The Traveling Salesman (& $\mathcal{J}$)

Peter G. Anderson,

Computer Science Department

Rochester Institute of Technology,

Rochester, New York

anderson@cs.rit.edu  http://www.cs.rit.edu/

June 11, 2003
Abstract

The traveling salesman problem (TSP) and its variations and disguises.

Also, introducing the J programming language, showing that it is highly suitable for GA programming, and demonstrating an ordered greed implementation for the traveling salesman.
TSP

Given a symmetric $N \times N$ matrix of positive values (costs), $C$. (This is an edge weighted complete graph.)

Find the permutation, $p$, of $\{0, 1, 2, \cdots, N - 1\}$ so that the following cost is minimized:

$$cost(p) = \sum_{i=0}^{N-1} c_{p(i), p(i+1)}$$

This problem is NP-hard.

The following problem is NP-complete:

Is there a $p$ such that $cost(p) < C$?
TSP Restrictions—Still NP Hard/Complete

Require $C$ to be a metric

1. $c_{ij} \geq 0, \forall i, j$

2. $c_{ij} = 0 \Rightarrow i = j$

3. $c_{ij} + c_{jk} \leq c_{ik}, \forall i, j, k$ (triangle inequality)
TSP Restrictions—Still NP Hard/Complete

Use Euclidean distance for $N$ points in the plane or higher dimensional space.
An Approximation to An Optimal Path ($\leq 100\%$ Error)

Visit the cities along the edges of the minimum spanning tree (MST).

Vertices with degree $d$ in the MST will be visited $d$ times in this circuit.

Algorithm: Remove the offending multiple visits to nodes $q$. The path $p$ to $q$ to $r$ can be replaced by a shorter step $p$ to $r$

Can this be done by an analogy to ordered greed.
Kruskal’s Efficient MST Algorithms

Sort the graph’s edges by cost (use a heap). Cost = $O(N^\varepsilon \log N^\varepsilon)$

Build up a forest (a graph without cycles) using the shortest edge that does not form a circuit.
Prim’s Efficient MST Algorithms

Start the tree, $T$, with an arbitrary vertex.

Sort the edges that touch $T$ by length (use a heap).

Add the shortest to $T$ that does not form a cycle.
Keyboard Design—TSP In Disguise

Goal: to speed up typing—improve over QWERTY and Dvorak.

Suppose $F_{ab}$ is the frequency of the letter pair $a - b$. And $C_{pq}$ is the cost of hitting keys in order $p - q$.

Find an optimum assignment, $\lambda$, of letter to keys. Minimize

$$\sum_{a,b} F_{ab} C_{\lambda(a),\lambda(b)}$$
Wallpaper Cutting—TSP In Disguise
Multi-part Forms Printing—TSP In Disguise
Job Shop Scheduling—TSP In Disguise
**Croes’s \((k = 2)\) and Lin’s \((k = 3)\) Algorithm**

Start with a circuit.

Iteratively, replace \(k\) edges in the circuit.

When \(k = 2\), the greedy algorithm will remove crossing edges.

Idea: combine this with simulated annealing. Accept moves early in the process that degrade the fitness.
Elastic Net Inspired By Kohonen’s Self-Organizing NN
Hitachi Crossover

Form a child from two parents by growing the path from city 0.

Parent 1 has $0 \rightarrow a$. Parent 2 has $0 \rightarrow b$.
So child gets its start as $a, 0, b$.

Parent 1 has $a \rightarrow c$. Parent 2 has $b \rightarrow d$.
So child grows as $c, a, 0, b, d$.

Continue this process as long as possible, then fill in remainder randomly.
Traveling Salesman

This is another type of experiment.

I use my favorite programming language: $\mathcal{J}$, a new dialect of APL.
Traveling Salesman

Given $N$ cities, determine the shortest circuit.

$NP$-complete problem!

The greedy heuristic:
- Given a permutation of cities
- Build a circuit with the first three cities
- Add each city to the circuit least expensively

This depends on an ordering of the cities
- What are the first three?
- What order are the cities entered?

Use ordered greed!
The Approach

Gain more experience with the $J$ language.

Maintain a population matrix: each row is a permutation.

Implement OX.

Try a novel generation-making strategy.
Generation Making

Sort the population by fitness.

Kill off the worst half ... “on the average.”

Duplicate and scramble the population.

Circularly adjacent individuals create a child.

Slightly mutate the new population.
Test Plan

$N$ randomly chosen cities.

$N \times M$ cities in a grid ($N \cdot M$ must be even).

