Problem set 6

Jonas Ishøj Nielsen, join@itu.dk

19/11/2021

Contents

1 TSP

1.1 DF TSP

The code can be seen in `DP_TSP.py`. The basic idea is to set node 0 as the start node and then create all the 2^{n-1} subsets of nodes 1,...,n-1.
The input for the funciton is: V, set and it returns the minimum cost of going from node 0 to node V visiting each node in the set exactly once along with the last visited node in the set. The function is recursive and the base case is when the set consists of a single element $\{x\}$ and it is stored in the cache as the (x, Ø) with value of the edge from 0 to x and parent 0.
For each subset, in increasing order based on length, for each node e in the subset the value of (e, subset$\backslash\{e\}$) by comparing the cost of (x, subset$\backslash\{e, x\}$) + edge cost $w_{e,x}$. The lowest value x is then the value of (e, subset$\backslash\{e\}$) and the value along with the parent x is stored in the cache.

Analysis
There are 2^{n-1} subsets and for each it goes through O(n^2) cache lookups and also O(n) for the list duplication, which could have been avoided if trying to optimize further.
The total is thus $O(2^n \cdot n^3)$.

1.2 Metric TSP approximation

The 2-approximation algorithm works by computing the MST, doubling the edges of the MST, finding a eulerian tour on that graph and then outputs the tour after applying shortcuts.
This method creates a 2-approximation since the following inequalities hold:

$$w(ALG) <= w(Q) = w(D) = 2w(T) <= 2 \cdot (w(T' + edge) = OPT - edge <= OPT$$

Where Q is the Eulerain tour in the Double edged tree D, and T is the MST.

The 3/2-approximation algorithm works by computing the MST T, adding to T the edges from a minimum cost perfect matching M on the odd degree vertices of T, finding a eulerian tour on that graph and then outputs the tour after applying shortcuts.
This method creates a 3/2-approximation since the following inequalities hold:

$$w(ALG) <= w(Q) = w(TUM) <= w(T) + w(M) <= (OPT) + (OPT/2) = (3/2)OPT$$

Both implementations can be seen in `MetricTSP.py`.

1.3 Comparison

The methods for comparison can be found in `TSP_Comparisons.py`. The 2-approximaiton on random points never gets twice the DF and the 3/2-approximation doesn't go above $3/2 \cdot$ DF. A graph of the cities of Denmark can be seen in figure 1 and a graph of random points can be seen in figure 2. In the graphs the black lines are the MST created and the red lines are the final path.

The DF picks the path:
['Copenhagen,', 'Greve,', 'Svendborg,', 'Faaborg,', 'Odense,', 'Horsens,', 'Vejle,', 'Kolding,', 'Aabenraa,', 'Esbjerg,', 'Holstebro,', 'Thisted,', 'Aalborg,', 'Randers,', 'Aarhus,', 'Grenaa,', 'Copenhagen,']

The 2-approx picks the path:
['Copenhagen,', 'Greve,', 'Grenaa,', 'Aarhus,', 'Horsens,', 'Odense,', 'Faaborg,', 'Svendborg,', 'Vejle,', 'Kolding,', 'Aabenraa,', 'Esbjerg,', 'Holstebro,', 'Thisted,', 'Randers,', 'Aalborg,', 'Copenhagen,']

The 3/2-approx picks the path:
['Copenhagen,', 'Greve,', 'Grenaa,', 'Aarhus,', 'Horsens,', 'Randers,', 'Aalborg,', 'Thisted,', 'Holstebro,', 'Esbjerg,', 'Kolding,', 'Aabenraa,', 'Vejle,', 'Odense,', 'Faaborg,', 'Svendborg,', 'Copenhagen,']

