A Simple Algorithm for the Metric Traveling Salesman Problem

M. J. Grimm
Communications Systems Research Section

An algorithm was designed for a wire list net sort problem. A branch and bound algorithm for the metric traveling salesman problem is presented for this. The algorithm is a best bound first recursive descent where the bound is based on the triangle inequality. The bounded subsets are defined by the relative order of the first K of the N cities (i.e., a K city subtour). When K equals N, the bound is the length of the tour. The algorithm is implemented as a one page subroutine written in the C programming language for the VAX 11/750. Average execution times for randomly selected planar points using the Euclidean metric are 0.01, 0.05, 0.42, and 3.13 seconds for ten, fifteen, twenty, and twenty-five cities, respectively. Maximum execution times for a hundred cases are less than eleven times the averages. The speed of the algorithm is due to an initial ordering algorithm that is a N squared operation.

The algorithm also solves the related problem where the tour does not return to the starting city and the starting and/or ending cities may be specified. The algorithm can easily be extended to solve a nonsymmetric problem satisfying the triangle inequality.

I. Introduction

The Digital Projects Group uses an in-house development aid program (Ref. 3) for specifying the interconnections of the I/O pins of integrated circuits placed on wire wrap boards. The program produces nets of (x, y) coordinates of points that must be interconnected. The coordinates are the locations of socket pins upon which at most two wires may be placed. An insulated wire electrically connects two socket pins. The length of a net connecting the N socket pins is the sum of the effective lengths of the $N-1$ wires connecting pairs of points in that net. When a wire wrap machine connects the point $(x1, y1)$ with the point $(x2, y2)$, the effective length of the wire is $|x1 - x2| + |y1 - y2|$. It is desirable to minimize the length of the nets.

For a typical logic design, a board has two thousand nets, with each net containing an average of three points. Approximately eight percent of the nets have ten or more points, and a few exceed fifteen points. With such small nets, it would seem that a simple algorithm could be specified that finds minimum length nets using a reasonable amount of computer time. The algorithm to be presented satisfies the requirement, and also

solves the metric traveling salesman problem (TSP). It also allows one or both end points of the net to be specified, facilitating the routing of electrically terminated nets.

II. The Combinatorial Problem

The TSP can be defined as follows. Given an N by N matrix $\mathbf{D}$ of non-negative integers; find an order vector $\mathbf{O}$ such that

$$\mathbf{D}(\mathbf{O}[N], \mathbf{O}[1]) + \sum_{I=1}^{N-1} \mathbf{D}(\mathbf{O}[I], \mathbf{O}[I+1])$$

is minimal. The order vector may be any permutation of the numbers one through N. If $\mathbf{D}[I, J] = \mathbf{D}[J, I]$ then the problem is called the symmetric TSP. If, additionally, $\mathbf{D}$ has a zero diagonal and satisfies the triangle inequality ($\mathbf{D}[I, J] \leqslant \mathbf{D}[I, K] + \mathbf{D}[K, J]$) the problem is called the metric TSP. If the first term in the minimization equation is omitted, the salesman does not have to return home, and the problem becomes the net sort problem. The distance defined in the introduction is a metric; so that problem is a metric net sort problem.

The algorithm to be presented solves both the metric TSP and the net sort problem. It can easily be extended to solve a nonsymmetric problem given that the triangle inequality holds.

III. The Algorithm

Bently (Ref. 2) describes approximate solutions to the TSP that are feasible for $N = 1000$. Smith (Ref. 4) defines the branch and bound algorithm but does not give execution times. Bellmore and Malone (Ref. 1) give execution times for random Euclidean problems that are three orders of magnitude inferior to the algorithm to be presented. The algorithm is a simple application of branch and bound using the triangle inequality. The novelty of the algorithm is a worst possible ordering presort which enhances execution times by about three orders of magnitude for $N = 20$.

The branch and bound algorithm (Refs. 1 and 4) can be defined recursively as follows. Given a set of nets and an upper bound (current minimum) on the absolute minimum length net: Partition the set into subsets and compute lower bounds on the lengths of the nets in each subset. If the lower bound of a subset is not less than the current minimum, discard that subset because it cannot contain a net shorter than the current minimum. If the lower bound of a subset is less than the current minimum and the subset contains more than one net, explore that subset. Otherwise, if the set has only one net, then the length of that net becomes the current minimum, and that net is saved as a potential minimum length net. The algorithm terminates when the set has been explored, and the potential minimum length net is then the minimum length net.

The algorithm to be presented is a best bound first algorithm; that is, when a set is partitioned, the first subset to be explored is the subset with the smallest lower bound. The algorithm also has the property that subsets are mutually exclusive and collectively exhaustive and that the subsetting process ultimately yields a subset with just one net in it, whose length is the lower bound.

