An Experimental Expert System For Routing Problems

MS-CIS-94-15

Jian Zhang

University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

March 1994

An Experimental Expert System

For Routing Problems

Technical Report

Jian Zhang

Department of Computer and Information System

University of Pennsylvania

March 31, 1994

AN EXPERIMENTAL

VEHICLE ROUTING SYSTEM

Jian Zhang

University of Pennsylvania

(April 23, 1994)

Abstract --- This paper describes an experimental system which provides heuristic solutions to vehicle routing problems. The system basically contains two components: an initial routes generator and a route improver. The route generator consists of an modified version of the sweep algorithm, and a new TSP algorithm, called the shrink algorithm. It is observed that the shrink algorithm is much more computationally efficient than Lin's exchanged method, while giving outputs comparable to those from the latter. The route improver in turn consists of two sub-components. The first is a small yet efficient expert system, which identifies and remedies specific problems in the initial routes, with the experience of human routers incorporated in the improvement process. The second sub-component is a pair-wise re-router, which runs the saving algorithm on neighboring-route pairs, trying to reduce the total costs. The system has been tried on several typical routing problems, and yields optimal or near-optimal solutions.

I. Introduction

Vehicle routing represents a wide range of logistics management problems. Out of the great variety, this paper specifically considers the following version. A number of customers are located around a central depot, each requiring a certain quantity of goods from the latter. A number of vehicles, each with a fixed capacity, deliver goods to satisfy the customer demands. The problem is to determine the number and the paths of a set of routes with the minimum total travel cost and without load constraint violation, while satisfying all customer orders.
For simplicity, the capacities of all the vehicles are assumed to be identical, and there is no restriction on route travel time and distance.

Table 1.1 gives the notations and definitions uses in the whole article, for the convenience of the discussion.

TABLE 1.1 General Notations and Definition

K = number of vehicles.
N = number of locations.
b = capacity of each vehicle.
q_i = demand at location i.
d_{ij} = travel cost between locations i and j.
x_i, y_i = rectangular coordinates of location i.
r_i = radius of location i, i.e., distance from depot to location i;
a_i = polar coordinate angle of location i, defined as

$$a_i = \arctan\left(\frac{y_i - y_0}{x_i - x_0}\right)$$

where: $0 <= a_i <= \pi$ when $y_i > y_0$ and $\pi < a_i < 2\pi$ when $y_i < y_0$.
(depot is indexed as 0)

Given the prohibitive time complexity for searching global optimality in the exact algorithms, most of the working algorithms of vehicle routing take heuristic approaches. A major category of the heuristics is the two-phase technique (Christofides 1985), and a well-known two-phase algorithm is to use the sweep algorithm (Gillett 1971) for clustering locations into groups and to use the k-opt exchange method (Lin 1965,1973) for solving the TSP for each group.

While capable of giving optimal or near-optimal solutions to problems, the above algorithm has its weakness. on the one hand, it suffers from its low computational efficiency (Gillett 1971). The defect stems largely from the fact that both the sweep algorithm and the k-opt method are highly iterative. On the other hand, when very tight capacity constraint exists, the algorithm often fails to fine any solution, or gives solutions of poor quality, due to the angular sequentialness in location sweeping and the limited range for location replacement.

An new system for vehicle routing has been developed to tackle the problems. The system has two basic components: a route generator and a route improver. The route generator provides an initial solution for a problem. While following the basic sweep procedure, it utilizes several new techniques in clustering locations, including route cost estimation and look-ahead/look-back assignment. Meanwhile, a new and highly efficient TSP algorithm is used for routing each cluster, in place of Lin's algorithm.

The route improver is made up of two sub-components: a small route-improving expert system, called Route Doctor, which makes diagnosis on, and treatments to, problems in the initial solution; and a pair-wise re-router which applies the saving algorithm on pairs of neighboring routes in order to find new pairs with less cost.

II. Improved Sweep Algorithm

The original sweep algorithm takes the following procedure (Gillett 1971).

1) All customer locations are rearranged according to their polar-coordinate angles.

2) From a starting location, the first route begins and takes in successive locations until the vehicle capacity is full. The next route starts at the next unassigned location.

3) After the formation of a route, inter-route location replacements are tried to reduce cost.

4) After all the routes are formed, the total cost is recorded. The location next to the current starting location is chosen as the new header, and a new cycle begins from Step 2.

5) The whole procedure stops when all locations have been tried as the starting location. The minimum-cost solution is selected as the final solution.

The sweep algorithm finds a good solution by taking pains in making an exhaustive search for a best sweeping starting location, although most of the starting points will proved to be quite inappropriate. We must evaluate all the solutions with the exchange algorithm, which is also iterative in nature and time consuming. The expense will double if backward sweep is also conducted. Meanwhile, since the initial solutions before location replacement are produced by strict angularly sequential assignment, their availability is low when very tight vehicle capacity constraint exist, i.e., when the ratio of total customer demand to total vehicle capacity is high.

Improvements seems possible in several aspects. First, instead of using the conventional time-consuming TSP algorithm to evaluate solutions, we may use an estimator to quickly approximate the solution costs. To compensate for the error brought in by the estimation, we establish a heap, which accommodates a certain number of feasible solutions (each made of a set of routes) that have the smallest estimated costs. The size of the heap should be such that it will capture the true minimum-cost solution most of the time. We apply the TSP algorithm only to those solutions in the heap to find the real best solution.

The second enhancement is for dealing with tight vehicle capacity constraint. It is possible that when vehicle capacity constraint is high enough, and when customer demands vary greatly, the standard sweep algorithm will fail in finding any feasible solution. To remedy this, a "look-ahead" method can be used. When the load of a route is still low yet assigning next location will result in capacity overflow, we may jump over one or more high-demand locations to continue the sequential assignment, so as to ensure high utilization of vehicle capacity. On the other hand, when a route is completed yet even after looking ahead the load is still low, we start to "look-back", trying to take location over from the previous route. The purpose is to delete unnecessary jumped assignments in the previous route, since jumped assignments often cause route overlap and therefore increase cost.

The new sweep algorithm operates in steps as follows:

1: Sort locations with their polar angles as the first sorting index and their radiuses as the second, i.e., i comes before j if $a_i < a_j$, or $a_i = a_j$ yet $r_i < r_j$.

2: Set LA = 0, where LA = number of locations to look ahead. (LA = 0 means no look-ahead). Set H = 1, where H = index of the header(starting location) for a sweeping cycle.

3: Set I = H, where I = starting location of a route.

4: Assign locations I,I+1,...,L to route k (k=1,2,...K), where L is the last location that can be added to route k without capacity violation.

5. See if w>=avgw, where w = accumulated cargo weight for current route, avgw = average route weight = $\Sigma q/K$. If not, then begin **looking ahead** to check if $w+q_{L+a}<=b$ (a=2,..LA+1). If yes, add location L+a to the route. Take Step 5 again, until w>avgw or a>LA+1.

6. If w<avgw, that is, the route is somewhat underloaded, begin **looking back**. Check if any location l in the range of route k, i.e., $a_l > a_{I+1}$ and $a_l < a_L$, has been assigned to route k-1. If found, try to take it over to route k.

7. Estimate the cost of route k (we will discuss the estimation soon). Let c_k be the estimated cost. Add c_k into the total cost of the current solution.

