. On the other hand, if such a determination is made before that cargo is picked up, the reassignment recommendation is potentially implementable. New demands should be able to be inserted quickly into existing schedules. This means that the consequences of such an insertion (or of all potential insertions) should be able to be accounted for quickly. Efficient list processing techniques should be implemented for fast database manipulations.

(3) The procedure should be hierarchically designed, that is, allow the user to start the decision making process with "first-cut" gross feasibility analyses (possibly in several levels of aggregation) and only then proceed with detailed scheduling. Such a feature is considered important because a "quick and dirty" feasibility analysis may establish that, say, some due dates are infeasible, and hence allow the user to inquire for adjustments before further decisions are made.

(4) Finally (and perhaps obviously) we consider it important that the procedure be user-friendly. This is much more important in a dynamic setting than it is in a static one. In particular graphics aids are significant features that can enhance the efficiency of the man-machine interaction.

It is clear from the above list that in a dynamic vehicle routing procedure, information management and user interface issues become probably even more important than the theoretical performance and efficiency of the routing/scheduling (or, decision) module of the procedure. This module may still be considered to be the "core" of the overall procedure. However, in light of the discussion thus far, it is important to conclude that an efficient dynamic routing procedure implies much more than just an efficient core module. Actually, the core module itself must be designed in such a way that the above features can be easily implemented. This may have profound implications on the methodology used in the "core" module.

Adaptations of Static Approaches

Can a static routing approach, after suitable modifications, be efficiently used in a dynamic setting? The answer of course depends on the specific approach and setting.

A successful real-time implementation of a static approach has been reported in Bell et al. (1983), for the routing and scheduling a fleet of vehicles delivering a bulk product stored at a central depot. The routing "core" of the procedure is the static algorithm of Fisher et al. (1982), which is based on a mixed integer programming formulation of the problem and a solution using Lagrangian relaxation and a multiplier adjustment method. The core algorithm first heuristically generates a menu of possible vehicle

routes taking into account the geographical location of customers and the amounts of demands and truckloads. A set packing problem is then formulated so as to select from this menu of routes the subset that would actually be driven, specifying the time each route should start, the vehicle to be used, and the amount to be delivered to each customer. A typical scenario is for this overall system to be run once a day, so as to determine the schedules for the next two to five days. A separate "schedule change" module takes care of updates in input data that may occur in real-time.

In general, and in all fairness to a specific static vehicle routing algorithm, it would be unreasonable to expect that the procedure, as it stands, can handle any dynamic situation. In many cases, the algorithm would have to undergo a significant degree of redesign, most of it heuristic, to be tailored to the nature of the dynamic scenario.

There are generally two ways to adapt a static algorithm to a dynamic case. The first is to rerun the procedure virtually from scratch each time a (significant) revision of the input occurs (say, a new customer appears, or another one cancels his request, or a vehicle breaks down, etc.). This would involve generating a new set of routes at each input update, while guaranteeing that decisions already made (e.g., allocation of cargoes already enroute) are not compromised. Both steps would involve "freezing" many of the variables of the problem to values determined at previous iterations. Running a static algorithm in such a way could present several nontrivial challenges, one of which would be how to cope with the excessive computational burden of rerunning the algorithm over and over again while results are needed in real-time.

An alternative and more commonly used adaptation would be to handle dynamic input updates via a series of "local" operations, applied via the execution of an insertion heuristic (possibly followed by an interchange heuristic), after the static core algorithm is run. This would involve running the static algorithm just to initialize the process (say, once every day), and rely on "local" operations for all subsequent input updates. This author believes that this approach would work reasonably well if both the time horizon of the initial input is relatively long and subsequent input updates are infrequent (that is, if the overall problem is closer to static than dynamic). If the time horizon of the initial input is short or if subsequent input updates are numerous, the overall schedule would be less influenced by its initial solution and more by the subsequent local improvements. Such a scenario would drastically reduce the role of the static core in a dynamic situation, and shift the emphasis to the efficiency of the local operation method.

