INFO F424 Project: The Symmetric Traveling Salesman Problem (STSP)

kaoutar chankhar

Update: August 19, 2019

Abstract

in this report we try to study one compact formulation of the TSP problem proposed for subtours elimination, an exact resolution method based on integer programming (IP) guaranteeing an optimal solution has been applied to several data sets (different distance matrices randomly generated), in addition to, an approximation algorithm that not guarantee the optimality of solutions, however, it has the advantage of being time-efficient, afterward and finally, a comparative analysis based on numerical experimentations is discussed to highlight both weaknesses and strengths of the two approaches.

Keywords: Traveling Salesman Problem, two-commodity flow formulation, Christofides algorithm

1 Network flow based formulation of TSP

1.1 The FCG formulation

The standard traveling salesman problem (TSP) can be stated mathematically by means of a complete graph $G = (V, E)$, where set $V = 1, 2, ..., n$ refers to the cities (aka nodes) and E represents the set of the distances between each possible pair usually called edges , the challenge is to find the best possible way (shortest) of visiting all the cities exactly once and returning to the starting point, such circuit is known by Hamiltonian cycle.

let us consider the basic mathematical formulation of the the TSP problem

$$\underset{x}{\text{minimize}} \quad \sum_{i,j \in V} c_{ij} x_{ij} \tag{1a}$$

$$\text{subject to} \quad \sum_{i \in V} x_{ij} = 1, \quad j \in V, \tag{1b}$$

$$\sum_{j \in V} x_{ij} = 1, \quad i \in V, \tag{1c}$$

$$x_{ij} \in \{0, 1\}, \quad i, j \in V \tag{1d}$$

where x is a dummy variable

$$x_{ij} = \begin{cases} 1 & \text{if city } j \text{ is visited directly after city } i \\ 0 & \text{otherwise} \end{cases}$$

and c_{ij} represents the distance between i and j .

The objective function $(1a)$ aims to minimize the total distance of a complete tour passing by all of the cities, the first constraint $(1b)$ guaranthee that each city is arrived at from exactly one other city, in the same manner, the second constraint $(1c)$ ensure that from each city there is a departure to exactly one other city, the last constraint $(1d)$ defines the integrality constraint.

For any solution $X = (x_{ij})_{n*n}$ of the above assignment problem, consider the set of solutions S . Clearly, S represents a tour or a collection of subtours in G which means a set of unconnected paths. In order to avoid such cases, Finke, Claus and Gunn have added additional restrictions to make sure that S does not contain any subtours (also named cycle free). These restrictions are called subtour elimination constraints.

Hence we get a new compact formulation of the TSP problem, called the two-commodity flow model.

$$\underset{x,\,y,\,z}{\text{minimize}} \quad \sum_{i,j\in V} c_{ij}x_{ij} \tag{2a}$$

$$\text{subject to} \quad \sum_{i\in V} x_{ij} = 1, \quad j \in V, \tag{2b}$$

$$\sum_{j\in V} x_{ij} = 1, \quad i \in V, \tag{2c}$$

$$x_{ij} \in \{0,1\}, \quad i,j \in V, \tag{2d}$$

$$\sum_{j\in V} y_{1j} - \sum_{j\in V} y_{j1} = n-1,, \tag{2e}$$

$$\sum_{j\in V} y_{ij} - \sum_{j\in V} y_{ji} = -1, \quad i \in V\backslash\{1\}, \tag{2f}$$

$$\sum_{j\in V} z_{1j} - \sum_{j\in V} z_{j1} = -(n-1),, \tag{2g}$$

$$\sum_{j\in V} z_{ij} - \sum_{j\in V} z_{ji} = 1, \quad i \in V\backslash\{1\}, \tag{2h}$$

$$\sum_{j\in V} y_{ij} + \sum_{j\in V} z_{ij} = n-1, \quad i \in V, \tag{2i}$$

$$y_{ij}, z_{ij} \geq 0, \quad i,j \in V, \tag{2j}$$

$$y_{ij} + z_{ij} = (n-1)x_{ij}, \quad i,j \in V \tag{2k}$$

To understand the intuition behind the formulation above, let's make the situation more realistic, and consider for example a salesman that delivers a full container of some commodity (bottled gas), and picks up an empty container for each costumer. at all times he will have $(n-1)$ containers (empty + full), here we suppose the salesman doesn't deliver to himself. In other words, the salesman starts his journey with $(n-1)$ full containers, and each time he arrived to a costumer, he give one full and get back one empty, until he returns back to his start point with $(n-1)$ empty containers.

y_{ij} and z_{ij} are here two continuous variables that represent the flow of full containers in edge (i,j) and the flow of empty containers in edge (i,j) respectively.

