A ROUTING SYSTEM BASED ON SPACEFILLING CURVES

John J. Bartholdi, III *

April 11, 1995; revised April 1, 2003

Abstract

We describe some fundamental issues in routing and distribution and illustrate them in a case study, which shows how to build a commercial-quality routing system in one day and with no computer.

1 Routing and distribution

Distribution is the getting the product to the customer. The management of distribution has continued to change rapidly after deregulation in the early 1980's and with the development of new technologies such as on-board computers, radio, and global-positioning systems.

It is cheapest to ship long distance by train and so railroads are capturing an increasing amount of long-haul freight. Small, urgently-needed goods move by more expensive plane. But currently the bulk of freight, and all local freight, is handled by truck.

The basic problem faced by a distribution manager, then, is this: How best to coördinate a fleet of vehicles to deliver goods to their destination[1]. One would like to accomplish many things, some of which work at cross-purposes, such as

- Allocate the goods among the vehicles so that not too many trucks are required.

 The author may be contacted at `john.bartholdi@isye.gatech.edu`.

[1]For a journalist's view of one trucking company, see Rounds, 1993.

- Route each truck to reduce the travel time.

In addition, there may be constraints of various types that limit one's choices:

- A trailer can hold only a limited volume of freight.
- The axle weight of a trailer must lie within certain limits specified by law.
- It may be that delivery to a customer is allowed only within a certain "time-window", such as between 08:00 and 10:00.
- A driver may work only for a limited time (say no more than ten hours).

Routing is just one aspect of the problem of distribution. There must be sufficient room in the shipping department to accumulate the freight in advance so it can be loaded on the trailer in reverse order of delivery. (Otherwise either the trailer must be unloaded and reloaded at each stop; or else the delivery sequence is arbitrarily determined by the order in which the freight happened to have been loaded.)

The simplest abstraction of the problem of routing is the "Travelling Salesman Problem" (TSP) which can be formalized as: Given n locations, find a tour of minimum total length. (A tour is a path that visits all the required locations and then returns to its origin). The TSP is a famous problem because it is simple to understand but hard to solve, at least for instances of realistic size. For a survey of solution techniques see Lawler et al., 1985.

Let us look at a commercial routing problem. . .

2 Meals on Wheels

Senior Citizen Services, Inc. is a private, nonprofit corporation in Atlanta, Georgia whose purpose is to provide social services for the elderly, especially the elderly poor, in Fulton County. One of their major services is the "Meals on Wheels" program (MOW), which delivers prepared lunches to people who are unable to shop or cook for themselves. As for many charitable organizations, the funding for MOW is unstable, chronically insufficient, and occasionally desperate. Any additional resources are

used to purchase more food for needy people, so the administrative facilities of MOW remain the bare minimum necessary to function.

MOW operates Monday through Friday each week. At 8:30 am the prepared meals are delivered by an institutional caterer to Senior Citizen Services in mid-Atlanta. There they are heated in a holding oven until about 9:15 am when four paid, part-time employees arrive. They load the meals into insulated bulk containers and then into their four vehicles. Each driver is given a "route manifest" which lists all of his delivery locations in a suggested order of visitation. Each then drives his route, delivering 40–50 meals to 30–40 locations between 10 am and 2 pm

Because the delivery vehicles are usually station wagons, they can easily carry sufficient meals, so vehicle capacity is not an effective limitation. The only constraint is that all meals must be delivered within four hours, the length of time the insulated containers will keep the meals properly warm (at least 140°, as required by the public health department). However, drivers complete their routes within the limit so time constraints are usually not active. In fact, neither vehicle capacity nor delivery time is likely to become an active constraint unless the system grows considerably, an unlikely event for a charitable organization during lean times.

MOW maintains two lists of clients: an active list of those to whom meals are currently delivered, and a waiting list of those hoping to join the system when space or additional resources become available. These lists are volatile. They change at a rate of about 14% each month because of the nature of the clients: most are elderly or ill. They may die, or recover from illness, or receive care elsewhere (in a hospital, nursing home, or family) and so leave whichever list they are on. Clients may be added to the active list either from the waiting list, or as emergency special cases (perhaps referred to MOW by a social worker).