Local operations provide a reasonable way to handle dynamic input updates,

their principal advantage being execution speed. The fastest local operation method is an insertion approach, in which a new request is inserted within the current schedule, without perturbing the sequence of visits already planned. An insertion approach can also work in reverse, that is, whenever a request is deleted from the list for some reason (e.g., a cancellation). However, as mentioned in the previous section, the main drawback of the insertion method is that it cannot take care of the need of possible resequencing or reassignment operations. Among the various insertion methods we refer to the work of Wilson et al. (1971, 1976, 1977) that was specifically developed for the dynamic version of the dial-a-ride problem, and its more sophisticated variant for the advance request case with time windows (Jaw et al. 1986). Both algorithms keep track of how much (up or down the schedule) the service time of a customer can be shifted so that feasibility is maintained. The second reference can be readily implemented in a "mixed-demand" scenario.

Interchange methods can be used after the insertion to further improve the set of routes and schedules. Such methods are based on the concept of "k-interchange" made widely known by Lin (1965) and by Lin and Kernighan (1973) for the TSP (plus, their extensions for the multiple vehicle case). Resequencing and reassignment can be effectively performed by such methods, some variants of which are particularly powerful (e.g., for the TSP the $k = 3$ case is much more powerful than the $k = 2$ case, and the $k = 4$ case is only marginally better than the $k = 3$ case). Unfortunately, a drawback of such methods is that they tend to become computationally expensive as k increases. Also, computational effort has to be spent to check whether the improved route maintains feasibility. Sophisticated adaptations of the interchange concept that do not result in a substantial CPU time increase (in an order-of-magnitude sense) to check feasibility are due to Psaraftis (1983) for the dial-a-ride problem and Savelsbergh (1985) for the time-window TSP. Also, we note the approach of Or(1976) that looks only at a subset of possible interchanges and ignores those that are unlikely to result in an improvement. All of the above methods can be adapted in a dynamic situation.

The MORSS Algorithm

The remainder of this section is devoted to giving an overview of MORSS, a dynamic vehicle routing algorithm developed by this author, Jim Orlin, and their colleagues to assist schedulers of the U.S. Military Sealift Command (MSC) to route cargo ships in an emergency situation. Complete details on the MSC problem and MORSS (which stands for MIT Ocean Routing and Scheduling System) can be found in Psaraftis et al. (1985).

The MSC is the agency responsible for providing sealift capability for the Department of Defense. To do this, it provides peacetime logistical sealift

support of U.S. military forces worldwide, it develops contingency plans for the expansion of the peacetime sealift cargo fleet in case a military emergency occurs, and it has the operational control of this expanded fleet in mobilization situations.

Under conditions of military emergency, the objective of the MSC is to allocate cargo ships under its control (which can be as many as 1,000 in serious situations) to cargoes (whose number can be several thousands) so as to ensure that all cargoes, dry and liquid, arrive at their destinations as planned. Constraints that have to be satisfied include time windows for the cargoes, ship capacity, and cargo/ship/port compatibility. In addition, the scheduler has to allocate ships to cargoes so that three criteria are satisfied: First, cargoes should not be delivered (too) late. Second, ship utilization should be high. And third, port congestion should be avoided. The problem is dynamic in nature, as in a mobilization situation anything can change in real time. After extensive discussion with MSC personnel, an abstract model was developed for this problem. A simplified characterization of the input is the following (see Psaraftis et al. (1985) for more details).

For each cargo:

POE : Port of embarkation (origin)
POD : Port of debarkation (destination)
EPT : Earliest pickup time at POE (hard constraint)
EDT : Earliest delivery time at POD (hard constraint)
LDT : Latest delivery time at POD (soft constraint)
WEIGHT: Weight
VOL : Volume
SQFT : Deck surface area

For each ship:

LOC : Initial geographical location at time zero
W : Weight capacity
V : Volume capacity
S : Deck area capacity
SPEED: Speed
LOAD : Cargo loading/unloading rate
D : Draft

For each port:

DRAFT: Draft
THRU : Throughput characteristics of berths and terminals
DIST : Distances to all other ports and ship initial locations.

MORSS is based on the "rolling horizon" principle. In Figure 4, t_k is the "current time", that is, the time at the k^{th} iteration of the procedure. At t_k, MORSS considers only those known cargoes whose EPT's are between t_k and t_k+L, where L, the length of the rolling horizon, is another user input (say, L = 2 weeks). It then makes a tentative assignment of those cargoes to eligible ships (more on how assignments are made shortly). However, only cargoes within the "front end" of L are considered for permanent assignment. Those are those whose EPT's fall between t_k and $t_k + aL$, where a is another user input between 0 and 1. Thus, if L = 2 weeks and a = 0.5, MORSS will "look" at two weeks of cargo data into the future, but will commit to assign cargoes only within the first week. In such a fashion, MORSS places less (but, still, some) emphasis on the less reliable information on future cargo movements, since such information is more likely to change as time goes on.

