A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem

HARILAOS N. PSARAFTIS

Massachusetts Institute of Technology, Cambridge, Massachusetts

An investigation of the single-vehicle, many-to-many, immediate-request dial-a-ride problem is developed in two parts (I and II). Part I focuses on the "static" case of the problem. In this case, intermediate requests that may appear during the execution of the route are not considered. A generalized objective function is examined, the minimization of a weighted combination of the time to service all customers and of the total degree of "dissatisfaction" experienced by them while waiting for service. This dissatisfaction is assumed to be a linear function of the waiting and riding times of each customer. Vehicle capacity constraints and special priority rules are part of the problem. A Dynamic Programming approach is developed. The algorithm exhibits a computational effort which, although an exponential function of the size of the problem, is asymptotically lower than the corresponding effort of the classical Dynamic Programming algorithm applied to a Traveling Salesman Problem of the same size. Part II extends this approach to solving the equivalent "dynamic" case. In this case, new customer requests are automatically eligible for consideration at the time they occur. The procedure is an open-ended sequence of updates, each following every new customer request. The algorithm optimizes only over known inputs and does not anticipate future customer requests. Indefinite deferment of a customer's request is prevented by the priority rules introduced in Part I. Examples in both "static" and "dynamic" cases are presented.

Dial-A-Ride involves the dispatching of a fleet of vehicles to satisfy

Transportation Science
Vol. 14, No. 2, May 1980

0041-1655/80/1402-0130 $01.25

demands from customers who call a vehicle operating agency requesting service, requesting that is, to be carried from specified points to other similarly specified points. Such demand-responsive transportation systems have been in operation in several metropolitan areas of the United States as well as abroad during the recent years: Rochester, New York; Ann Arbor, Michigan; San Jose, California; Tokyo, Japan, etc.

Several versions of dial-a-ride service exist today, giving rise to several types of what we shall call the "*dial-a-ride problem.*" The particular version of the problem we shall examine is a single-vehicle, many-to-many, immediate-request one. "Many-to-many" means that the origins (pickup points) as well as the destinations (delivery points) of the various customers are all distinct points. "Immediate-request" means that every customer requesting service wishes to be serviced as soon as possible.

We shall study a generalization of the objective function of the classical Traveling Salesman Problem (TSP), namely, the minimization of a weighted combination of the time needed to service all customers and of the total degree of "dissatisfaction" they experience until their delivery. This dissatisfaction is assumed to be a linear function of the time each customer waits to be picked up and of the time he spends riding in the vehicle until his delivery. In addition, vehicle capacity constraints are part of our problem.

Our study is organized in two parts. Part I focuses on the "static" case of the problem. Part II focuses on the corresponding "dynamic" case.

I. THE "STATIC" CASE

IN THE "STATIC" CASE, each customer requests service by telephone and thus a chronological list of requests can be formed. At a certain point in time ($t = 0$) a single vehicle becomes available. The task of the vehicle operator is to provide service to the customers the list contains at $t = 0$, and only to those. In other words, at $t = 0$ our list is closed and no new customer requests are appended to it during the execution of the route. Any such requests can be arranged in another list for consideration after the completion of the route or for the scheduling of another vehicle. In the "dynamic" case, new customer requests occurring during the execution of the route are automatically eligible for consideration at the time they appear.

The following two points will be made in the formulation of the "static" problem in order to facilitate its subsequent extension to the equivalent "dynamic" case:

First, vehicle routes will be *open paths*, ending with the delivery of a (last) customer, instead of tours, as in the classical TSP. This causes no

loss of generality in our formulation, as the open and closed route problems are reducible to one another in linear time.[1]

Second, *special priority constraints will be imposed.* Their description goes as follows.

In any particular vehicle route we can identify the sequence of pickups and the sequence of deliveries, sequences which, in general, will be *merged* with one another. The position (1st, 2nd, etc.) that a particular customer holds in the sequence of pickups will not in general be the same as his First-Come-First-Served (FCFS) position in the initial list of customer requests, the difference of these two positions constituting the *pickup position shift* of that customer. For instance, a customer holding the 5th position in the initial list and being picked up 3rd has a pickup position shift of +2 while this shift becomes −1 if the customer is picked up 6th. A *delivery position shift* can be similarly defined as the difference between the position of a customer in the sequence of deliveries and the FCFS position of that customer in the initial list of requests.

Under our special priority constraints and for every customer, *we shall not allow either one of the two position shifts defined above to be greater in magnitude than a prescribed nonnegative integer MPS.* MPS stands for Maximum Position Shift and is an input to our problem. If MPS = 0, each customer should be picked up and delivered according to his FCFS position in the initial list. On the other hand, for N customers, MPS $\geq N - 1$ means that our priority constraints are superfluous and customers may be picked up or delivered in any order we wish.

The purpose of these constraints, although somehow vague for the "static" case, will be seen to be very important for the "dynamic" case, namely in preventing the possibility that the service of any particular customer will be indefinitely deferred by the algorithm. For the moment, however, these constraints can be considered as "service guarantees" for all the customers. For example, the customer who holds the ith position in the initial list is guaranteed that his position in the sequence of pickups and in the sequence of deliveries will be between i − MPS and i + MPS.

Solution procedures for the dial-a-ride problem depend in general on the particular version examined. Due to the fact that the simplest version of the problem cannot be easier to solve than the classical Traveling Salesman Problem,* the bulk of research efforts for solving the dial-a-ride problem has been directed toward heuristic procedures, such as in [2–5].

In this paper, we shall present an *exact* approach for the solution of

* The simplest version of the problem is a single vehicle, many-to-one, immediate request, "static" problem, where the objective is to minimize total distance traveled and no capacity or other constraints are imposed. This version is, in fact, a classical TSP.

the version of the dial-a-ride problem described above. The approach is based on Dynamic Programming and derives from the classical D.P. solution to the TSP, first suggested by HELD AND KARP.[6] It is, however, more sophisticated in that it examines the constraints and the special structure of the problem in a fashion particularly suited to the recursive nature of the procedure. For example, the algorithm is structured in such a way, so that the route legitimacy requirement (the origin of each customer precedes his destination on the route) is taken care of automatically. The vehicle capacity and MPS and constraints are also incorporated into the recursive procedure. In addition, the algorithm can examine several different (i.e. nonreducible to one another) specialized objectives with equal ease.

The exact way in which the computational effort of our algorithm grows is a complicated function of the values of N (number of customers), MPS and C (vehicle capacity). Storage requirement and running time are bounded by exponential functions of N, but the algorithm's worst case performance is asymptotically better than that of the classical D.P. algorithm,[6] when the latter is applied to a TSP on a graph of the same size.

"Static" Case Formulation

In our problem formulation, we assign numbers to customers according to the order in which they have telephoned the agency for service and we arrange them in the list in the same order. Thus, customer i holds the ith position in the list. Let N be the total number of customers. Also let "$+i$" be the pickup point (origin) and "$-i$" the delivery point (destination) of customer i ($i = 1, 2, \cdots, N$) and let A be the starting point of the vehicle (location of vehicle at $t = 0$). We shall assume that the time to go from any one of the $2N + 1$ points of our problem, i, directly to any other point j is a known and fixed quantity $t(i, j)$.

Our goal is to find a vehicle route starting from A and ending at one of the delivery points so that the following hold:

1) The quantity

$$w_1 \sum_{j=1}^{2N} T_j + w_2 \cdot \sum_{i=1}^{N} [\alpha \cdot WT_i + (2 - \alpha) RT_i] \qquad (1)$$

is minimized. Our notation is the following:

w_1, w_2: Given "weights"
α: Customers' time preference constant ($0 \leq \alpha \leq 2$)
T_j: Duration of the jth leg of the route
WT_i: Waiting time of customer i, from $t = 0$ until his time of pickup
RT_i: Riding time of customer i, from his time of pickup until his time of delivery.

2) Vehicle routes should be legitimate, namely each customer should be picked up before he is delivered.