8. Check if all locations have been assigned. If not, set I = L+1, go back to Step 4 to build another route;

9. Check if the number of routes exceeds K. If not, try to put the current solution (current route set) into the best-solution heap; otherwise, try to join routes in the solution and put the solution into the heap. If the joining fails, discard the solution.

10. Increment H, i.e., choose a new header, go back to Step 3 for another sweep cycle. If no more header, check if there are enough feasible solutions in the heap. If not, increment LA, and go to Step 2 for a whole new round with a larger looking ahead range.

11. Apply the TSP algorithm to the routes of all the solutions in the heap, choose the one with the minimum cost as the final output.

The solution heap contains only unique solutions. So if a current solution has a route set that is identical to any solution already in the heap, the solution will be thrown away.

III. Route Cost Estimation

The application of the **route cost estimator** plays a major role in cutting down the computing time of the revised sweep algorithm. Here the length of the **"outline"** of a route is used as the estimator of its true cost. The outline of a route is defined as follows. Suppose a route contains locations j,j+1,...L (ranked in polar angle). We define the "peak" of the route as

$$p = \max_{i=j}^{L} r_i \qquad (3\text{-}1)$$

Suppose $r_P = p$, then location P divides the route into two sections L1 and L2, where L1 = {j,j+1,...P} and L2 = {P+1,...L}. A "mountain-climbing" method is used to form the outline. From L1, we find the "upslope" part of the outline, which initially contains the depot and the first location j. we scan forwardly from location j+1 to P-1, only taking into the outline those locations whose radius is not smaller than that of the previous one on the outline. Let O be a list of on-outline locations, which currently contains m locations. Then location l should be included into O only when $r_l >= r_{Om}$. In other words, the distance from the depot should be monotonously increasing during the formation of the outline. The "downslope" of the outline is formed in a similar way, only that we scan backwardly from L to P+1.

An illustration is shown in Figure 1.1. In A, location 3 is the peak. Initially only location 2 is on the outline. Location 1 is not on the outline, since its radius is smaller than that of location 2.

Part B shows the outlines of some routes in a problem. It is seen that the outlines keep the general shapes of the routes. The computation effort for their lengths is trivial. It is true that the estimation is underestimated, but what is more important is whether this underestimation is constant, so that the ranking order of solutions in terms of cost can be preserved in the heap.

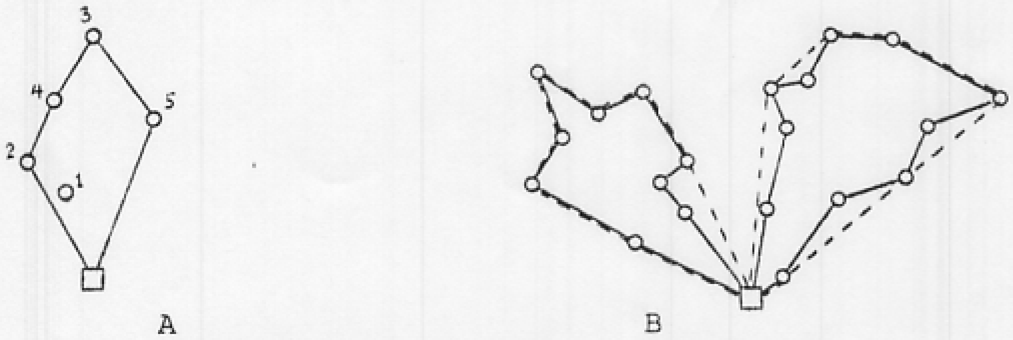

Figure 1.1 Outline of a Route

Table 3.1 displays the results from computer runs over four problems. For each problem, the estimated costs of the five solutions in the heap are compared with their actual costs. Please note the high consistency in the rates of underestimation across the solutions within each problem. Despite the small degree of freedom, the correlation coefficients are all above 0.95. To a large extent, the ranking orders of the estimated costs agree with those of the actual. This is the most important, since the bottom line here is to capture the actual minimum-cost solution. Experiments show that a heap with the size of 5% of the total number of the locations would be quite safe, which means a nearly 95% cut in computing time, since the TSP module takes most of the CPU time.

Some variations seen in the underestimation rates across the problems are likely to be related to certain features of the problems, such as the average length of the routes, the location density, and the average location radius.

It should be pointed out that the outline produced by the above algorithm is not necessarily convex. Experiments shows that a non-convex outline is often a better cost estimator, since it captures some "dents" in the upper part of a route. However, the new TSP algorithm discussed in the next section, the **shrink algorithm**, needs a convex outline as a starting point.

TABLE 3.1. Statistics of Route Cost Estimation

Prob 1.	n=50					Prob 2	n=71				
Sol.	$C_{est.}$	Rank	$C_{act.}$	Rank	Err(%)	Sol.	$C_{est.}$	Rank	$C_{act.}$	Rank	Err(%)
1	485	1	532	1	-8.8	1	251	1	280	2	-10.3
2	485	2	533	2	-8.9	2	252	2	279	1	-9.9
3	504	3	554	4	-9.0	3	255	3	284	3	-10.0
4	504	4	551	3	-8.6	4	258	4	285	4	-9.7
5	516	5	565	5	-8.8	5	260	5	287	5	-9.5
				R_2 =	0.997					R_2 =	0.972

Prob 3.	n=75					Prob 4.	n=134				
Sol.	$C_{est.}$	Rank	$C_{act.}$	Rank	Err(%)	Sol.	$C_{est.}$	Rank	$C_{act.}$	Rank	Err(%)
1	844	1	878	1	-3.9	1	1279	1	1357	2	-5.8
2	845	2	883	2	-4.3	2	1279	2	1358	3	-5.8
3	853	3	892	3	-4.3	3	1285	3	1359	4	-5.5
4	862	4	905	5	-4.8	4	1285	4	1353	1	-5.0
5	863	5	901	4	-4.2	5	1302	5	1374	5	-5.3
				R_2 =	0.945					R_2 =	0.993

Note: 1. $C_{est.}$ -- estimated cost; $C_{act.}$ -- actual cost;
2. R^2 is between $C_{act.}$ and $C_{est.}$;
3. For comparability, the heaps for the four problems have the same size, although they are different in real runs.

IV. The Shrink Algorithm

Closely looking at Figure 1.1 B, we may find that the solid lines, which are the links in the optimal routes, often run fairly close to the dotted lines, which are the outlines. We may say that the outline is an embryo to the final optimal route. Since an outline is easy to obtain, it is possible, and beneficial as well, to take it as the starting point to build the optimal route.

It must be pointed out that when forming an outline for building the optimal route, we should include only those locations into a outline so that all the vertices on the outline are convex. This guarantees the sequence of the locations in the outline is the same as that in the optimal route.

The TSP within the sweep algorithm is somewhat different from the TSP in general. Rather than having customer locations scattering all around, the sweep algorithm slices the whole network plain into a number of route cones fanning out from the depot. Intuitively, a good slicing would have comparatively wide spaces between those route sectors, and have locations close to the outline within each route sector. The former phenomenon means the routes are "narrow" so that costs due to links crossing the inter-route spaces are saved, while the latter means the absence of extensive zigzags in the route. These facts favor building routes from outlines.

The new algorithm, called the shrink algorithm, is very efficient in solving the TSP for the sweep algorithm. In general, it takes the following procedure:

1) Find the outline (see above) of a route, which becomes the initial partially-formed route.

