Advanced Algorithms	Lecture 12: May 9, 1995
Lecturer: Johan Håstad	*Scribe: Bengt Werner*

12.1 The Traveling Salesman Problem

12.1.1 Introduction

The traveling salesman problem (TSP) can be defined as follows: Given N cities where the distance between any two cities is defined as $d_{i,j}$, find the shortest tour that visits every city exactly once.

A special case is when the cities are points in a plane and the distances $d_{i,j}$ is the Euclidian distance between the points.

An optimal solution can be found in time 2^n using dynamic programming and for the Euclidian case there exists an algoritm that has the time complexity $2^{c\sqrt{n}logn}$.

In a paper by David S. Johnson [1] heuristic algorithms with emphasis on local optimization is surveyed.

12.1.2 Tour Construction Heuristics

If we are only interested in a tour close to the optimal one we might use a heuristic algorithm which will run considerably faster. Eight heuristics are described below.

1. Nearest Neighbour: Visit the nearest neighbour that has not yet been visited.
2. Double Minimum spanning tree: Find a minimal spanning tree. Walk every edge twice to form a tour but shortcut revisits to a city.
3. Strip: Divide the cities into $\sqrt{N}$ equal width strips and walk every strip top-to-bottom bottom-to-top alternating.
4. Greedy: Add the shortest edge between two cities that wont create a vertex of with higher degree than two and wont form a cycle with less than N cities.

5. Spacefilling curve: Visit the cities in the order they appear along a spacefilling curve.

6. Nearest insertion: Start with a degenerate tour using the two closest cities. Add the closest non-tour city to the present tour and insert it where it causes the minimum increase in tour length.

7. Farthest insertion: Start with a degenerate tour using the two farthest cities and insert the farthest non-tour city where it will cause the minimum increase in tour length.

8. Christofides: Construct a graph using the minimal spanning tree. Match vertices with an odd degree using minimum length matching and add these edges to the graph. Proceeed as in the double minimum spanning tree following the edges and skipping cities allready visited. Minimal matching can be found in time $O(n^3)$.

A minimal spanning tree can be found in $O(mlogn)$ time or $O(n^2)$ where n is the number of cities and m the number of edges. Probabilistically the minimal spanning tree can be found in time $O(m)$. If we restrict the problem to euclidian distances in a plane there are more efficient methods to compute a MST.

12.1.3 Evaluation

There are a number of criteria how to evaluate TSP algoritms. Theoretically we may compute:

- Worst case behaviour of algorithm compared to optimal tour.
- Expected behaviour for n randomly points in the unit square.

From a practical perspective we may compute the tour for random or given instances of the problem.

Teoretical comparison to optimal

Here is an (incomplete) table of worst case and expected behavior compared to an optimal tour.

Algorithm	Worst case	Expected
Nearest neighbour	$\Theta(logn)$	
Double MST	2	
Strip	Bad	C_1
Greedy	$\Theta(logn)$	
Spacefill	Bad	C_2
Nearest Insert	2	
Farthest Insert		
Christofides	1.5	

Two expected tourlengths

For the unit square with N random points the expected tourlength is $\Theta(\sqrt{N})$. If the distances $d_{i,j}$ are picked from a rectangular distribution from 0 to 1 the expected length of the tour is constant.

Experimental evaluation

Below is the result of running the algorithms with N points being uniformly distributed in the unit square. Results are averages over 10,5,5,5,3,1,1 instances of each size.

The running times in the table is seconds on a Sequent Balance Computer. Asterisks indicate time on a VAX-8550 computer, multiplied by 10 so as to be roughly comparable.

	Number of cities						
Algorithm	10^2	$10^{2.5}$	10^3	$10^{3.5}$	10^4	$10^{4.5}$	10^5
Spacefill	0.2	0.8	2.7	8.7	28.4	92.3	302.3
Strip	0.3	1.0	3.5	11.9	39.7	133.6	445.7
Nearest Neighbour	0.6	2.2	9.8	27.1	110.2	316.8	1031.2
Greedy	1.8	6.2	24.4	78.6	252.8	933.3	2930.3
Double MST	2.4	8.8	32.1	105.7	356.8	1772.0*	9168.0*
Nearest Insert	3.4*	12.2*	44.3*	163.8*	543.6*	2177.5*	10869.3*
Farthest Insert	3.4*	13.8*	56.0*	229.6*	805.5*	3240.8*	14925.3*
Christofides	7.1	151.6	4053.1	108000.0*			

The numbers below is the average percent excess over the Held-Karp lower bound on the optimal tour length. For these instances the lower bound appears to be within

0.8% of the true optimum.

