Min. Metric Travelling Salesman Problem

DAPO TP1 2017/2018

Manuel Henriques 42546, Sebastião Pamplona 42735

1. Problem

A real function $c : S \times S \rightarrow \mathbb{R}$ is a *metric cost* if it satisfies the non-negativeness, strict positiveness, symmetry and triangle inequality properties, which are as follows, respectively, for every $u, v, w \in S$:

- $c(u,v) \geq 0$
- $c(u,v) = 0 \Leftrightarrow u = v$
- $c(u,v) = c(v,u)$
- $c(u,v) \leq c(u,w) + c(w,v)$.

In the Min. Metric Travelling Salesman Problem, we are given a complete undirected graph $G = (V, E)$, with a *metric cost* $c(u,v)$ associated with each edge $(u,v) \in E$, and we must find a hamiltonian cycle of G with minimum cost.
The Min. Metric Travelling Salesman Problem is an NP-hard problem.

2. Algorithms

2.1 Greedy Algorithm, using MST

2.1.1 Algorithm Description

Given an instance $G = (V, E)$ of the Metric Travelling Salesman Problem, the algorithm can be described in pseudocore as follows:

1. Select a vertex r to be the root.
2. Compute an MST T from r using Prim's Algorithm.
3. List the vertices visited in preorder p from the constructed T.

2.1.2 Algorithm Time Complexity

The algorithm has polynomial time complexity. The Prim's algorithm runs in $\Theta(V^2)$ and the preorder traversal runs in $\Theta(V)$.

2.1.3 Algorithm Space Complexity

The algorithm has polynomial space complexity. The structures used for the Prim's algorithm:

- A hashset, $\Theta(V)$.
- A priority queue, $\Theta(V)$.
- A boolean array, $\Theta(V)$.

The structure used for the permutation of nodes was an ArrayList, $\Theta(V)$.

The structure used for the MST was a LinkedList, $\Theta(V)$.

2.1.4 Algorithm Approximation Ratio

This algorithm has an approximation ratio of 2. The proof for this can be found in "c03-Greedy-part2.pdf", on the course page in *CLIP* under "Acetatos", slides 17-20.

2.2 Christofides Algorithm

2.2.1 Algorithm Description

Given an instance $G = (V, E)$ of the Metric Travelling Salesman Problem, the algorithm can be described in pseudocore as follows:

1. Create a minimum spanning tree T of G.
2. Let O be the set of vertices with odd degree in T. By the handshaking lemma[1], O has an even number of vertices.
3. Find a minimum-weight perfect matching[2] M int the induced subgraph given by the vertices from O.
4. Combine the edges of M and T to form a connected multigraph *H* in which each vertex has even degree.
5. Form an Eulerian circuit in *H*.
6. Make the circuit found in previous step into a Hamiltonian circuit by skipping repeated vertices (*shortcutting*).

NOTE: regarding the step 3 of this algorithm, we were not able to implement the minimum-weight perfect matching, only a greedy perfect matching, therefore it only gives us the minimum-weight perfect matching with very few (and small) instances of the graph. However, we encountered an implementation of this algorithm online, so we decided to run each instance of the graph used for testing, with that

---

[1] *Handshaking lemma: every finite undirected graph has an even number of vertices with odd degree.*

[2] *Perfect matching: a matching which matches all the vertices of the graph. That is, every vertex of the graph is incident to exactly one edge of the matching.*

implementation as well, for better comparison with *Greedy Algorithm, using MST*.

Regarding the step 5, we implemented two different algorithms for calculating the Euler Tour. This is due to the fact that sometimes one algorithm is better than the other (depending on the instances used), and frankly we were not able to implement an optimal solution for the Euler Tour. Again, the other implementation of Christofides that we found online[3] gives us a better solution, that's why we include it in our tests.

2.2.2 Algorithm Time Complexity

The algorithm has polynomial time complexity. The MST algorithm, has stated, has polynomial space complexity; Finding the nodes with odd degree has time complexity of $\Theta(V)$; For the perfect matching we used a greedy approach with complexity $\Theta(E)$; Combining two graphs has complexity $\Theta(E)$; The Hierholzer's algorithm for the Eulerian path has complexity $\Theta(E)$; The Hamilton cycle has complexity $\Theta(V)$.

2.2.3 Algorithm Space Complexity

The algorithm has polynomial space complexity.The MST algorithm, has stated, has polynomial time complexity.

The data structures used to find odd degrees were an array of integers and a list $\Theta(V)$.

