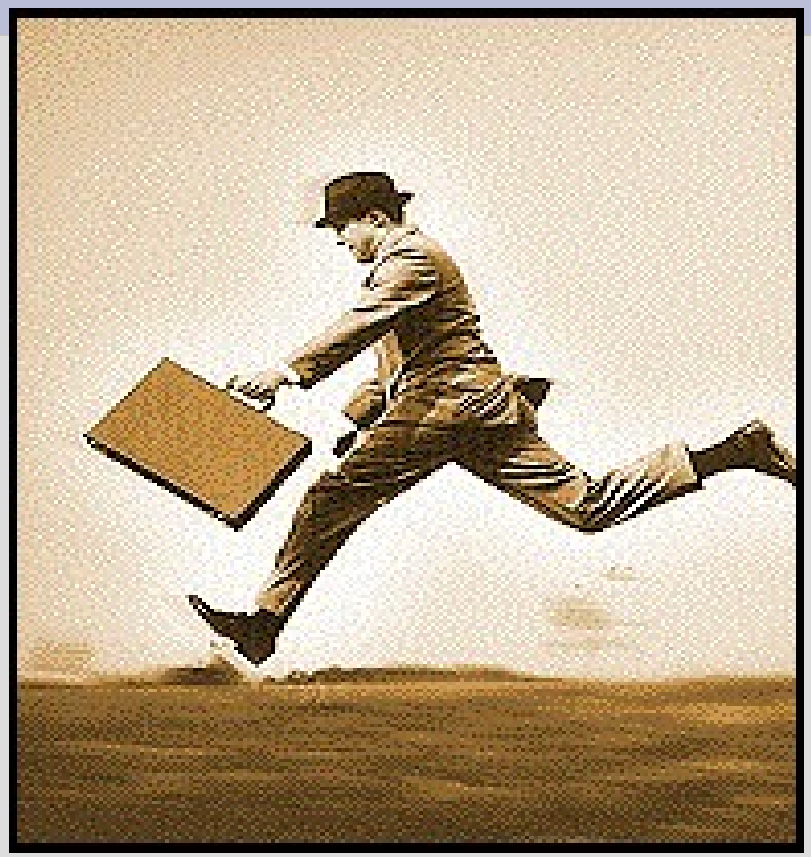
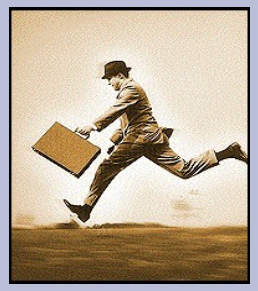
The Travelling Salesman Problem