The algorithm computes an initial ordering for the points, and it labels them one through N. The initial ordering will be described in the next paragraph; this one describes the subsets and bounds used by the algorithm. The subsets are defined by the relative order of the first K points in the net, and the lower bounds are computed directly from the triangle inequality. The depth of a subset is defined to be the number of points considered (K). A convenient label for a depth K subset is its K point order vector. For example, the depth three subset 1-3-2 is the set of all nets for which point three is between points one and two. If N is greater than three, 1-3-2 can be partitioned into the four depth four subsets: 4-1-3-2, 1-4-3-2, 1-3-4-2, and 1-3-2-4. For the net sort problem, the lower bound for the set 1-3-2 is $\mathbf{D}[1, 3] + \mathbf{D}[3, 2]$. For the TSP the lower bound is $\mathbf{D}[1, 3] + \mathbf{D}[3, 2] + \mathbf{D}[2, 1]$. The triangle inequality guarantees that no net in the subset is shorter than the lower bound. Moreover a depth N subset contains just one net whose length is the lower bound.

The algorithm consists of recursively partitioning depth K subsets into depth $K + 1$ subsets until either the lower bound exceeds the current minimum or a new current minimum is found. The speed of the algorithm is found to be extremely data sensitive, and an initial ordering of the points is required. In order to maximize the lower bounds of depth K subsets, and hence tend to eliminate subsets without having to partition them, the first K points should be chosen to be maximally separated. This is accomplished with the following presort algorithm. For the net sort problem, if both end points are specified, they are labeled one and two. If one end point is specified, it is labeled one, and point two is the one farthest from it. Otherwise, and for the TSP, the first two points are chosen as the ones farthest apart. Point $P[J + 1]$ is chosen as a point not already chosen which is the farthest distance from all of the J points already chosen. That is, point $P[J + 1]$ is the point K such that: $\text{MIN}(\mathbf{D}[P[I], K]: I = 1, \ldots, J)$ is maximal for $1 \leqslant K \leqslant N$ and K not in $P[S]: S = 1, \ldots, J$.

This initial ordering is a N squared operation and significantly enhances the execution speed of the algorithm.

The branch and bound algorithm is applied to the presorted points and is as follows. Start with the set of all nets to be considered, the lower bound for the set, and a current minimum of infinity. From the set's lower bound, compute lower bounds for each one deeper subset. Record the two best (smallest) lower bounds. If the best lower bound is not less than the current minimum, discard the entire set because it cannot contain a net shorter than the current minimum. Otherwise if the subset depth is N, record the single net in the best subset and set the current minimum to be the subset's lower bound (length); and the set has been explored. Otherwise if the best subset's lower bound is less than the current minimum, explore it. Having explored the best subset, if the second best subset's lower bound is not less than the current minimum, the entire set has been explored. Otherwise, successively explore each subset whose lower bound is less than the current minimum. The algorithm terminates when the initial set has been explored. At this time, the current minimum is the length of an absolute minimum length net, and the recorded net is one of the absolute minimum length nets.

IV. The Computer Realization of the Branch and Bound Algorithm

The recursive algorithm is easily implemented as a recursive subroutine (SUBSET) whose arguments are as follows:

LP1	The depth (L + 1) of the resulting one deeper subsets
LEN	The lower bound of the depth L subset to be explored
D	Row LP1 of the distance matrix

The subset being explored is globally defined by the singly linked list LINK. For the TSP, the initial set to be explored is 1-2-3 which is defined as a circular list (LINK[1] = 2; LINK[2] = 3; LINK[3] = 1). Other global variables used by SUBSET are as follows:

N	The number of points in a net
MIN	The current minimum
WIN	The linked representation of the current minimum net.

The increase in lower bound of the depth LP1 subset formed by placing point LP1 between points I and LINK[I] is

$$D[I, LP1] + D[LP1, LINK[I]] - D[I,LINK[I]]$$

To facilitate the recursive computation of lower bounds in the net sort problem, the distance matrix is augmented with a column of zeros so that D[J, 0] = 0 for all J. For the net sort problem, the initial set to be explored is 1-2 which is defined by

$$LINK[0] = 2; LINK[2] = 1; LINK[1] = 0$$

Notice that the lower bound equation is now also valid for end points of the net. The valid indices (I) for the net sort problem with S preselected end points are S, . . . , L. The valid indices for the TSP are 1, . . . , L.

For a symmetric distance matrix, only one row of D is required to compute the increase in lower bound provided that D[I,LINK[I]] is maintained. DL[I] is defined to be D[I,LINK[I]]. Now the lower bound equation becomes

$$D[I] + D[LINK[I]] - DL[I]$$

where D[J] is D[LP1, J]; i.e., D is now just a row of the distance matrix.

