Practice 16. Approximation algorithms

Mayra Cristina Berrones Reyes

May 14, 2020

1 Introduction

In the subject of optimization, most problems presented are $\mathcal{NP}$-hard, thus have no polynomial algorithms to find their solutions that we know of. Approximation algorithms are an efficient way to find approximate solutions to this type of problems.

An α-approximations a polynomial time algorithm that for all instances of the problem it produces a solution, whose value is within the factor of α and the value of the optimal solution. The factor α is called the approximation ratio [Williamson and Shmoys, 2011].

2 A 2-Approximation Algorithm for TSP

In other practices we establish that it is $\mathcal{NP}$-complete to decide if a given graph $G = (V, E)$ has a Hamiltonian cycle. An approximation algorithm for TSP can be used to solve a Hamiltonian cycle, since both problems are very similar [Schaeffer, 2020].

The input for this algorithm is a distance matrix of $n \times n$. The numbers inside the matrix represent d where $d_{i,j}$ denotes the cost of travelling from city i to city j. This matrix is defined by the following features [Biswas, 2015]:

1. $d_{i,j} \geq 0$ for all $i, j \in \{1, \ldots, n\}$
2. $d_{i,j} = d_{j,i}$ for all $i, j \in \{1, \ldots, n\}$
3. $d_{i,k} \leq d_{i,j} + d_{j,k}$ for all $i, j, k \in \{1, \ldots, n\}$
4. $d_{i,i} = 0$ for all $i \in \{1, \ldots, n\}$

The number one feature is the non negative rule of the TSP. The second rule aplies to the symmetric form of the matrix, which means it is the same distance from node i to j as is from j to i. Then we have the triangle inequality feature.

This basically states that the distance between node i and k is less or equal to the distance between node i and j, plus the distance between j and k. This can be seen in Figure 1.

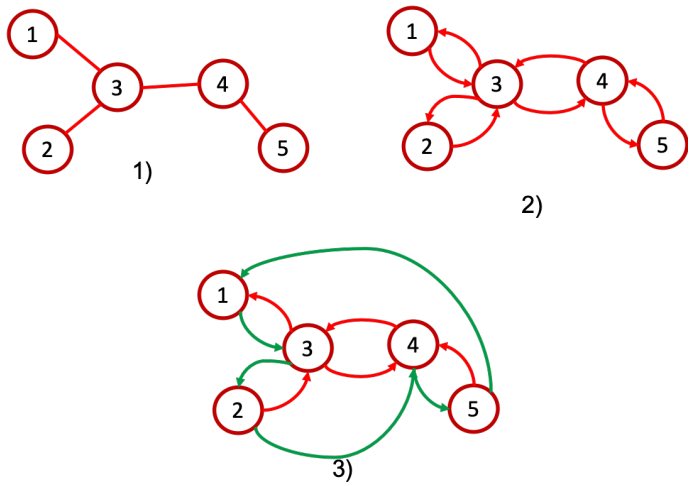

Figure 1: Example of the 2-approximation algorithm. In step one we see our MST. In step two we transverse our tree and form sequence S_1. Finally step three we have our shortcuts made and form sequence S_2.

The Triangle-Inequality holds in many practical situations. When the cost function satisfies the triangle inequality, we can design an approximate algorithm for TSP that returns a tour whose cost is never more than twice the cost of an optimal tour [Cormen et al., 2009].

Now, knowing this features, we can begin to compute our path. We can think of d as a complete weighted graph G whose vertex set is $V = \{1, \ldots, n\}$ and for which the weight of the edge ij is $d_{i,j}$. So now we first need to find the minimum spanning tree (MST) of this graph. The MST can be found in $\mathcal{O}(n^2)$, being n the number of vertex. There are several algorithms to find this MST, for example, the Prim's algorithm.

The steps to build a MST are as follows:

1. Choose a starting node or root node for the starting and ending point of the salesman.
2. Construct a MST from that root node.
3. List the vertices visited in preorder walk of the constructed MST and add the root node at the end.

In the example on Figure 1 we can see the diagram of a given graph, and then we show the MST constructed taking the node 1 as root. The pre order transversal of this example is

$$S_1 = 1, 3, 2, 3, 4, 5, 4, 3, 1 \ .$$

For transversing the tree, in this example, we obtain the sequence

$$S_1 = 1, 3, 2, 3, 4, 5, 4, 3, 1 \ .$$

Now, remove all but the first occurrence of each vertex in his sequence (except 1):

$$S_2 = 1, 3, 2, 4, 5, 1 \ .$$

Note that S_2 is a permutation of $\{1, \ldots, n\}$ since S_1 contains every element of $\{1, \ldots, n\}$ at least once and therefore S_2 contains every element exactly once. Therefore, S_2 is a valid solution for the TSP. So we have T that will represent the MST which is the connected subgraph of G of minimum weight.