Brett D. Estrade
estrabd@mailcan.com

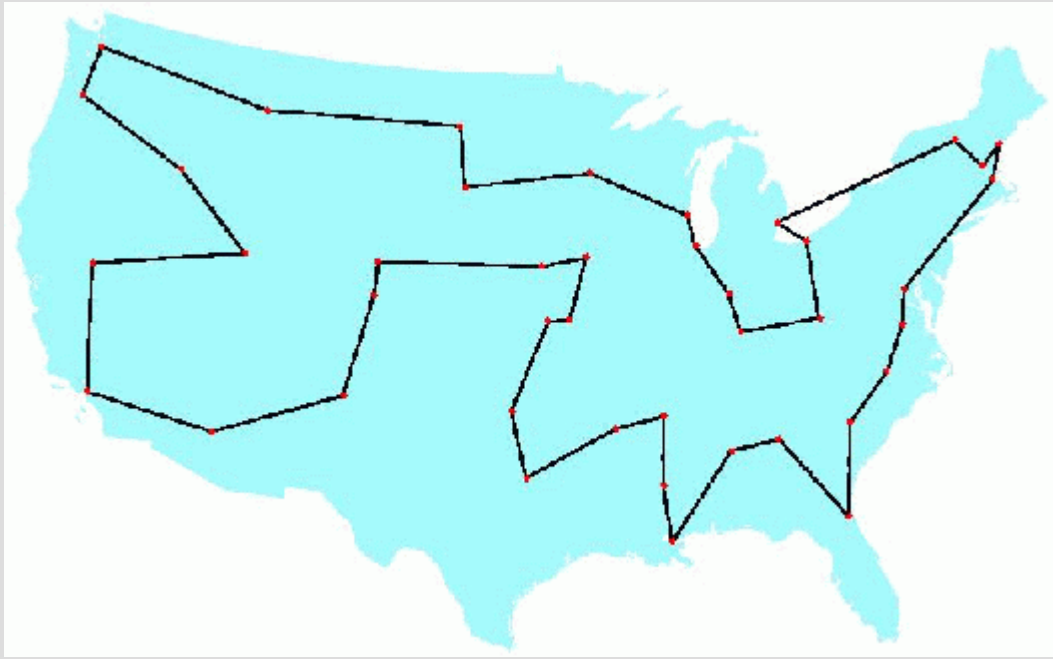
CS616 – Spring 2004



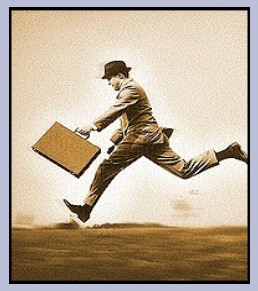
Overview

The goal of the Traveling Salesman Problem (TSP) is to find the "cheapest" tour of a select number of "cities" with the following restrictions:

- You must visit each city once and only once
- You must return to the original starting point



49 city solved by George Dantzig, Ray Fulkerson, and Selmer Johnson (1954)



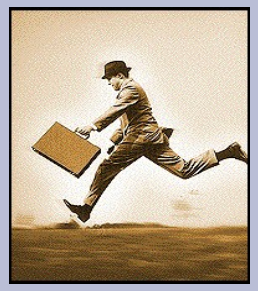
Why Study TSPs?

NP-complete problems are the hardest problems in NP, in the sense that they are the ones most likely not to be in P.

The reason is that if you could find a way to solve an NP-complete problem quickly, then you could use that algorithm to solve all NP problems quickly.

Common NP-complete problems:

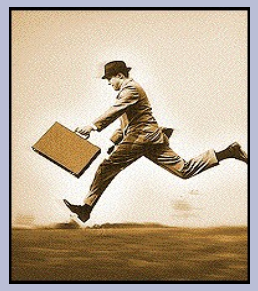
* Boolean satisfiability problem (SAT)
* Fifteen puzzle
* Knapsack problem
* Minesweeper
* Tetris
* Hamiltonian cycle problem
* Traveling salesman problem
* Subgraph isomorphism problem
* Subset sum problem
* Clique problem
* Vertex cover problem
* Independent Set problem



Hamiltonian Circuit

TSP is similar to these variations of Hamiltonian Circuit problems:

- Find the shortest Hamiltonian cycle in a weighted graph.
- Find the Hamiltonian cycle in a weighted graph with the minimal length of the longest edge. (bottleneck TSP).

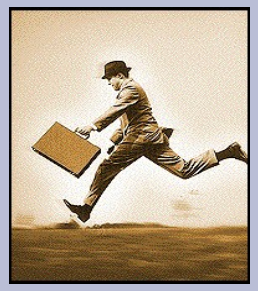


Brief History

In the 1800s, similar problems were looked at by Sir William Rowan Hamilton and Thomas Penyngton Kirkman.

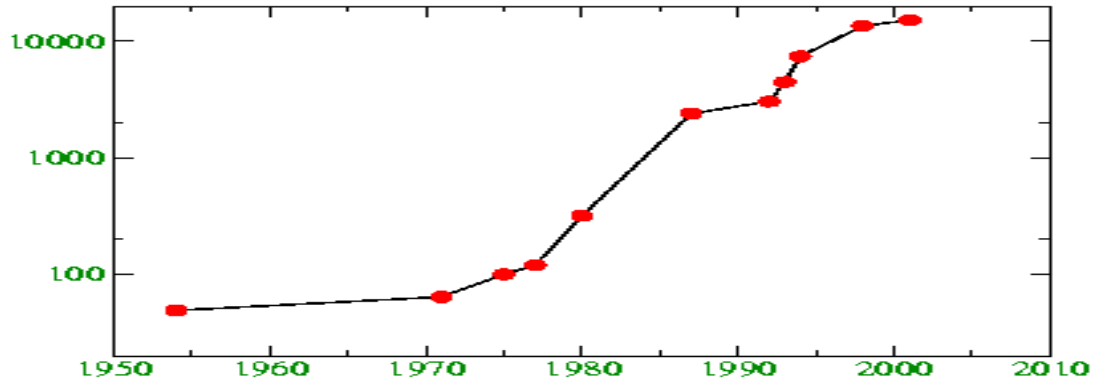
In the 1930s, Karl Menger began looking at the general form of TSP by Hassler Whitney and Merrill Flood at Princeton.