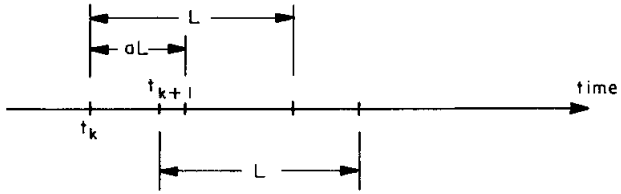

FIGURE 4

Iteration k+1 will move the "current time" to t_{k+1} (see Figure 4), which is equal to the time a significant input update has to be made (say, new cargo movement requirements are made known), or to the lowest EPT of all yet unassigned cargoes, whichever of the two is the earliest.

More formally, MORSS works as follows:

STEP 0 : Initialize locations of available ships.
Initialize "master list" of unassigned cargoes.
Select length L of individual time horizons.
Select fraction a (0 < a < 1).
Set k = 1, t_1 = 0.

STEP 1 : Set up next horizon (t_k, t_k+L).
Form list of cargoes eligible for assignment (all cargoes in master list whose EPT's are between t_k and t_k+L).

STEP 2 : Calculate assignment utilities for all eligible cargo/ship pairs (see Note (1) below).

STEP 3 : Form and optimize a transportation network using assignment utilities as arc costs. Resulting assignment forms the "tentative assignment" for (t_k, t_k+L) (see Note (2) below).

STEP 4 : Return to master list of unassigned cargoes (a) all unassigned cargoes by Step 3, (b) all tentatively assigned cargoes whose EPT's are between $t_k + aL$ and $t_k + L$ and (c) all tentatively assigned cargoes which "interact unfavorably" with one another or with cargoes assigned at previous iterations (see Note (3) below).
Make all other cargo/ship assignments in $(t_k, t_k + aL)$ "permanent". Remove permanently assigned cargoes from master list of unassigned cargoes.

STEP 5 : "Roll" time horizon. Set t_{k+1} = min (lowest EPT of cargoes in master list of unassigned cargoes, time "significant" input update occurs). Update ship locations at t_{k+1}.
Set k = k+1 and go to Step 1.

The above sequence of steps refers only to the "core" of MORSS, that is, does not include descriptions of preprocessing and postprocessing modules. Preprocessing modules conduct gross feasibility analyses, and postprocessing modules perform local improvements to the schedules produced up to Step 5. Also in Steps 3 and 4 the human scheduler can override the procedure and "force" it to make assignments of his/her choice.

The following notes provide additional details on the algorithm:

Note (1): The "utility" u_{ij} of assigning ship i to cargo j (Step 2) is defined by:

$$u_{ij} = u_{ij}(1) + u_{ij}(2) + u_{ij}(3) + u_{ij}(4).$$

(a) $u_{ij}(1)$ measures the assignment's effect on the delivery time of cargo j and by ship i. Dropping subscripts i and j and defining t as the tardiness of cargo j (= arrival time - LDT if > 0, zero otherwise), and V_{min}, V_{max}, t_0 and b as user-specified (and, generally, cargo-dependent) parameters, a "reasonable" functional expression for $u_{ij}(1)$ is

$$u_{ij}(1) = V_{min} + (V_{max} - V_{min})e^{-2(t/t_0)^b} \qquad (1)$$

The motivation of this functional form, which is typically bell-shaped with the utility very close to V_{max} if t is small and then dropping to a level very close to V_{min} if t increases ($u_{ij}(1) = V_{min} + 0.135(V_{max} - V_{min})$ for $t = t_0$, independent of b), -is the following. If the cargo is delivered early or on time (t = 0), its utility is maximum (V_{max}). If it is only a few days late (say, 1 or 2 days after its LDT), we assume that its value is close to V_{max}, but a bit lower. If it is more than a few days late, we assume its utility decreases rapidly with tardiness, until it reaches a "bottom" value (V_{min}) which is practically independent of t, if t is large ("if it's delivered after two weeks, it might as well be delivered after a month"). We have arrived at this form after discussions with MSC personnel, however we have also experimented with other, less CPU-intensive formulas that essentially exhibit the same features (see Psaraftis et al. (1985) for more details).

