A Heuristic Algorithm for the Vehicle-Dispatch Problem

Billy E. Gillett

University of Missouri, Rolla, Missouri

and

Leland R. Miller

Bowling Green State University, Bowling Green, Ohio

(Received April 11, 1971)

This paper introduces and illustrates an efficient algorithm, called the sweep algorithm, for solving medium- as well as large-scale vehicle-dispatch problems with load and distance constraints for each vehicle. The locations that are used to make up each route are determined according to the polar-coordinate angle for each location. An iterative procedure is then used to improve the total distance traveled over all routes. The algorithm has the feature that the amount of computation required increases linearly with the number of locations if the average number of locations for each route remains relatively constant. For example, if the average number of locations per route is 7.5, the algorithm takes approximately 75 seconds to solve a 75-location problem on an IBM 360/67 and approximately 115 seconds to solve a 100-location problem. In contrast, the time to solve a problem with a fixed number of locations increases quadratically with the average number of locations per route. The sweep algorithm generally produces results that are significantly better than those produced by GASKELL's savings approach and are generally slightly better than CHRISTOFIDES AND EILON'S results; however, the sweep algorithm is not as computationally efficient as Gaskell's and is slightly less so than Christofides and Eilon's.

THERE ARE many problems that fall in the general category of vehicle-dispatch or delivery problems; however, this paper specifically considers the following type of vehicle-dispatch problem. A number of customers located at different locations desire a certain quantity of goods that can be delivered from a single depot. A single vehicle may not be able to supply every demand because of load and distance constraints. The problem then is to determine the number of routes and the paths in each route that will minimize the total distance traveled by all vehicles in supplying all demands, subject to the load and distance constraints on each vehicle. Of course, the distance constraint could be replaced by a time constraint without changing the problem. Likewise, the problem could be thought of in terms of picking up customers or merchandise at each location.

Basically there are two types of algorithms that can be used to solve the vehicle-dispatch problem: exact and heuristic. Exact algorithms are ones that, in the absence of round-off or other errors, will yield an exact solution in a finite number of steps. CHRISTOFIDES AND EILON[2] developed an exact algorithm based on a branch-and-bound approach that was limited to problems involving relatively few locations because of the computer time involved. This is generally true of all exact algorithms that solve the vehicle-dispatch problem. They are all restricted by com-

puter time and/or storage. Heuristic algorithms, on the other hand, are quite often faster and capable of obtaining optimum or near-optimum solutions to much larger problems in a reasonable amount of computer time. This point is illustrated by methods B and C in Christofides and Eilon[2] and by the sweep algorithm presented in this paper.

Heuristic methods developed to date[2,6,7,14] have been limited to problems with no more than 100 locations. Each method attempts to solve the vehicle-dispatch problem as one big problem, whereas the sweep algorithm divides the locations into a number of routes and then operates on the individual routes until an optimum or near-optimum solution is obtained. When the problem is broken into a number of smaller subproblems, the computation time involved increases somewhat in a linear rather than in an exponential manner as more locations are added to a given problem. Consequently, problems with many more than 100 locations are well within the computational limits of the sweep algorithm. This concept will be treated in detail in the next section.

SWEEP ALGORITHM

THE ALGORITHM DISCUSSED in this section will yield an optimum or near-optimum solution to the single-depot vehicle-dispatch problem. The following notation and definitions will aid in its development.

N: number of locations including the depot (the depot is always location 1),

$Q(I)$: demand at location I, $(I=2, 3, \cdots, N)$

$(X(I), Y(I))$: rectangular coordinates of Ith location, $(I=1, 2, \cdots, N)$

C: capacity of each vehicle,

D: maximum distance each vehicle can travel,

$A(I, J)$: distance between locations I and J,

$\mathrm{An}(I)$: polar coordinate angle of Ith location $(I=2, 3, \cdots, N)$ defined as

$$\mathrm{An}(I)=\arctan[(Y(I)-Y(1))/(X(I)-X(1))],$$

where $-\pi<\mathrm{An}(I)<0$ if $Y(I)-Y(1)<0$, and $0\leqq\mathrm{An}(I)\leqq\pi$ if $Y(I)-Y(1)\geqq 0$.