Constraint $(2e)$ guaranthee a path from source to each node. Constraint $(2f)$ completes $(2e)$ so that there is no other path starting from another node (eliminating subtours). We can say that these two constraints conserve the flow of full containers from breaks.
In the same manner, Constraints $(2g)$ and $(2h)$ guaranthee a return path to source and conserve the flow of empty containers from breaks. Constraint $(2i)$ ensures the synchronization between the two flows, its an immediate result of equation $(2k)$.

$(2j)$ gives non negativity of the two flows. And finally, last constraint $(2k)$ incorporate the fact of having exactly $(n-1)$ containers (empty + full) in each edge of the tour.

1.2 Experimental Results

The proposed formulation was initially dedicated to asymmetric traveling salesman problem (general case), namely when $distance(i,j) \neq distance(j,i)$, however the model is valid for symmetric TSP which illustrates our case.
To evaluate the run-time of our model, we use *pyomo* environment for implementation and GLPK solver to solve it, several data instances are utilized whether being randomly generated or provided by tsp library.

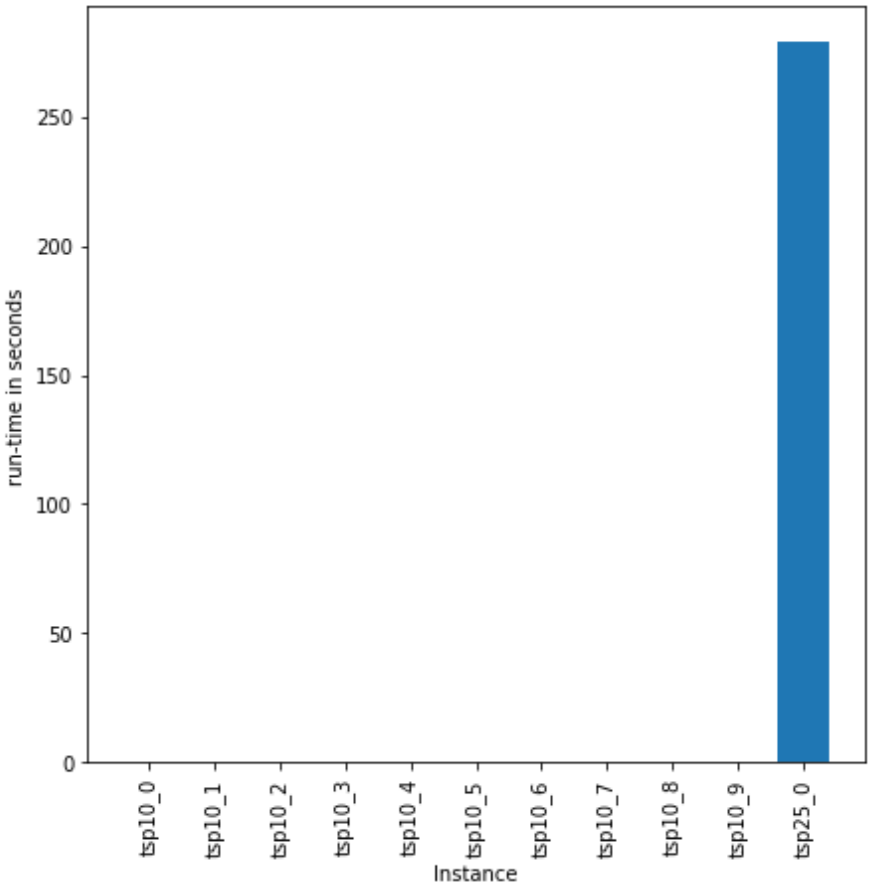

Figure 1: computational time of some random instances.

In Figure 1, We can see that for very small size data sets (tsp10-0), the FCG algorithm is rationally efficient regarding computational time (around some milliseconds), however, more data size increases more the algorithm takes quite a longtime to be solved (instance tsp25-0) and that is due to the

polynomial increase of the number of decision variables and constraints in the problem, For the time being, we can be a little reserved about making any judgment about the algorithm efficiency since the solver used here (GLPK) is not very powerful compared to commercial solvers like CPLEX specially when dealing with mixed integer linear problems containing very high number of variables and constraints.

Unfortunately and as expected, we couldn't make any test on tsplib instances due to the its large volume. .