(b) $u_{ij}(2)$ measures the assignment's effect on the delivery times of all other cargoes already assigned to ship i. It does this by calculating the net change (if any) of the utilities of all such cargoes. These are computed according to the same formula suggested above (1).

(c) $u_{ij}(3)$ measures the assignments effect on the "efficiency" of use of ship i. Such an efficiency has two dimensions. First, we would like the ship to sail as full as possible. Second, we would like to keep some slack in the ship's schedule, so that additional cargoes can be carried by the ship in future iterations. Again dropping subscripts i and j and defining as C the ship's capacity, as R the residual capacity of the ship after the cargo has been picked up (C and R are expressed in weight, volume or area units depending on which of the three capacities is binding), and as F the slack in the ship's schedule averaged over all future stops, a reasonable formula for $u_{ij}(3)$ is:

$$u_{ij}(3) = V_s e^{-2(R/C)^c(1-fF/L)^d} \qquad (2)$$

where V_s, c,d and f are user-specified (and, generally, ship-dependent) parameters (c, d >0, $0 < f < 1$).

The motivation for this function (which is two-dimensionally bell-shaped) is that the utility reaches its maximum value (V_s) if R = 0 (ship is already full, so ship utilization is maximum), independent of the slack in the ship's schedule. It drops to lower values (for F = constant) if R increases. At the same time, the utility is a non-decreasing function of F, for R = constant, for, everything else being equal, one would prefer more flexibility in a ship's schedule than less. See Psaraftis et al. (1985) for more details.

(d) Finally, $u_{ij}(4)$ measures the assignment's effect on the system's port resources, as manifested by the increase in port queueing and congestion

caused by ship i's visit to the POE and POD of cargo j. Calculating queueing delays in this problem is very complicated. Instead, we chose to use the following empirical formula:

$$u_{ij}(4) = \begin{cases} V_p & \text{if cargo j's POE/POD coincide with ports that have to be visited anyway,} \\ V_p e^{-2(mN/P)^n} & \text{otherwise.} \end{cases} \quad (3)$$

In (3), N is the anticipated number of visits in cargo j's port during the current time horizon (by all ships), P is the throughput capacity of the port (expressed in equivalent number of visits), and V_p, m, and 1 are non-negative user-specified parameters. According to (3), $u_{ij}(4)$ drops from its maximum value V_p to zero if cargo j's port is not in the previous schedule and if N is high (see Psaraftis et al. (1985) for more details).

It is obvious that finding an effective set of values for the numerous user-specified parameters used in (1), (2) and (3) is in itself an extremely difficult (and probably data-dependent) task. We have carried out such a calibration to some extent, many times assigning "reasonable" but essentially arbitrary values to these parameters. More calibration would be carried out in a possible implementation phase (see also at the end of this section).

Note (2): The problem that determines the maximum utility "tentative assignment" (Step 3) is formulated as follows:

$$\begin{aligned} \text{Maximize} \quad & \sum_i \sum_j u_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_i x_{ij} \leq 1 \quad \text{for all j} \\ & \sum_j x_{ij} \leq K \quad \text{for all i} \\ & x_{ij} \geq 0 \quad \text{for all i and j} \end{aligned} \quad (4)$$

where $x_{ij} = 1$ if ship i is assigned to cargo j and zero otherwise. In (4), K is a user-specified integer (usually no more than 2 or 3). This formulation forbids more than K cargoes to be simultaneously assigned to the same ship (per iteration). This "artificial" constraint has been imposed so that one can justify adding all utilities in the objective function and limit the chance of "bad" cargo interactions. Note also that (4) does not explicitly incorporate ship capacity constraints. These constraints come into play in Step 4 of the procedure (moving from tentative to permanent assignment - see also Note (3)).

Note (3): It is clear that the "true" utility of a cargo-ship assignment is directly dependent upon the assignment of other cargoes to the same ship. This nonlinearity is patently neglected in our formulation (4) and may cause significant errors in case multiple cargoes are selected to be assigned to the same ship (put in another way, if each of cargoes 1 and 2 alone is a good

assignment for ship 3, it does not necessarily mean that both of them together are). Setting a low value for K limits (but cannot eliminate) the chance of unfavorable cargo interactions. However, should such interactions occur, some of the cargo assignments will have to be cancelled, and the corresponding cargoes will have to return to the pool of unassigned cargoes (see also example below).