$R(I)$: radius from location 1 to location I. $(I=2, 3, \cdots, N)$

The constraints on the problem are

$$\begin{aligned} Q(I)&\leqq C, && \text{for all } I,\\ A(I, J)&>0, && \text{for all } I\neq J,\\ A(I, I)&=0, &&\\ A(I, 1)+A(1, I)&\leqq D && \text{for all } I. \end{aligned}$$

Assume the locations are renumbered according to the size of their polar-coordinate angle. That is, assume $\mathrm{An}(I)<\mathrm{An}(I+1)$ for all I. If there exist an I and J such that $\mathrm{An}(I)=\mathrm{An}(J)$, then $I<J$ if $A(1, I)<A(1, J)$. This determines a unique ordering.

The vehicle-dispatch problem is to minimize the total distance traveled in supplying all demands while satisfying all constraints.

The *sweep algorithm* consists of two parts, a forward sweep and a backward sweep. In the *forward-sweep algorithm*, the locations are partitioned into routes beginning with the location that has the smallest angle, namely, location 2. Recall that the locations were renumbered according to the size of their polar-coordinate angle and the depot is location 1. The first route consists of locations $2, 3, \cdots, J$, where J is the last location that can be added without exceeding the vehicle capacity or distance constraint. The second route contains locations $J+1, J+2, \cdots, L$, where L is the last location that can be added to the second route without exceeding the vehicle capacity or distance constraint. The remaining routes are formed in the same manner. The total distance traveled then is just the sum of the distances for each route.

The procedure is then modified to consider replacing one location in route K with one or more locations in route $K+1$ for $K=1, 2, \cdots, m-1$, where m is the number of routes formed. The replaced location is left unassigned, to be picked up by a later formed route. Of course, a replacement is made only if the total distance is decreased. The location to be deleted from route K is obtained by minimizing a function of the radius $R(I)$ and the angle $\mathrm{An}(I)$ of each location in route K. This provides a location that is close to the depot and also close to the next route. A function that works very well is $R(I)+\mathrm{An}(I)\cdot AVR$, where AVR is the average of the radii for all locations. The first location, say location p, that is considered for inclusion in route K is the location in route $K+1$ that is nearest to the last location that was added to route K. The second location considered for inclusion in route K is the location in route $K+1$ that is nearest to location p. If one or more locations are added to route K by this scheme, then the next location in route $K+1$ is also checked to see if it can be included in route K. This process is continued until J is the last location in route $K+1$ that can be added to route K. Location $J+2$ is then checked to see if it can be included in route K. Choosing locations in this manner may not give the best solution; however, it is a very fast scheme for selecting the locations and it produces good results.

The process of adding one or more locations to route K and deleting another location continues until no improvement is found ($K=1, 2, \cdots, m$). The X and Y axes are then rotated counterclockwise so the first location becomes the last, the second becomes the first, and so forth. The above procedure of partitioning routes and interchanging locations between routes is then repeated. Again a minimum total distance is calculated. The process of rotating the X and Y axes is continued until all possibilities have been exhausted. Each time a minimum total distance is calculated. The smallest of these minimums provides a good heuristic solution for the vehicle-dispatch problem.

A second algorithm, called the *backward-sweep algorithm*, is exactly like the forward-sweep algorithm except it forms the routes in reverse order. Initially route 1 contains locations $N, N-1, N-2, \cdots, L$, route 2 contains locations $L-1, L-2, \cdots, M$, and so forth. In most cases the two procedures produce different routes and consequently in some cases different minimum total distances. Of course, the smallest of these two minimums is the best approximate solution.

A step procedure for the forward-sweep algorithm is given in Appendix A.

The sweep algorithm was used to solve a large number of vehicle-dispatch problems where the number of locations ranged from 10 to 250. The time to solve a

given problem is strictly a function of the number of routes and the number of locations per route. However, the number of locations per route has the greatest effect on the total computer time. In fact, the time to solve the vehicle-dispatch problem increases linearly with the number of locations if the number of locations for each route is approximately the same, whereas the computer time increases quadratically with the average number of locations per route if the total number of locations remains relatively constant. This explains why one 40-location problem may take 10 seconds to solve while another 40-location problem may take 50 seconds to solve; the first problem may have approximately five locations per route, while the second may have only two routes with 20 locations each.