2 Christofides's Algorithm

2.1 Definition

The Christofides algorithm is an algorithm for finding approximate solutions to the travelling salesman problem, on instances where the distances form a metric space (X, d) . Here X is a finite space and $d : X^2 \longrightarrow \mathbf{R}$ is the distance function satisfying symmetry and triangle inequality

- $d(i, j) \geq 0$ and $d(i, j) = 0$ if $i = j$;
- $d(i, j) = d(j, j)$;
- $d(i, j) \leq d(i, t) + d(t, j)$;

and data used in applications here respond perfectly to these conditions.

Algorithm :

.

1. The starting point of the algorithm is a minimum spanning tree (MST) T that connects all vertices together without cycles with minimum possible total edge weight, if we consider H the optimal solution, then we have $cost(T) \leq cost(H)$, there are several techniques to find the MST, one of the most known is the Kruskal algorithm. an illustrative plot is given using instance tsp10-0.

   .

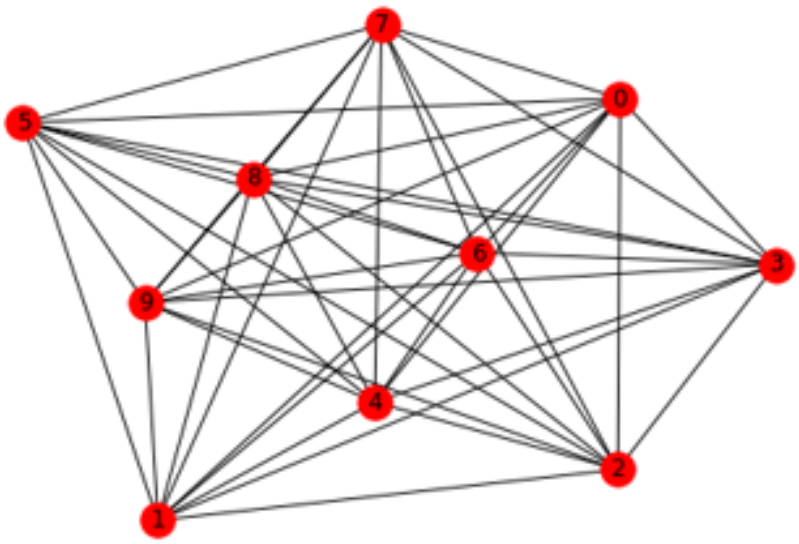

Figure 2: graph G

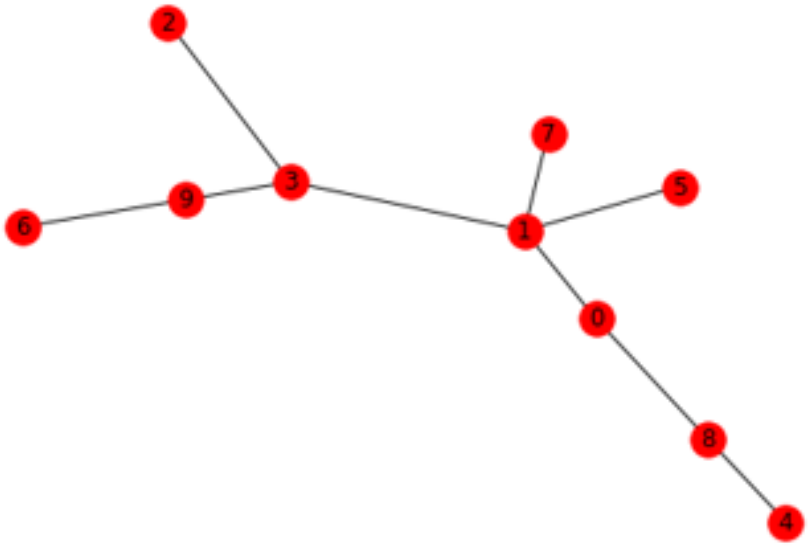

Figure 3: MST extracted from G

2. Find odd degree vertices in T and save them into a set.

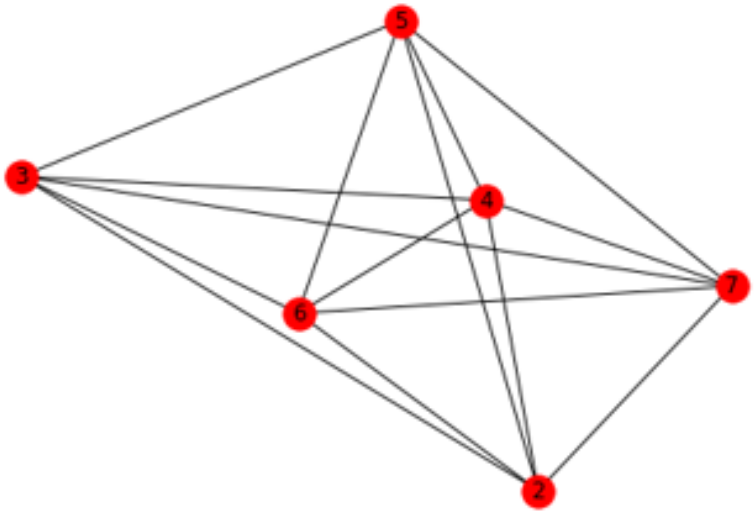

Figure 4: subgraph using odd degree vertices