2) For each location not yet assigned to the partially-formed route, calculate its assignment cost. The cost is associated with a specific link on the partial route, and the link represents a potential position for inserting the location into the partial route.

3) Choose the lowest-assignment-cost location, break its associated link, and insert the location in between. The partially-formed route expands by one location.

4) if there are more locations to be assigned, go to Step 2; otherwise, the route is completed.

Figuratively speaking, the route at first takes on a "swollen" form, and then bit by bit shrinks to its best shape. An important issue here is how to calculate the assignment cost for each free (not yet assigned) location, which determines the shrinking sequence. The first and the simplest way is to use the minimum insertion cost. Suppose at a certain point, the route contains locations 0,I,I+1,...,L, (the depot is location 0). Define

$$AC_J = IC_J = \min(d_{i,J} + d_{J,k} - d_{i,k}), \qquad (4-1)$$

$$i = 0, I, I+1, \ldots L \; ; \; k = I, I+1, I+2, \ldots, L, 0.$$

where AC_J = the assignment cost of location J, and IC_J = the minimum insertion cost of location J. If the insertion cost comes to the minimum when i=I+m and k=I+m+1 (m>=0, m<=L-I), then the associated link is the one between locations I+m and I+m+1.

The second choice defines

$$AC_J = IC_J \,/\, IC'_J \qquad (4-2)$$

where IC'_J = the second smallest insertion cost of location J. The smaller the ratio, the more likely that the current associated link is where the location should be inserted. Therefore, the first criterion is greedy in nature, while the second is farther sighted. Experiments show that the second criterion works often better than the first one.

Of course, we may use more complicated formulas, a possible candidate would be

$$AC_J = \alpha * IC_J \,/\, AVL + \beta * IC_J \,/\, IC'_J \qquad (4-3)$$

where AVL = average link length of the route; α and β are parameters, and $\alpha + \beta = 1$. The advantage of using this criterion is that we adjust the parameters to suit a unique problem. CPU time will increase a little.

In order to reduce computational cost further, we store intermediate results during the shrink procedure. Instead of calculating after each insertion the insertion costs of a location to all the links on the partially formed route and choose among them the smallest one as its assignment cost, we do that only at the beginning, and store the assignment cost of each free location (also the second smallest cost if we use equation 4-2). Each actual insertion creates two new links. So we just calculate the insertion costs of all free locations to these two new links, and compare the results with the stored data. If the new costs are less than the stored costs, we change the associated links of the related free locations. Only for those locations whose assignment costs are associated with the link broken by the insertion, we need to do a whole round of recalculation.

The shrink algorithm with the second assignment cost calculation method has been tried on all the above-mentioned four problems, and yields solutions identical to those best solutions from Lin's algorithm. However, the reduction in computing time is drastic (see Table 4.1). It should be pointed out that in order to save time in the experiments, the route cost estimator is also used in the "sweep+lin" method. Otherwise, the difference in CPU time between the two would be even greater.

The last two columns in Table 4.1 show that the computation saving of the shrink algorithm increases with the average number of locations on the routes ($R^2 > 0.99$). The fact makes the shrink algorithm a useful tool to solve large-scale problems. Another merit of the algorithm is that it always gives the same solution for the same problem, while the random generation of initial routes renders the output from Lin's exchange algorithm non-deterministic.

TABLE 4.1 Comparison on Computing time

Problem	N	A) Sweep Shrink	B) Sweep-Lin	Ratio(A/B)	Avg. # of Loc./route
1	50	8	75	0.106	10.0
2	71	25	732	0.034	17.7
3	75	15	120	0.125	7.5
4	134	100	3390	0.029	19.1

Note: 1 -- Computing times are on a 386SX PC without math coprocessor
2 -- Without running Route Improver.
3 -- Ratio of number of iterations to number of locations for Lin's algorithm is set to 0.3.
4 -- All times are in seconds.

V. The Route Improver

Although the route generating component produces an initial solution quite efficiently, there is no guarantee that the solution is optimal, due to the heuristicity of the algorithm. Hence, under most circumstances, further improvement is possible. It is especially true in our case when tight constraints are present, which severely restricts the route generator from finding enough feasible solutions to select from.

In the standard sweep algorithm, inter-route location replacements is the sole measure to improve the initial solution. For each route, locations are tried to be inserted into its front, and deleted from its rear, so as to reduce the total cost. The benefit from the action is limited. Often, some routes are good and do not need location replacement, so blind trials only waste computing efforts. Or, high vehicle capacity constraint makes deleted locations unacceptable by routes at rear.

The new system provides much more opportunities for the refinement of initial solutions, with its Route Doctor module and its pair-wise re-routing module. Normally, there are two flows of improvement: 1) initial solution -> Route Doctor -> improved solution ; 2) initial solution -> pairwise re=router -> Route Doctor -> improved solution. Out of the two improved solutions, the better one will be the final output. Figure 5.1 displays the general structure of the system.

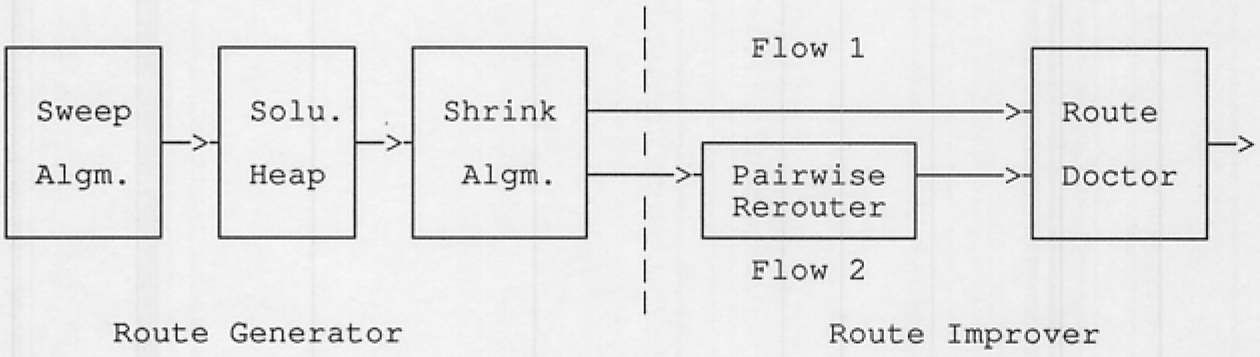

Figure 5.1 The General Structure of the System

V.I Route Doctor

Knowledge-based expert systems (KBES) provide a good means for route improvement (Duchessi 1988). Typically, a KBES is composed of two components: a knowledge base, and an inference engine. The knowledge base stores domain knowledge in form of facts, production rules, etc. Each fact describes an attribute of the problem at hand, and each rule represents a piece of human knowledge that may be applicable to solve the problem. The inference engine contains general problem-solving knowledge in form of meta-rules, which determine the way to identify symptom from the existing facts (interpreter) and the sequence to apply relevant rules to find the remedy (scheduler) (see Figure 6.1). According to the way of reasoning, inference engines in expert systems are categorized as backward chaining (from goal to contributing facts), and forward chaining (from facts to desired goal). Typical routing expert systems use forward chaining, working from an initial set of conditions and attempt to reach a goal state.