Each edge of T is used exactly twice S_1, so

$$w(S_1) = 2w(T) \ .$$

By the triangle inequality

$$w(S_2) \leq w(S_1) \ .$$

Remember that T is a minimum spanning tree; it's a minimum weight connected subgraph of G. But a travelling salesman tour is also a connected subgraph of G, so

$$w(T) \leq w(C^\star) \ ,$$

where $C^\star$ is any optimal travelling salesman tour. Putting all these together, we have

$$w(S_2) \leq w(S_1) \leq 2w(T) \leq 2w(C^\star) \ .$$

We have just found a solution, S_2, to the TSP whose weight is at most twice of the optimal solution.

3 Christofides algorithm for TSP

The strategy for this algorithm is to construct an Eulerian tour whose cost is at most α, the short cut it to get an α-approximation solution. A connected graph is Eulerian if and only if every vertex has even degree. The steps to carry out this algorithm are as follows:

1. Find minimum spanning tree (T).
2. Let O be a set of nodes with odd degree in T. Find a minimum cost perfect matching on O.

3. Add the set of edges of the perfect matching to T and find an Eulerian tour.

4. Shortcut the Eulerian tour.

This means that we have found a solution to the traveling salesman problem whose cost is

$$w(T) + w(M)$$

We already know that $w(T) \leq w(C^\star)$. For M, we rely on an algorithm for the minimum weight perfect matching problem, which runs in $O(n^3)$ time. Still, we need to relate $w(M)$ to $w(C^\star)$.

Finally, by triangle inequality, shortcutting previously visited vertices does not increase the cost $w(C) \leq w(C^\star)$. What's more, C can be partitioned into two perfect matching by alternately coloring its edges red and blue. Since the total cost of the two matchings is at most $w(C)$, the cheaper one has cost at most $1/2$ of $w(C)$. In other words, one of these two matchings has weight at most $w(C)/2$

$$w(M) \leq w(C)/2 \leq w(C^\star)/2 \ .$$

So in the end we found a solution that has a weight of

$$w(T) + w(M) \leq w(C^\star) + w(C^\star)/2 = \frac{3}{2} w(C^\star)$$

In Figure 2 we see an example of the Christofides algorithm.

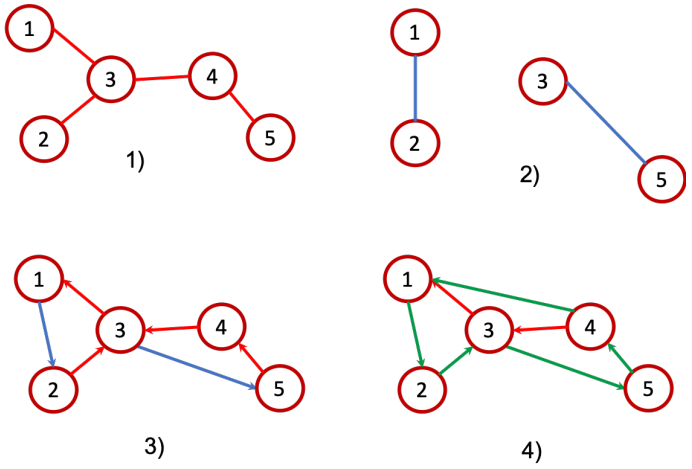

Figure 2: Example of the Christofides algorithm. In step one we have our MST. In step 2, we do or two matching in color blue. In step three we add the other vertices, and finally in step four we see our tour.

4 Conclusions

In several other practices where we realize the difficulty or the computational time it takes to solve some algorithms, it is really interesting to see the ways that exist to approach the $\mathcal{NP}$-hard problems. It seems relatively more easy to understand this subject after practices with similar subjects such as reductions and other complexity problems we made along some other practices.

References

[Biswas, 2015] Biswas, A. S. (2015). R9.approximation algorithms: Traveling salesman problem.

[Cormen et al., 2009] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press.

[Schaeffer, 2020] Schaeffer, E. (2020). Complejidad computacional de problemas y el analisis y diseño de algoritmos. Available `"https://elisa.dyndns-web.com/teaching/aa/pdf/aa.pdf"`.

[Williamson and Shmoys, 2011] Williamson, D. P. and Shmoys, D. B. (2011). *The design of approximation algorithms*. Cambridge university press.