When the sweep algorithm is used to solve a vehicle-dispatch problem, the only significant amount of time is the sum of the times associated with each route. The initial allocation of locations to routes takes a very insignificant amount of time. The time T_K associated with the Kth route is the time required to check to see if a location should be deleted from route K and/or if other locations from route $K+1$ should be added to route K, $K=1, 2, \cdots, m-1$, where m is the number of routes. This process involves solving the traveling-salesman problem each time a check is made. If the route contains relatively few locations, T_K is fairly small. On the contrary, T_K increases rapidly as the number of locations per route increases. The computer time needed to solve the traveling-salesman problem increases quadratically with the number of locations, even with the best algorithm.

If the number of locations per route is relatively constant, say h, then $T_K=B=$ constant for all $K=1, 2, \cdots, m$, and the total time for the sweep algorithm is $\sum_{K=1}^{K=m} T_K$. Thus, if h additional locations are added to the original problem it only requires approximately B additional units of time to solve the new problem. In fact, if any number of locations, say p, is added to a given problem, the approximate time to solve the new problem is the time to solve the original problem plus the time to solve a separate vehicle-dispatch problem with p locations.

The computer time for the sweep algorithm increases quadratically with the average number of locations per route if (1) the total number of locations remains relatively constant, (2) the demand per location is uniformly distributed, and (3) the locations are uniformly distributed in the (X, Y) plane. To illustrate, locations were generated that met these three constraints. One case involved 75 locations and another 100 locations. The demand constraint was varied so that the average number of locations per route was 5, 10, and 15 for each case. Table I gives the results.

Note that, if the average number of locations per route remains constant, the computer time per route remains relatively constant and the total computer time increases linearly as the number of locations increases from 75 to 100. Also note that, for the 75-location problem, the total computer time increases quadratically as the average number of locations per route increases from 5 to 10 to 15. A least-squares equation to predict the total computer time needed to solve the 75-location problem as a function of the average number of locations per route was calculated:

$$y=227.2-68.57x+6.33x^2,$$

where x is the average number of locations per route and y is the total computer time in seconds. The multiple correlation coefficient R^2 was equal to 0.96 for this model. Similar results hold for the 100-location problem.

TABLE I

COMPARISON OF COMPUTER TIMES FOR 75 AND 100 LOCATIONS

Average number of locations/route	Number of locations	Total computer time (sec)	Computer time per route (sec)
5	75	41	2.73
5	100	59	2.95
10	75	174	23.20
10	100	233	23.30
15	75	622	124.40
15	100	856	128.52

TRAVELING-SALESMAN ALGORITHMS

EACH TIME A set of locations is considered for a given route, a traveling-salesman problem must be solved to determine the minimum path to service each of the locations in the route. Highly successful heuristic algorithms as well as exact ones are available to solve the traveling-salesman problem.

The heuristic algorithms are generally considerably faster, but sometimes sacrifice accuracy, whereas exact algorithms may be extremely slow but always get the optimum solution if time permits. Specifically, LIN AND KERNIGHAN[13] have developed a heuristic algorithm that will produce an optimum or near-optimum solution to 100-location problems in reasonable times. The HELD AND KARP[10] procedure is an exact method that must rely on a branch-and-bound procedure in many cases. Consequently, the computer times reported are excessive even when compared to heuristic results reported by LIN.[12] Table II shows the comparisons.

Lin's[12] algorithm failed to give the optimal solution in only one case. However, Lin and Kernighan[13] report that their revised algorithm gets the optimum solution for each of the cases in Table II. In addition, their revised algorithm is faster in most cases and is always at least as accurate as the original Lin[12] procedure.

TABLE II

COMPARISON OF THE LIN AND HELD-KARP PROCEDURES

Problem	Lin procedure[12]		Held and Karp[10]	
	Distance	Time (sec)	Distance	Time (sec)
25 cities; Held and Karp[8]	1,711	5.24		12.0
33 cities; Karg and Thompson[11]	10,861	10.7		
42 cities; Dantzig and Fulkerson[4]	699	36.3	699	54.0
48 cities; Held and Karp[8]	11,461	63.0	11,461	84.0
57 cities; Karg and Thompson[11]	12,985	107.2	12,955	780.0
Computer[a]	GE 635		IBM 360/91	

[a] Exact comparison between the IBM 360/91 and the GE 635 cannot be made, but the IBM 360/91 is several times faster.