To improve a solution, the Route Doctor module works in a way similar to that of a human router. In general, Route Doctor loops in two-stage cycles. On the **diagnosis stage**, it examines the network and the current solution, obtaining information on general characteristics of the solution, and searching for symptoms indicative of specific problems. When multiple problems are found and categorized, the program picks out the most "serious" one, as is defined by meta-rules. On the **treatment stage**, the program tries to take a remedial action. If there are multiple actions which have the potential to solve the problem, Route Doctor tests them in an order in accordance with the meta-rules. An action is executed only if it leads to a better feasible solution. The loop ends when no problem exists or no treatments are found for existing problems. Human domain knowledge is incorporated into the above interpreting and scheduling processes. The advantage of using an routing expert system like Router Doctor rather than a human router is the former's high speed and consistency.

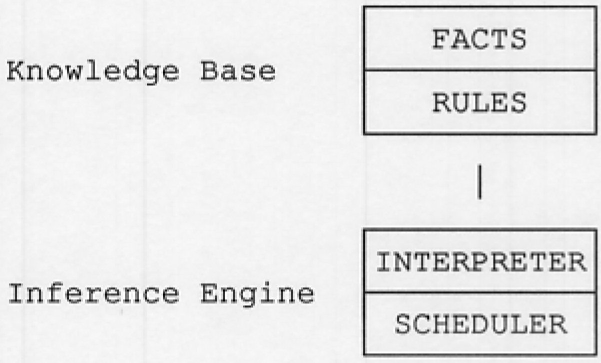

Figure 6.1 Structure of Expert Systems.

Additional notations and definitions used in the this section are summarized in Table 6.1.

TABLE 6.1 Route Doctor Notations and Definition

La_r = angle vicinity of route r;
Ld_r = distance vicinity of route r;
Ma_r = mean of inter-location angle of route r;
Md_r = mean inter-location distance of route r;
Mr_r = mean location radius of route r;
σa_r = standard deviation in inter-location angles of route r;
σd_r = standard deviation in inter-location distances of route r;
$\Theta_{i,j}$ = absolute value of the angular difference between location i and location j;
h_r/t_r = the first/last location (ranked by polar angle) of route r;
Cn = constant as parameters (n=1,2,...);
o = A location/cluster which is outlying;
p/q = locations immediately before/after o.

We use the notations of **angle vicinity** and **distance vicinity** of a route. Angle vicinity is the weighted sum of the mean inter-location angle and the standard deviation in inter-location angles of the route (location list is sorted by polar angle), i.e.

$$La_r = Ma_r + C1 * \sigma a_r; \tag{6-1}$$

while distance vicinity is the weighted sum of the mean inter-location distance and the standard deviation in inter-location distances of the route (scheduled location list), i.e.

$$Ld_r = Md_r + C2 * \sigma d_r. \qquad (6\text{-}2)$$

Angle vicinity is used to determine route adjacency and outlying sectors, and distance vicinity is used to determine outlying locations and clusters.

Currently, Route Doctor uses two types of facts --- the **condition-descriptive facts** and the **problem-indicative facts**. The former, shown in Table 6.2, define geometric relationship among routes and locations; while the latter categorize problematic phenomena in the current solution, as are described in Table 6.3.

TABLE 6.2 Condition-descriptive Facts

A. Describing route-route relationship

1. Containing: Route 1 contains Route 2 if the latter's angular range (from a_{h1} to a_{t1}) is entirely within that of the former, i.e.

   $$\Theta_{h1,h2} < \Theta_{h1,t2} < \Theta_{h1,t1};$$

2. **Overlapping**: Route 1 and Route 2 are overlapping each other when part of their angular ranges overlap, i.e.

   $$\Theta_{h1,h2} < \Theta_{h1,t1} < \Theta_{h1,t2};$$

   and the overlapping area between the two routes are defined as

   $$alo_{12} = a_{h_2} - ALa_2$$

   $$aup_{12} = a_{t_1} + ALa_1$$

   where: alo_{12}/ahi_{12} = polar angle of the lower/upper boundary

3. Detached: Route 1 and Route 2 are detached from each other when their angular ranges do not overlap, i.e.

   $$\Theta_{h1,t1} < \Theta_{h1,h2} < \Theta_{h1,t2};$$

4. Adjacent: exists only between detached routes. Moreover, Route 2 is adjacent to Route 1 when

   $$\Theta_{t1,h2} <= Ala_1;$$

5. Near-to-Join: Route 1 and Route 2 are near-to-join to each other only when they are close on the average, i.e.

   $$\Theta_{t1,h2} <= 2\pi/K \quad (K = \# \text{ of routes})$$

B. Describing route-location relationship

Near-to-receive: Route r is near-to-receive to an outlying location o if

$$\min(\theta_{o,p}, \theta_{o,q}) > \theta_{o,h_r}$$

when o is forward, or

$$\min(\theta_{o,p}, \theta_{o,q}) > \theta_{t_r,o},$$

when o is backward;

Route r is near-to-receive to an outlying cluster o with Locations i,i+1,...m (in schedule order) if

$$\min(\theta_{i,p}, \theta_{m,q}) > \theta_{i,h_r}$$

when o is forward, or

$$\min(\theta_{i,p}, \theta_{m,q}) > \theta_{t_r,i},$$

when o is backward;

Route r is near-to-receive to an outlying sector o with Locations i,i+1,...m (sorted by polar angle) if

$$\theta_{i-1,i} > \theta_{m,h_r}$$

when o is forward, or

$$\theta_{m,m+1} > \theta_{h_r,i}$$

Also, assuming the outlier o (either an outlying location, or an outlying cluster, or an outlying sector) is on route r2, then r1 is near-to-receive to o if

1. r1 contains, or is contained by, r2; or
2. r1 overlaps r2 in the same direction as the direction of o

C. Location-location relationship

1. Clusters: Locations i,i+1,...m (in schedule order) on route r form a cluster if the first location i is close to all the rest locations, i.e.

   $\forall j\ d_{ij} =< C3 * Ld_r;$ (j=i+1,i+2,...m)

2. Swappable: Location i on route r1 is swappable with Outlying location/cluster o on route r2 when

   $\Theta_{u,i} * r_i < \min(\Theta_{o,p},\Theta_{o,q}) * r_o;$

   where: u is a location in r2, and is the angularly closest location to r1. The value of the left-hand-side product is called the swap-gap value of location i.

Note: 1. It is assumed that Route 1 is before Route 2 by polar angle.
2. For definitions of outlying location/cluster/sector, see Table 6.3.

Figure 6.2 is the demonstration of certain relationships between two routes. In Part A, route 1 contains route 2; in Part B, the two routes are overlapping; and in Part C, route 2 is adjacent to Route 1, but not vise versa, since the inter-route gap is smaller than the angle vicinity of route 1 but is greater than that of route 2.