3) The vehicle has a certain capacity of C customers that cannot be exceeded.

4) Maximum Position Shift constraints must be satisfied. Thus if p_i is the position customer i holds in the sequence of pickups and d_i the position he holds in the sequence of deliveries, then for a given non-negative integer MPS, we should have:

$$|i - p_i| \leq \text{MPS} \quad \text{for} \quad i = 1, 2, \cdots, N \tag{2}$$

$$|i - d_i| \leq \text{MPS} \quad \text{for} \quad i = 1, 2, \cdots, N \tag{3}$$

Some clarifying observations follow.

In (1), we can recognize $\sum_{j=1}^{2N} T_j$ as the time to service all customers. In fact, a special case ($w_1 = 1$, $w_2 = 0$) of our generalized objective calls for the minimization of the above quantity, an objective identical to that of the classical TSP.

The quantity $\sum_{i=1}^{N} [\alpha \cdot WT_i + (2 - \alpha) \cdot RT_i]$ is the assumed form of the total degree of "dissatisfaction" experienced by the customers until their delivery. Note that the waiting and riding times are weighted unequally in general (except if $\alpha = 1$, when customers are indifferent between the two). Note also that the restriction $0 \leq \alpha \leq 2$ causes no loss of generality, since the ratio $\alpha/(2 - \alpha)$ can take any value between zero and infinity and thus, the *relative* values of the customers' waiting and riding times can have any specified ratio. A special case ($w_1 = 0$, $w_2 = 1$) of our generalized objective calls for the minimization of the total (or average) customer "dissatisfaction."

One should note that our generalized objective function does not include terms reflecting the customers' waiting times from the instant of call until the instant the vehicle becomes available ($t = 0$). This apparent omission has no effect on our formulation since these terms, one for each customer, would appear as a constant (sunk cost) in our objective and would not affect our subsequent decisions. The linearity of our objective function is crucial in the validity of the above argument.

A clarifying observation for (2) and (3) is that the particular form of these is not binding in our formulation. For example, the values of MPS in (2), (3) need not be the same. Neither of the inequalities need to be two sided or symmetric. Finally, each customer i may have his own upper and lower bounds for his MPS_i.

"Static" Case Solution

Our Dynamic Programming approach goes as follows. The state vector is $(L, k_1, \cdots, k_N)$ with the following conventions:

a) L : Point the vehicle is currently visiting. We assume that
$L = 0$: Vehicle is at starting point A
$1 \leq L \leq N$: Vehicle is at point $+L$, i.e. picks up customer L
$N + 1 \leq L \leq 2N$: Vehicle is at point $-(L - N)$, i.e. delivers customer $L - N$.

We also assume that a customer boards on or gets off the vehicle immediately upon arrival at point L.

b) k_j: "status" of customer j ($j = 1, \cdots, N$). We assume that:

$k_j = 3$: Customer j has not been picked up so far
$k_j = 2$: Customer j is in the vehicle
$k_j = 1$: Customer j has been delivered.

Feasibility considerations. Not all state combinations will be feasible. The following is a list of necessary conditions for the feasibility of a particular state $(L, k_1, \cdots, k_N)$:

1) The state should be *consistent*, namely,

i) If $L = 0$, then $k_j = 3$ for all $j = 1, \cdots, N$ (4)

ii) If $1 \leq L \leq N$, then $k_L = 2$ (5)

iii) If $N + 1 \leq L \leq 2N$, then $k_{L-N} = 1$. (6)

The above follow directly from the definition of $(L, k_1, \cdots, k_N)$.

2) The state should satisfy the vehicle capacity constraints. Letting x_2 be the number of k's in the state vector which are equal to 2, these constraints can be formulated as follows:

i) If $1 \leq L \leq N$, then $x_2 \leq C$ (7)

ii) If $N + 1 \leq L \leq 2N$, then $x_2 \leq C - 1$ (8)

We can see the validity of the above through the observation that x_2 is the number of customers inside the vehicle. We can also observe that we actually need impose only one of (7) and (8). For example if L is a delivery stop, (8) will always be satisfied if (7) is satisfied for the state corresponding to the first pickup stop preceding this delivery stop and vice versa.

3) The state should satisfy the MPS constraints. In addition to the number x_2 defined above, let x_1 be the number of k's in the state vector which are equal to 1. Then the MPS constraints can be formulated as follows:

i) If $1 \leq L \leq N$, then $|L - (x_1 + x_2)| \leq \text{MPS}$ (9)

ii) If $N+1 \leq L \leq 2N$, then $|(L-N)-x_1| \leq \text{MPS}$ (10)

We can see the validity of (9) by the observation that $x_1 + x_2$ is the number of customers either delivered or currently in the vehicle, hence the number of customers picked up so far. But this number is nothing more than the position customer L holds in the sequence of pickups, hence $L - (x_1 + x_2)$ is his corresponding pickup position shift and thus, (9) is equivalent to (2). A similar argument can explain that (10) is equivalent to (3).

Conditions (4) through (10) are necessary *but not sufficient* for the feasibility of a particular state $(L, k_1, \cdots, k_N)$. To complete our feasibility test we must look at the feasibility of its "next" states. If none of them is feasible, then our original state is infeasible. If on the other hand, there exists at least one that is feasible, then our original state is feasible as well. The fact that for determining the feasibility of any particular state we have to know *a priori* about the feasibility of its "next" states constitutes the *recursive nature of feasibility*, a property which is particularly well-suited to the nature of Dynamic Programming.

The set of states $(L', k_1', \cdots, k_N')$ which are "next" to a given consistent state $(L, k_1, \cdots, k_N)$ is such that L' belongs to the union S of the following two sets:

a) The set of "next" pickup stops

$$S_3 = \{i: 1 \leq i \leq N \text{ with } k_i = 3\} \tag{11}$$

b) The set of "next" delivery stops

$$S_2 = \{i: N+1 \leq i \leq 2N \text{ with } k_{i-N} = 2\}. \tag{12}$$

Also the new k-vector (k') is given by

$$k_j' = \begin{cases} k_j - 1 & \text{if } j = L \text{ or } j = L - N \\ k_j & \text{otherwise} \end{cases} \tag{13}$$

for all $j = 1, \cdots, N$.

The case where both S_2 and S_3 are empty corresponds to the case where $k_j = 1$ for all $j = 1, \cdots, N$, meaning that all customers have been delivered. Any such state is called a *terminal* state and obviously has no "next" states. For terminal states, conditions (4) through (10) *are* sufficient for feasibility.

Summarizing, the following is a sequence of "Screens" to determine whether a particular state $(L, k_1, \cdots, k_N)$ is feasible or not.

Screen 1 (Consistency): If any one of (4), (5), (6) (check the one that applies) does not hold, then STOP: $(L, k_1, \cdots, k_N)$ is infeasible. Other-

wise, if $L = 0$, STOP: $(L, k_1, \cdots, k_N)$ is feasible. Otherwise, go to Screen 2.

Screen 2 (Vehicle Capacity): If any one of (7), (8) (check the one that applies) does not hold, then STOP: $(L, k_1, \cdots, k_N)$ is infeasible. Otherwise go to Screen 3.

Screen 3 (Maximum Position Shift): If any one of (9), (10) (check the one that applies) does not hold, then STOP: $(L, k_1, \cdots, k_N)$ is infeasible. Otherwise go to Screen 4.

Screen 4 ("Next" states): Form S, the union of the sets defined in (11) and (12). If S is empty, then STOP: $(L, k_1, \cdots, k_N)$ is feasible. Otherwise do the following: Examine one by one the "next" states $(L', k_1', \cdots, k_N')$ with $L' \in S$ and k_j' obeying (13). At the first feasible state encountered, STOP: $(L, k_1, \cdots, k_N)$ is feasible. If no feasible state is encountered, STOP: $(L, k_1, \cdots, k_N)$ is infeasible.