The Lin-Kernighan[13] algorithm appears to be the best algorithm to use to solve the traveling-salesman phase of the sweep algorithm because of its near accuracy, with much less computer time, and its availability.

COMPUTATIONAL RESULTS

CHRISTOFIDES AND EILON[2] solved ten vehicle-dispatch problems from the literature and compared their results with those obtained by one of GASKELL'S[6] procedures for the same problems. Even though their method took approximately three to four times longer for most of the problems, it produced significant decreases in the total distances traveled in most cases and was able to reduce the number of routes in three of the problems.

TABLE III
LIST OF PROBLEMS

Prob. no.	Source	No. loc.[a]	Max. load	Max. distance	Ave. no. loc. per route[b]
1	Gaskell[6]	21	6000	210	5.25
2	Gaskell[6]	22	4500	240	4.4
3	Gaskell[6]	29	4500	240	7.25
4	Gaskell[6]	32	8000	240	8.0
5	Christofides & Eilon[2]	50	160	Unlimited	10.0
6	New problem	75	100	"	5.0
7	Christofides & Eilon[2]	75	140	"	7.5
8	New problem	75	180	"	9.37
9	New problem	75	220	"	10.71
10	New problem	100	112	"	7.14
11	Christofides & Eilon[2]	100	200	"	12.5
12	New problem	250	120	"	10.0

[a] Excludes depot.
[b] Using best solution.

Seven of the problems proposed by Gaskell[6] and Christofides and Eilon[2] along with five variations of these problems are presented in Table III. Problems 6, 8, and 9 are the same as problem 7, except that the load constraint is changed. The same is true with respect to problems 10 and 11. This was done to illustrate that the time to solve a given problem is highly dependent on the average number of locations per route and much less on the total number of locations.

Table IV shows a comparison of results from the vehicle-dispatch algorithms mentioned in this paper. In each case where a comparison with other algorithms was possible, the sweep algorithm was able to produce better solutions for the larger problems (50, 75, and 100 locations), and was able to obtain the same solution as Christofides and Eilon[2] in problems 3 and 4. The sweep-algorithm solutions for problems 1 and 2 were very close to the best available solutions. Christofides and Eilon[2] report that 10 runs of 36 seconds of IBM 7090 time each were needed to obtain their results for problem 1, which was just 0.7 percent better than that ob-

tained using the sweep algorithm, and 10 runs of 48 seconds each for problem 3, where the result was the same as the result from the sweep algorithm.

CONCLUSIONS

THE SWEEP ALGORITHM has the following attributes:

(a) It is a new approach to the solution of the single-depot vehicle-dispatch problem.

(b) The computer time increases linearly with the total number of locations if the number of locations for each route remains relatively constant.

TABLE IV

COMPARISON OF VEHICLE-DISPATCH ALGORITHMS

Prob. no.	No. loc.[a]	Method A			Method B			Method C		Method D		Best Sweep Sol.		
		Sol.	Rts.	min[b]	Sol.	Rts.	min[b]	Sol.	Rts.	Sol.	Rts.	Sol.	Rts.	min[c]
1	21	598	4	0.1	585	4	6.0	602	4	591	4	591	4	0.21
2	22	955	5	0.1	949	5	0.5	970	5	956	5	956	5	0.17
3	29	963	5	0.2	875	4	8.0	875	4	888	4	875	4	0.51
4	32	839	5	0.2	810	4	0.8	810	4	817	4	810	4	0.62
5	50	585	6	0.6	556	5	2.0	574	5	546	5	546	5	2.0
6	75							1127	15	1128	15	1127	15	0.68
7	75	900	10	1.3	876	10	4.0	865	10	884	10	865	10	1.23
8	75							765	8	754	8	754	8	2.23
9	75							723	7	715	7	715	7	3.68
10	100							1176	14	1170	14	1170	14	1.83
11	100	887	8	2.5	863	8	10.0	864	8	862	8	862	8	6.0
12	250							5794	25	5911	25	5794	25	9.7

[a] Excludes depot.

[b] IBM 7090.

[c] IBM 360/67—total time for Methods C and D.

Note: Method A—Gaskell's saving, multiple approach; Method B—Christofides and Eilon's 3-optimal approach; Method C—forward-sweep algorithm; Method D—backward-sweep algorithm.

(c) The computer time increases quadratically with the average number of locations per route if the total number of locations remains relatively constant.

