

Group members:

- Fizzah Mansoor
- Hanaa Hashim Gatta
- Shams Arfeen

## Background:

### 1. Problem Description

The **travelling salesman problem (TSP)** asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?". Until now, researchers have not found a polynomial time algorithm for the travelling salesman problem. The most direct solution would be to try all permutations (ordered combinations) and see which one is cheapest, using brute-force search. The running time for this approach lies within a polynomial factor  $O(n!)$ , of the factorial of the number of cities, so this solution becomes impractical even for only 20 cities. One of the earliest applications of dynamic programming is the Held-Karp algorithm that solves the problem in time  $O(n^2 2^n)$ . This bound has also been reached by Exclusion-Inclusion in an attempt preceding the dynamic programming approach.

### 2. Held-Karp Algorithm

There is an optimization property associated with TSP problem that provides a recursive formulation to the problem given as:

*"Every sub-path of a shortest path is itself of shortest distance."*

The Held-Karp algorithm suggests a dynamic programming procedure based on the above property, stated as: Compute the solutions of all sub-problems starting with the smallest. Whenever computing a solution requires solutions for smaller problems using the above recursive equations, look up these solutions which are already computed. To compute a minimum distance tour, use the final equation to generate the 1st node, and repeat for the other nodes. For this problem, we cannot know which sub-problems we need to solve, so we solve them all. The exact time complexity, in terms of number of addition and comparison operations, can be shown to be  $(n - 1)(n - 2)2^{n-3} + (n - 1)$ .

### 3.i Held-Karp Instructions

Number the cities  $1, 2, \dots, N$  and assume we start at city 1, and the distance between city  $i$  and city  $j$  is  $d_{ij}$ . Consider subsets  $S \subseteq \{2, \dots, N\}$  of cities and, for  $c \in S$ , let  $D(S, c)$  be the minimum distance, starting at city 1, visiting all cities in  $S$  and finishing at city  $c$ .

- First phase: if  $S = \{c\}$ , then  $D(S, c) = d_{1,c}$ .  
Otherwise:  $D(S, c) = \min_{x \in S - c} (D(S - c, x) + d_{x,c})$ .
- Second phase: the minimum distance for a complete tour of all cities is  $M = \min_{c \in \{2, \dots, N\}} (D(\{2, \dots, N\}, c) + d_{c,1})$
- A tour  $n_1, \dots, n_N$  is of minimum distance just when it satisfies  $M = D(\{2, \dots, N\}, n_N) + d_{n_N,1}$ .

Our implementation:

#### 3.ii) Held Karp Algorithm

An iterative approach of the above algorithm, that executes oppositely by solving sub-problems first and then utilizing their solutions to solve higher-problems, can be written as:

- Set  $D(\emptyset, c) = d_{1c}$  for all  $c = 2, 3, \dots, n$ , and set  $i$  to 2 ( $i$  for length phase).
- Calculate  $i + 1$  length paths from  $i$  length paths, by adding one by one each unvisited node to each  $i$  length path i.e. set the distance

$$D(\{m_1, m_2, \dots, m_i\}, c) = D(\{m_1, m_2, \dots, m_{j-1}, m_{j+1}, \dots, m_i\}, m_j) + d_{c, m_j}$$

*for all unvisited nodes  $c$*

If  $D(\{m_1, m_2, \dots, m_i\}, c)$  already has a value, set it to the smaller value of the two. After all such paths are calculated, set  $i$  to  $i + 1$ .

- Repeat above step, unless  $i = n$ .
- Connect the two ends in each  $n$  length paths to get Hamiltonian cycles. Return the minimum cost cycle.