The volatility of the active list is further increased by the special way in which MOW is funded. Senior Citizen Services receives operating revenues from three primary sources, the federal government, the state of Georgia, and United Way. Unfortunately, all three administer their grants under different fiscal calendars. The federal government begins its fiscal year on October 1, the State of Georgia on July 1, and United Way on January 1. The multiple fiscal years cause continuous turmoil at MOW

because each grant must be spent during its respective fiscal year. Consequently it is not unusual for a large number of people to be added to the active list during the close of a fiscal year then to be removed to the waiting list during the beginning of a new fiscal year.

MOW is managed by a devoted, energetic woman who is a full-time employee of Senior Citizen Services. As in many charitable organizations, the manager is overworked. Her responsibilities are many, including management of the MOW budget and meal delivery: planning menus, ordering meals from the caterer, monitoring the quality of meals, maintaining the insulated containers for the meals, and supervising part-time employees. She also recruits and trains volunteers, and coördinates her services with social workers. The manager has little time and no additional resources to devote to routing.

We set out to design a method to help the busy manager quickly generate efficient routes from a volatile list. This method could not rely on a computer, nor even on appreciable clerical efforts, for such resources are not within the means of MOW.

3 The "Travelling Salesman Problem"

Because there are no severe constraints on vehicle capacity or trip length, the MOW problem is (almost) a pure routing problem and so the "Travelling Salesman Problem" (TSP) is a suitable abstraction. Recall that in the TSP you are given the locations of some number n customers and you must devise a sequence in which to visit them and return to the starting location so that the total distance travelled is minimized. Like most abstractions, this does not perfectly capture the real world problem. In particular, it is different from the Meals-on-Wheels problem in that:

- There is only one person/vehicle/salesman travelling in the TSP.
- MOW wants to minimize travel *time*, not necessarily distance.

Nevertheless, the TSP is a good place to begin building an understanding of the delivery problem and its solution.

The TSP is a famous hard problem: It requires complicated optimization techniques and expensive computers to find minimum-cost solutions. This is impractical for this application for at least two reasons.

- The client cannot afford to purchase or maintain such a demanding system.
- Optimization is of doubtful use in this case: Ideally we would minimize travel time; but travel time is not well-defined as it may vary with driver, time of day, day of week, and so on. And even if travel time were well-defined, it would take much too long to measure all the required times (travel times between all pairs of locations)! Consequently it makes engineering sense to use travel distance as an approximation to travel time and to assume some approximate conversion factor, such as 25 miles-per-hour. Since all this is only approximate, it seems questionable to expend great expense in optimizing.

An alternative to optimization is to use *heuristics*; that is, procedures that produce reasonable good answers but quickly. Here are some of the simpler of the standard TSP heuristics. Assume that n locations are to be visited.

Nearest Neighbor. Starting from an arbitrarily chosen location, repeat the following: Travel to the closest location that has not yet been visited. Once all locations have been visited, return to the starting location.

Greedy algorithm. Rank all pairs of locations in order of non-decreasing distance. Pass through this list, adding the corresponding edge to the tour whenever doing so will neither curtail the tour prematurely nor revisit a location.

Nearest Insertion. Starting from a degenerate tour consisting of the two closest locations, repeatedly choose a location that is not yet on the tour but is closest to the tour; insert that location between the two consecutive tour locations for which such an insertion causes the minimum increase in total tour length.

Strip. Divide the map into $\sqrt{n}$ strips of equal width; then, starting at the leftmost strip, proceed strip by strip to visit the locations by order of latitude (“height”)

within each strip, alternately descending and ascending. Finally, connect the last location in the rightmost strip to the first location in the leftmost strip.

As you can see, with the possible exception of the Nearest Neighbor, even these heuristics require some care in implementation and would almost certainly require execution by computer for a problem of realistic size.

The routing system we implemented is based on a TSP heuristic that is extremely simple and yet provides good tours on the average (Bartholdi and Platzman, 1982 and Platzman and Bartholdi, 1989) . The idea behind this algorithm is a "spacefilling curve", the construction of which is indicated in Figure 1. You will notice that the curve may be described recursively: To produce the next refinement, shrink the present picture to quarter size (that is, shrink by half in each dimension), copy into each quadrant, and connect the pieces at the center. The spacefilling curve is the limit of this process.

