Advanced Algorithms - Mandatory Assignment 4

Frank Andersen [fand@itu.dk]

September 2020

Sheet 4

1. Experiments with TSP approximations:

   - Describe a dynamic programming algorithm for finding a TSP in a general graph G. The algorithm should run in $O(2^n p(n))$ time for some polynomial p and output a Hamiltonian cycle of minimum total weight in G. Argue for the correctness of this algorithm and implement it.

     Algorithm Given as input a directed edge weighted graph G(V,E)
     Let C be a dictionary with *tuple* of visited vertices as key, and a value of a another dictionary with a vertex as key and value tuple (infinity, list).
     C Lets us keep track of what we have visited and at what cost.

     ① Name vertices $(1 \dots N)$
     ② Pick a starting vertex $s \in V$
     ③ Compute all permutations, s.t. subsets be $S \subseteq V$ of size $(2 \dots N)$
     for each $s \in S$ where s not start
     compute $OPT(s,t)$, the minimal cost from *start* to *target* as the path in each permutation.
     $C \leftarrow min[OPT(S \backslash \{t\}, k) + cost(k,t)] \ \forall k \in S \backslash \{t\}$
     ⑥ Output $min[OPT(\{2 \dots n\}, t) + cost(start, t)] \ \forall t \in \{2 \dots n\}$

     Analysis The TSP-DP algorithm runs in $O(2^n E \log V)$, the cost lies in the enumeration of all subsets.
     My implemtation can be found here:
     `https://github.com/dkfrankandersen/ITU_AdvancedAlgorithms/blob/master/4fourth/DpTSP.py`

   - Implement the 2-approximation algorithm for Metric TSP discussed in the lecture. Your implementation may use inefficient data structures when implementing the MST algorithm.
     Solution
     My solution for 2-approximation Metric TSP, is applying the steps form lecture.
     ① setup graph using cities input data
     ② build MST from graph with Prim's algorithm
     ③ find Euler path (circuit) with Hierholzer's algorithm
     ④ remove reoccurring vertices from Euler path
     ⑤ return path and trip distance (using haversineDistance on lat/lon)

     My implementation can be found here:
     `https://github.com/dkfrankandersen/ITU_AdvancedAlgorithms/blob/master/4fourth/MetricTSP.py`

   - Compare the output of the 2-approximation algorithm and the DP when given as input the list of Danish cities on the next page. (Using the haversine formula or the Spherical Law of Cosines to compute the distances.)

Metric TSP	**DP TSP Tour**
Aabenraa	Copenhagen
Esbjerg	Grenaa
Holstebro	Aarhus
Thisted	Randers
Aalborg	Aalborg
Grenaa	Thisted
Randers	Holstebro
Aarhus	Esbjerg
Horsens	Aabenraa
Vejle	Kolding
Kolding	Vejle
Odense	Horsens
Faaborg	Odense
Svendborg	Faaborg
Greve	Svendborg
Copenhagen	Greve
Aabenraa	Copenhagen
1057358 meters	964225 meters

- For a bonus, plot the two tours from the previous part on a map.

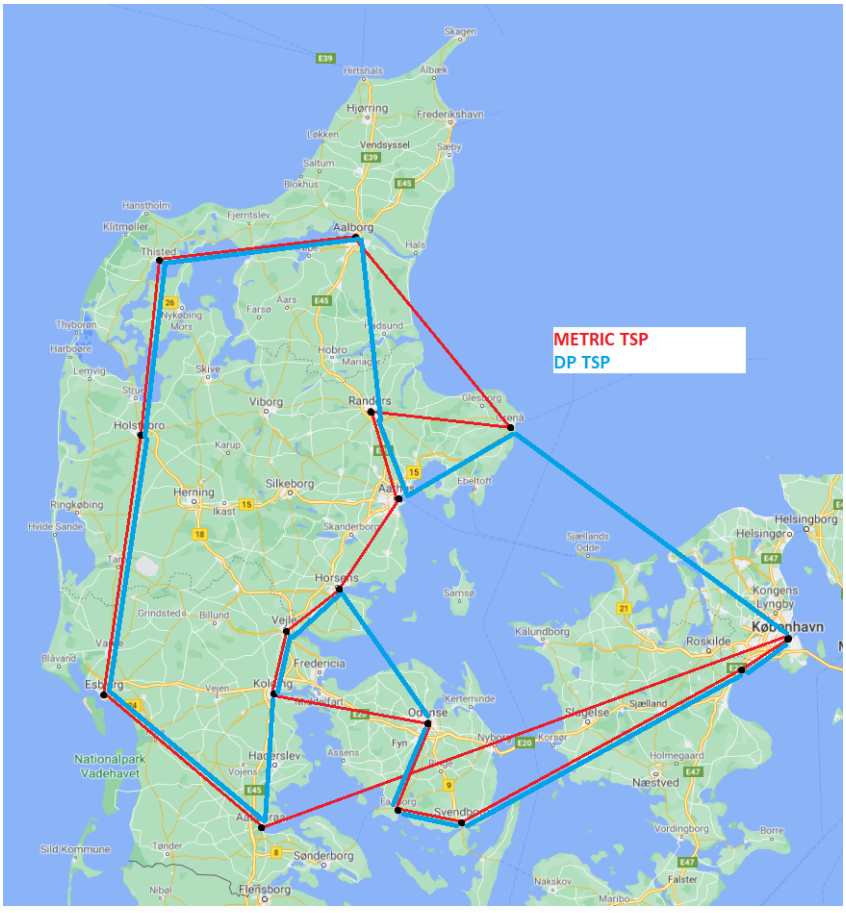

Figure 1: Comparison Metric vs. DP TSP

2. Experiments with LPs:

   - Implement the 2-approximation algorithm for Vertex Cover based on maximal matchings.
   - Implement the LP-based rounding algorithm for Vertex Cover using calls to an LP solver of your choice.