Also when a permanent assignment is made, it may happen that the cargo is too large for the available residual ship capacity. In this case the cargo is split. As much as possible of it goes on the ship, and the remaining amount is returned to the pool of unassigned cargoes for the next iteration.

We now illustrate the MORSS approach by a rather rudimentary example. Figure 5 shows two ships (S1 and S2) and four cargoes (1 to 4, pluses are origins, minuses are destinations) in a Euclidean geographical area. Sailing times (days) are also shown in the figure. Assume that cargoes 1 to 3 are known at time t=0, but that cargo 4 appears only at time t=12. EPT's, EDT's and LDT's are given as follows:

Cargo	EPT	EDT	LDT
1	0	0	12
2	1	1	21
3	0	0	22
4	12	12	32

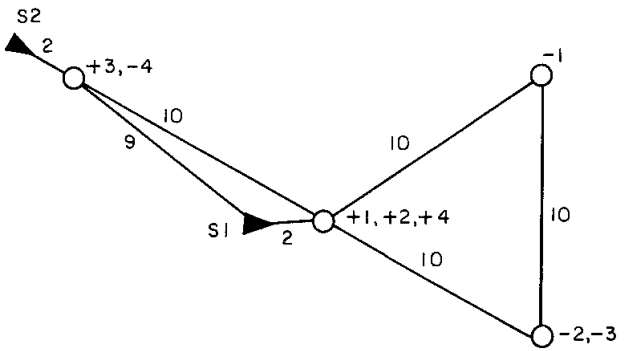

FIGURE 5

For the sake of the example, assume that no capacity constraints exist, and that the only relevant components of the utility are the delay-related ones, that is, the ones given by formula (1) shown earlier. Also assume the following values for MORSS parameters: L = 5, a = 0.5, V_{min} = 0, V_{max} = 1, t_0 = 5, b = 2, and K = 2.

Let us now see how MORSS behaves. The first time horizon (k = 1) is the interval (0, 5), which includes cargoes 1 to 3 (cargo 4 is not known at that time). MORSS then calculates utilities for all eligible cargo/ship pairs. It is straightforward to check that these are as follows:

Ship 1
Cargo 1: Tardiness = 0, Utility = 1
Cargo 2: Tardiness = 0, Utility = 1
Cargo 3: Tardiness = 7, Utility = 0.02

Ship 2
Cargo 1: Tardiness = 10, Utility = 0.00
Cargo 2: Tardiness = 1, Utility = 0.923
Cargo 3: Tardiness = 0, Utility = 1.

Notice the relatively small decrease in utility if the tardiness is small (cargo 2 by ship 2), and the precipitous utility drop if the tardiness increases considerably (cargo 3 by ship 1 and cargo 1 by ship 2).

The maximization of total utility via transportation problem (4) results in the following tentative assignment:

Ship 1: Gets cargoes 1 and 2 (utility = 2),
Ship 2: Gets cargo 3 (utility = 1).

Notice however that the simultaneous assignment of both cargoes 1 and 2 on ship 1 actually results in a lower total utility than the sum of individual utilities, because cargoes 1 and 2 interact unfavorably with one another (see Figure 5). Thus, MORSS will have to return either cargo 1, or cargo 2, to the pool of unassigned cargoes.

Ties can be broken arbitarily, or by using some secondary criteria. One of such criteria is to cancel the assignments of cargoes with higher EPT values. If this is so, the assignment of cargo 2 is cancelled, and MORSS will complete iteration 1 by permanently assigning cargo 1 to ship 1 and cargo 3 to ship 2. Cargo 2 is still unassigned.

Iteration 2 is for horizon interval (1, 6). MORSS here assigns cargo 2 to ship 2, because this is clearly the maximum utility assignment, given the previous assignments. The resulting schedules are established as follows:

Ship 1: Pick up cargo 1 on day 2, deliver it on day 12,
Ship 2: Pick up cargo 3 on day 2, pick up cargo 2 on day 12, deliver these cargoes on day 22.

The third iteration of MORSS will occur at time t = 12, when cargo 4 appears. At that time, ship 1 is just delivering cargo 1, and ship 2 is just picking up cargo 2. Despite the fact that the appearance of cargo 4 coincides exactly (location- and time-wise) with ship 2, it turns out that it is better in terms of utility to assign cargo 4 to ship 1, and let ship 2 proceed with the deliveries of cargoes 2 and 3. The rest of the schedules are thus established as follows:

Ship 1: Pick up cargo 4 on day 22, deliver it on day 32,
Ship 2: Pick up cargo 2 on day 12, deliver cargoes 2 and 3 on day 22.