The data structures used to make a greedy perfect match were a list, a priority queue and a map, $\Theta(V+E)$.

Combining two graphs only requires the already used data structures for the perfect match and the data structures from the MST algorithm.

We used two solutions for the eulerian path algorithm and as such, the space complexity for each solution is:

1. The first solution requires two maps, one with the information of which nodes are available from a node key and the other the available edges, $\Theta(E)$, and a queue with all the node keys that have nodes to explore, $\Theta(V)$.
2. The second solution requires two graphs, $\Theta(V)$, a data structure created by us (DAPOLinkedList), which is a classic linked list, $\Theta(V)$ and an array of booleans, with size V, so in total $\Theta(V)$.

The hamilton cycle only requires a list, $\Theta(V)$.

2.2.4 Algorithm Approximation Ratio

This algorithm has an approximation ratio of 3/2. The link for the proof can be found in References[2].

2.3 Algorithms for Christofides Algorithm

The christofides algorithm is an algorithm that requires the use of three algorithms to work. It requires the solution of the Minimum Spanning Tree problem, the Minimum-Weight Perfect Match problem and the Eulerian Path of the combined graphs.

For the MST problem we used a greedy algorithm explained in topic 2.1.

For the MWPM problem we found that the Blossom algorithm by Jack Edmonds was created with complexity $\Theta(V^2E)$. Since 1961 this algorithm has been improved to a complexity of $\Theta(V.E.logV)$. Unfortunately, this algorithm is known to have a complex implementation and we did not find any usable source code for our program. Instead of implementing a Minimum-Weight, we only implemented a greedy Perfect Match, that pairs a node with another node that has the immediate lowest weighted path (and that does not belong to another pair).

For the Eulerian Path we used the Hierholzer's algorithm. The fact that our combined graph of the MST with the Perfect Match only has odd degree nodes, we can always trace a path from a root node to itself, without the possibility of getting stuck in another vertice. Doing this multiples times with unused nodes gives us a path that uses every vertice. Removing repeated vertices (*shortcutting*) will give us the Hamilton cycle for the output of Christofides.

3. Tests

As explained in 2.2.1, we use 3 different solutions of the Christofides algorithms for our tests. Two of this solutions where implemented by us and the third solution was found on a git repository[3] .

3.1 Tests

We used a different set of instances for our tests. The files have a name according to the number of vertices. An input instance with 5 vertices has a file name of "5.txt".

The first instance, “5.txt”, was created by us using a graph given in the theoretical classes for the MST problem.
The second set of instances, from “15.txt” to “128.txt”, where found in the online git repository for the Christofides algorithm[3].
The third set of instances where obtained from a website[4].

3.2 Tests Result

This table compares the obtained cost from the algorithms and the different input instances.
The *MST* represents our greedy approximation with an approximation ratio of 2.
The *Christ.1* and *Christ.2* the Christofides algorithm implemented by us but using different approaches to calculate the Euler Path.
The *Christ.Online* is the Christofides algorithm found in the git repository[3].

Nodes (*n.txt*)	MST	Christ. 1	Christ. 2	Christ. Online
5	12	11	11	11
15	494	452	441	380
17	3278	3037	3103	2293
26	1154	1340	1096	1095
48	76667	61317	58538	43174
59	2225	2213	2232	1225
128	103686	61373	74137	57807
423	2664	2806	2696	1982
813	8760	10682	8506	4690
2036	14238	15294	13976	8840

By analysing these results, we verify that the solutions of the Christofides implementations are usually better, given its better approximation ratio. However, the solution of the Christofides implementation found online are always better than ours because its algorithm for calculating the Euler Tour and the minimum-weight perfect match is better implemented (as said in the *NOTE*, on section 2.2.1).

4. REFERENCES

[1] Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. pp. 1112. [Online]. Available: http://ressources.unisciel.fr/algoprog/s00aaroot/aa00module1/res/%5BCormen-AL2011%5DIntroduction_To_Algorithms-A3.pdf

[2] Algorithm Design and Applications by Goodrich, Michael T., Tamassia, Roberto. pp. 514. [Online]. Available: http://canvas.projekti.info/ebooks/Algorithm%20Design%20and%20Applications%5BA4%5D.pdf

[3] Christofides by faisal22. [Online]. Available: https://github.com/faisal22/Christofides/tree/master/src/Datasets

[4] http://www.math.uwaterloo.ca/tsp/vlsi/index.html