Spacefilling curves were devised around the turn of the century by mathematicians such as Peano, Sierpinski, Hilbert, and others, who found that they seemed to defy intuition. For example, the Sierpinski curve is everywhere continuous curve but nowhere differentiable. It is also surprising that a 2-dimensional figure may be completely filled with a 1-dimensional curve. (For more about spacefilling curves, see Sagan, 1994.)

We can imagine a spacefilling curve to be the route of an obsessive delivery person who visits all the points of the unit square. The curve seems a reasonably economical route for such ambition. Accordingly, we use it to suggest a route to visit a subset of locations; that is, we visit only the required locations, but in the same sequence as the obsessive salesman. This will be easy to do because the obsessive route is already defined. Our only concern is that our route inherit the economy of the obsessive route.

The algorithm has the following structure:

Step 1. For each location (x, y) calculate its relative position θ along the spacefilling curve. The index θ is a number between 0 and 1.

Step 2. Sort the locations from the smallest to largest θ.

Thus the points to be visited are sequenced according to the order in which they appear along the spacefilling curve, as illustrated in Figure 2.

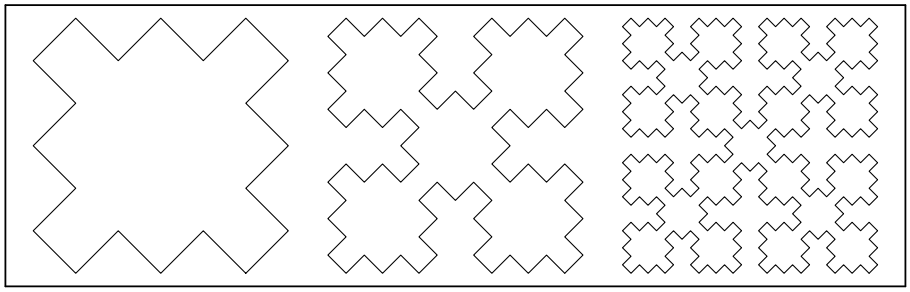

Figure 1: The Sierpinski spacefilling curve is the limit of the series of recursively-constructed figures shown above. Each is built upon the preceding figure by dividing the square into quadrants and filling each quadrant with a shrunken copy of the preceding figure. The limiting figure is a 1-dimensional curve that is continuous and visits every point in the 2-dimensional square.

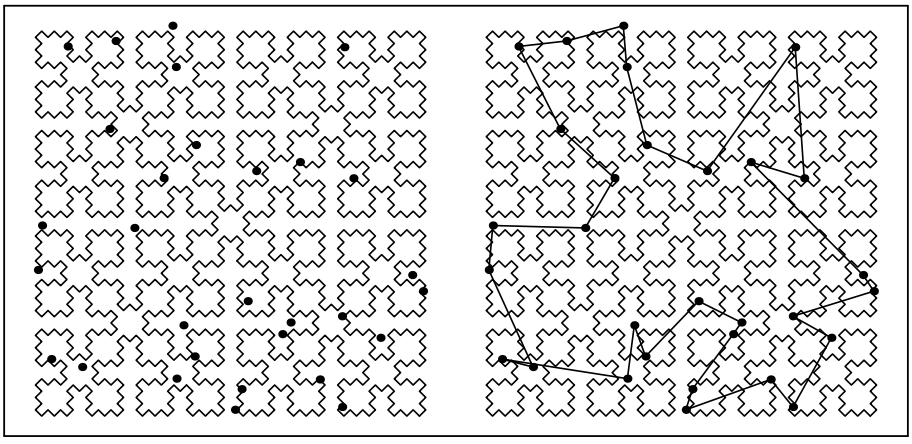

Figure 2: A set of random points and a TSP tour according to their relative position along the spacefilling curve.

3.1 Computing position on the curve

A spacefilling curve may be considered a mapping from the (1-dimensional) unit interval *onto* the (2-dimensional) unit square. Computing the position of a point along the curve is equivalent to computing a "pseudo-inverse" of that function. There are many ways to do this. Here is one of the simplest.