3. From odd degree vertices, make minimal-weight perfect matching M, matching M in a graph G is a set of edges E without common vertices. Perfect matching is a matching which matches all vertices of the graph. That also means that every vertex of graph is incident to exactly one edge of the matching M, to do so we will proceed as following:
    - use a simple trick and substract each weight from a very large number (= max(weights) for example), and then replace the current weights by these values.
    - calculate maximum weight perfect matching for these new weights. In fact, if we find the maximum-weight matching in a graph with edge weights $Q - w_{ij}$, with $Q \geq max(w_{ij})$ then we have also found the minimum-weight matching in the graph with edge weights w_{ij}.

Figure 5: minimum weight perfect matching M

4. The fourth step is to unite matching M and minimal spanning tree MST to form an Eulerian multigraph $E = M \cup MST$.

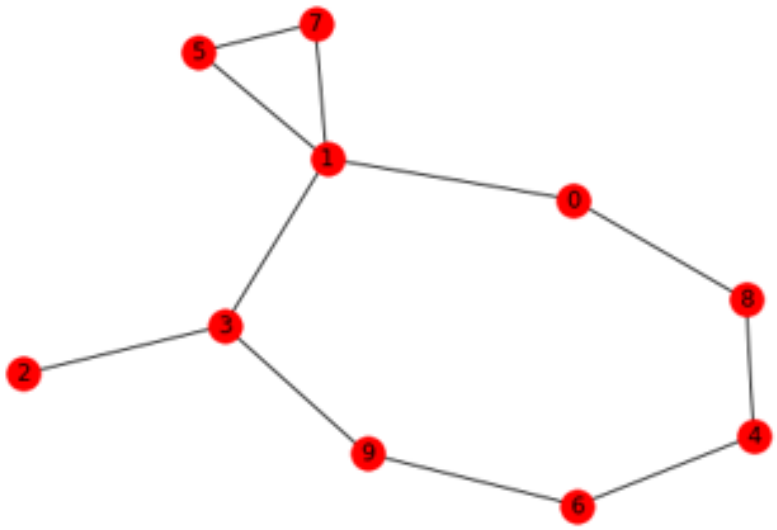

Figure 6: Eulerian tour

.

5. The last step in the algorithm is to remove repeated vertices that were included in the euler tour and get the hamiltonian tour.

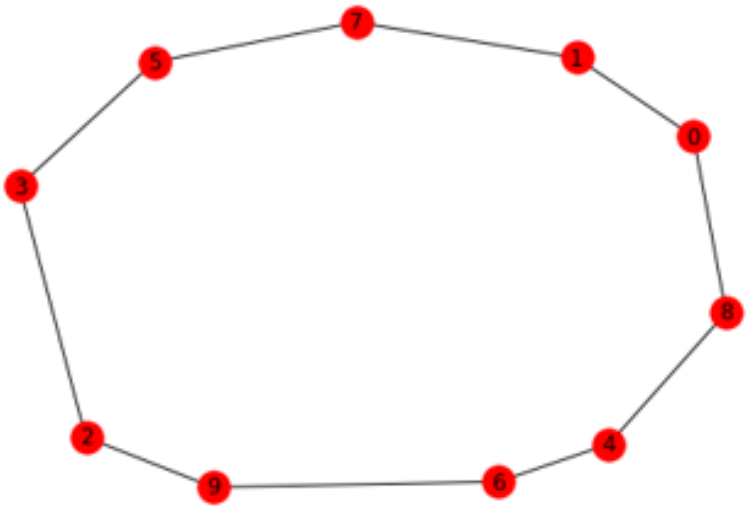

Figure 7: hamiltonian tour

networkx python-based package provides implemented algorithms for each step above, thus we just combined them to get the entire algorithm.