It can be seen from the above that making simultaneous cargo-to-ship assignments is advantageous from a computational viewpoint, (vis-a-vis a one-by-one sequential assignment procedure), but that care should be taken to avoid unfavorable cargo interactions. Again, since K is low, such bad interactions are expected to occur relatively rarely.

There has been more refinement, testing, calibration, and computational experience with MORSS. The procedure has been coded in Pascal, and developed on both an IBM mainframe system (CMS) and on an Apollo workstation. Details can be found in Psaraftis et al. (1985) and in forthcoming publications (in preparation). The MSC has been very satisfied with the structure and generic features of the procedure, and is planning to proceed with an implementation phase in the foreseeable future. This implementation phase would link MORSS with "real" databases on ships, cargoes and ports, and would typically use MORSS for simulation and training purposes, so as to be prepared for the (undesirable) case in which a real mobilization situation occurs. Of course, many details on the actual operation of MORSS (such as, for instance, the time lag between execution of the algorithm and implementation of scheduling decisions, or how frequently the solution will be updated, or the degree of aggregation of the solution, or what kind of computer system would be used, etc.), will be determined in the implementation phase.

4. DIRECTIONS FOR FURTHER RESEARCH

As mentioned earlier, the state-of-the-art in dynamic vehicle routing methodologies is nowhere near that of the equivalent static case. This paper has attempted to put dynamic vehicle routing into perspective within the broader area of vehicle routing, as well as identify methodological and algorithmic design issues that are likely to be important in the dynamic case.

Methodologically, some effort should be spent to first develop a taxonomy of dynamic vehicle routing problems that parallels the traditional static classification structure. On this score, this author believes that one

should really start from scratch, for there isn't that much there to begin with. For instance, the classical (static) TSP is considered to be the "archetypal" (static) vehicle routing problem, in the sense that most other vehicle routing problems are its extensions and generalizations. What is known about the equivalent "dynamic" TSP? To this author's knowledge, the dynamic TSP (DTSP) is a problem that has not even been explicitly defined, let alone investigated or solved. We shall give one definition of the DTSP below, so as to hopefully stimulate the development of such a dynamic vehicle routing taxonomy, and the subsequent buildup of a methodological base in this area.

The Dynamic Traveling Salesman Problem (DTSP)

Let G be a complete graph of n nodes. Demands for service are independently generated at each node of G according to a Poisson process of parameter λ. These demands are to be serviced by a salesman who takes a (known) time of t_{ij} to travel from node i to node j of G, and spends a (known) time of t_0 servicing each demand (on location). If at time zero the salesman is at node 1, what should his "optimal" routing policy be? "Optimal" here may be with respect to a number of objectives (as will be further clarified below).

With the possible exception of the Probabilistic Traveling Salesman Problem (see also below), we are aware of no work by others on problems similar to the DTSP, as defined above. The work of this author is no exception, and he claims no special expertise on this problem. However, a very cursory investigation can reveal a number of interesting issues:

(a) As in dynamic routing in communications networks (see chapter 5 of Bertsekas and Gallager (1987)), there are two main classes of performance measures that are affected by routing decisions in this problem: (i) throughput measures and (ii) delay measures. According to (i), we may want to maximize the average expected number of demands serviced per unit time, that is, the limit, as T goes to infinity, of the ratio of the expected number of demands serviced within T, divided by T. According to (ii), we may want to minimize the average, over all demands, expected time from the appearance of a demand until its service is completed.

(b) Each of the two measures defined above is relevant or irrelevant, in the following sense. If the demand rate λ is "relatively low", then the vehicle will be able to keep up with the demand, and the throughput will be equal to $n\lambda$, irrespective of the routing policy. However, the average expected delay will definitely depend on the routing policy. Figure 6 further clarifies this point (Euclidean travel times are assumed and shown on the network links-in hours). Assume that t_0 = 1 hour and that λ = 0.01 demands/hour. Then it is clear that both the policy "service the (probably

sole) demand as soon as it appears and then wait" and the policy "service demands as-you-go, by performing tour 1-2-3-4-1 ad infinitum" achieve a throughput of 0.04 demand services/hour (other routing policies can exhibit the same throughput as well). However, in terms of average expected delay, the above two policies differ. This delay is approximately equal to (1/4) x (1 + 2 + 2.4 + 2) = 1.85 hours for the first policy (there are 4 equal-chance possibilities for the location of the next demand, relative to the current location of the vehicle, and the service time is one hour in all cases). For the second policy, the delay is approximately equal to 3 hours (2 hours average waiting time until vehicle comes to the demand point plus one hour service time). In the above calculations the probability of more than one active demand was ignored.