A final observation concerning feasibility is that the definition of a "next" state in (11) through (13) *automatically* takes care of the route legitimacy constraints. Thus, a customer's pickup (change of k from 3 to 2 for that customer) will always precede his delivery (change of k from 2 to 1).

Optimality considerations. Let $V(L, k_1, \cdots, k_N)$ be the optimal value (measured in terms of our specific objective function) of all subsequent decisions from $(L, k_1, \cdots, k_N)$ till the end of the route (i.e. till a terminal state is reached). *V is defined only if the corresponding state is feasible.* It can be seen that V must obey the following optimality recursion:

$$V(L, k_1, \cdots, k_N) = \begin{cases} \operatorname{Min}_{L' \in S}[t(L, L') \cdot M + V(L', k_1', \cdots, k_N')] & \text{if } S \neq \emptyset \\ 0 & \text{if } S = \emptyset. \end{cases} \tag{14}$$

In (14), S is the union of S_3 and S_2 defined in (11) and (12) and k_j' is defined by (13). M is the factor of proportionality by which the time $t(L, L')$ needed to go from L to L' should be multiplied.

To estimate M, we consider all the marginal contributions to the value of our objective due to the fact that the vehicle travels from L to L'. From (1) we can see that these can be divided into two categories:

a) Those reflected into the term $w_1 \cdot \sum_{j=1}^{2N} T_j$. The corresponding marginal contribution is simply $w_1 \cdot t(L, L')$.
b) Those reflected into the term $w_2 \cdot \sum_{i=1}^{N} [\alpha WT_i + (2 - \alpha) \cdot RT_i]$.

Letting x_3 and x_2 be the cardinalities of the sets S_3 and S_2 defined in (11) and (12), we can see that when the vehicle travels from L to L', the total waiting time is simply $x_3 \cdot t(L, L')$, and the total riding time is $x_2 \cdot t(L, L')$.

Thus, the marginal contribution to this term is $w_2 \cdot [\alpha \cdot x_3 + (2 - \alpha) \cdot x_2] \cdot t(L, L')$.

We can see therefore that:

$$M = w_1 + w_2[\alpha \cdot x_3 + (2 - \alpha) \cdot x_2]. \tag{15}$$

All the parameters shown above are either known *a priori*, or readily obtainable from the state vector.

Structure and Complexity of the "Static" Case Algorithm

The structure of our Dynamic Programming algorithm is as follows:

1) There is a "Screening" part where the sequence of "screens" 1 through 4, described in the previous paragraph is performed, using *backward recursion*. We *tabulate* feasibility information into the logical array FEASBL(L, k_1, $\cdots$, k_N) with values "true" or "false" depending on whether the corresponding state is feasible or not.

2) There is an "Optimization" part, which consists of the application of (14) using (11), (12), (13), and (15). Backward recursion is used here in the same way as in "Screening." (14) is applied *only* for those states already characterized as feasible by "Screening." The element L' of S which minimizes $M \cdot t(L, L') + V(L', k_1', \cdots, k_N')$ is stored into the array NEXT (L, k_1, $\cdots$, k_N).

3) Finally, we have the "Identification" part which uses the information of the array NEXT created previously. We start from the state (0, 3, 3, $\cdots$, 3) and then *move forward* to identify our optimal route. This part takes $2N$ steps.

It is thus necessary to reserve $(2N + 1)3^N$ memory locations for each of the arrays FEASBL, V and NEXT. If no space is reserved for inconsistent states, we can bring this figure down to $2N \cdot 3^{N-1} + 1$ locations. The algorithm's running time is bounded by $(2N + 1)^2 3^N$, for in both "Screening" and "Optimization" we have to look at a maximum of $(2N + 1)$ "next" states for each particular state. Comparing the storage requirement of $(2N + 1)3^N$ with the corresponding figure $(2N + 1)2^{(2N+1)}$ of the classical D.P. approach to a TSP of $(2N + 1)$ nodes, we see that the ratio $(2N + 1)3^N/(2N + 1)2^{(2N+1)} = 1/2\ (3/4)^N$ goes to zero as $N \to \infty$. The reason for this difference in performance is that our algorithm takes advantage of the fact that not all TSP tours are feasible for the dial-a-ride problem because of the route legitimacy constraints.

The vehicle capacity C and the Maximum Position shift MPS also play roles in determining the algorithm's computation effort. Qualitatively we should expect the latter to be an increasing function of C and MPS. The corresponding quantitative analysis is somewhat complicated. It is presented in [7], along with additional suggestions on algorithm refinements.

Computational Experience

The exponential growth of the computational effort of the algorithm can be best illustrated in Table I. Table I represents typical CPU times incurred when a computer program implementing this algorithm was run. The VAX/VMS timeshared interactive computer system of the MIT Joint Computer Facility was used for that purpose. It should be noted here that possible algorithm refinements suggested in the previous section and in [7] *were not* implemented in the computer program in question. For instance, storage space was reserved for $(2N + 1)3^N$ rather than $2N3^{N-1} + 1$ memory locations for each of the arrays V, NEXT and FEASBL.

All the figures of Table I refer to the problem's unconstrained (worst) case, namely to MPS $\geq N - 1$ and $C \geq N$. Variations of CPU times when MPS $< N - 1$ and/or $C < N$ were also examined. A typical example is represented in Table II, where CPU times for $N = 7$ and various values of MPS and C are displayed. (Note that for MPS ≥ 6 or for $C \geq 7$ the corresponding constraints are essentially superfluous.) Referring to Table II, one can verify the earlier anticipated behavior, namely that CPU time increases as MPS or C increase, everything else being held constant.

In general, the figures in Tables I and II tend to confirm the increasingly higher price one has to pay for *exactly* solving many-to-many dial-a-ride problems of increased size. The tractable problem size will depend, among other things, on the tightness of the constraints, on possible refinements in the computer code, on the computer facility used (dedicated or timeshared) and on how quickly solutions are needed. In that last respect, implementing the algorithm in a "dynamic" environment, where calculations should be performed in a reasonable amount of time, will certainly

TABLE I

Typical CPU Times (Seconds) Versus N (Number of Customers)[a]

N	2	3	4	5	6	7	8	9
CPU	0.6	0.7	1.1	2.7	9.6	46.8	149.7	591.4

[a] The unconstrained (worst) case is displayed (MPS $\geq N - 1$, $C \geq N$).

TABLE II

Typical CPU Times (Seconds) Versus MPS and C (N = 7)

C	MPS		
	0	3	≥6
1	12.1	13.7	15.3
4	13.0	25.5	36.9
≥7	13.1	29.6	46.8

be more difficult than doing so in a "static" setting. On the other hand, as it will be seen in Part II of this paper, the "dynamic" nature of the real-world scheduling/dispatching process is such that it may not be absolutely essential to obtain exact solutions *over a long scheduling horizon* (i.e., simultaneously consider *many* customers per vehicle), particularly in view of the fact that the arrival of new customers will necessitate the continual update of each vehicle's schedule. In any event, customers exceeding the algorithm's limit can be arranged in a "standby" buffer and become "active" as soon as possible, or be assigned to other vehicles.

An Example of the "Static" Case

We present now a series of cases regarding the Euclidean graph of 15 nodes (7 customers) of Figure 1. The cases vary as far as the objective, the capacity C and the value of MPS are concerned. In all cases the input value of α is 1.00 (customers are indifferent between waiting for and riding in the vehicle). The starting point A can be thought of as a depot.