## 5. Held-Karp Example

Let  $C(a, S, b)$  be the cost of the shortest path from city  $a$ , through the cities in set  $S$ , to city  $b$

$$\text{For distance matrix} = \begin{pmatrix} 0 & 3 & 1 & 1 \\ 3 & 0 & 2 & 5 \\ 1 & 2 & 0 & 6 \\ 1 & 5 & 6 & 0 \end{pmatrix}$$

Starting with 2 cities one-way paths, setting the cost to the corresponding entry

$$C(1, \emptyset, 2) = 3$$

$$C(1, \emptyset, 3) = 1$$

$$C(1, \emptyset, 4) = 1$$

3 cities one way paths obtained by adding one by one, all unvisited nodes to each 2 cities-paths

$$C(1, \{2\}, 3) = 3 + 2 = 5$$

$$C(1, \{2\}, 4) = 3 + 5 = 8$$

$$C(1, \{3\}, 2) = 1 + 2 = 3$$

$$C(1, \{3\}, 4) = 1 + 6 = 7$$

$$C(1, \{4\}, 2) = 1 + 5 = 6$$

$$C(1, \{4\}, 3) = 1 + 6 = 7$$

Similar step is repeated to compute 4 cities paths, upon remembering that for the occurrence of cost of the same set  $S$ , we take the minimum of the costs.

$$C(1, \{2, 4\}, 3) = 8 + 6 = 14, \text{ discarded since } C(1, \{2, 4\}, 3) > C(1, \{4, 2\}, 3)$$

$$C(1, \{2, 3\}, 4) = 5 + 6 = 11, \text{ discarded since } C(1, \{2, 3\}, 4) > C(1, \{3, 2\}, 4)$$

$$C(1, \{3, 2\}, 4) = 3 + 5 = 8$$

$$C(1, \{3, 4\}, 2) = 7 + 5 = 12, \text{ discarded since } C(1, \{3, 4\}, 2) > C(1, \{4, 3\}, 2)$$

$$C(1, \{4, 2\}, 3) = 6 + 2 = 8$$

$$C(1, \{4, 3\}, 2) = 7 + 2 = 9$$

Computing the cost of each Hamiltonian path resulting from  $n=4$  cities paths

$$C(1 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 1) = 8 + 1 = 9$$

$$C(1 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 1) = 8 + 1 = 9$$

$$C(1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1) = 9 + 3 = 12$$

Thus,  $1 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 1$  is the minimum cost Hamiltonian cycle, appeared in clockwise as well as counter-clockwise due to undirected graph.

Note: The above time complexity only counts two types of operations (number addition and comparison) and does not takes into account the list-copy and list-checking operations, which are limitations of the python implementation of the algorithm. Applying them, requires a greater time complexity, formulated as  $(n - 1)^2 2^{n-2} + 3(n - 1)$ .

## 6. A Modification of Held-Karp Algorithm

We present a modified version of the above algorithm with time complexity reduced by 2 times and giving an ideal solution. The algorithm uses “path joining”, instead of addition of just one node, in a recursive call. Let  $P(a, S, b)$  represent the shortest path from city  $a$ , through the cities in set  $S$ , to city  $b$ . Let

$f(A, c) = S - A - \{c\}$  be a function where the set  $A \subset S$  with city  $c \in S$  and  $c \notin A$ . Then the following recursion defines the problem to sub-problem transition in our algorithm.

$$P(a, S, b) = \text{Min}\{ P(a, A, c) + P(c, f(A, c), b) \\ \text{for each } c \text{ in } S - A \}$$

Where “+” is the joining operation between paths that preserves the order of two paths as in the normal sense. We can see that while the Held-Karp algorithm computes all shortest  $i + 1$  length paths by adding one node to each  $i$  length path, the new algorithm can jump directly to length  $2i - 1$  i.e. when  $2|A| + 1 = |S|$ . However, the above technique is only applied in the last phase in the algorithm i.e. computing  $n$  length Hamiltonian cycles ( $c = b$ ) from  $\frac{n+2}{2}$  length one-way paths.