(d) It is extremely useful for very large problems that have only a few locations per route on the average. Based on (b) above, it is practical timewise even up to 250 locations with 10 locations per route and 1000 locations with five locations per route. The 250-location problem with an average of 10 locations per route was solved in 9.7 minutes of IBM 360/67 time. The literature is void of algorithms that can solve problems this large.

(e) Attribute (c) above could restrict the algorithm to problems as small as 60 locations if there are approximately 30 locations per route.

(f) For large problems from the literature (50 locations or more) it has proved to be better than the best known algorithms and possibly requires just slightly

more computer time. (The IBM 360/67 is roughly three to four times faster than the IBM 7090, but may vary considerably depending on the program and programmer).

(g) For small problems it is comparable with the best known algorithms with respect to both time and accuracy.

APPENDIX A

A Step-by-Step Presentation of the Forward-Sweep Algorithm

If location J is in a given route and location $J+1$ cannot be added to it, then location $J+2$ is not checked. The notation in the body of the paper is assumed. Instead of relabeling the locations, we let $K(2)$ denote the location with the smallest angle, $K(3)$ denote the location with the second smallest angle, and so forth.

Step 1. Evaluate the polar coordinates for each location with the depot at (0, 0). Let $\mathrm{An}(I)$ represent the angle and $R(I)$ the radius for location I, $I=2, 3, \cdots, N$.

Step 2. Determine $K(I)$ for $I=2, 3, \cdots, N$ such that $\mathrm{An}(K(I))$ is less than or equal to $\mathrm{An}(K(I+1))$.

Step 3. Begin the first route with $J=2$ and $SUM=Q(K(2))$.

Step 4. Increment the angle by making $J=J+1$.

Step 5. If $SUM+Q(K(J))>C$, then go to step 7.

Step 6. Augment the route with location $K(J)$ by making $SUM=SUM+Q(K(J))$. If $J=N$, then go to step 16. If $J\neq N$, then go to step 4.

Step 7. Calculate the minimum distance D_1 for the route by means of a traveling-salesman algorithm. Check the distance constraint. If the distance capacity is exceeded, then eliminate $K(J-1)$ from the route. Make $SUM=SUM-Q(K(J-1))$ and $J=J-1$. Check the distance constraint again. Continue this procedure until the distance constraint is satisfied.

Step 8. Determine JJX so that $K(JJX)$ is the nearest location to $K(J-1)$ and not in a route. Find JII so that $K(JII)$ is the nearest location to $K(JJX)$ and not in a route. Likewise determine I so that $R(K(I))+\mathrm{An}(K(I))\cdot AVR$ is a minmum for all locations in the route. Let KII denote this I. Determine the minimum distance D_2 for the route with $K(JJX)$ added to the route and $K(KII)$ deleted from it.

Step 9. If $D_2\leqq D$ and the load constraint is satisfied, then go to step 11. Otherwise go to step 10.

Step 10. Record the route and start a new route by setting $SUM=Q(K(J))$. Go to step 4.

Step 11. Evaluate the minimum distance D_3 for starting at 1, traveling through locations $K(J), K(J+1), \cdots, K(J+4)$ and ending at $K(J+5)$. Determine the distance D_4 for traveling through the same locations, except eliminate $K(JJX)$ and inject $K(KII)$. If $K(JJX)$ is not $K(J), K(J+1), \cdots,$ or $K(J+4)$, then go to step 10. If $D_1+D_3<D_2+D_4$, then go to step 13. Otherwise, go to step 12.

Step 12. Place $K(JJX)$ in the route and remove location $K(KII)$. Go to step 4.

Step 13. Evaluate the minimum distance D_5 for the route with $K(JJX)$ and $K(JII)$ substituted for $K(KII)$. If $K(JJX)$ and $K(JII)$ are not $K(J), K(J+1), \cdots,$ or $K(J+4)$, then go to step 10. If $D_5<D$ and the load constraint is satisfied, then go to step 14. Otherwise, go to step 10.