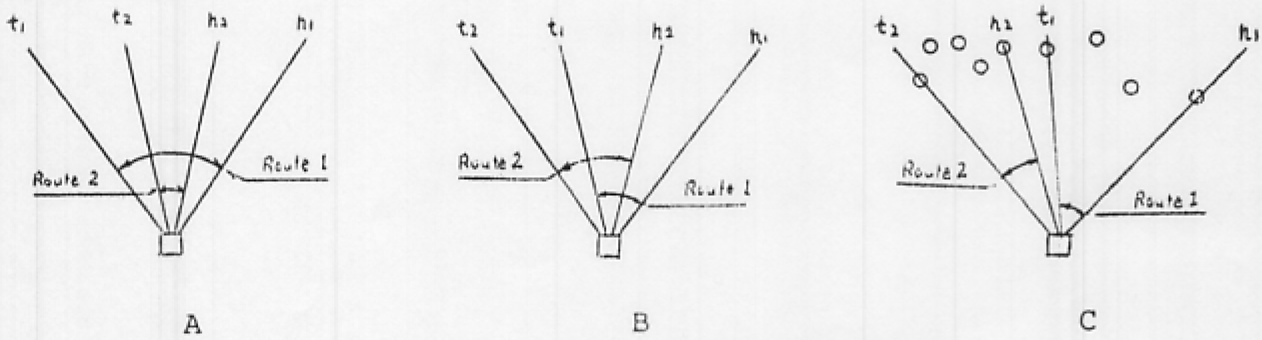

Figure 6.2 Route Relationships

TABLE 6.3 Problem-indicative Facts

1. Underloaded: Route r is underloaded if its load is below the average load too much, i.e.

$$w_r =< C4 * \sum_i q_i / b$$

where: w_r = total demand of locations on Route r;

2. Outlying location: Location o on Route r is outlying if:

1) $d_{poq} > Ld_r$, and
2) $d_{po}/d_{poq} >= C5$, and
3) $d_{oq}/d_{poq} >= C5$, and
4) $d_{poq}/d_{pq} >= C6$;

where: $d_{poq} = d_{po} + d_{oq}$, the total distance of a location, equals the sum of the length of the two links connected to the location;

Outlying location/cluster Categories:
Upward if $r_o>r_p$ and $r_o>r_q$ and $a_q<a_o<a_p$;
Downward if $r_o<r_p$ and $r_o<r_q$ and $a_q<a_o<a_p$;
Forward if $a_o>a_p$ and $a_o>a_q$;
Backward if $a_o<a_p$ and $a_o<a_q$;

3. Outlying Cluster: A cluster containing Locations i,i+1,...m is outlying if all the three conditions in B are satisfied if we substitute o with i and let q be the location immediately after m.

4. Outlying Sector: In a route, Locations i,i+1,...,m (ranked by polar angle) on Route r form an backward outlying sector if:

1) $\Theta_{m,m+1} > La_r$, and (a wide gap)
2) $m-i+1 < N_r / 2$; (not too many locations)

where: N = # of locations in Route r.

Locations m,m+1,...,n forms a forward outlying sector if:

2) $\Theta_{m-1,m} > La_r$
3) $n-m+1 < N_r / 2$;

5. Border Clash: There is a border clash between two routes if

1) the two routes are overlapping, and
2) within the overlap area, the lower route has an upward outlying location/cluster, or the higher route has downward outlying location/cluster (the height of route r is indicated by Mr_r).

Note: An outlier can be an outlying location or an outlying cluster, or an outlying sector.

There can be more than one outlying sectors in either the front half or the back half of a route. Suppose Route r has locations i,i+1,...m. While scanning the front half (from m back to the middle location j) (j=(m-i)/2), if we find a gap larger than the angle vicinity between two of the locations, say k and k+1, we mark k+1 through m as an forward outlying sector, and then continue the scanning. If another big gap is found between l and l+1 (l>j and l+1<k), then we mark l+1 through m as another outlying sector. The scanning on the front half stops when j is hit. Scanning on the back half for backward outlying section is done in a similar way.

Figure 6.3 demonstrate certain problem-indicative facts. Part A shows a backward outlying location o1 and an forward outlying cluster o2; Part B shows an forward outlying sector o3; and Part C presents a case of border clash.

In determining the **swappability** of a location in a receiving route with an outlying location/cluster (see Table 6.2, C.2), we use the product of its radius and the angular gap between this location and the outlier-giving route (without the outlier) as the criterion, for we want to consider those close-to-depot locations in the receiving route. In Figure 6.4, a is swappable with o, but b is not. c is also swappable with o, since c's small radius offsets the large gap between c and route 2. When the value of gap is negative (in the case of overlapping), we just use gap as the criterion, to prevent those far locations with a negative gap from being tested first.

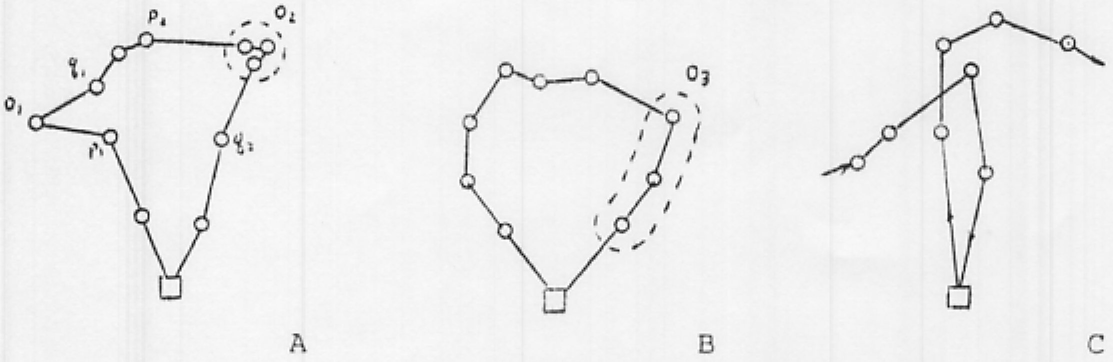

Figure 6.3 Problem-indicative Facts

For the presence of many parameters used in the above fact definitions, one may worries about the proper setting of their values. In fact, using these parameters is convenient rather than troublesome. The parameters have straightforward meanings, as are shown in Table 6.4, and their values are easy to determine. Meantime, they provide leverages for the user to control the running of the system. For instance, we may set parameters C5 and C6 low, so that the program will have greater diagnostic power to identify less conspicuous outlying locations/clusters.

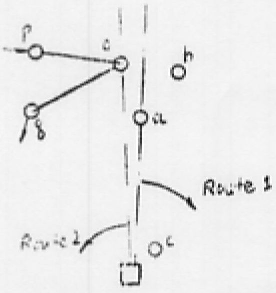

Figure 6.4 Location Swappability

TABLE 6.4 Meaning of Parameters

C1 ---	Weight of standard deviation in angle vicinity (=1)
C2 ---	Weight of standard deviation in distance vicinity (=1)
C3 ---	Maximum ratio of distance between two locations over the route distance vicinity for the locations to be regarded as belonging to a cluster (=0.2)
C4 ---	Maximum ratio of a route's total load over the average load for a route to be considered underloaded (=0.5)
C5 ---	Minimum ratio of the length of an location's either link over the location's total distance for the location to be considered outlying (an outlying location/cluster must be detached on both sides) (=0.3)
C6 ---	Minimum ratio of a location's total distance over the length of the bypass link for the location to be considered outlying (an outlying location/cluster can not be on a fairly straight path) (=1.2)
Note:	Values in parentheses are the values currently being used for the parameters.

Table 6.5 lists the **actions** that Route Doctor currently takes for route improvement. In attempting to eliminate an outlier (either an outlying sector, or an outlying location, or an outlying cluster), the **shift** action is always tried first, which is the simplest action.