To explore the depth L + 1 subset formed by placing L + 1 between I and LINK[I], the required updates to LINK and DL are

```
LK = LINK[I]
LINK[I] = LP1
LINK[LP1] = LK
DT = DL[I]
DL[I] = D[I]
DL[LP1] = D[LK]
```

To restore DL and LINK to the depth L subset,

```
DL[I] = DT
LINK[I] = LK
```

The C language realization of this algorithm is given in Fig. 1.

V. Example of Algorithm Execution

Consider the five-point net with its corresponding distance matrix in Fig. 2. The problem to solve is the net sort with no preselected points. The initial ordering algorithm gives the points labeled as shown. The initial set 2-1 is partitioned and lower bounds are calculated until a new current minimum of 7 is generated from 5-2-3-1-4. Sets 2-3-1-4 and 2-3-1 both have second best lower bounds not greater than the current

minimum, so they have been explored. The second best subset of 2-1 yields a new current minimum of 6 for 3-2-5-4-1. The rest of the subsets fail the second best test and the algorithm terminates. In this example, only twenty-one lower bounds were computed to determine the minimum of sixty (5!/2) possible nets.

VI. Execution Times

Table 1 gives average and maximum execution times for randomly selected planar points using the Euclidean metric for the TSP. These measurements were made on a VAX 11/750 running UNIX.

References

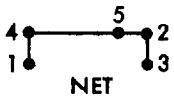
1. Bellmore, M. and Malone, J. C., Pathology of Traveling-Salesman Subtour Elimination Algorithms. *Oper. Res.* 19(1971), 278–307.
2. Bently, J. L., A Case Study in Applied Algorithm Design. *IEEE Computer* 17:2, Feb. 1984, 75–87.
3. Lushbaugh, W. A., Quicklist – The Basis for a Computer Aided Design System. *DSN Progress Report 42-58*, 67–71, May-June 1980, Jet Propulsion Laboratory, Pasadena, California.
4. Smith, D. R., Random Trees and the Branch and Bound Procedures. *J. ACM 31* (Jan. 1984) 163–188.

Table 1. Execution times

Number of Points	Number of Cases	Average, s	Maximum, s
10	100	0.01	0.07
15	100	0.05	0.23
20	100	0.42	3.87
25	100	3.13	33.55
30	25	55.36	400.37
35	14	263.63	1786.93

```
subset(lp1,len,d)
int lp1,len,*d;
{
        int dd[SIZ1],i,lk,dt,lim,lim2;
        for(lim=lim2=30000000,lk=s;lk<lp1;lk++)
                if((dd[lk]=d[lk]+d[link[lk]]-dl[lk])<lim2)
                        if(dd[lk]<lim){
                                i=lk;
                                lim2=lim;
                                lim= dd[lk];
                        }
                        else
                                lim2=dd[lk];
        if((lim+=len)>=min)
                return;
        lk=link[i];
        link[i]=lp1;
        link[lp1]=lk;
        if(lp1==n){
                for(dt=0;dt<=n;dt++)
                        win[dt]=link[dt];
                link[i]=lk;
                min=lim;
                return;
        }
        dt=dl[i];
        dl[i]=d[i];
        dl[lp1]=d[lk];
        subset(lp1+1,lim,d+n+1);
        dl[i]=dt;
        link[i]=lk;
        if((lim=min-len)<=lim2)
                return;
        for(dd[i]=30000000,i=s;i<lp1;i++)
                if(dd[i]<lim){
                        lk=link[i];
                        link[i]=lp1;
                        link[lp1]=lk;
                        dt=dl[i];
                        dl[i]=d[i];
                        dl[lp1]=d[lk];
                        subset(lp1+1,len+dd[i],d+n+1);
                        dl[i]=dt;
                        link[i]=lk;
                }
}
```

Fig. 1. The C Language realization of branch and bound algorithm



NET

	0	1	2	3	4	5
1	0	0	5	4	1	4
2	0	5	0	1	4	1
3	0	4	1	0	5	2
4	0	1	4	5	0	3
5	0	4	1	2	3	0

AUGMENTED DISTANCE MATRIX (D)

SUBSET	LOWER BOUND	CURRENT MINIMUM	DELTA LOWER BOUND 0	1	2	3	4
2-1	5		1	4	0		
2-3-1	5		4	1	8	2	
2-3-1-4	6		1	6	2	2	3
5-2-3-1-4	7	7					
3-2-1	6		5	1	0	8	
3-2-4-1	6		2	4	0	2	6
3-2-5-4-1	6	6					

Fig. 2. Example of algorithm execution for a five-point net and its corresponding distance matrix