A photograph of Hamilton's Icosian Game that requires players to complete tours through the 20 points using only the specified connections.



Milestones

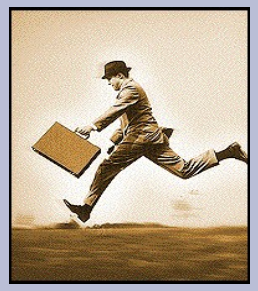
Milestones in the Solution of the TSP



Year	Cities	Team
1954	49	Dantzig, Fulkerson, and Johnson
1971	54	Held and Karp
1975	100	Camerini, Fratta, and Maffioli
1977	120	Groetschel
1980	318	Crowder and Padberg
1987	2,392	Padberg and Rinaldi
1992	3,038	Concorde
1993	4,461	Concorde
1994	7,397	Concorde
1998	13,509	Concorde
2001	15,112	Concorde

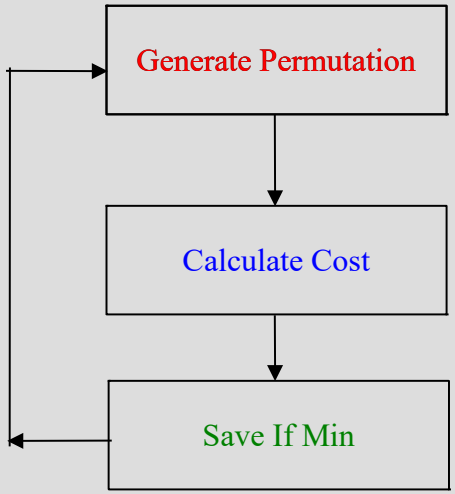
Factor of 126

2025: 1.9 million?



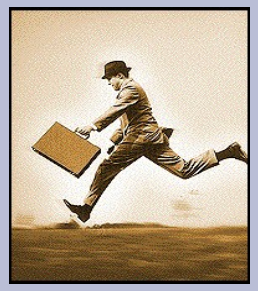
Brute force Method

- Description
  - Is written in Perl
  - Tries every possible path
  - Has a complexity of O(n!)




```
#!/usr/bin/perl -w
use strict;
# Brett D. Estrade
# CSC 616 - Automata Final Project

my @cities = (1...11);
my @combos = ();
my @dist_matrix = ();

my $MIN_DISTANCE = 0.0;
my @MIN_PATH = ();

#
#  Build look up table
#
while (<DATA>) {
  my @line = split('  ',$_);
  push (@dist_matrix,[@line]);
}

sub compute_cost {
  my (@path) = @_;
  my $path = \@path;
  my $sum = 0.0;
  for (my $i = 0; $i <= $#path; $i++) {
     my $y = 0;
     if ($i == $#path) {
       $y = $path[0]-1; # start
     } else {
       $y = $path[$i+1]-1;
     }
     my $x = $path[$i]-1;
     $sum += ($dist_matrix[$x]->[$y]);
  }
  if ($sum < $MIN_DISTANCE || $MIN_DISTANCE
+ == 0) {
    $MIN_DISTANCE = $sum;
    @MIN_PATH = @path;
  }
}
```