	Number of cities						
Algorithm	10^2	$10^{2.5}$	10^3	$10^{3.5}$	10^4	$10^{4.5}$	10^5
Double MST	36.9	37.7	39.0	39.3	39.5	39.7	39.9
Spacefill	25.4	28.0	31.2	33.4	34.2	34.8	34.9
Strip	22.8	30.4	30.3	30.7	30.3	30.5	30.2
Nearest Insert	23.0	24.4	26.5	26.4	26.5	27.4	27.1
Nearest Neighbour	29.3	26.7	25.1	25.7	24.2	24.2	23.8
Greedy	22.1	18.6	16.9	16.8	16.5	15.5	14.7
Farthest Insert	8.3	10.4	12.5	12.6	13.3	13.2	13.6
Christofides	8.8	8.9	9.9	9.8			

12.1.4 Local Optimization

A tour constructed by the above heuristics might be optimized using local optimization. Five different techniques will be described:

1. 2-opt: for each pair of edges in the tour try to exchange them.
2. 3-opt: delete three edges and put them back again to create a shorter tour.
3. Lin-Kernighan: variable opt(described below).
4. Simulated Annealing. Allow the tour to occasionally become longer. The acceptability of a solution with a higher cost is affected by a parameter t called the temperature. Typically the temperature is gradually reduced from a very high value to a very low.
5. Genetic algoritms. One performs some fixed number of independent local optimization runs and derive new starting solutions based on a transfer of information between the solutions found.

Lin-Kernighan

Consider the change as a switch from edges x_1 $x_2..x_s$ to y_1 y_2 .. y_s where y_i has a vertex common with x_i and $x_i + 1$ and y_s has a vertex common with x_s and x_1 (See fig 12.1).

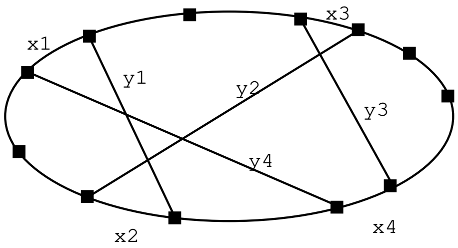

Figure 12.1: Exchanging x's to y's on a tour.

The decision of x_1 y_1 x_2 y_2 ... can be viewed as searching a tree. If the entire tree is searched we will find an optimal solution.

Lin-Kernighan wants to describe a part of the tree as follows;

1. The cost $\sum_{i=1}^{t} c(x_i) - c(y_i)$ should always be positive (which does not limit the algorithm; if there is a global improvement then there is a local improvement).

2. Choose x_t such that if y_t goes to x_1 we will get a tour.

3. Do total backtracking on level 1 and 2, otherwise choose the y_i with the lowest cost or an y_i that completes the tour.

Evaluation of local optimizations

Finally, here is the running time and percentage excess of the Held-Karp lower bound for Local Optimization algorithms. Running time is on a VAX-8550. Starting tours for 2-opt and 3-opt is Nerest Neighbour. Starting tour for Lin-Kernighan is a random tour.

	Running time				Percentage excess			
N:	10^2	10^3	10^4	10^5	10^2	10^3	10^4	10^5
2-opt	0.4	5.0	55.0	746	6.2	6.4	6.4	6.5
3-opt	0.5	6.6	71.5	980	2.6	3.5	3.6	3.6
Lin-Kern	1.1	16.7	340.2	17776	1.5	2.1	2.2	2.2

References

[1] David S. Johnson. Local Optimization and the Traveling Salesman Problem. *Proc of 17'th Colloquium on Automata, Languages & Programming*, 446-461, 1990.