For example, let us compute, for the point $p = (0.74, 0.58)$, its percentage of the way around a Sierpinski curve. By convention we assume that the Sierpinski curve begins and ends at the southwest corner of the unit square, which we label accordingly as 0%, as well as 100%, of the way around the curve.

Step 1: From Figure 1 you will notice that the Sierpinski curve visits all points in the northwest triangle before visiting any points in the southeast triangle. This means the northeast corner must be 50% of the way around the curve; and so our point p, which lies within the southeast triangle, must be 50–100% of the way around the curve (Figure 3).

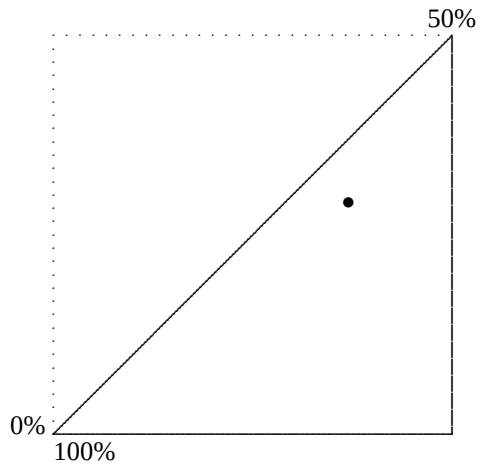

Figure 3: Computing percentage around the curve by successive refinement: first iteration

Step 2: The southeast triangle, which contains our point p, may be partitioned into two subtriangles as shown in Figure 4; and the spacefilling curve visits all the points of

the eastern subtriangle before visiting the southern subtriangle. Furthermore, the eastern and southern subtriangles are the same size and so their common southeasternmost point must be $(50 + 100)/2 = 75\%$ of the way around the curve. We see that p lies within the eastern subtriangle and so must be 50–75% of the way around.

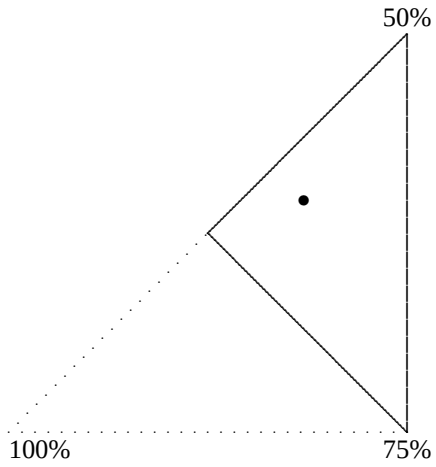

Figure 4: Computing percentage around the curve by successive refinement: second iteration

Step 3 and beyond: Now simply continue this process to as fine a level of accuracy as desired: Examining the eastern triangle we see that it may be partitioned into upper and lower subtriangles such that the curve visits all the points of the upper subtriangle before visiting the lower (Figure 5). The unlabeled vertex of the eastern subtriangle must be $(50 + 75)/2 = 62.5\%$ of the way around the curve. The point p lies in the upper triangle and so must be 50–62.5% of the way around the curve. (Problem: Finish the computation to two significant digits!)

The process is as follows. At each step the point p is known to lie within a triangle for which the percentages of the way around the curve are known for two of the vertices.

- Determine percentage of way around the curve for the unlabeled vertex of the triangle by averaging the percentages of the other two vertices.
- From this vertex bisect the opposite side of the triangle to partition it into two

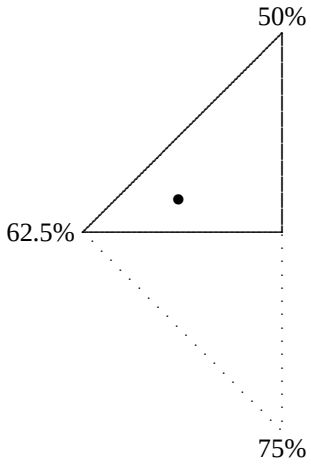

Figure 5: Computing percentage around the curve by successive refinement: third iteration

subtriangles.

- Determine which of the two subtriangles contains the point. If it is small enough, stop and report the label of one of its vertices; otherwise continue.

See the Appendix for a realization of this as a compilable computer program. Also, a table of pre-computed values for all points of a 100×100 grid is available from the first author on request.