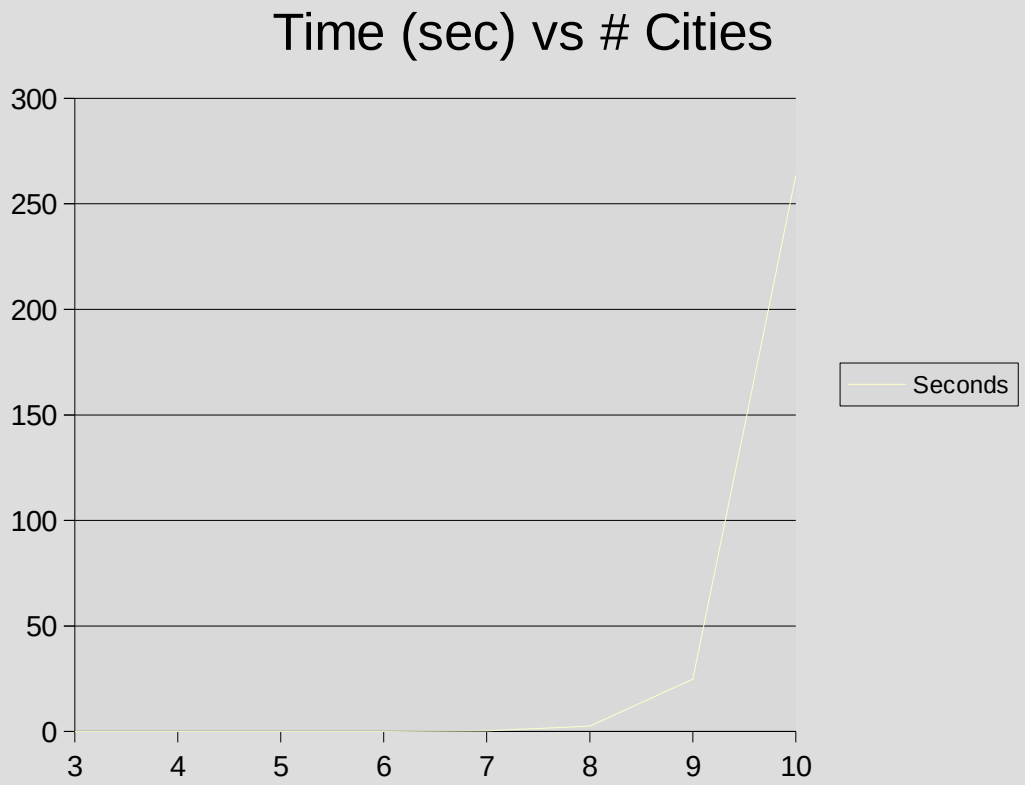
```
#
# Create all combinations - recursive
#http://www.perlmonks.com/index.pl?node=618
sub permute {
    my @items = @{ $_[0] };
    my @perms = @{ $_[1] };
    unless (@items) {
        #print "@perms\n";
        compute_cost(@perms);
    } else {
        my(@newitems,@newperms,$i);
        foreach $i (0 .. $#items) {
            @newitems = @items;
            @newperms = @perms;
            unshift(@newperms,
splice(@newitems, $i, 1));
            permute([@newitems], [@newperms]);
        }
    }
}
permute(\@cities, []);

print "The shortest distance is: $MIN_DISTANCE\
n";
my $pathStr = join(" to ",@MIN_PATH);
```



Benchmarks

Platform Specs:
- CPU: 2.4ghz Xeon
- 4 GB RAM

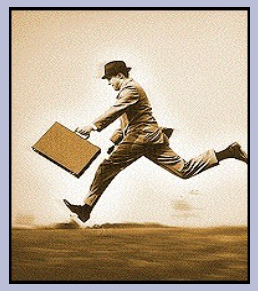
- 3 – 6
- 4 – 24
- 5 – 120
- 6 – 1440
- 7 – 5040
- 8 – 40320
- 9 – 362880
- 10 – 3628800
- 11 – 39916800
- 12 – 479001600



Avg time per cost sum = 6.79 x 10^-5 sec*
* Increases nominally as more cities are added

This means that 11 cities is projected
to take 2710.35 seconds – that is 45 minutes.

12 cities is projected to take 9.03 hours!



Heuristic Approaches

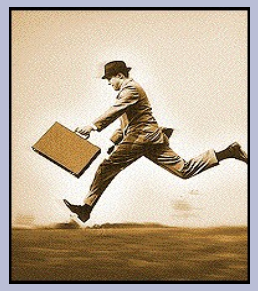
By using approaches based on heuristics, practical applications of TSP can be solved since the optimal solution is not always needed. By using heuristics, sub-optimal solutions can be found, and often times just having a solution is “good enough”.