Christofides algorithm has the best approximation ratio that has been proven so far. The solution provided by Christofides guarantees to be withing 3/2 of the optimum solution, ie;if T is a christofides solution of TSP and T^* is a optimal solution then $cost(T) \leq 1.5cost(T^*)$

Proof of the 1.5 approximation ratio:

- Let H^* be the optimal TSP cycle for input (X, d) and let T be the minimum spanning tree, by definition we have $cost(T) \leq cost(H^*)$ because T contains only edges with minimum costs covering all vertices without necessity of being a complete closed tour, and deleting an edge from H^* gives a spanning tree of G.
- we know that the sum of degrees of all the vertices in a graph is equal to twice the number of edges, $\sum_{v \in V} deg(v) = 2 * E$
  and we have by definition:
  $\sum_{v \in V} deg(v) = \sum_{v \in V_{odd}} deg(v) + \sum_{v \in V_{even}} deg(v)$
  in the other side we know that $odd + even = odd$, it follows that
  $\sum_{v \in V_{odd}} deg(v)$ is even and $\sum_{v \in V_{even}} deg(v)$ is even
  Therefore, since the sum of the odd degrees is an even number, there must be an even number of vertices of odd degree. and this means that the minimum perfect matching M of subgraph

induced by odd degree vertices satisfy $cost(M) \leq 1/2cost(H^*)$

- E is an euler cycle in $M \cup T$ then $cost(E) = cost(T) + cost(M) \leq 1.5cost(H^*)$
- since the hamiltonian tour C^* is exctracted from E , it's obvious that $cost(C^*) \leq 1.5cost(H^*)$

2.2 comparative analysis of results

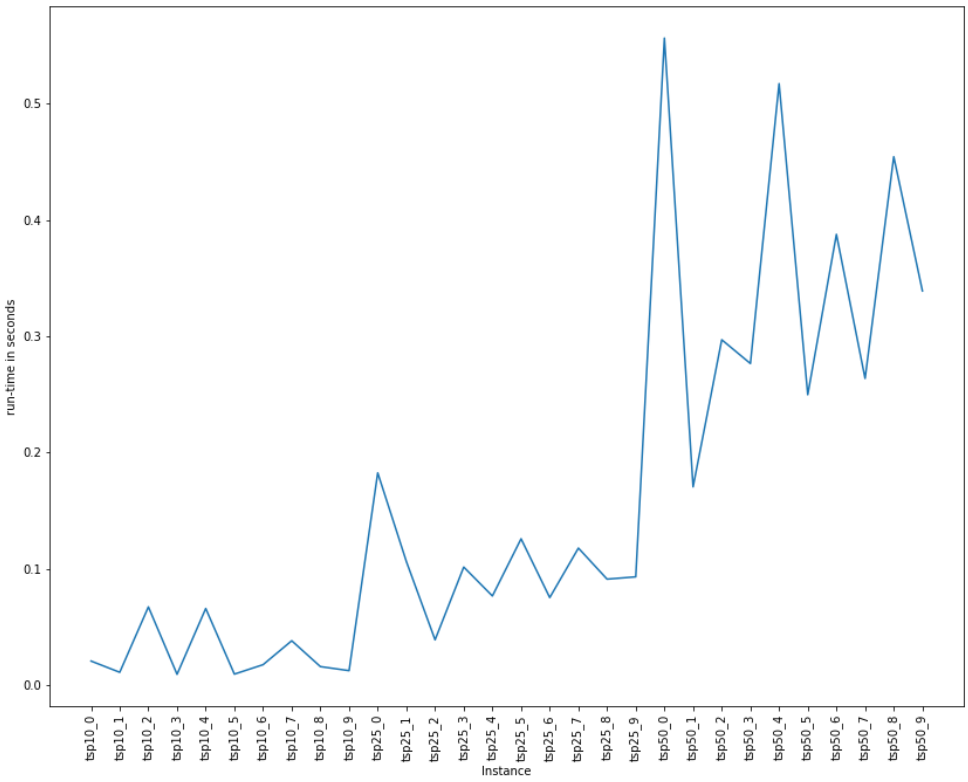

Figure 8: computational time of Christofides algo on random data

In Figure 8, We can see that for very small size data sets (tsp10-i), the christofides algorithm is highly efficient(around some milliseconds), Although time is increasing with the increasing volume of data (tsp25-iand tsp50-i), but, this difference is inconsiderable and can not be taken into account since it does not exceed 0.5 seconds.