If shifting leads to capacity violation, the **swap** action will be tried. First an array of swappable locations in the outlier-receiving route is established, sorted ascendingly by their swap-gap value. Combinations of locations in the array which incur no capacity violation are tested as the back-shift group. For computational efficiency, the size of the array (n1) and the maximum size of the back-shift group (n2) should be small. Currently, n1 = 5 and n2 = 3;

If swapping again fails to yield a feasible and better solution, the **relay** action will then be attempted as the last measure to eliminate the outlier. Suppose an outlier is found in route A, and route B is near-to-receive to the outlier. If the shift action will cause an overload in B, and the exceeding weight can not be shifted back to A, we try to shift some of B's locations to a third route, route C. The relay action is successful if the net saving of the two shiftings is positive. To reduce computing time, the two shiftings should be in the same direction. The locations shifted from B to C can be an outlier, or locations close to C, selected in a way similar to that of selecting back-shift group in the swap action.

While outliers indicate conflicts in horizontal (or more precise, in periphery), border clashes represent conflicts in vertical. In the **vertical-sweep** action, therefore, we apply the sweep algorithm vertically. Suppose Locations i,i+1,...m (sorted ascendingly by their radii) are in the overlapping area between route 1 and route 2 (route 1 is lower than route 2). We append the locations sequentially to the non-overlap section of route 1 until the capacity of route 1 is violated. Suppose the last location appended to route 1 is j. We are done if appending j+1 to m to route 2 does not lead to a capacity violation in route 2, otherwise one-to-one location swaps are tried between the two sections of i-j and j+1-m for removing the violation. Again, some limits are needed to maintain computation efficiency (see preconditions in Item 5, Table 6.5).

TABLE 6.5 List of Actions

1. Join	
Description:	Combine two routes into one route
Purpose:	Eliminate underloaded routes
Preconditions:	Between two near-to-join routes; no vehicle capacity violation
2. Shift	
Description:	Shift an outlier from route A to route B
Purpose:	Eliminate the outlier in route A
Preconditions:	Route B must be near-to-receive to the outlier; no vehicle capacity violation
3. Swap	
Description:	Shift an outlying location/cluster from route A to route B, meanwhile shift some location(s) from route B back to route A
Purpose:	Eliminate the outlying location/cluster in route A
Preconditions:	Route B must be near-to-receive to the outlier; no vehicle capacity violation

4. Relay

Description:	Shift an outlier from route A to route B, and meanwhile shift some location(s) in route B on to route C
Purpose:	Eliminate the outlier in route A
Preconditions:	Route B must be near-to-receive to the outlier; route C must contains/be contained by route B, or route C overlaps with or is adjacent to route B in the same direction as the outlier shifting

5. Vertical-sweep

Description:	In the order of their radius, assign locations in the overlapping area of routes A and B to the non-overlapping sections of routes A and B
Purpose:	Eliminate border-clash condition between routes A and B
Preconditions:	Routes A overlaps with route B; overlapping area is not too wide ($<1/6\pi$) Not too many locations in the overlapping area (<15)

Note: The TSP algorithm is applied to all the routes changed in the above actions.

The following is a simplified example of Route Doctor's rules for taking improvement actions:

```
1. IF   (problem = outlying location)
   AND  (route-location-relation = close-to-receive)
   AND  (route-load + location-demand < vehicle-capacity)
   THEN (action = shift)
2. IF   (action = shift)
   AND  (saving <= 0)
   AND NOT (problem = outlying sector)
   THEN (action = swap)
```

As has been mentioned above, a meta-rule serves as either an interpreter, to identify the most serious problem from the existing facts, or as an scheduler, to determine the sequence of applying relevant rules to that problem. Table 6.6 displays the principal of some meta-rules employed in Route Doctor.

TABLE 6.6 Principal of Some Route Doctor Meta rules

- If an underloaded route exists, try to eliminate it first, regardless of any outliers or border-clashes.
- If both outliers and border clashes exist, first try to eliminate the former.
- If both outlying sectors and an outlying locations/clusters exist, try to eliminate the former.

- If multiple outlying sector exists, deal with the one with the widest gap first.
- If multiple outlying locations/clusters exist, deal with the one with the greatest saving first.
- If there are multiple near-to-receiving routes for an outlier, try the nearest one.
- Try the shift action before the swap and relay actions.
- Try the swap action before the relay action.

Table 6.7 reports the results of a Route Doctor improvement run on the sweep output for the 100-location problem. From the table it is observed that the relay actions yield the largest saving. A possible reason is that often one relay action can eliminate two outliers (in two routes). It is similar with the swap action.

TABLE 6.7 The Result of a Route Doctor Run

Cycle	Action	# Loc. Shifted	Saving
1	relay	2+3	20.71
2	shift	1	1.01
3	swap	3+3	18.14
4	relay	3+2	28.51
5	shift	3	0.92
6	vertical-sweep	–	10.95
7	shift	3	6.44
8	shift	1	4.07

Route Doctor uses the shrink algorithm to solve the TSP for every route changed in the remedial actions. Currently, the module is written in the C language. The advantage is the high speed, especially in the mathematical calculations; while the disadvantage is the difficulty in updating the program.

VII Pairwise Re-router

The sweep algorithm slices the whole network plain into K route cones, assigning locations into routes in sequence of their polar angles, regardless of their radii. On the other hand, the saving algorithm is very sensitive to location radius, and tends to link locations close to each other yet far from the depot, which results in large savings. Therefore, if there are both far locations and near locations (relative to the depot), the sweep algorithm will produce "high" routes with mixed far and near locations; while the saving algorithm will create both "high" and "low" routes, the high ones containing mostly far locations, while the low ones having near locations. The routes from the saving method are usually "fatter" than those from the sweep method. Figure 7.1 gives a good example.

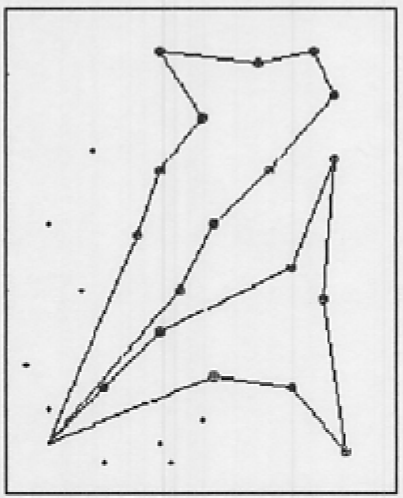

Figure 7.1.a The Sweep Output

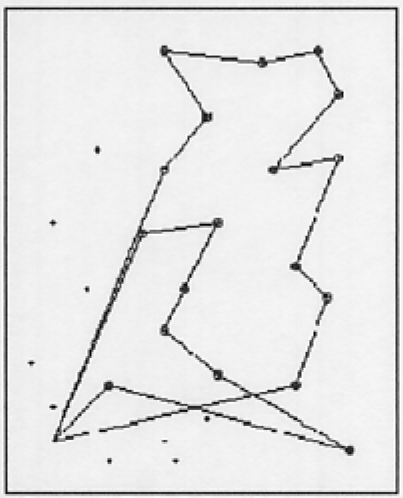

Figure 7.1.b The Saving Output

In some problems, there is great disparity in the radii of locations, and some far locations are fairly close to one another. Due to vehicle capacity constraint, the sweep algorithm may fails to assign those neighboring far locations into the same route, thus incurring extra cost for multiple trips to and from the remote areas. It is especially true when some near locations with high demands are in between those far locations in terms of polar angle. On the other hand, the saving algorithm is more likely to assign those far locations into one route, thus reducing the cost. An intuitive scheme for a better routing heuristic, therefore, is the cooperation between the sweep and the saving algorithm.