Cases 1 and 2 (Figs. 2 and 3) represent the unconstrained case, where no MPS and capacity constraints exist. Case 1 has $w_1 = 1$, $w_2 = 0$ and

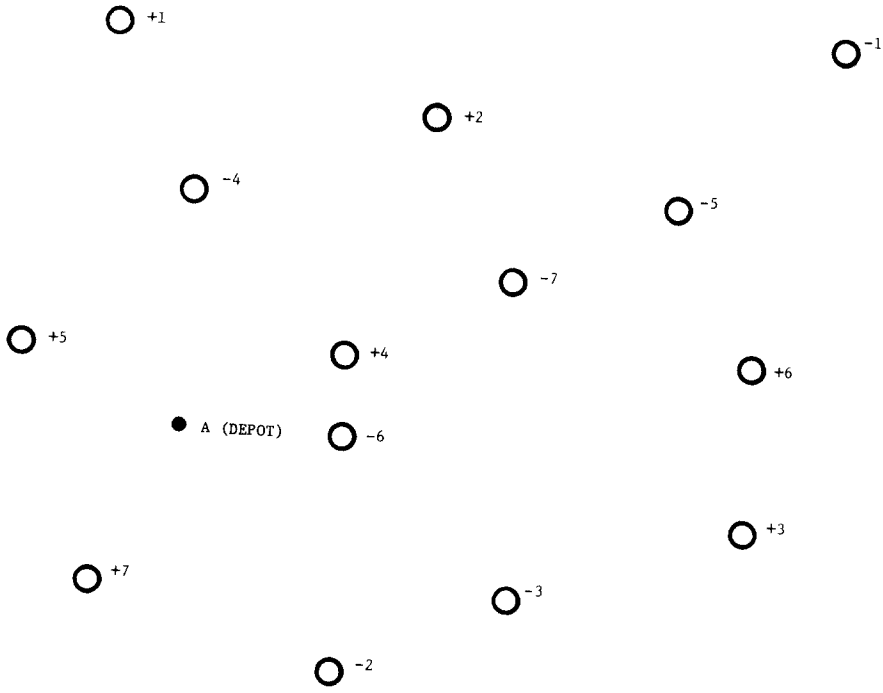

Fig. 1. Origins (+), destinations (−) and starting point (A) of the example.

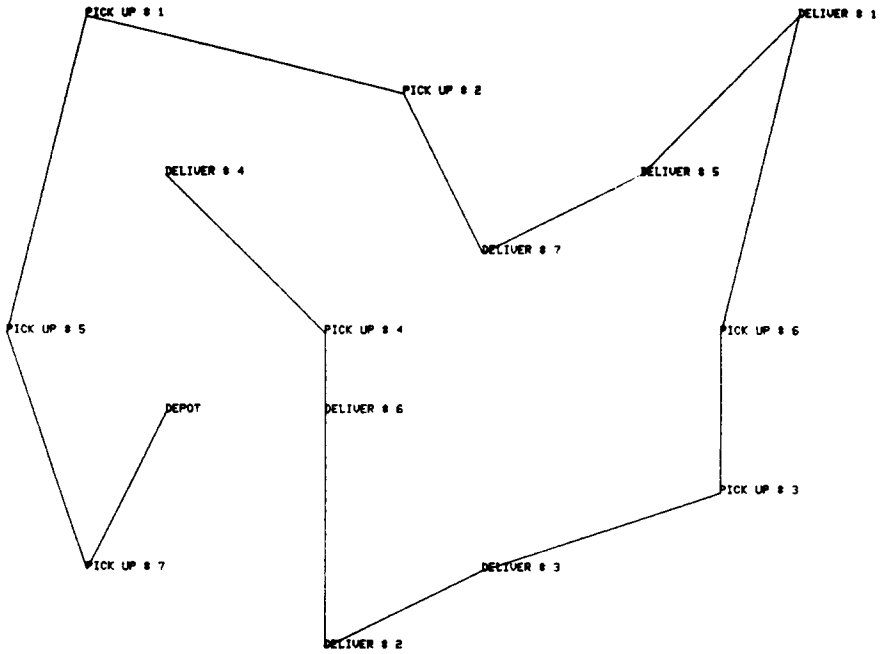

Fig. 2. Case 1: $w_1 = 1$, $w_2 = 0$, no MPS or capacity constraints.

Case 2 has $w_1 = 0$, $w_2 = 1$. The difference in optimal routes can be observed. For instance, the route for Case 1 looks very much like a typical good route for a TSP, while this is not true for the route of Case 2.

We subsequently impose an MPS constraint of 2 and a capacity constraint of 3 for Case 3 (Fig. 4). Here $w_1 = 1$, $w_2 = 0$. We observe how these constraints change the optimal route from that of Case 1.

Our final case is Case 4 (Fig. 5) with $w_1 = 0$, $w_2 = 1$. Here MPS = 0, but no capacity constraints exist. Despite MPS = 0, there is still one degree of freedom for optimization left: How one should merge the two FCFS sequences of pick-ups and deliveries.

From Figures 4 and 5 we can see that for "random" locations of customers, the shape of the route becomes more and more complicated as the constraints become more tight.

II. THE "DYNAMIC" CASE

THE MAIN ASSUMPTION in Part I has been that at a given point in time ($t = 0$) the list of customer requests is closed and no new requests are further considered, until all the customers that this list contains at $t = 0$ are serviced. In Part II we investigate what happens if we drop the above

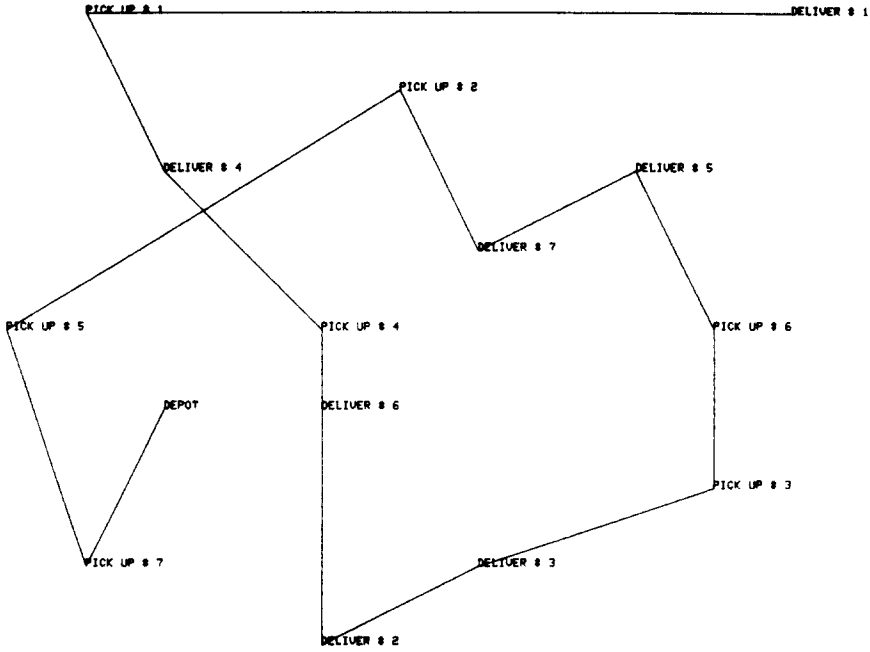

Fig. 3. Case 2: $w_1 = 0$, $w_2 = 1$, no MPS or capacity constraints.

assumption. The corresponding version constitutes the "dynamic" case of the problem.

In the "dynamic" case, each new customer request is automatically eligible for consideration at the time it appears. Each time such a request (immediate) appears, the input to our problem is modified and one should expect a corresponding modification to the problem's solution as well. This modification, for obvious reasons, cannot affect the portion of the route executed prior to the time the new request is made known, but will affect the remaining portion of the route. Since such new requests may be arriving for an indefinite period of time, our "dynamic" case is by nature *open-ended*, and the corresponding solution procedure must be active as long as these new inputs keep entering our problem.

We can, at this point only qualitatively, suggest several principal features that such "dynamic" procedure should possess.