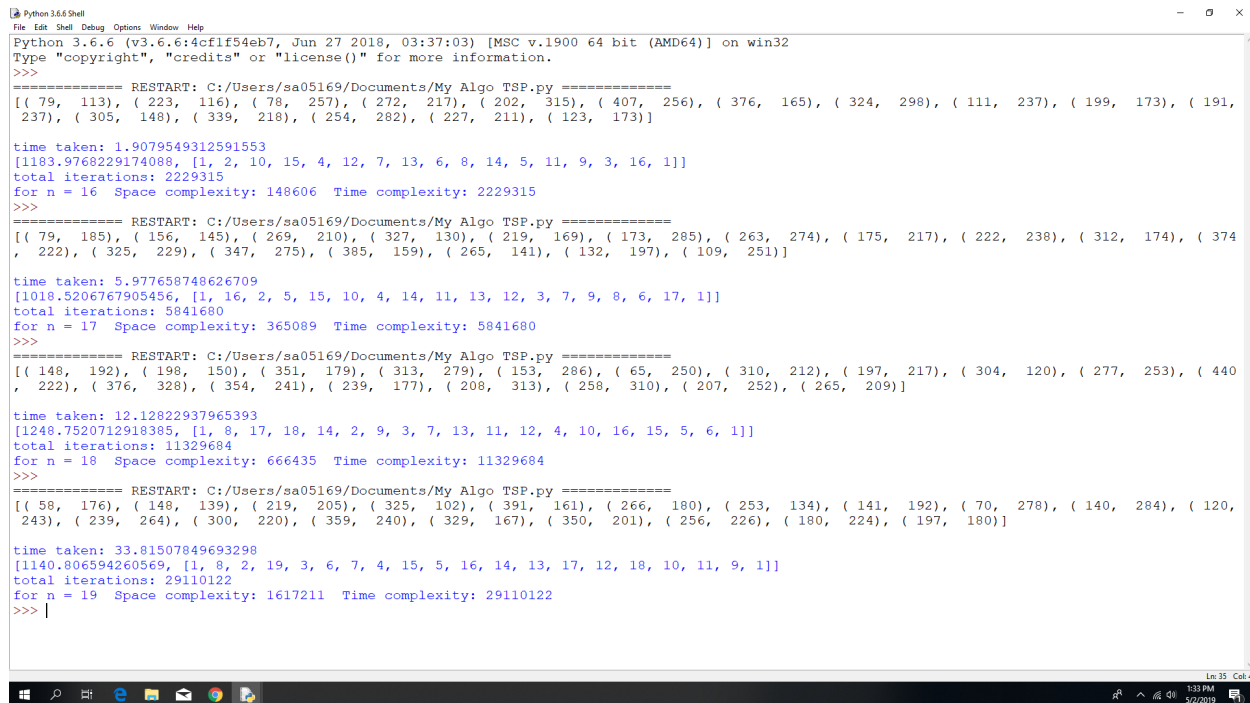
The reason for that is if the same technique is applied for each recursive call, one can only solve TSP for  $n = 3, 5, 9, 17, \dots, 2i - 1$  and so on. Even for the last phase, an

efficiency of twice execution speed has been observed, reflected in the following chart.

Number of cities, n	Time noted for Held-Karp algorithm in seconds	Time noted for the new algorithm in seconds
16	4.2	1.9
17	10.4	6.0
18	24.6	12.1
19	57.4	33.8