My implementation can be found here:
https://github.com/dkfrankandersen/ITU_AdvancedAlgorithms/blob/master/4fourth/VertexCover.py
https://github.com/dkfrankandersen/ITU_AdvancedAlgorithms/blob/master/4fourth/VertexCoverRando
py

- Compare the output of the 2-approximation algorithm and the LP-based algorithm on the following graphs and interpret the results:

  (a) A random graph with 200 vertices in which each edge is sampled independently with probability 0.9.

  (b) The attached 200-vertex graph (encoded as a list of edges).

  My LP Solver implementation is based the python PIP package cPlex, unfortunately I found out that it required licensed version to be used on larger problems, I managed to get a student licens and install it, but the implementation was different from cPlex so was unable to get it to work with in a reasonable time frame.
  Result on a small graph with only 7 vertices:

```
(py36cplex2) frank@FA-SKYLAKE:/mnt/e/repository/itu/msc/AdvAlg/ITU_AdvancedAlgorith
ms/4fourth$  python3 VertexCover.py < small_vc_graph.txt
Result from 2-approximation
(6, [1, 1, 0, 1, 1, 1, 1])

Result from LP-based algorithm
Problem Type: MILP
Version identifier: 12.10.0.0 | 2019-11-27 | 843d4de
CPXPARAM_Read_DataCheck                          1
Found incumbent of value 7.000000 after 0.00 sec. (0.00 ticks)
Tried aggregator 1 time.
MIP Presolve eliminated 12 rows and 6 columns.
All rows and columns eliminated.
Presolve time = 0.00 sec. (0.02 ticks)

Root node processing (before b&c):
  Real time             =    0.01 sec. (0.02 ticks)
Parallel b&c, 4 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.01 sec. (0.02 ticks)
Solution result is: integer optimal solution
[1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0]
```

  Result on the larger 200-vertex graph.

```
(py36cplex2) frank@FA-SKYLAKE:/mnt/e/repository/itu/msc/AdvAlg/ITU_AdvancedAlgorith
ms/4fourth$ python3 VertexCover.py < sheet4_graph.txt
Result from 2-approximation
(90, [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0,
1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1
, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
 1, 0, 0, 0, 1, 0, 0, 0])

Result from LP-based algorithm
Problem Type: MILP
CPLEX Error  1016: Community Edition. Problem size limits exceeded. Purchase at htt
p://ibm.biz/error1016.
Traceback (most recent call last):
  File "VertexCover.py", line 63, in <module>
    cplexLpSolver(g)
  File "VertexCover.py", line 53, in cplexLpSolver
    prob.solve()
  File "/home/frank/miniconda3/envs/py36cplex2/lib/python3.6/site-packages/cplex/__
init__.py", line 1329, in solve
    _proc.mipopt(self._env._e, self._lp)
  File "/home/frank/miniconda3/envs/py36cplex2/lib/python3.6/site-packages/cplex/_i
nternal/_procedural.py", line 692, in mipopt
    check_status(env, status)
  File "/home/frank/miniconda3/envs/py36cplex2/lib/python3.6/site-packages/cplex/_i
nternal/_procedural.py", line 236, in __call__
    raise CplexSolverError(error_string, env, status)
cplex.exceptions.errors.CplexSolverError: CPLEX Error  1016: Community Edition. Pro
blem size limits exceeded. Purchase at http://ibm.biz/error1016.
(py36cplex2) frank@FA-SKYLAKE:/mnt/e/repository/itu/msc/AdvAlg/ITU_AdvancedAlgorith
ms/4fourth$
```

  The 2-approximation returns a cover of 90, which I was unable to compare with the LP solvers result

for the larger graph..

- If your LP solver outputs an integral solution, what can you conclude?

3. In a Set Cover instance with universe $U = \{x_1, \ldots x_n\}$ and sets $S_1, \ldots, S_m$, the frequency of element x_i is the number of sets among $S_1, \ldots, S_m$ that contain x_i. Let f be the maximum frequency over all elements in U.

   (a) Give the ILP formulation of Set Cover and describe how to obtain an LP relaxation of this ILP.

   (b) Describe an LP-based rounding algorithm for Set Cover that results in an f-approximation.

   ILP formulation of Set Cover
   First we define the *objective function for minimising* as:

$$min \sum_{s \in S} wt_s \cdot x_s$$

   Subject to constraints:

$$\sum_{S \ni i} x_s \leq 1 \quad \forall i \in U$$

   We either pick or do not pick a set. $x_s \in \{0, 1\}$
   But all i elements of the universe U, must be cover by a set.

   To obtain an LP relaxation, we relax the integer constraint on ILP $x_s \in \{0, 1\}$ to be a value in the range $[0 \ldots 1]$
   Let f denote the frequency of the most frequent element.
   Pick all sets S for which $x_S \geq \frac{1}{f}$.

   Proof
   Let C denote the collection of sets picked by the algorithm.
   Choose an arbitrary element $i \in U$. Assume it belong to the sets $(S_1, S_2, \ldots, S_r)$, where $r \leq f$.
   Since $\sum_{j=1}^{r} \geq 1$, at least one of the $x_j \geq \frac{1}{r} \geq \frac{1}{f}$.
   Thus, the corresponding set will be picked and i will be covered, i.e., C is a valid cover.

   The rounding process increases x_S for each S by at most a factor of f.

   Thus, the cost of C is at most f times the cost of the optimal fractional cover and hence at most f times the cost of the optimal integer cover.