1) At any point in time and for the current set of the problem inputs (current position of the vehicle and the current status of all customers) the procedure should be able to produce upon request what we shall call the *tentative optimal route*, and do so in a reasonably short period of time. The tentative optimal route is the optimal solution, according to a prescribed objective, corresponding to the current set of inputs and only

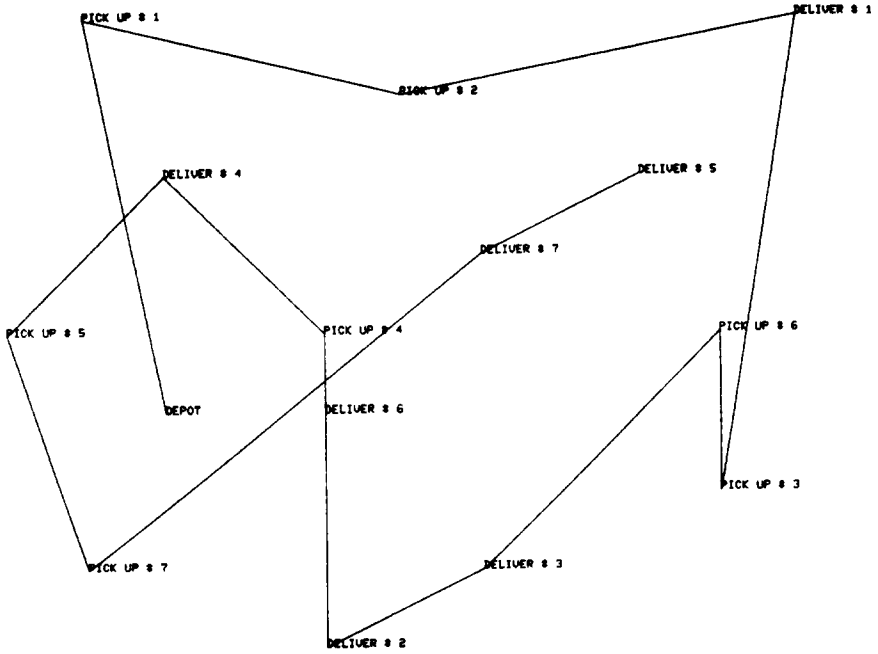

Fig. 4. Case 3: $w_1 = 0$, $w_2 = 1$, MPS = 2, $C = 3$.

to those. This route will indeed be totally optimal if and only if no customer request appears during its execution.

2) Each time a new customer request appears, the procedure should be able to incorporate this new information as part of the problem's input and somehow solve the new problem again, also in a reasonable amount of time. This updating process will produce a new tentative optimal route each time a new customer request is made.

3) The procedure should be flexible enough so that it is to some extent *able to reconsider its past decisions.* Any procedure which bases its updates on *local* reoptimizations of prior solutions will be suboptimal. For instance, a *local* reoptimization would occur if, upon appearance of a new customer request, one decides to examine only the possible *insertions* of the pickup and delivery stop of this customer into the *current* sequence of other pickup and delivery stops and therefore *a priori exclude* the possibility that the appearance of the new input may render the current sequencing of already existing customers no longer optimal. Algorithms using the above "insertion" idea have been developed in [2] and [4] and are, in that respect, suboptimal.

4) Despite the above, the reoptimization capability of a "dynamic" procedure is a two-edged sword: A *highly flexible* procedure which is

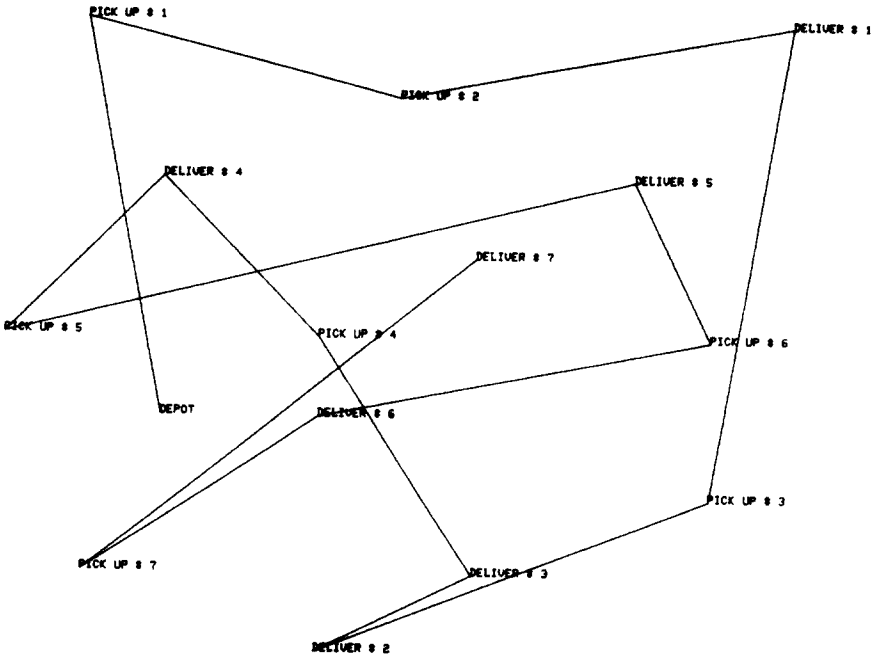

Fig. 5. Case 4: $w_1 = 0$, $w_2 = 1$, MPS = 0, no capacity constraints.

solely based on *global* reoptimizations may present other complications. In addition to representing a difficult combinatorial problem, such a procedure may conceivably continually defer the service of any particular customer for an indefinite period of time. *Indefinite deferment* of a customer's request can happen whenever a customer is being continually assigned to the last position in the pickup or delivery sequence because of his unfavorable geographical characteristics with respect to other customers. The service (pickup or delivery) of such a customer will be consistently given a low priority by the procedure, as long as other, more favorable inputs keep entering our problem.

The issue of indefinite deferment is *fundamental* not only for the "dynamic" case of the dial-a-ride problem, but for any other "dynamic" scheduling problem. Any viable "dynamic" scheduling algorithm must incorporate a mechanism to prevent, or at least discourage this possibility. In [2] and [4], this possibility was partially kept under control, yet not unambiguously eliminated, both through the assumed quadratic form of the objective function and through use of heuristic criteria.

A simple example on how a "dynamic" procedure might operate is depicted in Figure 6. The procedure is given a set of initial inputs (A: initial position of vehicle, "+": origins, "−": destinations) and subse-

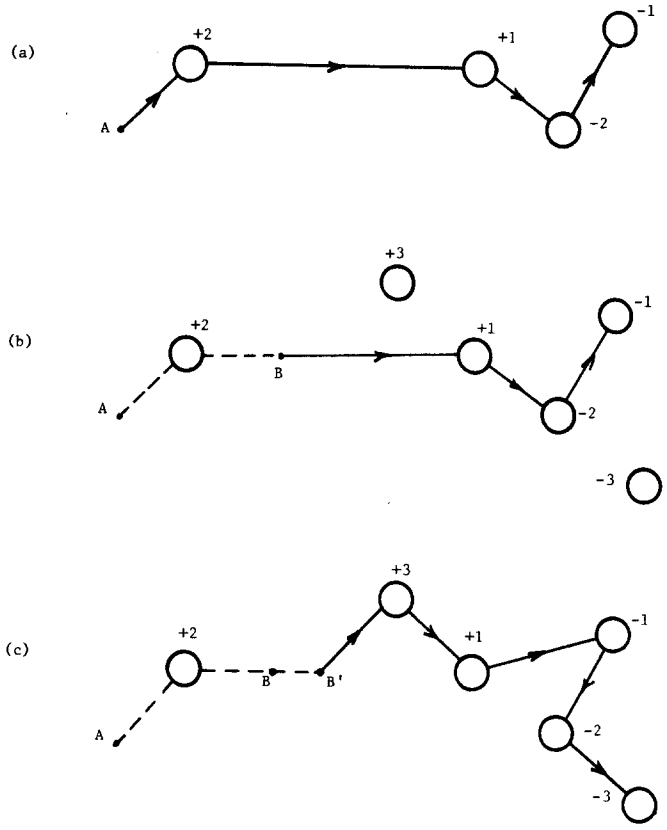

Fig. 6. New request (customer 3) appears when vehicle is at B. New tentative optimal route is produced when vehicle is at B'.

quently produces the tentative optimal route shown in Figure 6a. At the time the vehicle is at B, a new request by customer 3 is made known (Fig. 6b). At this point in time the procedure revises the nonexecuted portion of the route and produces the new tentative optimal route shown in Figure 6c. Note that the new route does not have B as origin, but a point B', slightly ahead of B. This is due to the fact that it will in general take some time for the procedure to process the new input and reoptimize and this time is reflected by the segment BB'. The process is repeated indefinitely as long as new inputs keep entering our problem.