## Our Results:

### 6.ii. Screen-shot for some instances by the New Algorithm



```
Python 3.6.6 Shell
File Edit Shell Debug Options Window Help
Python 3.6.6 (v3.6.6:4c1f54eb7, Jun 27 2018, 03:37:03) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/sa05169/Documents/My Algo TSP.py =====
[( 79, 113), ( 223, 116), ( 78, 257), ( 272, 217), ( 202, 315), ( 407, 256), ( 376, 165), ( 324, 298), ( 111, 237), ( 199, 173), ( 191, 237), ( 305, 148), ( 339, 218), ( 254, 282), ( 227, 211), ( 123, 173)]

time taken: 1.9079549312591553
[1183.9768229174088, [1, 2, 10, 15, 4, 12, 7, 13, 6, 8, 14, 5, 11, 9, 3, 16, 1]]
total iterations: 2229315
for n = 16 Space complexity: 148606 Time complexity: 2229315
>>>
===== RESTART: C:/Users/sa05169/Documents/My Algo TSP.py =====
[( 79, 185), ( 156, 145), ( 269, 210), ( 327, 130), ( 219, 169), ( 173, 285), ( 263, 274), ( 175, 217), ( 222, 238), ( 312, 174), ( 374, 222), ( 325, 229), ( 347, 275), ( 385, 159), ( 265, 141), ( 132, 197), ( 109, 251)]

time taken: 5.977658748626709
[1018.5206767905456, [1, 16, 2, 5, 15, 10, 4, 14, 11, 13, 12, 3, 7, 9, 8, 6, 17, 1]]
total iterations: 5841680
for n = 17 Space complexity: 365089 Time complexity: 5841680
>>>
===== RESTART: C:/Users/sa05169/Documents/My Algo TSP.py =====
[( 148, 192), ( 198, 150), ( 351, 179), ( 313, 279), ( 153, 286), ( 65, 250), ( 310, 212), ( 197, 217), ( 304, 120), ( 277, 253), ( 440, 222), ( 376, 328), ( 354, 241), ( 239, 177), ( 208, 313), ( 258, 310), ( 207, 252), ( 265, 209)]

time taken: 12.12822937965393
[1248.7520712918385, [1, 8, 17, 18, 14, 2, 9, 3, 7, 13, 11, 12, 4, 10, 16, 15, 5, 6, 1]]
total iterations: 11329684
for n = 18 Space complexity: 666435 Time complexity: 11329684
>>>
===== RESTART: C:/Users/sa05169/Documents/My Algo TSP.py =====
[( 58, 176), ( 148, 139), ( 219, 205), ( 325, 102), ( 391, 161), ( 266, 180), ( 253, 134), ( 141, 192), ( 70, 278), ( 140, 284), ( 120, 243), ( 239, 264), ( 300, 220), ( 359, 240), ( 329, 167), ( 350, 201), ( 256, 226), ( 180, 224), ( 197, 180)]

time taken: 33.81507849693298
[1140.806594260569, [1, 8, 2, 19, 3, 6, 7, 4, 15, 5, 16, 14, 13, 17, 12, 18, 10, 11, 9, 1]]
total iterations: 29110122
for n = 19 Space complexity: 1617211 Time complexity: 29110122
>>> |
```

### 6.iii. Screen-shot for some instances by Held-Karp Algorithm

```

Python 3.6.6 Shell
File Edit Shell Debug Options Window Help
Python 3.6.6 (v3.6.6:4cf1f54eb7, Jun 27 2018, 03:37:03) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/sa05169/Documents/Held-Karp.py =====
[ ( 79, 113), ( 223, 116), ( 78, 257), ( 272, 217), ( 202, 315), ( 407, 256), ( 376, 165), ( 324, 298), ( 111, 237), ( 199, 173), ( 191, 237), ( 305, 148), ( 339, 218), ( 254, 282), ( 227, 211), ( 123, 173) ]
4.2276387214660645
[1183.9768229174088, [16, 3, 9, 11, 5, 14, 8, 6, 13, 7, 12, 4, 15, 10, 2, 1]]
time complexity: 3686445
for n = 16 Space complexity: 245746 Time complexity: 3686445
>>>
===== RESTART: C:/Users/sa05169/Documents/Held-Karp.py =====
[ ( 79, 185), ( 156, 145), ( 269, 210), ( 327, 130), ( 219, 169), ( 173, 285), ( 263, 274), ( 175, 217), ( 222, 238), ( 312, 174), ( 374, 222), ( 325, 229), ( 347, 275), ( 385, 159), ( 265, 141), ( 132, 197), ( 109, 251) ]
10.449647426605225
[1018.5206767905456, [16, 2, 5, 15, 10, 4, 14, 11, 13, 12, 3, 7, 9, 8, 6, 17, 1]]
time complexity: 8388656
for n = 17 Space complexity: 524273 Time complexity: 8388656
>>>
===== RESTART: C:/Users/sa05169/Documents/Held-Karp.py =====
[ ( 148, 192), ( 198, 150), ( 351, 179), ( 313, 279), ( 153, 286), ( 65, 250), ( 310, 212), ( 197, 217), ( 304, 120), ( 277, 253), ( 440, 222), ( 376, 328), ( 354, 241), ( 239, 177), ( 208, 313), ( 258, 310), ( 207, 252), ( 265, 209) ]
24.568708896636963
[1248.7520712918385, [8, 17, 18, 14, 2, 9, 3, 7, 13, 11, 12, 4, 10, 16, 15, 5, 6, 1]]
time complexity: 18939955
for n = 18 Space complexity: 1114096 Time complexity: 18939955
>>>
===== RESTART: C:/Users/sa05169/Documents/Held-Karp.py =====
[ ( 58, 176), ( 148, 139), ( 219, 205), ( 325, 102), ( 391, 161), ( 266, 180), ( 253, 134), ( 141, 192), ( 70, 278), ( 140, 284), ( 120, 243), ( 239, 264), ( 300, 220), ( 359, 240), ( 329, 167), ( 350, 201), ( 256, 226), ( 180, 224), ( 197, 180) ]
57.373440742492676
[1140.806594260569, [9, 11, 10, 18, 12, 17, 13, 14, 16, 5, 15, 4, 7, 6, 3, 19, 2, 8, 1]]
time complexity: 42467382
for n = 19 Space complexity: 2359279 Time complexity: 42467382
>>> |