48 state capitals.
Lists

x =. i.10
x
0 1 2 3 4 5 6 7 8 9
y =. 10?10
y
8 2 4 3 7 5 1 0 9 6
x+y
8 3 6 6 11 10 7 7 17 15
x<y
1 1 1 0 1 0 0 0 1 0
x=y
0 0 0 1 0 1 0 0 0 0
+/x=y
2
### Tables

\[ z = 5 \ 5 \ \$ \ 2 \]

\[
\begin{array}{cccccc}
2 & 2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 & 2 \\
\end{array}
\]

\[
\begin{array}{cccccc}
?z \\
1 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccc}
?z \\
1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 \\
\end{array}
\]
\[ J \] Functions

\[
\begin{align*}
\text{plus} &=. + \\
5 \text{ plus } 6 &= 11 \\
\text{add\_one} &=. +&1 \\
\text{add\_one \ 17} &= 18 \\
5^2 &= 25 \\
\text{square} &=. \ ^&2 \\
\text{square \ 6} &= 36
\end{align*}
\]
Idiom: Tilde

\[ + \sim 5 \]
\begin{align*}
  10 & \sim 5 \\
  0 & \sim 5 \\
  25 & \sim 1 2 3 4 5 \\
  1 2 3 4 5 1 2 3 4 5 & \sim 5 \\
  3125 & 5 \sim 20 \\
  15 & 5 \% \sim 20 \\
  4 &
\end{align*}
J Idiom: Tilde

5 ? 5 NB. ? is deal
4 0 1 3 2
?~ 5
4 1 0 2 3
] x =. ?~ 5
2 0 4 1 3
 x {~ 2 NB. { is the subscript operator
4
Idiom: Masking

] list =. ?~ 10
8 2 4 3 7 5 1 0 9 6
<./list
0
list = <./list
0 0 0 0 0 0 0 1 0 0
i. 10
0 1 2 3 4 5 6 7 8 9
(list = <./list) # i. 10
7
# list
10
(list = <./list) # i. # list
7
My $J$ OG GA Program for TSP

The $J$ program determines a collection of $N$ cities' coordinates and their pairwise Euclidean differences.

It tries to create a short circuit.

It uses the heuristic:
- Build a circuit with the first three cities.
- Add each city to the circuit least expensively.

This depends on an ordering of the cities
- what are the first three?
- what order are the cities entered?
Program Components

Define several utility functions.

Define some GA functions.

Create cities list and distance matrix.

Run the GA with specified population size and number of generations.
A Few Utility Functions In $J$

Convert internal binary to printable ASCII:

\[
\text{fmt} =. ":\n\]

Print to standard output.

\[
\text{print} =. 1!:2 \& 2
\]

We can print inside functions using:

\[
\text{print someValues}
\]
A Few Utility Functions In $\mathcal{J}$

Write an array to a named file:

\[
\text{WRITE} =: 4 : 0
\]

NB. $x$. is an array to be written.
NB. $y$. is the file name.

\[
\text{linefeed} =. 10 \{ a.
\]
\[
(, x. ,. \text{linefeed}) 1! : 2 < y.
\]

Call this function using:

\[
(\text{fmt values_array}) \text{WRITE 'my_file'}
\]
Some TSP Functions

Creation of cities (here are two ways to do it).

30 random cities using the coordinates 0..59:
   cities =. (N,2) \$ ?~ 2*N =. 30

$N^2$ cities on an $N \times N$ grid:
   cities =. ,/,"0/^/~ i. 10
   N =. #cities

Length of a vector is the square-root of sum of squares:
   norm =. %: @: +/ @: *:

Create the distance matrix, D:
   D =. norm "1 -"1 1/^/~ cities
Functions to Support the Heuristic

How much does it cost to insert city \( a \) into a circuit between cities \( a \) and \( b \)?

\[
delta = \begin{array}{c} 4 : 0 \\ a = (0 \{ x. ) \\ b = (1 \{ x. ) \\ c = y. \\ (a \{ c \{ D) + (b \{ c \{ D) - (a \{ b \{ D) \\ \end{array}
\]

Determine the best position to insert a new city into the circuit:

\[
\text{best_pos} = \begin{array}{c} 3 : '0 \{ (y. = <./y.) \# i. \# y.' \\end{array}
\]
The Fitness Function

Convert an ordered list of cities into a path.

\[
\text{evaluate} = 3 : \\
\text{path} = 3 \cdot y. \quad \text{NB. head} \\
L = 3 \cdot y. \quad \text{NB. behead} \\
\]

while. 0 < #L do.
\[
c = \cdot L \\
\text{best} = \text{best_pos} (\text{path} , \cdot \_1|\cdot \text{path}) \text{ delta } "1 0 \cdot c \\
\text{path} = (\text{best} \cdot \text{path}), c, (\text{best} \cdot \text{path}) \\
L = \cdot L \\
\]
end.

\[
\text{path} \\
\]
\)
Here is a $J$ session:

```
] path =. ?~ 10
3 9 0 1 2 4 6 8 7 5
_1 |. path
5 3 9 0 1 2 4 6 8 7
```

```plaintext
path ,. _1 |. path
3 5
9 3
0 9
1 0
2 1
4 2
6 4
8 6
7 8
5 7
```
Note: Another Idiom

(path ,. _1| path) delta "1 0 c =. { . L

path is a \( N - k \times 2 \) array.