3.2 Quality of the routes

The spacefilling curve heuristic generates tours that are about 25% longer than optimum for sets of points that are uniformly distributed (Platzman and Bartholdi, 1989). In real life, delivery locations tend to be clustered around urban areas and so the algorithm produces even better solutions because it tends to visit all the locations in a region before moving on the another region. The quality of this solution is competitive with other simple heuristics and it is an order of magnitude faster. Other attractive properties include:

- The algorithm requires minimal data: to route to n locations requires only the $2n$ values of the coordinates (x, y). In fact, the $O(n^2)$ distances between points are never required!

- Locations are easily inserted into or removed from the heuristic tour: It is as simple as inserting or removing cards from the card file. In particular, there is no need to re-solve the entire routing problem. In contrast tours generated by other methods are not so easy to modify.

4 The Routing System

The routing system we built for MOW is inexpensive but fairly accurate; consequently it has more implementations throughout the world than any other commercial routing system. It consists simply of a map, a table of θ values, and two Rolodex card files. The map is a standard Department of Transportation street map of Atlanta. It is mounted under a plastic grid so that one can read the (x, y) coordinates for any location.

We pre-computed a table for MOW so that they do not need to calculate θ. It allows one to enter with values for (x, y) and read a corresponding θ value. It fits easily on six sheets of paper. The map and table in effect place a spacefilling curve over the metropolitan Atlanta area, as suggested by Figure 6.

Each of the two card files contains a complete list of active clients. A card lists a client's name, address, and telephone number, together with miscellaneous notes, such as special handling required for meals, etc. In addition, each card has the client's θ number. Each client is represented by two identical cards, one for each file. The first file is sorted alphabetically by name, and the second is sorted numerically by θ.

The card files permit simple insertion and removal of clients so that the system can easily handle the volatility of the lists. To remove a client, one just looks up his name in the alphabetical list, notes his θ value, and removes the card, then looks up his θ value in the route list and removes that card also. To insert a client, one simply goes to the map and measures the (x, y) coordinates of the clients location, then enters the table with (x, y) and reads the corresponding θ value. Two identical file cards are prepared