The algorithm we shall present for the "dynamic" dial-a-ride problem derives from the D.P. algorithm presented in Part I for the corresponding "static" case. Several modifications of the "static" algorithm are necessary to accomplish this. Indefinite deferment is eliminated by the MPS priority constraints already introduced.

"Dynamic" Case Formulation and Solution

A basic assumption, which actually reflects our optimization philosophy for the "dynamic" case is that each time we update the problem solution, we shall only be considering as the problem's inputs all the information we have on *specific and known* customer requests. By contrast, we *shall not* be concerned with any, possibly available, probabilistic information describing the spatial or time distribution of other *future* requests. In this spirit, *no customer request will be either anticipated or in any other way taken into account before it actually occurs.* A consequence of the above assumption is that *at each update we shall be optimizing only over existing and active customer requests.* Obviously, customers delivered prior to a specific update are no longer part of our input at that update.

Several observations concerning our formulation are now made:

First, it can be seen that there is no need to perform an update (i.e. reoptimize) except when a new customer request occurs. In fact, letting $(A \equiv P_0, P_1, \cdots, P_n)$ be a particular sequence of stops having A as origin and which constitutes a tentative optimal route, then it can be shown that for any index i $(0 \leq i \leq n)$, the sequence $(B, P_{i+1}, P_{i+2}, \cdots, P_n)$ with its origin B being *any intermediate point on the arc* (P_i, P_{i+1}), is also a tentative optimal route for the set of stops $(P_{i+1}, P_{i+2}, \cdots, P_n)$, *provided* no new customer request appears while the vehicle travels between A and B. The above property is easily shown to be valid by contradiction (see for example [7]).

Second, at any particular update, the objective function can be defined as follows: Suppose that at the instant of our update, S_2 is the set of customers on board the vehicle and S_3 the set of customers waiting to be picked up. It is not difficult to see that these are the same S_2 and S_3 defined by (11) and (12) in Part I. Letting now $x_2 = |S_2|$ and $x_3 = |S_3|$, it is clear that the route to service all of the above customers (and only those) will consist of $x_2 + 2x_3$ legs, each of duration T_j $(j = 1, \cdots, x_2 + 2x_3)$. It is also clear that at the instant this update is performed there is no reason to worry about the portion of the route which has already been executed because no subsequent decisions on our part can influence what has happened in the past. Thus, all "costs," i.e. changes in the value of the objective function that have incurred in the past are "sunk costs," and since these costs are additive, they can be neglected from further consideration. The last argument allows us to reset our time origin $(t = 0)$ at the instant of the update and consider only what our decisions can influence thereafter. In that respect, a customer i belonging to the set S_2 will only experience an additional "dissatisfaction" equal to $(2 - \alpha) \cdot RT_i$, with the riding time RT_i being measured from the new time origin $(t = 0)$ until the delivery of the customer. By contrast, a customer k belonging

to the set S_3 will experience an additional "dissatisfaction" equal to $\alpha \cdot WT_k + (2 - \alpha) \cdot RT_k$, with the waiting time WT_k being measured from the new time origin ($t = 0$) until pickup and the riding time RT_k from pickup until delivery. Thus, we can write our objective function for a specific update of the "dynamic" case as follows:

$$\text{Minimize } w_1 \sum_{j=1}^{x_2+2x_3} T_j + w_2\left(\sum_{i \in S_3} \alpha WT_i + \sum_{i \in S_2 US_3} (2 - \alpha)\, RT_i\right). \quad (16)$$

It is easy to see that we can reduce (16) to a formulation which is *exactly the same* as the "static" case. We can do this by *resetting* the origins of all customers belonging to S_2 to coincide with the location of the vehicle at the instant of the update. In this way, we assume that the vehicle is empty at the instant of the update. However, the algorithm will immediately execute a sequence of x_2 "dummy" pickup stops, separated from one another by a zero time interval, and thus adding these customers to the vehicle at once. Renumbering our customers from 1 to $N = x_2 + x_3$, our objective function reduces exactly to the "static" form examined in Part I, i.e.,

$$\text{Minimize } w_1 \sum_{j=1}^{2N} T_j + w_2 \sum_{i=1}^{N} (\alpha WT_i + (2 - \alpha) RT_i). \quad (17)$$

The solution of this problem will obviously have $T_j = 0$ for $j = 1, \cdots, x_2$ and $WT_i = 0$ for $i = 1, \cdots, x_2$.

As far as constraints are concerned, we shall maintain the vehicle capacity and MPS constraints introduced already in Part I. The difference in the MPS constraints between the "static" and "dynamic" cases is that in the "dynamic" case we also have a mechanism to *keep track* of the various position shifts from one update to another, as these will in general change. The mechanism for doing this is straightforward to develop and is particularly helpful for the reformulation of relations (9) and (10) for the "dynamic" case. Details can be found in [7]. The capacity constraints are formulated in exactly the same way as in Part I (see relations (7), (8)).

Finally, some words are necessary concerning the determination of the location of the origin B' of the new tentative route (see Fig. 6c): Recall that due to the noninstantaneous processing of the new input, an interval of time will elapse between the instant t_B of the request (when the vehicle is at B) and the instant t_B' when any decision on our part can be actually implemented. This time interval $(t_B' - t_B)$ is reflected in the segment BB'. So actually, at the instant t_B when the vehicle is at B, an estimate of the time needed to "run" the algorithm, produces the point B' and it is for this new origin that the subsequent update is done. For long

distances, small number of customers or fast computers one may assume that B' practically coincides with B.

An Example of the "Dynamic" Case

We shall now present an example to which we shall apply our "dynamic" algorithm. Without loss of generality we shall assume that we wish to minimize the total customer "dissatisfaction" [$w_1 = 0$, $w_2 = 1$ in (17)]. α is assumed equal to 1, MPS = 3, $C = 4$. The distance metric of the problem is assumed Euclidean and the vehicle is assumed to be traveling at a constant speed of 30 mph. Finally, we assume that each update the interval $(t_{B'} - t_B)$ to process the new inputs and produce the new solution is negligibly small.

At $t = 0$, the vehicle becomes available at point (1.000, 4.000) (Cartesian Coordinates in miles). At that time, 6 customers have *already* requested service. (See Table III and Figure 7.)

In addition, we shall assume that a chronological sequence of additional customer requests will occur in the future (see Table IV).

It is our assumption, of course, that we are not able to anticipate these requests ahead of time and act accordingly. These requests become part of our input only after the customer files a request at the indicated time.

Figure 8 represents the tentative optimal route for our first update at $t = 0$.

At $t = 20$ minutes, customer 7 appears. At that time, the vehicle is at point (4.418, 4.388), point marked by (*) in Figure 8.

Figure 9 shows the new tentative optimal route for this second update at $t = 20$ minutes. The portion of the route between $t = 0$ and $t = 20$ has already been executed and is shown by dotted lines.

At $t = 40$ minutes, customer 8 appears. At that time, the vehicle is at point (3.863, 8.000).

Figure 10 shows the new tentative optimal route for this third update at $t = 40$ minutes.