L is the list of \( k \) unprocessed cities

c is the first element of L

delta is a function that takes two arguments:

    list delta node

list is a short list of two nodes
delta gets the cost of inserting node in the middle of list.
array delta "1 0 c

determines the vector of $N - k$ costs.
"1 0 " denotes that delta applies to the 1-cells on the left and the 0-cells on the right.
The Length of a Path

\[
\text{length} = 3 : 0
+ (0 \{"1 s\} \{"0 1 (1 \{"1 s = . y. , . 1 | . y.) \{ D\}
\]
Ordered Crossover: “OX”

Create a child permutation whose prefix (of a randomly chosen length) is taken from one parent, and the remainder according to the ordering in the other parent.

\[
\text{OX} = 4 : 0
\]
\[
\text{NB. first, choose a random cut point}
\]
\[
pos = 1 + ?1 \sim N
\]
\[
\text{rest} = \text{pos} \}. \text{x}.
\]
\[
(\text{pos} \}. \text{x}., (((-1 ((\text{y} \ i. \ \text{rest}) \}) \text{y}.)) = _1)) \# y.)
\]
The GA Program

evolve =. 4 : 0
    P =. x.      NB. population size
    pop =. ?~ "0 P#N NB. Create population

    NB. Compute all fitnesses, and print best 10
    fitness =. length "1 evaluate "1 pop
    print 10 {. /:~ fitness
    'path_0' WRITE~ ": (, (0&{)) cities {~ evaluate 0 { pop
    i =. 0

continued on next page...
The GA Program, cont.

while. y. > 0 do.
    pop =. pop /: fitness NB. Sort by fitness
    i =. 1 + i

    f_name =. 'path_', "i
    f_name WRITE~ "": (, (O&{})) cities {~ evaluate 0 { pop

NB. Delete worst half; replicate; shuffle
pop =. (?~P) /:~ ,~ (-:P) {. pop

NB. Crossover
pop =. pop OX "1 1 (1 |. pop)
NB. Compute all fitnesses, and print best 10
fitness =. length "1 evaluate "1 pop
print 8.2 ": 10 {. /:~ fitness
y. =. y. - 1
end.
)

NB. Call the evolve function.
60 evolve 60
Apply the permutation-making function to the 0-cells:

NB. Create population
pop =. ?~ "0 P#N

Sort the elements of pop:

NB. Sort by fitness
pop =. pop /: fitness

Note the function composition:

NB. Shuffle
pop =. (?~P) /:~ ,~ (-:P) {. pop
fmt =. "": NB. Convert to printable ASCII
print =. 1!:2 & 2
WRITE =. 3 : 0 :
  NB. x. is an array to be written.
  NB. y. is the file name character string.
  linefeed =. 10 { a.
  (, x. ,. linefeed) 1!:2 < y.
)

best_pos =. 3 : '0 { (y. = <./y.) # i. # y.'

cities =. ,/"0/~i. 10
cities =. (N,2) $ ?~ 2*N =. 100
] N =. #cities

'cities' WRITE~ "": cities

norm =. %: @: +/ @: *: NB. Length of a vector

D =. norm "1 -"1 1/~ cities NB. Distance matrix

delta =. 3 : 0
  :
  a =. (0 { x.) [ b =. (1 { x.) [ c =. y.
  (a { c { D) + (b { c { D) - (a { b { D)
  )
The Code

eval =. 3 : 0
    path =. 3 {. y.
    L =. 3 }. y.
    while. 0 < #L do.

        c =. {. L
        best =. best_pos (path ,. _1|. path) delta "1 0 c
        path =. (best {. path), c, (best }. path)

        L =. }. L
    end.
    path
)
length =. 3 : 0
   '+/ (0 {"1 s) {"0 1 (1 {"1 s =. y. ,. 1 |. y.) { D'
)

OX =. 3 : 0 NB. Ordered crossover :
   pos =. 1 + ?1 -~ N NB. random cut point
   rest =. pos }. x.
   (pos {. x.), (((_1 ((y. i. rest) }y.) = _1) # y.)
)
evolve =. 3 : 0

: 
  P =. x. NB. population size 
  pop =. ?. "0 P#N NB. Create population 

  print 10 {. /:~ fit =. length "1 eval "1 pop 
  'path_0' WRITE~ ": (, (0&{)) cities {~ eval 0 { pop 
  x =. 0

while. y. > 0 do.
  pop =. pop /: fit NB. Sort by fitness 
  x =. 1 + x 
  ('path_', ":x) WRITE~ ":(, (0&{)) cities {~ eval 0{pop
NB. Delete worst half; replicate; shuffle
pop =. (?~P) /:~ ,~ (=:P) {. pop

pop =. pop OX "1 1 (1 |. pop) NB. Crossover

print 8.2 ": 10 {. /:~ fit =. length "1 eval "1 pop
y. =. y. - 1
end.
)
60 evolve 60
30 Random Cities

![Diagram of 30 random cities](image-url)
100 Random Cities
100 Cities in a Grid
48 State Capitals
48 State Capitals
48 State Capitals