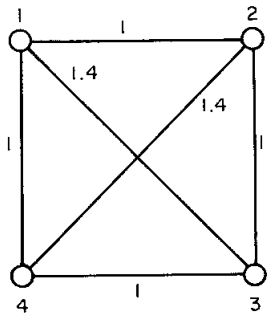

FIGURE 6

(c) Things become more complicated if λ takes on higher values. In this case, not only will the delay depend on the routing policy (as before), but the vehicle may not be able to cope with the demand, that is, the throughput may be forced to be lower than $n\lambda$. In this case, the queue of demands will become unstable, and then the option will be for customers to either suffer an infinite delay, or be denied service (rejected from the system). It is important to realize that whereas for very high values of λ no routing policy will be able to cope with the demand, for intermediate values of λ it is precisely the routing policy which will determine whether the system can handle the demand (and result in realistic delay values), or cannot handle it (and result in infinite delays). For instance, with respect to the previous example (Figure 6), and if λ is high, the policy "perform tour 1-2-3-4-1 ad infinitum, servicing only one demand at a time" achieves a throughput of 4/8 = 0.5 demand services/hour. However, if this policy is applied via tour 1-3-4-2-1, the throughput drops to 4/8.8 = 0.45 demand service/hour. Notice

that both throughputs are independent of λ and much lower than the theoretical maximum value of 4λ, if λ is high. In both cases, delays are expected to be infinite (if demands are not rejected). What happens however, if λ is in the vicinity of 0.1 demands/hour? It seems that the former policy is superior to the latter, because of the higher throughput. Does this mean that the optimal policy if λ is very high is always to perform the optimal TSP tour?

(d) The other side of the coin is the following. If λ is low, it may make sense for the vehicle to move to a strategically located node (such as the graph's median, for instance), in anticipation of the next demand. To our knowledge, this issue has been addressed in the context of facility location in a congested network (see Berman et al. (1985) and Chiu et al. (1985)). It would seem that if λ is extremely low and if the objective is to minimize average expected delay, the DTSP resembles the 1-median problem. However, and in contrast to the locational problems examined in the above two references, in the DTSP the server is not restricted to return to a prespecified node after the service of a given demand. Further research is necessary to explore the relationship between these two classes of problems (especially for intermediate values of λ).

Several other variants of the DTSP can be considered. For instance, the graph can be incomplete, symmetric, or Euclidean. Morever, each node can have its own Poisson parameter λ_i. Actually, the demand generation process does not have to be Poisson.

As stated above, the closest problem to the DTSP that comes to mind is the Probabilistic Traveling Salesman Problem (PTSP) (the reader is referred to Jaillet (1985) and to the paper by Jaillet and Odoni in this volume). In the PTSP, a demand at each node occurs (with probability p), or does not occcur (with probability 1-p) during a given day. The PTSP calls for the construction of a route R through all of the nodes so that the expected travel time of the actual route that will be traveled is minimized. The convention here is that the actual route will be based on R so that nodes that have no demand on a particular day will simply be skipped.

Note that for all the resemblance between DTSP and PTSP, the PTSP is really a static problem, for the determination of the optimal R has to be made before actual dispatching of the salesman, even though the actual route to be traveled depends on which (if any) of the demands is actually "active" that day. By contrast, in the DTSP the salesman's decisions have to be based on the current, and, generally, on the probable future states of the system as well.

How can this basic problem be solved? Under what circumstances is a myopic policy (optimize over known demands only) optimal? What happens if

service time is zero? Does it make sense to let demands accumulate before the vehicle departs from a node? And so on. Answers to these (and similar) questions would increase our knowledge about dynamic vehicle routing problems, and might motivate further work on more realistic (and more difficult) variants.

ACKNOWLEDGEMENTS

The author is indebted to the Editors and referees for their comments, and to Amedeo Odoni for pointing out the relationship between the Dynamic Traveling Salesman Problem and some locational problems on congested networks.