```

### 7. Instructions for the New Algorithm

Let  $S$  be the set of all cities and  $d_{ij}$  represent the distance between city  $i$  and city  $j$ . Let  $D(1, S, c)$  be the minimum distance starting at city 1, visiting all cities in  $S$  and finishing at city  $c$ .

- Calculate length 2 paths, by setting  $D(1, \emptyset, c) = d_{1c}$  for all  $c = 2, 3, \dots, n$ , and set  $i$  to 2 ( $i$  for length phase).
- Calculate  $i + 1$  length paths from  $i$  length paths, by adding one by one each unvisited node to each  $i$  length path i.e. set the distance

$$D\left(1, \{m_1, m_2, \dots, m_i\}, c\right) = D\left(1, \{m_1, m_2, \dots, m_{j-1}, m_{j+1}, \dots, m_i\}, m_j\right) + d_{c, m_j}$$

for all unvisited nodes  $c$  in the path  $1 \rightarrow \{m_1, m_2, \dots, m_{j-1}, m_{j+1}, \dots, m_i\} \rightarrow m_j$

If  $D\left(1, \{m_1, m_2, \dots, m_i\}, c\right)$  already has a value, set it to the smaller value of the two. After all such paths are calculated, set  $i$  to  $i + 1$ .

- Repeat above step, unless  $i = \frac{n+2}{2}$  for even  $n$  or  $i = \frac{n+3}{2}$  if  $n$  is odd.
- Join each pair of  $i$  length paths to get Hamiltonian cycles. Two paths form a pair when they have the same end city and disjoint city-set i.e.  $D(1, A, c)$  with  $D(1, S - A, c)$  which represents the Hamiltonian cycle  $\{1 \rightarrow (\text{cities in } A) \rightarrow c \rightarrow (\text{cities in } S - A) \rightarrow 1\}$ .
- Return the optimum route.

## 8. Why the New Algorithm is Faster

Both algorithms start from computing 2-cities paths, which require  $n - 1$  number of value assigning tasks. Each of these paths then make 3-cities paths by adding one by one of each  $n - 2$  number of unvisited cities, until we have  $(n - 1)(n - 2)$  number of 3-length paths. Upto this stage, time-complexity in both the algorithms is given by  $(n - 1) + (n - 1)(n - 2)$ . Following is asserted a general case to help simplify the explanation.

Let there be  $k$  number of all  $i$  length paths that start from our initial city 1. To compute all  $i + 1$  length paths from these  $k$  paths, we add each of the remaining  $n - i$  unvisited nodes to each of these paths. When a path already has a value we compare and replace the old value only if it is greater than the new value. This require  $k(n - i)$  number of addition and comparison operations. Now consider an arbitrary  $i + 1$  length path  $D(1, \{m_1, m_2, \dots, m_{i-1}\}, c)$ , we show that  $i - 1$  of the aforementioned operations have contributed to compute it. Following are such operations:

First assignment:

$$D(1, \{m_1, m_2, \dots, m_{i-1}\}, c) = D(1, \{m_2, m_3, \dots, m_{i-1}\}, m_1) + d_{c, m_1}$$

Second assignment:

$$D(1, \{m_1, m_2, \dots, m_{i-1}\}, c) = D(1, \{m_1, m_3, \dots, m_{i-1}\}, m_2) + d_{c, m_2}$$

⋮ (note the continuum)

$(i - 1)$ th assignment:

$$D\left(1, \{m_1, m_2, \dots, m_{i-1}\}, c\right) = D\left(1, \{m_1, m_2, \dots, m_{i-2}\}, m_{i-1}\right) + d_{c, m_{i-1}}$$

Of course, we assign the minimum of such values of assignment. Since, there are  $k(n - i)$  number of total operations, this makes  $\frac{k(n - i)}{(i - 1)}$  number of  $i + 1$  length

paths. Thus, in the Held-Karp, which executes the process to  $n - 1$  length paths which are then connected end-to-end to get Hamiltonian cycles. Total complexity becomes:

$$(n - 1) + \sum_{k=2}^{n-1} k(k - 1) \binom{n - 1}{k} = (n - 1)(n - 2)2^{n-3} + (n - 1)$$

On the other hand, our algorithm continues this process to  $i = \frac{n + 2}{2}$  for even  $n$  or  $i = \frac{n + 3}{2}$  if  $n$  is odd. Time complexity becomes:

$$(n - 1) + \sum_{k=2}^i k(k - 1) \binom{n - 1}{k}$$

Thus, the binomial expansion of  $2^{n-3} = (1 + 1)^{n-3}$  has been reduced to about half of the original expansion of terms. Since, it is known that in a binomial expansion, the coefficients are repeated after the middle term i.e. for the  $r$ th term  $\binom{n}{r} = \binom{n}{n - r}$ , one can imagine that the time complexity have been halved, as one set of the repeating terms are vanished.

## 9. Example Solved by the New Algorithm

Consider the following undirected distance matrix, of 4 nodes

$$\begin{pmatrix} 0 & 3 & 1 & 1 \\ 3 & 0 & 2 & 5 \\ 1 & 2 & 0 & 6 \\ 1 & 5 & 6 & 0 \end{pmatrix}$$

The computer calculates 3-cities one-way paths as done in the Held-Karp algorithm.

$$C(1, \{2\}, 3) = 3 + 2 = 5$$



$$C(1, \{2\}, 4) = 3 + 5 = 8$$

$$C(1, \{3\}, 2) = 1 + 2 = 3$$

$$C(1, \{3\}, 4) = 1 + 6 = 7$$

$$C(1, \{4\}, 2) = 1 + 5 = 6$$

$$C(1, \{4\}, 3) = 1 + 6 = 7$$

Now, we join each pair of 3-cities paths with same end city and disjoint set S, to get  $3(2) - 1 = 5$  cities Hamiltonian paths (with 1 at both ends). As an example, consider the following join operation:

$$C(1 \rightarrow 2 \rightarrow 3) + C(3 \rightarrow 4 \rightarrow 1) = 5 + 7 = 12$$

Needless to say, we will discard the paths with greater costs. We have the following Hamiltonian paths:

$$C(1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1) = 5 + 7 = 12$$

$$C(1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1) = 8 + 7 = 15$$

$$C(1 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 1) = 3 + 6 = 9$$

Hence, the optimum solution is taken as  $1 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 1$

### Conclusion:

The Held Karp modified is a better algorithm for solving the TSP problem in comparison to the traditional factorial method and slightly better than the original Held Karp method. The Held Karp modified method solves the TSP in less amounts of operations which subsequently reduces its complexity to an exponential value. Although, the formula is far from perfect, it is significantly better than its conventional counterparts. Our results suggest that the modified Held Karp method takes 30 percent less time than the original held Karp method and 47 percent less than the factorial method. It is suffice to know that the experiment was run in a limited scale of cities less than 20. Hence, we are not able to deduce if the method can be suitable for larger scale. But for the most part, the modified held Karp method has a unique way of counting paths by joining them and calculating the shortest output.