A4 programming – Andy Xu, Jonathan Chan, Jerry Wang

1. Runtime complexity in Big-O of brute force should be O(n!). Starting with city 1, there are n-1 possibilities in the first traversal. For each n-1 traversal in the first recursion, there are n-2 possibilities for the next recursion. This repeats until n-n=0 as in we have traversed every possibility. The number of total computations is therefore (n-1)! = O(n!).
2. I implemented Held-Karp’s algorithm.

TSP(n)

For k from 1 to n

C(all sets within S = {1, 2…n}, k) = infinity

C({1},1) = 0

For all subsets T ⊆ S

For all elements j ∈ T

For all elements k ∈ S

If(k ∉ T and ( C(T+{k}, k) > C(T,j) + dist(k, j) )

C(T+{k}, k) = C(T,j) + dist(k, j)

predecessor(T+{k}, k) = j

length = infinity

For all elements k ∈ S

If (length > C(S,k) + dist(1,k))

Length = C(S,k) + dist(1,k)

temp = k

Array[numCities+1] = 1

i = numCities

While S is not empty

Array[i] = temp

i = i – 1

temp2 = temp

temp = predecessor(S, temp)

S = S – {temp2}

return length

return Array

This is an implementation of the Held-Karp algorithm. Essentially, I start with the smallest possible sets, calculate the shortest distance for each combination of elements in each set, and work up until the set contains every city. At each step, the algorithm compares an element not in the set “k” and compares it with each element already in the set. If it is a shorter distance to travel to “k” first than an element within the set, the value for the shortest distance is updated. This occurs and works on bigger and bigger sets until all cities are accounted for.

The predecessor array is used to keep track of the routes so that the optimal tour can be returned. On the last iteration, the algorithm goes through all shortest paths and looks at which path is shortest after including the last traversal back to city 1 using “length”. The predecessor array returns the optimal tour in reverse order; the algorithm simply reorders the tour in the right direction into “Array”.

1. Held-Karp has O(2^n \* n^2). At each level of traversal k, every possible set must considered; this is simply the binomial coefficient of (n-1) choose k. The term is n-1 because city 1 is already included. For each set, each element within the set must be compared to every other element within the set. Therefore, the number of operations is k to account for every element multiplied by k-1 in order to compare each element to every other element: k\*(k-1). Together, the total number of operations per traversal level is k\*(k-1)\*((n-1) choose k). All the operations of each level 2 to (n-1) must be included and added together; therefore, the total complexity is

The n\*(n-1) gives the n^2 term while the sigma sum can be see as a series of binomial coefficients which gives the 2^n term. Therefore the complexity is O(2^n \* n^2).

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 | 25 |
| 8 | 47 | 266 | 1428 | 7298 | 36803 | 178081 | 853233 | 4029132 | 18468023 | 85739293 |

1. The plot above looks similar to an exponential graph. For small n the time elapsed is very small but ramps up very quickly as n increases. The data points have a difference of 2 cities per plot. Looking at the complexity O(2^n \* n^2), we can compare the times between two points.

The rational term approaches 1 as n increases. Therefore, each subsequent data point should be a bit more than 4 times the previous data point. This should approach 4x the previous time elapsed as n increases.

Looking at cities = 5 and 7, 47/8 = 5.88. Looking at cities = 23 and 25, 85739293/18468023=4.64.

The data seems to support my analysis above that the multiplicative difference approaches 4 between any two adjacent data points as n increases. As well, it shows that the complexity is indeed dominated by a 2^n term but also includes something else which is in this case a polynomial term. As expected, the effect of the polynomial term drastically reduces as n increases because the 2^n term dominates.