REFERENCES

Bell, W., L.M. Dalberto, M.L. Fisher, A.J. Greenfield, R. Jaikumar, P. Kedia, R.G. Macj, and P.J. Prutzman, 1983. Improving the Distribution of Industrial Gases with an On-Line Computerized Routing and Scheduling Optimizer. Inferfaces 13, 4-23.

Berman, O., R.C. Larson, and S. Chiu, 1985. Optimal Server Location on a Network Operating as an M/G/1 Queue. Operations Research 33, 746-771.

Bertsekas, D.P., and R. Gallager, 1987. Data Networks, Prentice-Hall.

Bodin, L.D., B.L. Golden, A. Assad and M.O. Ball, 1983. Routing and Scheduling of Vehicles and Crews: The State of the Art. Computers and Operations Research 10, 62-212.

Brown, G., and G. Graves, 1981. Real-Time Dispatch of Petroleum Tank Trucks, Management Science 27, 19-31.

Chiu, S., O. Berman, and R.C. Larson, 1985. Stochastic Queue Median Problem on a Tree Network. Management Science 31, 764-772.

Fisher, M.L., A.J. Greenfield, R. Jaikumar, and P. Kedia, 1982. Real-Time Scheduling of a Bulk-Delivery Fleet: Practical Application of Lagrangian Relaxation. Report 82-10-11, Decision Sciences Dept., University of Pennsylvania.

Golden, B.L., and A. Assad, 1987. Perspectives on Vehicle Routing: Exciting New Developments. Operations Research 34, 803-810.

Jaillet, P., 1985. The Probabilistic Traveling Salesman Problem. PhD thesis, Department of Civil Engineering, MIT.

Jaw, J.J., A.R. Odoni, H.N. Psaraftis, and N.H.M. Wilson, 1986. A Heuristic Algorithm for the Multi-Vehicle Advance-Request Dial-A-Ride Problem with Time Windows. Transportation Research 20B, 243-257.

Larson, R.C., and A.R. Odoni, 1980. Urban Operations Research, Prentice-Hall.

Lin, S., 1965. Computer Solutions to the Traveling Salesman Problem. Bell System Technical Journal 44, 2245-2269.

Lin, S., and B.W. Kernighan, 1973. An Effective Heuristic Algorithm for the Traveling Salesman Problem. Operations Research 21, 498-516.

Minkoff, A.S., 1985. Real-Time Dispatching of Delivery Vehicles. PhD Thesis, Operations Research Center, MIT.

Or, I., 1976. Traveling-Salesman-type Combinatorial Problems and Their Relation to the Logistics of Blood Banking. PhD Thesis, Dept. of Industrial Engineering and Management Sciences, Northwestern University.

Powell, W., 1985. An Operational Planning Model for the Dynamic Vehicle Allocation Problem with Uncertain Demands. Department of Civil Engineering Working Paper EES-85-15, Princeton University.

Psaraftis, H.N., 1980. A Dynamic Programming Solution to the Single-Vehicle Man-to-Many Immediate Request Dial-A-Ride Problem. Transportation Science 14, 130-154.

Psaraftis, H.N. 1983. k-Interchange Procedures for Local Search in a Precedence-Constrained Routing Problem. European Journal of Operational Research 13, 391-402.

Psaraftis, H.N., J.B. Orlin, D. Bienstock, and P.M. Thompson, 1985. Analysis and Solution Algorithms of Sealift Routing and Scheduling Problems: Final Report. Working Paper No. 1700-85, Sloan School of Management, MIT.

Savelsbergh, M.W.P., 1985. Local Search in Routing Problems with Time Windows. Annals of Operations Research 4, 285-305.

Tharakan, G.G., and H.N. Psaraftis, 1981. An Exact Algorithm for the Exponential Disutility Dial-A-Ride Problem. Working Paper, MIT.

Wilson, N.H.M., J.M. Sussman, H.K. Wang, and B.T. Higonett, 1971. Scheduling Algorithms for Dial-A-Ride systems. Urban Systems Laboratory Report USL TR-70-13, MIT.

Wilson, N.H.M., and H. Weissberg, 1976. Advanced Dial-A-Ride Algorithms Research Project: Final Report. Report R76-20, Dept. of Civil Engineering, MIT.

Wilson, N.H.M., and N.H. Colvin, 1977. Computer Control of the Rochester Dial-A-Ride System. Report R77-31, Dept. of Civil Engineering, MIT.