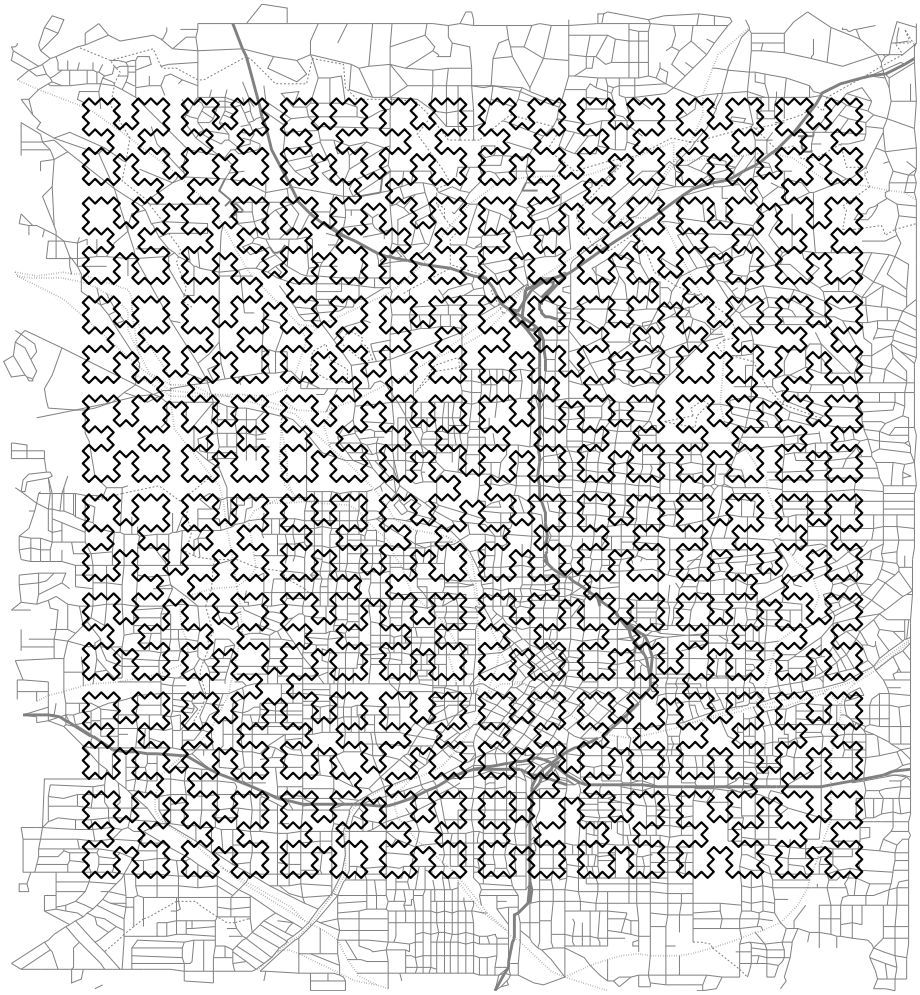

Figure 6: One can imagine Atlanta to be covered by a spacefilling curve, so that any specification of (x, y) coordinates can be equivalently given by the relative position along the curve.

for the client. One card is inserted into the alphabetical list and the other is inserted into the numerically sequenced list.

The second card file is the implementation of the heuristic. It maintains all client locations in the sequence that a single delivery vehicle would follow when obeying our heuristic. We convert this to subroutes for four vehicles by simply partitioning the cards of the card file into four roughly equal sets of contiguous cards. Then each vehicle drives approximately one-fourth of the single vehicle route. The idea of partitioning a single route into subroutes is similar to that of Fredrickson, Hecht, and Kim (1978), who proved nice worst-case bounds for stricter implementation of the method.

This route-partitioning scheme allows the manager to adapt the delivery plan to the number of vehicles available. Some of the vehicles were donated; some are undependable and might not start on a cold morning. On the other hand, sometimes a volunteer driver shows up in the morning. In any event, it is simple to partition the route list into sets of cards to immediately determine the appropriate number of routes.

Note that the system divides the management decisions appropriately:

- The card file sequences all the locations in advance; and this can be easily updated.
- The partitioning is done in the morning when it is known how many vehicles will be available.
- The system suggests only the sequence of deliveries (“go here first and then there”). The drivers choose the streets to travel because they know more about local driving conditions.

Finally, the drivers travel their routes clockwise to avoid making lefthand turns.

5 Implementation, Operation, and Performance

We had some initial difficulty in implementing the system because the drivers did not want to change their routes. Each was familiar with his area of the city and general sequence of locations. Moreover, there was concern that too much change would upset

the clients. Most of the clients are old, sick, and isolated, and for them the regular visit of a familiar driver is an important part of their day. However, because of budget reductions in July 1982, MOW had to severely reduce its active list and restructure the routes accordingly. Since major changes had to be made anyway, we implemented our system at that time.

We experienced one particular stumbling block in implementing the system. To help convince management to use our system, we had prepared a map that showed what good routes our system would produce for the current clients. Our mistake was that we left that map at MOW after training them on our system. It then happened that the manager, when rushed, occasionally made additions to the active list by "eyeballing" this map, thinking she would assign a θ value later. Eventually, under pressure of a heavy workload, she made all routing decisions simply by looking at the picture of the routes as we had drawn them early on. Unfortunately, the set of clients continued to change, so that our map became ever less representative of the real world. When we realized what was happening, we reclaimed our map and explained to the manager how failure to use the θ values would allow the system to degenerate to its previous inefficient state.

We discovered that partitioning the single route into subroutes caused some concern. The manager naturally wanted "ideal" routes, that is, those that look efficient on a map and balance total delivery time among the drivers. She was worried because the routes derived by partitioning tend to have somewhat large travel times to each first or from each last delivery. It seemed more reasonable, she felt, to go to someplace close first. But when we measured the routes, the relatively long initial and final legs of each were generally found to be an insignificant part of total deliver time. This was partly because the heuristic reduced total driving time to only 30-40% of total delivery time for each route. Thus occasional travel time aberrations from imperfectly partitioned routes are unimportant. However, the MOW manager preferred to hunt and make adjustments until she determined an acceptable partition. Since repartitioning is done only occasionally, this seemed satisfactory.