Step 14. Determine the minimum distance D_6 for starting at 1, traveling through locations $K(J), K(J+1), \cdots, K(J+4)$, and ending at $K(J+5)$, with $K(JJX)$ and $K(JII)$ ex-

TABLE V

PROBLEM 3 OF TABLE III

No.	X	Y	Demand	No.	X	Y	Demand	No.	X	Y	Demand
1	218	382	300	11	126	347	950	21	159	331	1500
2	218	358	3100	12	125	346	125	22	188	357	100
3	201	370	125	13	116	355	150	23	152	349	300
4	214	371	100	14	126	335	150	24	215	389	500
5	224	370	200	15	125	355	550	25	212	394	800
6	210	382	150	16	119	357	150	26	188	393	300
7	104	354	150	17	115	341	100	27	207	406	100
8	126	338	450	18	153	351	150	28	184	410	150
9	119	340	300	19	175	363	400	29	207	392	1000
10	129	349	100	20	180	360	300				

Maximum load: 4500. Maximum distance: 240.
Allow 10 miles extra for each location visited.
Depot coordinates: (162,354).

cluded and $K(KII)$ included. If $D_1+D_3<D_5+D_6$, then go to step 10. Otherwise, go to step 15.

Step 15. Place $K(JJX)$ and $K(JII)$ in the route and eliminate $K(KII)$ from the route. Go to step 4.

Step 16. Evaluate the minimum distance for the route and check the distance constraint. If not satisfied, then go to step 17. If satisfied, then that set of routes is complete. Check to see if another set of routes is needed. If no more are needed, then go to step 19. Otherwise go to step 18.

Step 17. Delete one from the route ($J=J-1$). Go to step 10.

Step 18. Increment the angle by one location [i.e., start with $K(3)$ for the second set of routes]. Go to step 2.

Step 19. Stop.

APPENDIX B

THE DATA FOR problem 3 in Table III and the corresponding routes using the sweep algorithm are given in Tables V and VI. The data for the other problems in Table III, along with the corresponding sweep-algorithm solutions can be obtained from the authors.

TABLE VI

THE FORWARD-SWEEP ALGORITHM SOLUTION OF PROBLEM 3 OF TABLE III

No.	Route	Miles	Load
1	0-18-10-11-12-9-17-7-13-16-15-0	227.21	2725.00
2	0-26-28-27-25-24-29-0	233.95	2850.00
3	0-20-3-6-1-4-5-2-22-0	236.59	4375.00
4	0-23-8-14-21-19-0	177.24	2800.00
	Total	874.99	

Note: The depot is location 0.

REFERENCES

1. M. Bellmore and G. L. Nemhauser, "The Traveling Salesman Problem: A Survey," *Opns. Res.* **16,** 538–558 (1968).
2. N. Christofides and S. Eilon, "An Algorithm for the Vehicle Dispatching Problem," *Opnal. Res. Quart.* **20,** 309–318 (1969).
3. G. Clark and J. W. Wright, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," *Opns. Res.* **11,** 568–581 (1963).
4. G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson, "Solution of a Large-Scale Traveling Salesman Problem," *Opns. Res.* **2,** 393–410 (1954).
5. ——— and J. H. Ramser, "The Truck Dispatching Problem," *Opns. Res.* **12,** 80–91 (1959).
6. T. J. Gaskell, "Bases for Vehicle Fleet Scheduling," *Opnal. Res. Quart.* **18,** 281–295 (1967).
7. Robert Hayes, "The Delivery Problem," Carnegie Inst. of Tech., Management Science Research Report No. 106 (1967).
8. M. Held and R. M. Karp, "A Dynamic Programming Approach to Sequencing Problems," *J. SIAM* **10,** 196–210 (1962).
9. ——— and ———, "The Traveling Salesman Problem and Minimum Spanning Trees," *Opns. Res.* **18,** 1138–1162 (1970).
10. ——— and ———, "The Traveling Salesman Problem and Minimum Spanning Trees: Part II," *Mathematical Programming* **1,** 6–25 (1971).
11. R. L. Karg and G. L. Thompson, "A Heuristic Approach to Solving Traveling Salesman Problems," *Management Sci.* **10,** 225–247 (1964).
12. Shen Lin, "Computer Solutions of the Traveling Salesman Problem," *Bell Syst. Tech J.* **44,** 2245–2269 (1965).
13. ——— and B. W. Kerninghan, "A Heuristic Algorithm for the Traveling Salesman Problem," Computer Science Tech. Report # 1, Bell Labs., April 1972.
14. F. A. Tillman and H. Cochran, "A Heuristic Approach for Solving the Delivery Problem," *J. Indust. Eng.* **19,** 354–358 (1968).