INTRODUCTION TO APX - HW1

TSP and friends

Every task is worth two points. Deadline: **10. 11. 2015 17:19.** Solutions can be sent via email or handed to me in person.

Exercise one

1. Find a class of graphs showing that the algorithm for metric TSP that uses the minimum spanning tree tour is no better than a 2-approximation.
2. Find a class of graphs showing that Christofides' algorithm for metric TSP is no better than a 3/2-approximation.

In both cases we look for an infinite class of graphs which has a strictly increasing number of vertices, i.e. we want $\{G_i | i \in \mathbb{N}\}$ so that $\forall i \in \mathbb{N}$: $|V(G_{i+1})| > |V(G_i)|$. That is a reasonable request; after all, if the tight bound would hold only for graphs with 20 vertices or less, and the algorithm would be 1.25-approximation for larger graphs, we would say that the algorithm is *asymptotically* a 1.25-approximation.

In this example we do not require tight bounds for small graphs, which means you can for instance prove that Christofides' algorithm on a graph G_i is no better than a $(3/2 - x_i)$-approximation, where $x_i \to 0$. In other words, if your example will „clearly show" that in the limit the bound is 3/2, you are done.

Solution. Our main gadget will be *a wheel* – n vertices $c_1, c_2, \ldots c_n$ connected in a cycle C and an extra vertex s (the center) with additional edges (called *spokes*) $sc_1, sc_2, \ldots, sc_n$.

In our first example, we set the edges of C to be of length 1 and the spokes to be of length 1. We choose the minimum spanning tree which takes all the spokes – of length n.

What is also important is the order of the vertices in the spanning tree, as we will use the order in the shortening part of the doubling algorithm. Since there was no restriction on the order in the exercise, we can assume a very bad order – for instance, we can take the order $s, c_1, c_3, c_5, c_7, \ldots$ followed by $c_2, c_4, c_6, c_8 \ldots$.

When we double the spanning tree, our total length will be $2n$. When we do the shortening, we note that we cannot shorten anything, because the distance $d(c_1, c_3) = 2$ is the same in the spanning tree and in the metric. (Here we used our assumption on the order of the vertices.)

The optimal TSP is of length $n + 1$, because the graph is hamiltonian. The ratio is $2n/n + 1$, which is 2 in the limit, as promised.

In our example for Christofides, we assume n is even, set the edges of C to be of length 1 and the spokes to be of length $1 - \varepsilon$ for some $0 < \varepsilon < 1/n$. The spanning tree is the same, except this time the spanning tree is of length $n - \varepsilon n$.

One of the minimum-cost matchings for the odd-degree vertices of our spanning tree is the following: we connect every odd-numbered vertex c_{2i-1} with the even-numbered vertex c_{2i}.

Our TSP tour is now of length $n + n/2 - \varepsilon n$, which is okay as the optimal TSP tour is of length $n+1-2\varepsilon$ (we could just traverse the cycle C of the wheel and visit s last). Except for the shortcutting, we have the desired bound $3n/2$.

Indeed, if we had a bad order on the vertices, we could visit s, c_1, c_2 and follow this by c_3, which we could shortcut. Therefore, we need to argue that our traversal visits some other vertex c_j afterwards. But, as we argued before,i we can choose the order of the vertices (the algorithm does not specify it), so we just choose a bad one.

Exercise two

Consider the following algorithm for asymmetric TSP on a graph $\vec{G}$ with a given distance function $d\colon \vec{E} \to \mathbb{R}^+$:

1. We find a directed circuit $\vec{C}$ in $\vec{G}$ which minimizes $\frac{\sum_{\vec{e}\in\vec{C}} d(\vec{e})}{|\vec{C}|}$.
2. We add all the edges $\vec{E}(\vec{C})$ to the solution.
3. We remove all vertices of $\vec{C}$ except one. We continue recursively until $\vec{G}$ is only a single vertex.

Your task is:

- Explain how we can achieve point 1 in polynomial time.
- Prove that the previous algorithm is an $\mathcal{O}(\log n)$-approximation for asymmetric TSP.

Solution: Point 1 in polynomial time. We use a dynamic program for computing the values $d^k(x, y)$, which will be equal to *the length of the shortest k-vertex walk between x and y.* Note that we allow vertices and edges to be repeated in this walk.

To compute $d^1(x, y)$ is easy – we just set it to be the distance $d(x, y)$. To compute it for larger k, we can use the recursion $d^k(x, y) = \min_z d^{k-1}(x, z) + d^1(z, y)$. It looks deceptively simple, but it actually can be computed this way. (Think for a moment why we cannot solve the Hamiltonian circuit problem in arbitrary directed graphs using this recursion.)

Alright, so what if we find some minimum $d^k(x, x)/k$ and the walk minimizing $d^k(x, x)$ is not actually a circuit? Here comes the trick: *if a walk attains the minimum average value but it is not a circuit, then it contains a shorter circuit that attains at most the same average value.*

This follows from the fact that if the closed walk W can be split into two closed walks A and B (which it always can if it is not a circuit itself), then $E[W] = E[A] + E[B]$ by linearity of expectation.

Therefore, our polynomial-time algorithm computes the table $d^k(x, x)$ and then finds the minimum value in it, going from smaller k to larger. The first occurence of the minimum value is exactly the value of the smallest circuit. To find the circuit itself is done by the standard dynamic programming argument.