The manager tries to balance the work of the four drivers. Because they are paid by the hour, none wants an unusually short route, and trading routes among the drivers

is not acceptable because that would disrupt the driver-client relationship. Fortunately, the partition scheme works quite well in this regard. There are two reasons for this:

- The spacefilling curve heuristic produces relatively short routes so that the total travel time has been reduced to only about 30% of total delivery time; therefore the total delivery time now depends more strongly on how many meals are to be delivered.

- The spacefilling curve heuristic produces routes that, at least for locations from a uniform distribution, tend to minimize the *longest* leg of the route. Alternatively, the locations tend to appear approximately uniformly along the route, so that any fraction p% of the route tends to include p% of the locations to be visited, and vice versa. (A formal proof of this appears in Platzman and Bartholdi, 1989.)

Because the client list changes so quickly, it was not possible to directly compare the driving times and distances of our routes with previous routes. We did, however, submit a previous client list to our heuristic and determined that our routes were about 13% shorter as measured by Euclidean distance.

The most important improvement, however, is the facility with which the system may now incorporate changes in the client list or the number of drivers. Additionally, the number of required vehicles was reduced by one, which made the delivery system more dependable.

6 Other applications

6.1 "Star Wars"

The following is based on private communications with a contractor for the Strategic Defense Iniative, a Department of Defense effort begun in the mid-1980's to protect the US from nuclear attack by space-based laser weapons, which would presumably destroy incoming missles.

The targeting of a space-based laser weapon can be modelled as a TSP problem: Since the laser beam travels much faster than its targets and does not diffuse appre-

ciably, we may imagine the targets arranged on a two-dimensional focal plane. The problem then is to sequence the targets to minimize the time spent re-aiming. Thus the sequence "aim-shoot-aim-shoot-. . ." is analogous to the sequence "drive-deliver-drive-deliver-. . .".

There are some special constraints on solution techniques to this problem.

- The solution technique must be capable of sequencing 1,000–30,000 targets in real time.
- Because of the potential for disaster, the solution technique must be verifiable; that is, a complete mathematical analysis must be known, including worst-case analysis. This rules out such techniques as expert systems or neural nets.
- The solution technique must run on hardware that is boostable to orbit and rad-hardened. (A Cray supercomputer is boostable to orbit but its power supply is not.)
- The solution technique must be parallelizable: the chosen computer was the Connection Machine, with 64,000 cpu's running in parallel. Because the spacefilling curve heuristic is essentially sorting, it can be sped up significantly: Each cpu assumes responsibility for one target and computes the position of that target along the spacefilling curve; then all the cpu's coördinate to sort the targets in parallel according to their positions along the curve. It is known that n items can be sorted in parallel within $O(\log n)$ time (that is, the time to sort increases only according to a logarithmic function of the size of the input).

  Interestingly, most other heuristics for solving the TSP, such as the "Nearest Neighbor" algorithm, are apparently *not* parallelizable.

6.2 Courier service

Courier services send trucks out to pick up packages and deliver them to their destinations. One way of doing this is to keep a list of requests for pickups until enough have arrived to justify a trip; then a truck is dispatched to make all the pickups. The packages are then sorted according to destination and delivered.

There is an obvious opportunity to improve this service: As long as the truck is out, it might be able to deliver some of the packages it has picked up. Why not integrate pickups and deliveries into the same route? In fact, now that many delivery fleets have on-board communications, this is increasingly being done; that is, new requests for pick-up and delivery are integrated into the route as the route is being driven. Thus the routing problem changes as it is being solved.

One way to manage such a system is as follows. (For simplicity we describe the system as if there were only one truck/driver.) The driver maintains a deck of index cards sorted according to location to be visited; this deck represents his intended route given his current knowledge of locations to be visited. When he receives (by radio) another request for pickup, he prepares a card for that location and files it into the set of cards according to its appearance on a spacefilling curve. The pickup has now been inserted into the route. When he eventually reaches that pickup location, he loads the package onto the truck, then turns the card over writes the destination of the package on the back, and inserts the card according to the appearance of the destination on the spacefilling curve, then resumes driving his route. In this way the route dynamically incorporates new requests for service even as it is being driven (Bartholdi and Platzman, 1988).