TABLE III

Initial Customer Requests (t = 0)

Customer	Pickup Point Coordinates (miles)		Delivery Point Coordinates (miles)	
	X	Y	X	Y
1	1	2	5	2
2	3	1	1	8
3	5	4	2	6
4	6	3	4	7
5	10	8	11	6
6	8	1	7	5

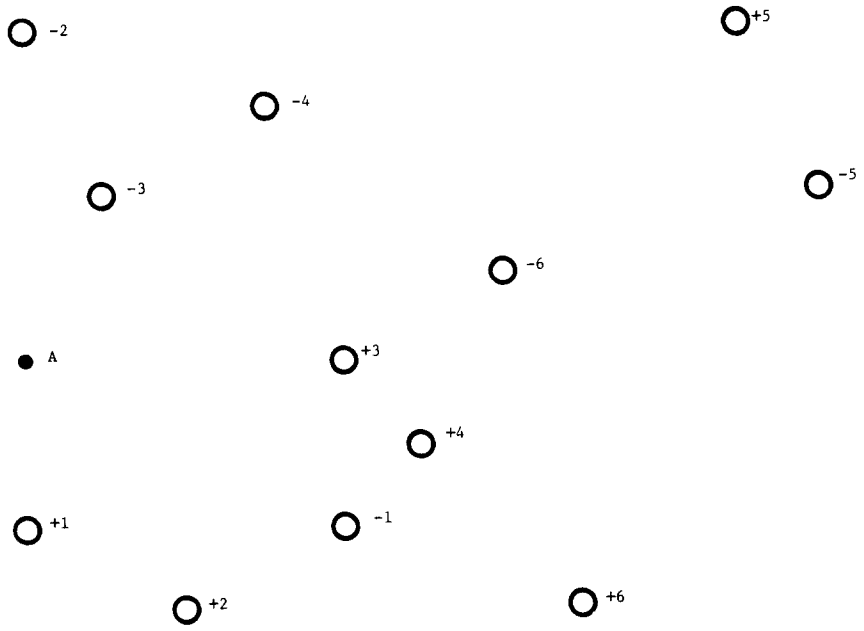

Fig. 7. Customer requests and vehicle location at $t = 0$.

TABLE IV

Chronological Sequence of Additional Customer Requests

Time of Request (minutes from $t = 0$)	Customer	Pickup Point Coordinates (miles)		Delivery Point Coordinates (miles)	
		X	Y	X	Y
20	7	4	6	11	3
40	8	7	7	9	4
55	9	4	1	5	3
80	10	5	4	7	4

Our fourth update occurs at $t = 55$ minutes, when customer 9 appears. At that time, the vehicle is at point (10.467, 7.066).

In *Figure 11* we show the new tentative optimal route for this fourth update at $t = 55$ minutes.

Our fifth (and final) update occurs at $t = 80$ minutes, when customer 10 appears. At that time, the vehicle is at point (5.090, 1.000).

In *Figure 12* we show the new tentative optimal route for this fifth update at $t = 80$ minutes.

An interesting observation we can make on Figure 12 is that the delivery of customer 6 *precedes* the delivery of customer 10, only because of the MPS = 3 constraint. In fact, if no such constraint existed, it would

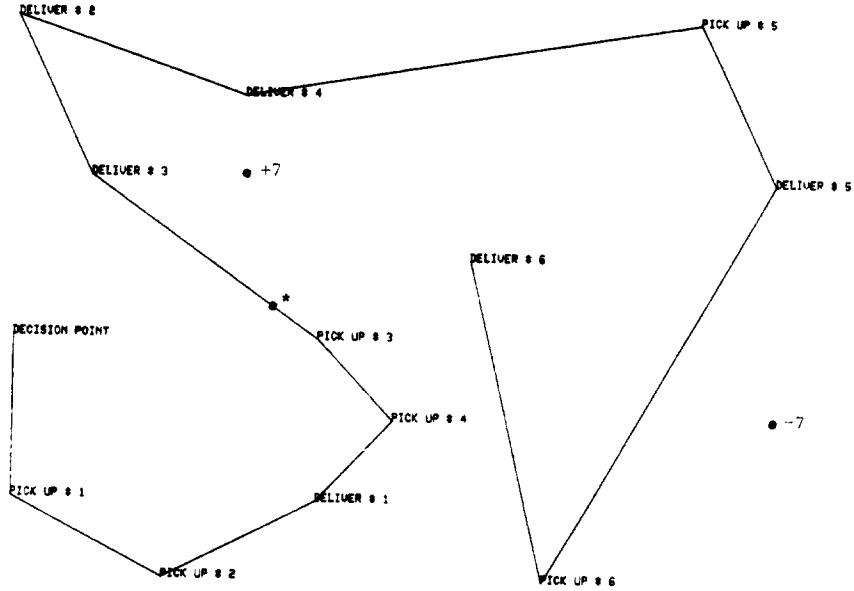

Fig. 8. $t = 0$, update #1. Customer 7 will appear when vehicle is at * ($t = 20$).

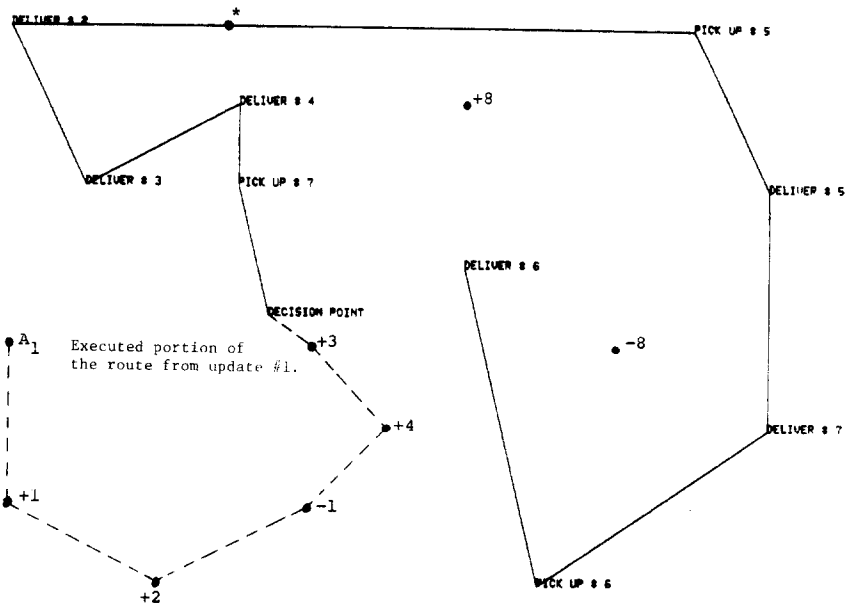

Fig. 9. $t = 20$, update #2. Customer 8 will appear when vehicle is at * ($t = 40$).

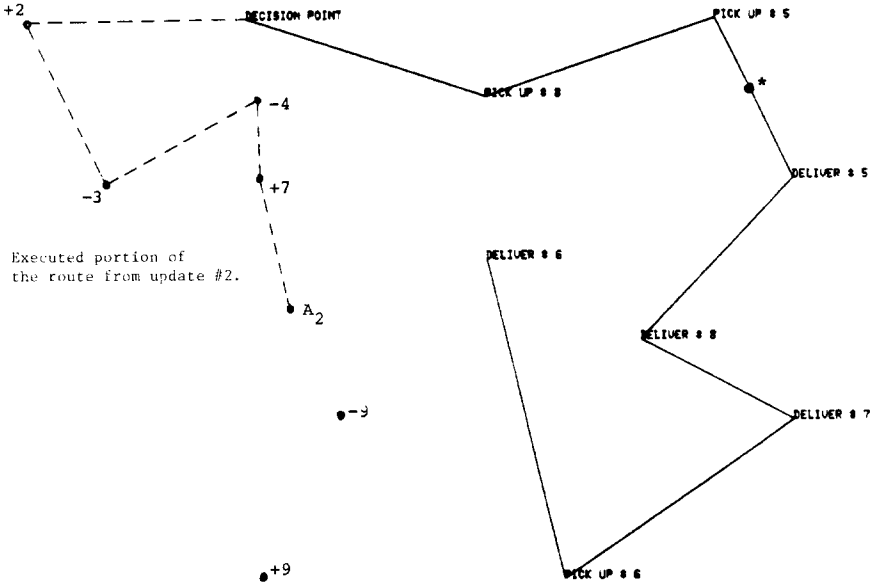

Fig. 10. $t = 40$, update #3. Customer 9 will appear when the vehicle is at * ($t = 55$).