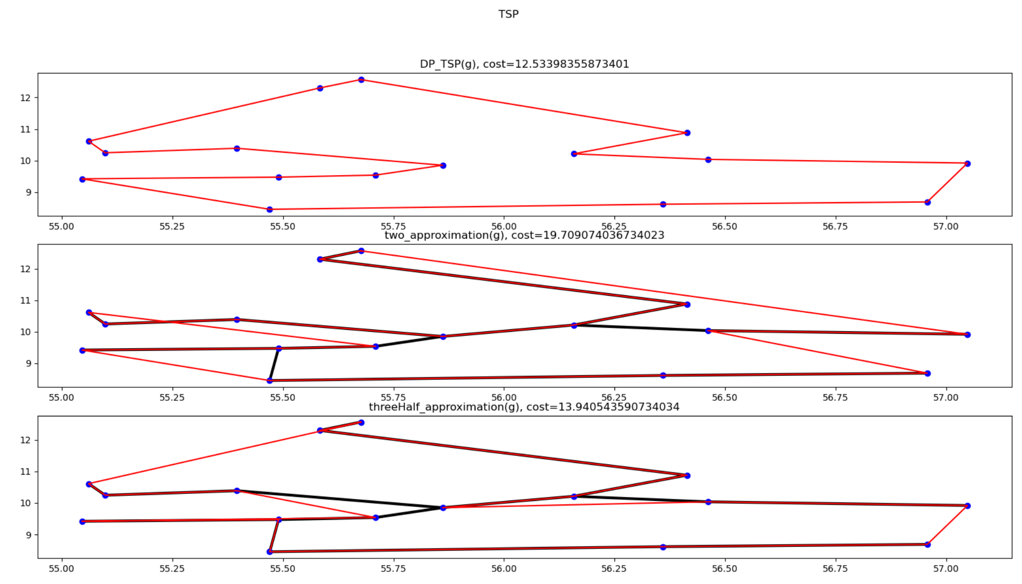

Figure 1: TSP of cities of denmark

2 Vertex cover and LP

2.1 2-approximation based on maximum matching

The implementation can be found in `VertexCover.py`. The algorithm works by computing a random maximal matching M and all the vertices in the matching.

2.2 LP-based rounding algorithm

The implementation can be found in `VertexCover.py`. It uses LP-rounding on value 0.5.

2.3 Conclusion of LP relaxation of ILP

The resulting vertices ALG is a vertex cover as for each edge (u,v) x_u+x_v>=1 meaning either u or v or both u and v gets rounded up and thus the edge gets covered.
The algorithm is a 2-approximation and thus: Cost(ALG)<=2OPT.

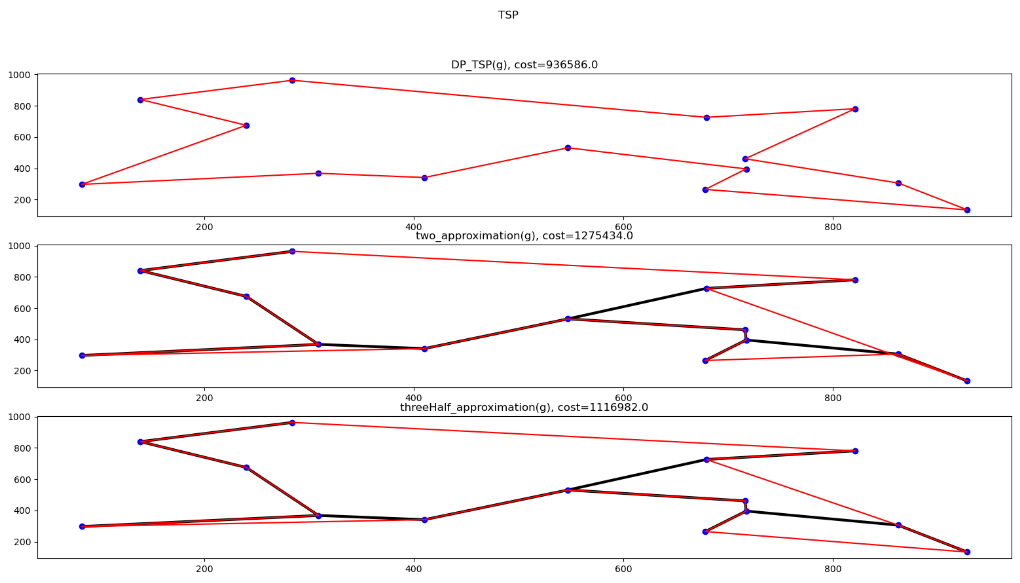

Figure 2: TSP of random points

2.4 Show OPT_{LP} can be arbitrary close to $2 \cdot OPT_{ILP}$

Since each variable X_v, when going from LP to ILP, gets increased with at most a factor of 2 it holds that:

$$ALG <= 2LP <= 2OPT$$

2.5 Exploratory exercise: find inputs

The LP algorithm outperforms the matching-based most of the time on graph:

$$[[0, 1, 1], [1, 0, 0], [1, 0, 0]]$$

And the matching-based outperforms the LP algorithm most of the time on graph:

$$[[0, 1, 1], [1, 0, 1], [1, 1, 0]]$$

3 Set cover ILP and LP

3.1 ILP for set cover

For the ILP the components are:
Variables: $x_s \in \{0, 1\}$ for s ∈ S=$\{S_1, ..., S_m\}$
Objective function: $f(x) = \sum_{s \in S} x_s$
Constraints: $\forall_{u \in U} \sum_{s \in S | u \in s} x_s >= 1$
Minimization problem
The LP relaxation of the ILP comes from removing the integer constraint $x_s \in \{0, 1\}$ and adding the constraint $0 <= x_s <= 1$ instead.

3.2 LP for set cover

The f-approximation is made by relaxing the ILP to get the LP, solving the LP and then do LP-based rounding. The implementation can be found in `SetCoverApproximation`
Each variable x_s is rounded up to 1 iff $x_s >= 1/f$ else it is rounded to 0.
The resulting sets ALG is a set cover as for element u $_{s \in S | u \in s} x_s >= 1$ meaning atleast one set containing u gets rounded up and thus the element gets covered.
The algorithm is a f-approximation since each variable X_s, when going from LP to ILP, gets increased with at most a factor of f it holds that:

$$ALG <= f \cdot LP <= f \cdot OPT$$