6.3 Miscellaneous

- If zip codes were assigned according to a Sierpinski curve then an effective delivery route could be generated simply by sorting on zip code. (This does not work now because zip codes have little logic behind them. A zip code region need not even be connected!) Extended zip codes could be generated as needed by simply increasing the resolution to which the spatial index is computed (proceed to sufficiently small triangles).

- Geographic databases are frequently quite large (think of a road map of Georgia). They are generally stored in what is effectively a one-dimensional storage medium, such as a computer disk; hence the problem of how to write 2-dimensional data onto a 1-dimensional structure so that such common 2-d tasks

as range queries are efficient (example range query: show me all the apparel manufacturing plants within Fulton, Dekalb, Gwinnett, Clayton, and Cobb counties). The trick is to store points based on a spatial index: one computed from a space-filling curve, of course! Then points that are close on the map will tend to be close on the disk (which means faster retrieval); and points which are close in index will be close on the map (which means good delivery routes).

7 Questions

Question 1 *If carrying capacity is not a limitation, a single truck can make all the required deliveries in fewer miles than can a fleet of trucks, but will take longer. Explain why.*

Question 2 *Before the system was redesigned the MOW drivers reported seeing each other on the routes several times a day. Why does this indicate an inefficient set of routes?*

Question 3 *How does the MOW system decide routes for, say, three vehicles? Why does this technique tend to balance work among the drivers?*

Question 4 *Why does the spacefilling curve heuristic tend to produce more-nearly optimum routes when locations are clustered, as in an urban area?*

Question 5 *The spacefilling curve heuristic seems to ignore travel time and distance but nevertheless produces relatively short routes; how is this possible?*

Question 6 *Can the spacefilling curve heuristic be used to route a vehicle in a city that has a large lake or park through which traffic cannot pass? What problems might this cause the routing system? Explain.*

References

[1] PLATZMAN, L. K. AND J. J. BARTHOLDI III (1989). "Spacefilling curves and the planar travelling salesman problem", *Journal of the Association for Computing Machinery* **36**(4):719–737.

[2] BARTHOLDI, J. J. III, AND L. K. PLATZMAN (1988). "Design of efficient bin-numbering schemes for warehouses", *Material Flow* **4**:247–254.

[3] BARTHOLDI, J. J. III, AND L. K. PLATZMAN (1988). "Heuristics based on spacefilling curves for combinatorial problems in the plane", *Management Science*, special issue on heuristics, **34**(3):291–305.

[4] LAWLER, E. L., J. K. LENSTRA, A. H. G. RINNOOY KAN, AND D. B. SHMOYS (1985). *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley and Sons.

[5] ROUNDS, D. (1993). *Perfecting a Piece of the World*, Addison-Wesley.

[6] SAGAN, H. (1994). *Space-Filling Curves*, Springer-Verlag.

A Code to compute Sierpinski index of a point

Input: The coordinates x, y of a point, both of which must be non-negative integers no larger than the constant `maxInput`.

Output: An integer giving the relative position of (x, y) along a Sierpinski curve that fills the square $[0, \texttt{maxInput}] \times [0, \texttt{maxInput}]$. Note that this number is *not* the percentage of the way around the curve; but sorting on this number gives the same sequence.

(The following code is given in Modula-2, which is very similar to Pascal. It should be straightforward to translate into C.)

```
CONST
        maxInput = 10000; (* Choose appropriate value *)

PROCEDURE Sierpinski-Index( x : CARDINAL, y : CARDINAL ) : LONGINT;
        VAR
                result : LONGINT;
                loopIndex : CARDINAL;
                oldx : CARDINAL;
BEGIN
        loopIndex := maxInput;
        result := 0;

        IF (x > y) THEN
                INC(result);
                x := maxInput - x;
                y := maxInput - y;
        END;

        WHILE (loopIndex > 0) DO
                result := result + result;
```

```
                IF (x + y > maxInput) THEN
                        INC(result);
                        oldx := x;
                        x := maxInput - y;
                        y := oldx;
                END;

                x := x + x;
                y := y + y;

                result := result + result;

                IF (y > maxInput) THEN
                        INC(result);
                        oldx := x;
                        x := y - maxInput;
                        y := maxInput - oldx;
                END;

                loopIndex := loopIndex DIV 2;
        END;

        RETURN result;

END Sierpinski-Index;
```