- Random Optimizations
  - Optimised Markov chain algorithms which utilise local searching heuristically sub-algorithms can find a route extremely close to the optimal route for 700-800 cities.

- Local Search
  - Start off with a valid tour, then use local moves to improve the tour.
  - Terminates at a ``local minimum.''

- Nearest Neighbor
  - Start at a node, and selected the nearest unvisited node; repeat until done. Worst case creates a complexity of **½ log n**.

- Greedy Algorithm
  - Start with empty partial tour. Add smallest edge that results in a valid partial tour. Repeat until complete tour reached. Worst case creates a complexity of **½ log n**.

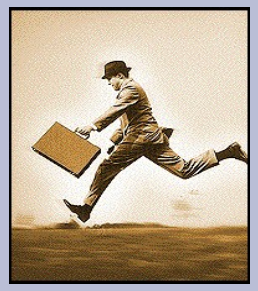


Exact Solutions

These algorithms find the exact solutions:

- Cutting Plane Algorithm
  - Technique based on linear programming
  - An exact solution for 15,112 Germany cities from TSPLIB was found in 2001 using this method proposed by George Dantzig, Ray Fulkerson, and Selmer Johnson in 1954.

- Branch and Bound Algorithms
  - Branching recursively divides the domain into feasible sub domains.
  - Bounding determines upper and lower bounds for the optimal solution in a feasible sub domain.
  - Can be used to process TSPs containing 40-60 cities.

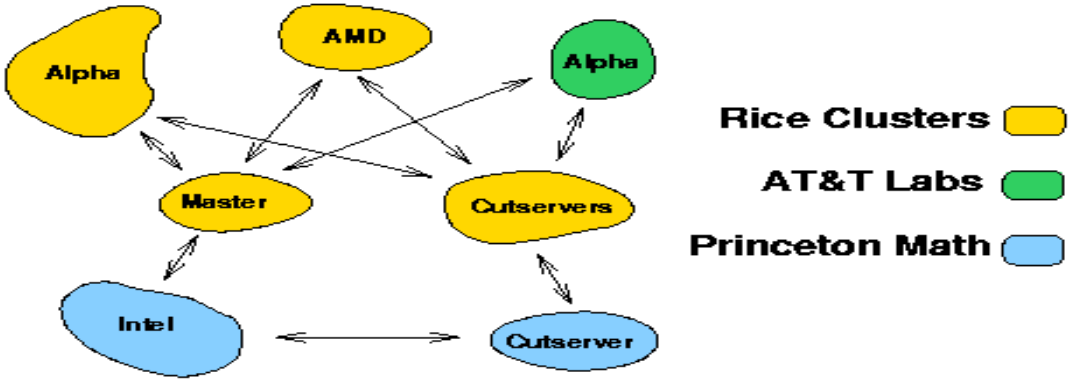


15,112 Cities!

Distributed Computing

Solution of d15112

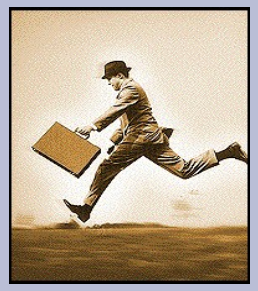
22.6 CPU Years (Alpha 500 MHz, EV6)
110 Distributed Processors



Quadratic Assignment Problems (QAP)
Anstreicher, Brixius, Goux, and Linderoth

Whizzkids'96 Vehicle Routing Problem
Applegate, C., Dash, and Rohe

General Branch–and–Cut Frameworks
Eckstein, Phillips, and Hart (2001)
Ralphs and Ladanyi (2001)



Special Cases

Restricted locations

• A trivial special case is when all cities are located on the perimeter of a convex polygon.

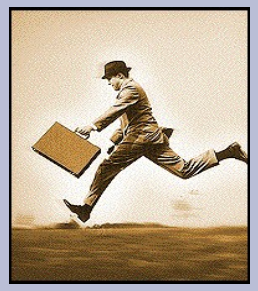
• A good exercise in combinatorial algorithms is to solve the TSP for a set of cities located along two concentric circles.

TSP With Triangular Inequality

•That is, for any 3 cities A, B and C, the distance between A and C must be at most the distance from A to B plus the distance from B to C. Most of natural instances of TSP satisfy this constraint.