This leads to the employment of the pairwise re-routing module in our system. The pairwise re-router applies the saving method to neighboring pairs in the output from the sweep algorithm. A neighboring pair has two routes in which one contains (is contained by), overlaps, or is adjacent to, the other. The module works in cycles. Each cycle is made up of three steps: 1) finding route relationship (see Table 6.2 A), 2) trying re-routing on neighboring pairs, and 3) saving a re-routed pair. To save time, in cycles other than the first, only the relationship between the two new routes and the other routes is renewed in Step 1.

Although there are only about K neighboring routes at the beginning, the number of neighboring routes is likely to grow as routes begin to overlap with, or contain, one another. Fortunately, it is unnecessary to try re-routing on every neighboring pair. The program can skip those pair

1. which has just been re-routed in the last cycle;
2. which has not been changed since last unsuccessful trial on it;
3. in which the nearest location in the higher route is farther away from the depot than the farthest location in the lower route; meanwhile the remaining capacity of the higher route is less than the smallest demand in the lower route;

A policy must be made for determining the trial order of the pairs. It is noted that usually cost is more likely to be reduced in high routes, especially when two high routes has about the same height. Hence, the product of the heights of two routes in a pair is used as the ranking index of the pair. In other words, the saving method is applied to neighboring pairs in the order of the magnitude of the product of their maximum location radii.

A pair re-routing trail is successful when it reduces cost. However, reduced cost does not necessarily mean the automatic adoption of the re-routing. Each actual pair re-routing affects the subsequent process, and thus affects the final result. A good policy here is that in the early n cycles, we keep the most successful trial so far (with the currently largest cost reduction) and look ahead into the next m successful trials for an even better result. If there is no better result, we adopt the current best result and end the cycle, otherwise we start looking ahead from the new best result. Large values of n and m put greater guarantee on the good quality of the final result, yet require more computational effort. Experiments show that a value of 2 is appropriate for both n and m.

To a large extent, results from the saving algorithm vary with the value of parameters it uses. The pertinent parameters include: 1) the number os neighbors considered (NB), 2) the saving weighting factor α, 3) the penalty weighting factor β (Tillman 1972) (typically $\alpha + \beta = 1$). , and 4) the route shape parameter Γ (Golden 1977). The optimal setting of these parameters are problem dependent. We may greatly increase the availability of good solutions by doing multiple runs with different parameter values. Results indicate that compared with the other, the shape parameter is less effective in altering results. In the re-routing model, therefore, variations are made only in NB and in α.

The model uses a very effective strategy to reduce the number of runs in search of the best solution. The user provides the upper and lower boundaries on NB and α, as well as the search increments. The program makes a two-level search:

1. Start searching at the mid-values of NB and α.
2. Make first-level search on NB, increment or decrement NB according to the direction in which a better solution is found, until the local optimal (NB1) is obtained.
3. Compare the costs of the two best solutions (C1 for NB1, and C2 for NB2). If $C2/C1 < r$ (currently $r=1.05$), make two second-level searches on α with NB1 and NB2 respectively; otherwise, search on α with NB1 only;
4. Output the best solution found in the search(es) on α.

In Step 3, the second best solution with a much higher cost is considered unpromising to lead to the optimal, and therefore discarded.

In order to cut down computing time further, the program stops searching as soon as one of the following conditions is satisfied:

1. no feasible solution is found in the first-level search,

or

2. the current best solution is reached for the third times.

It has been observed that the number of unique solutions found by the saving algorithm is often limited. The repeated appearance of a solution with the lowest cost may indicate that it is the optimal. If the original solution for the pair is already the optimal, then search will stop after it hits on that solution just twice.

Table 7.1 reports the results of a pairwise re-routing process for the 134-location problem. It is seen that a large proportion of cost reduction is realized in the first two cycles, in which the looking ahead method is used. Meanwhile, the later cycles see many unsuccessful runs. Although a large number of runs are involved, the total computational effort is small, due to the high speed of the saving algorithm and the shrink algorithms. Another test with longer looking-ahead duration (n=6, m=2, see above) is made, which costs 533 seconds and the final cost reduction is 173.6.

Figure 7.2 displays the relationship between cost reduction and computing time. It is again observed that in general earlier cycles are important in bringing down the cost. The last section of the curve on the right represents the work done by Route Doctor, while the horizontal section before it is the last cycle in the re-routing process, which makes no improvement and thus ends the process.

Cy-cle	Cost Reduc-tion	Comput-ing Time	# Runs	# Pairs Tried
1	84.7	19	13	3
2	26.0	28	13	3
3	0.1	8	5	1
4	5.1	24	8	2
5	0.3	23	8	2
6	12.6	27	10	2
7	6.5	35	7	1
8	23.6	31	10	3
9	4.0	93	17	3
10	-	70	15	3
Total	162.9	359	106	23

TABLE 7.1 Results of Re-routing Runs

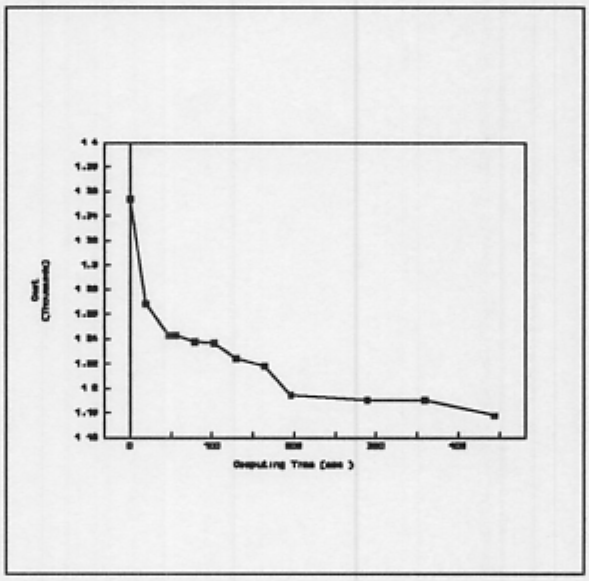

Figure 7.2 Cost Reduction and Computing Time

Note :
1. Parameter searching ranges: NB -- 3-5; α -- 0.8-1.0; (Γ=1)
2. Parameter increments: NB -- 1; α -- 0.1;
3. On a PC 386SX machine without math. coprocessor; time in seconds.

VIII. Computational Results

The system has been tried on some typical vehicle routing problems. Table 8.1 gives the summary characteristics of these problems. Problem 1 through Problem 6 are well-known test cases taken from the literature, with locations generated from a uniform distribution (Fisher 1981). Problem 7 through Problem 9 are taken from real routing applications (Fisher 1990). All of these problems have tight vehicle capacity constraints, as are seen in the fourth column of the table.

TABLE 8.1 Problem Characteristics

Problem	K	N	$\Sigma q/\Sigma b$
1	5	50	0.97
2	10	75	0.97
3	8	100	0.93
4	12	150	0.94
5	17	199	0.95
6	10	100	0.95
7	4	44	0.90
8	4	71	0.96
9	7	134	0.94

Table 8.2 through Table 8.4 report the computational results. Table 8.2 provides a comparison of solution quality among different algorithms on the randomly-generated problems. Out of the six problems, the system was able to find the best solutions on four, and the second best on the rest two. Its solutions are uniformly better than those from all the other algorithms except the Fisher-Jaikumar. In terms of the average solution cost, it outperforms all the other algorithms.