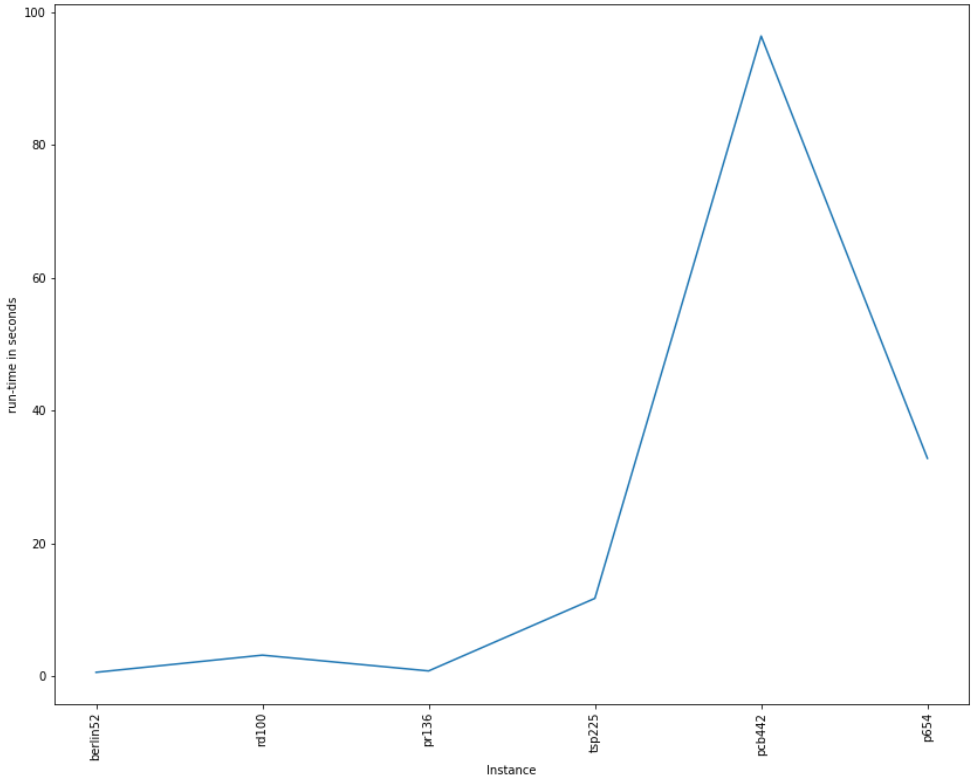

Figure 9: computational time of Christofides algo on some tsplib instances

In the second graph, Figure 9 , computational time is more significant since the data size becomes relatively big (at least 52 cities), however is still fairly efficient. as we can see almost 100 seconds for 442 nodes.

In contrary to FCG algorithm's performance, it's clearly that although not giving the optimal solution, christofides algorithm is much faster and provides very good (run-time/results quality) ratio Even for the complex instances containing hundreds of nodes. Which is well verified by the value of the the $\alpha-$approximation in the next summary table, we can easily note that it's widely lower than $3/2$,

Table 1

Instance	**FCG**		**Christofides**		α
	obj	time	obj	time	
tsp10-0	275.260	212 ms	293.996	7 ms	1.06
tsp25-0	415.87	267776 ms	453.96	51 ms	1.09
tsp50-0	-	∞	583.27	1895 ms	-

3 Conclusion

the work done during this project was very informative on two levels, firstly it was noticed that even if adding continuous soubtour elimination constraints to the basic formulation of TSP helps strengthening the linear relaxations and hence, the bounds used at each node of the branch and bound algorithm, the execution time remains very high because complexity is proportional to the overall dimension of the problem instance, Therefore the algorithm provided by Finke, Claus and Gunn is very expensive regarding the processing time. secondly the approximatif algorithm of Christofides seems to be more efficient especially for complex instances, this comparison between the two approaches regarding the runtime may be unfair and biased since the solver utilized is open source and performs poorly with large amounts of data, to get a rational judgment, it's likely better to test several optimization solvers known by their strength like CPLEX and GUROBI.

References

[1] Nicos Christodes. *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem*. Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976

[2] Gregory Gutin and Abraham P. Punnen, *eds. The Traveling Salesman Problem and Its Variations.* Combinatorial Optimization 12. New York: Springer, 2007. 830 pp

[3] http://www.cs.tufts.edu/ cowen/advanced/2002/adv-lect3.pdf

[4] http://www.cs.cornell.edu/courses/cs681/2007fa/Handouts/christofides.pdf