Euclidean TSP

•Is a TSP with the distance being the ordinary Euclidean distance. The problem still remains NP-hard, however many heuristics work better.



Neural Nets

A neural net is a mathematical model for information processing based on a connectionist approach to computation.

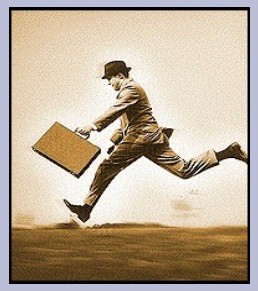
The original inspiration for the technique was from examination of bioelectrical networks in the brain formed by neurons and their synapses.

In a neural network model, simple nodes (or "neurons", or "units") are connected together to form a network of nodes

As applied to TSP:

Recently, several algorithms have relied on the promising Self-Organizing feature Map (SOM) which is a method for unsupervised learning based on a grid of artificial neurons. The map is not a map of the cities, but a grid of the artificial neurons, also referred to as a Kohonen map.

There are 2 phases – the training phase and the mapping phase. During the learning phase, the map is built using a competitive process. During the mapping phase, an input vector is quickly classified where there is a single winning neuron depending on what neuron's weight vector lies closest to the input vector.



Genetic Algorithms

A genetic algorithm (GA) is an algorithm used to find approximate solutions to difficult-to-solve problems through application of the principles of evolutionary biology to computer science.

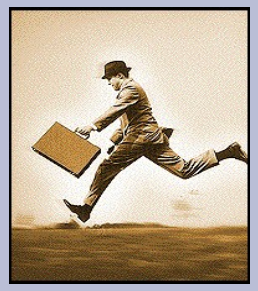
Genetic algorithms use biologically-derived techniques such as inheritance, mutation, natural selection, and recombination.

As applied to TSP:

The optimal solution will be the fittest individual of the final generation, being the product of many cycles of selection, reproduction, and even mutation.

And keep in mind that GA solutions are not necessarily the very best ones. However, they can be quite useful, and provide usable results.

One such algorithm that has enjoyed some success is the “Ant Algorithm”



The Ant Algorithm

The Ant Algorithm is a program that attempts to emulate the way that a colony of ants optimize the path between their nest and a food source.

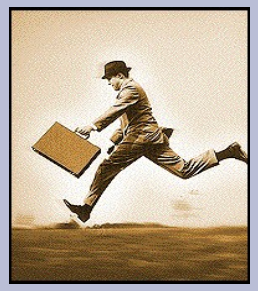
An ant is treated as a single agent among a colony of ants that follows a basic set of rules about how it is to traverse the graph of nodes. It also maintains a list of what nodes it has already visited so that the length of the trip can be extracted. As each ant travels, it deposits a “scent” or “pheromone” that tells other ants that it has been visited.

The initial population of ants is even distributed to different points among the graph. Movement of the ant is guided by a simple probability equation that takes into account the intensity of the pheromone scent that is on each node.

As time passes, the scent on each node begins to evaporate; therefore the most commonly traversed nodes maintain the scent while the least traveled nodes eventual lost their scent all together.

Eventually a single path emerges. A solution can be submitted once the emerging path continues to result after repeated trials.

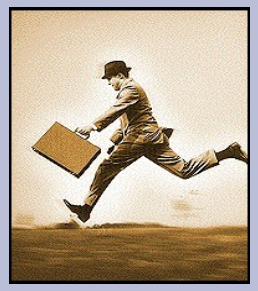
The ant algorithm can be applied to a variety of other problems. These include Quadratic Assignment Problems (QAP) and Job-shop Scheduling Problems (JSP).



Conclusion

The Traveling Salesman Problem has a long history and a strong tradition in academics. The continued study of this problem coupled with novel and creative approaches may some day yield a method that will lead to a polynomial-time solution for all NP-complete problems.

In the meantime, the study of this problem has yielded solutions that are “good enough” for practical application – so good that often the search for an optimal solution is not even worth the effort.



Sources

http://www.math.princeton.edu/tsp/

http://www.wikipedia.org

http://www.cs.duke.edu/

http://scef.unime.it/plebe/pubs/ICANN2002.ps.gz

Al Application Programming
by M. Tim Jones ISBN:1584502789