TABLE 8.2 Computational Results for Artificial Problems --- Solution Costs

Problem	N	Clarke-Wright	Sweep	Christofides tree search	Christofides 2 phase	Fisher-Jaikumar	Zhang
1	50	585	532	534	550	524	524
2	75	900	874	871	883	857	847
3	100	886	851	851	851	833	838
4	150	1204	1079	1064	1093	1014	1060
5	199	1540	1389	1386	1418	1420	1353
6	100	831	937	859	827	824	824
Avg.		1072.2	943.7	927.5	937.0	912.0	907.7

Table 8.3 displays the running times of the different algorithms on the above six problems. It is observed that the system is about four to seven times faster than the sweep algorithm, because of the high computational efficiency of the route cost estimator and the shrink algorithm. Although the system runs slower than some of the other algorithms, like the Clarke-Wright and the Fisher-Jaikumar algorithms, it should be noted that the system ran on a PC machine and the CPU time conversion is very conservative. Also, in order to test the diagnostic power of Route Doctor, the pertaining parameters are set with quite low values (see Table 6.4), resulting in large number of fruitless treatment runs in Route Doctor. Experiments show that it is almost always true that only the treatment of the most obvious symptoms lead to route improvement. In practical runs, therefore, we may increase the values of the parameters, thus reducing the running time of Route Doctor significantly. In addition, with some relatively easy improvements in the route generating module, which will be discussed later, the CPU time can be cut down further.

TABLE 8.3 Computational Results for Artificial Problems --- CPU time (in seconds)

Problem	N	Clarke-Wright	Sweep	Christofides tree search	Christofides 2 phase	Fisher-Jaikumar	Zhang
1	50	0.8	12.2	7.1	2.5	1.3	3.6
2	75	1.7	24.3	15.6	4.2	1.7	7.0
3	100	2.4	65.1	38.2	9.7	2.5	20.7
4	150	6.6	142.0	81.1	11.8	4.8	20.9
5	199	11.0	252.2	138.4	16.7	5.7	51.3
6	100	2.4	50.8	39.3	6.4	0.8	9.9

* The times for Zhang (originally on a PC 386SX without math coprocessor) have been converted to those for a CDC 6600 machine by dividing them with a speed difference factor of 11, which is quite conservative. All the other times are originally on the CDC 6600 machine, except for the Fisher-Jaikumar (see Fisher, 1982).

The computational results for the real vehicle routing problems are reported in Table 8.4. In all the three cases, the solution costs provided by the system are no more than 2.1 percent above the best solutions. The small differences in cost are in contrast to the drastic difference in computing time between this system and the k-tree algorithm, which is able to provide the best solutions (Fisher 1990). Considering the variance in computer speed, the system is 20 to 300 times faster than the k-tree algorithm for the problems tested. For lack of data, the other algorithms are not included.

TABLE 8.4 Computational Results for Real-world Problems

	Cost			Time	
Problem	Zhang	Best Solution	Difference	Zhang	Fisher(K-tree)
7	739	724	+2.1%	37	9900
8	242	242	0.0%	138	2220
9	1177	1164	+1.2%	627	15240

* The times for Fisher's algorithm are on Apollo Domain 3000 (20kz) (Fisher,1990), while the rest times are on PC 386SX (16kz)). All times are in seconds.

IX. Conclusion

The system discussed in this paper noticeably increases the computational efficiency of the sweep algorithm by adopting the route cost estimator and the shrink algorithm. Meanwhile, it produces quality routes with the help of an diagnostic expert system and an pairwise re-router. The pairwise re-router makes general changes, while Route Doctor chisels the details. Therefore, the first improvement flow (see Figure 5.1) is suitable for problems for which the route generator module has found the basic route structure. The second flow, on the other hand, is good for problems for which the solution from route generator deviate much from the optimal.

Despite the current satisfactory performance of the system, some further improvements are possible. For instance, when generating routes, solutions with neighboring head locations often differ only in the first one or several routes and in the last route. So we may save computational efforts by copying those identical middle routes from one solution to the next.

Furthermore, as an initial stage, the expert system limits its functions within route improvement. In fact, much of the E.S. power lies in its ability to handle a great variety of information, and find feasible and desirable solutions in complicated situations. Therefore, there exist two directions of its development. One is to dig more deeply into route amelioration, identifying more problematic patterns, finding more remedial actions, and carrying out sensitivity analysis. The other is to push it to a broader area, dealing with more factors met in reality, especially qualitative variables, in complement to mathematical programming applications.

REFERENCES

Belardo, S., P.Duchessi, and J.P.Seagle, "Microcomputer graphics in support of vehicle fleet routing," Interfaces, Vol.15, No.6, pp.84-92. 1985.

Bodin L.D., and L.Berman, "Routing and Scheduling of School Buses by Computer," Transportation Science, Vol.13., No.2, pp.113-129, 1979.

Clarke,G.and J.W.Wright, "Scheduling of vehicles from a central depot to a number of delivery points," Oper.Res. Vol.12, No.4, pp.568-581,1964.

Christofides, N., Chapter 12 "Vehicle routing", The Traveling Salesman Problem, John Wiley & Sons Ltd., 1985.

Christofides, N. and S.Eilon, "Algorithms for Large-scale Travelling Salesman Problems," Oper.Res.Quart. 23, pp.513-518, 1972.

Duchessi, P., S. Belardo, and J.P.Seagle, "Artificial Intelligence and the Management Science Practitioner: Knowledge Enhancements to a Decision Support System for Vehicle Routing," Interfaces, 18, Mar-Apr, pp.85-93 (1988).

Fisher, M. L., and R. Jaikumar, "A General Assignment Heuristic for Vehicle Routing," Networks, Vol.11, pp.109-124 (1981).

Fisher, M. L. "Optimal Solution of Vehicle Routing Problems Using Minimum K-trees", Working Paper 89-12-13, Department of Decision Sciences, University of Pennsylvania (1990).

Gillett, B.E. and L.R.Miller, "A Heuristic Algorithm for the Vehicle-dispatch Problem," Oper.Res. 22,pp.340-349 (1971).

Golden, B.L., T.L.Magnanti, and H.Q.Nguyen,"Implementing Vehicle Routing Algorithms," Networks,7(2),pp.113-148(1977).

Jessop, A., and J.W.Polak, "Towards Expert Systems for Goods Distribution Management," paper on the 5th IFAC/IFIP/IFORS Conference, Viena, Austria, Jul 8-11, 1986.

Lin, S., "Computer Solutions of the Traveling Salesman Problem", Bell System Tech. J.44,2245-2269.

Lin, S., and B.W.Kernighan, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem," Oper.Res.21,pp.498-516 (1973).

Ritchie, S., "A Knowledge-based Decision Support Architecture for

Advanced Traffic Management," Transportation Research A Vol.24A. No.1. 1990, pp27-37.

Tillman, F., and T. Cain, "An Upper Bounding Algorithm for the Single and Multiple Terminal Delivery Problem," Management Science, Vol. 18, No. 11, 1972, pp664-682.