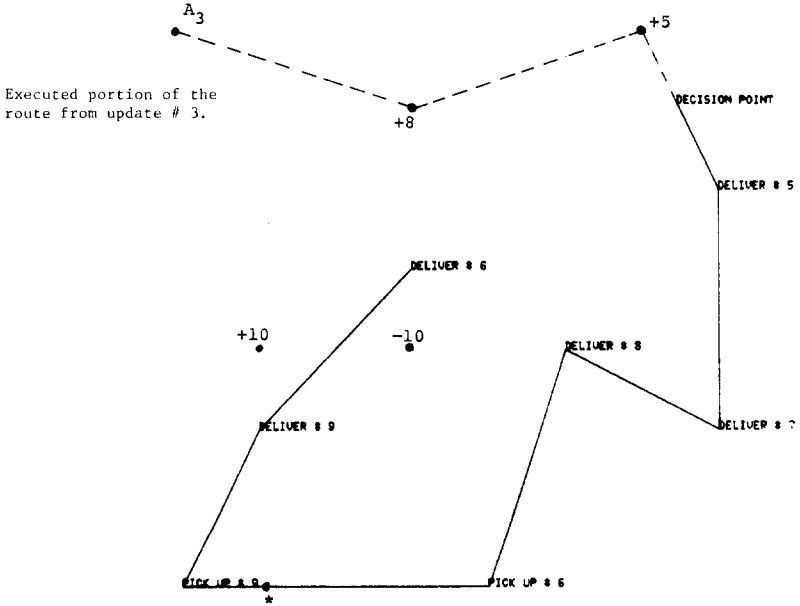

Fig. 11. $t = 55$, update #4. Customer 10 will appear when vehicle is at * ($t = 80$).

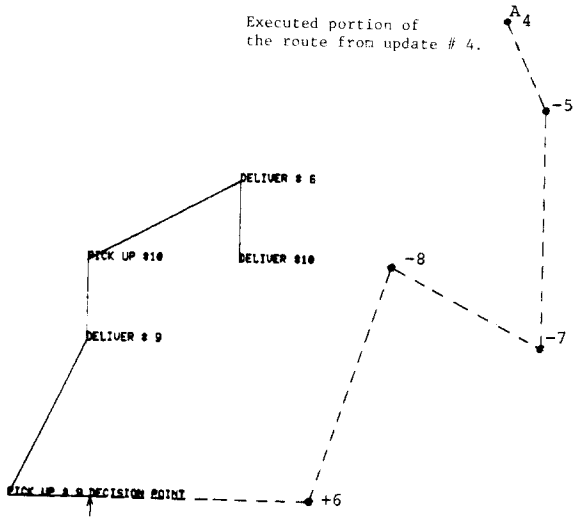

Fig. 12. $t = 80$, update #5. No more customers will appear. Note effect of MPS constraints on delivery of customer 6.

be better to deliver customer 10 first and then customer 6. One may have noticed already that in all our previous updates customer 6 was always assigned to be delivered *last*. In our fifth update, however, the MPS = 3 constraint prevents this deferment to continue and customer 6 is finally given priority over a more favorable customer.

In our example, no more customers appear after customer 10, so the tentative optimal route will be finally executed. If this were not the case, our updating process would be indefinitely pursued.

Conclusions and Suggestions for Possible Extensions

We have developed an exact optimization procedure, based on Dynamic Programming, to solve the "static" and "dynamic" versions of the single-vehicle, many-to-many, immediate request dial-a-ride problem. We have examined a generalized objective function and included vehicle capacity and MPS constraints. The algorithm's computational effort is (as expected) an exponential function of the size of the problem, yet asymptotically lower than that of the classical D.P. algorithm when the latter is applied to a TSP of the same size.

From an implementation point of view (particularly in a "dynamic" environment) there clearly exists a tradeoff between the fact that the algorithm is *exact* rather than heuristic (and hence there are potential benefits in overall system performance), and the fact that its tractable problem size is limited due to its exponential growth. Increasing this limit by algorithm refinements will certainly be a worthwhile effort.

A number of possible research directions that can be based on (or take advantage of) the work described in this paper are open. For instance, one might attempt to further refine the algorithms of this work by taking advantage of possible geographical proximities of customers. Also, one might want to test the performance of other *heuristic* routing algorithms for the same problem by comparison with that of the exact algorithm. Third, one might try to use the developed single-vehicle algorithm as a *subroutine* for the corresponding multivehicle problem. This has been attempted recently by the author with only partial success.[11] In that respect, the following issues were seen to be important:

1) The problem of how to *partition* a given set of customers among a given set of vehicles seems to be the crux of the difficulty of the multivehicle problem. The distinction of vehicle stops into origins and destinations tends to increase this difficulty even more. Thus, various algorithms developed for the scheduling of vehicles from a central depot to a number of delivery points do not fit the nature of the *many-to-many* multivehicle problem.[8–10]

2) In the "dynamic" case the appearance of a new customer will, in general, render some already made customer-to-vehicle assignments suboptimal. For instance, it may no longer be optimal to keep customer i scheduled to be picked up by vehicle j after the appearance of customer k. This reshuffling of customer-to-vehicle assignments in order to avoid suboptimal solutions will certainly add to the computational effort of a "dynamic" multivehicle algorithm.

3) Our priority rules should be redefined in a way compatible to the multivehicle configuration.

4) Concerning the problem's objectives, one can certainly see that the problem of servicing all customers as quickly as possible is a *minimax* problem. The problem of minimizing total customer dissatisfaction is certainly a different problem.

ACKNOWLEDGMENTS

THIS WORK WAS supported in part by a grant from the U.S. Department of Transportation, Urban Mass Transportation Administration. The author is particularly indebted to Nigel Wilson, Christos Papadimitriou, Jack Devanney and, most of all, Amedeo Odoni for their assistance.

REFERENCES

1. C. H. PAPADIMITRIOU, "The Euclidean Travelling Salesman Problem is NP-Complete," *Theoret. Comput. Sci.* **4**, 237–244 (1977).
2. N. H. M. WILSON AND H. WEISSBERG, "Advanced Dial-A-Ride Algorithms

Research Project: Final Report," Dept. of Civil Engineering, M.I.T. Report R76-20, 1976.

3. N. H. M. WILSON AND N. J. COLVIN, "Computer Control of the Rochester Dial-A-Ride System," Dept. of Civil Engineering, M.I.T. Report R77-31, 1977.
4. N. H. M. WILSON AND E. MILLER, "Advanced Dial-A-Ride Algorithms Research Project, Phase II: Interim Report," Dept. of Civil Engineering, M.I.T. Report R77-31, 1977.
5. D. M. STEIN, "Scheduling Dial-A-Ride Transportation Systems," *Trans. Sci.* **12,** 232–249 (1978).
6. M. HELD, AND R. M. KARP, "A Dynamic Programming Approach to Sequencing Problems," *J. SIAM* **10,** 196–210 (1962).
7. H. N. PSARAFTIS, "A Dynamic Programming Approach to the Dial-A-Ride Problem," Dept. of Civil Engineering, M.I.T. Report R78-34, 1978.
8. N. CHRISTOFIDES AND S. EILON, "An Algorithm for the Vehicle Dispatching Problem," *Opnl. Res. Quart.* **20,** 309–320 (1969).
9. C. CLARKE AND I. WRIGHT, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," *Opns. Res.* **12,** 568–581 (1964).
10. B. GILLETT AND L. R. MILLER, "A Heuristic Algorithm for the Vehicle Dispatch Problem," *Opns. Res.* **22,** 340–349 (1974).
11. H. N. PSARAFTIS AND G. G. THARAKAN. "A Dynamic Programming Approach to the Dial-A-Ride Problem; an Extension to the Multi-Vehicle Case," Department of Civil Engineering, M.I.T. Report R79-39, 1979.

(Received January 1979)