Input format. First, let us discuss the actual setting of the problem. Asymmetric TSP is usually defined on a complete directed graph $\vec{G}$ with edges in both directions with a pseudo-metric $d\colon \vec{E}(\vec{G}) \to \mathbb{R}^+$ which satisfies a triangle inequality (but not symmetry).

A second way to specify input for asymmetric TSP is to give a directed graph $\vec{H}$ with edges and non-edges and an arbitrary function $d\colon \vec{E}(\vec{G}) \to \mathbb{R}^+$ with no constraint on d; in this case, we think of working with the *shortest path pseudo-metric* on the graph $\vec{H}$.

In the exercise there is an algorithm specified for an asymmetric TSP on a graph $\vec{G}$. We can observe that our setting must be the first one; with $\vec{G}$ a complete directed graph and d a pseudo-metric. This is true because if $\vec{G}$ were defined as in the second case, then the algorithm does not work at all.

To see this, consider an undirected K_5 and add a random direction to every edge (so that it is a directed complete graph but every edge is present in only one direction). Then, set every edge to be of distance 1.

If we choose any circuit of length four as our initial $\vec{C}$ in step 1, then removing $\vec{C}$ except for one vertex leaves us with an acyclic graph of length 2, and so the algorithm cannot find any more cycles and it fails.

By this argument we now know that if the algorithm is to be a $\mathcal{O}(\log n)$-approximation, it has to be defined for the first setting, where $\vec{G}$ is a complete directed graph with edges in both directions and d a full pseudo-metric with triangle inequality.

Note: If you have trouble with the exact setting of an exercise in the future, do not be afraid to ask me via email.

Approximation factor. First of all, because of our problem setting, we know that OPT is a Hamiltonian circuit.

In the first step of the algorithm, we know that $d(\vec{C_1})/|\vec{C_1}| \leq d(OPT)/n$ from the definition of $\vec{C_1}$. After deleting the vertices of $\vec{C_1}$ except one, we want to bound $d(\vec{C_2})/n_2$ by some function of OPT, where $n_2 = n - |\vec{C_1}| + 1$. Again, because of our setting, we can use the fact that the remaining (not-yet deleted) edges in $OPT \setminus V(\vec{C_1})$ can be completed into a Hamiltonian circuit OPT' on the smaller graph $\vec{G} \setminus V(\vec{C_1})$ by shortcutting. We get that $d(OPT') \leq d(OPT)$ from the triangle inequality.

As OPT' is a valid circuit, the minimal choice of $\vec{C_2}$ gives us that $d(\vec{C_2})/|\vec{C_2}| \leq d(OPT')/n_2 \leq d(OPT)/n_2$. The same will hold for $\vec{C_3}, \vec{C_4}$ and so on, with $n = n_1 > n_2 > n_3 > n_4 > n_k > 1$ being the sequence of the remaining vertices in the graph $\vec{G}$, as we remove one circuit after another.

We now proceed to bound $d(T)$, the total distance of the ATSP tour T of our algorithm. The first circuit was of $d(\vec{C_1})$, which we can express as $d(\vec{C_1}) = |\vec{C_1}| \cdot \frac{d(\vec{C_1})}{|\vec{C_1}|} \leq |\vec{C_1}| \cdot \frac{d(OPT)}{n}$. Similarly, $d(\vec{C_i}) \leq |\vec{C_i}| \cdot \frac{d(OPT)}{n_i}$ for $i \geq 2$. Putting the bounds together, we see our total cost is

$$d(T) \leq \sum_{i=1}^{k} |\vec{C_i}| \cdot \frac{d(OPT)}{n_i} = d(OPT) \cdot \left(\sum_{i=1}^{k} \frac{|\vec{C_i}|}{n_i} \right).$$

We will be done when we show that the sum on the right is upper bounded by $\mathcal{O}(\log n)$. This is an easy exercise in combinatorics, but we include it here for completeness.

We first split the sum into two parts:

$$\left(\sum_{i=1}^{k} \frac{|\vec{C_i}|}{n_i} \right) = \left(\sum_{i=1}^{k} \frac{1}{n_i} \right) + \left(\sum_{i=1}^{k} \frac{|\vec{C_i}| - 1}{n_i} \right)$$

The first part is clearly bounded from above by $\sum_{i=0}^{n-1} \frac{1}{n-i} \leq \log n$.

The second part seems to be more difficult to bound, because it contains $|\vec{C_i}| - 1$ in the numerator. Suppose now for a second that $|\vec{C_1}| = 4$ and $|\vec{C_2}| = 3$. This means that $n_2 = n - 3$. In this case, the initial part of the sum can be written as:

$$\frac{3}{n} + \frac{2}{n-3} + \ldots = \frac{1}{n} + \frac{1}{n} + \frac{1}{n} + \frac{1}{n-3} + \frac{1}{n-3} + \ldots$$

The right-hand side of the last equation is bounded from above by $\sum_{i=0}^{n-1} \frac{1}{n-i}$, which is again at most $\log n$, and we have our total bound of $\mathcal{O}(\log n)$.

EXERCISE THREE In the *Steiner tree problem* we get on input a connected undirected graph $G = (V, E)$, an edge cost function $c : E \to \mathbb{R}^+$, and finally a list of *terminals* $S \subseteq V$. A feasible solution to our problem is any subset of edges $E' \subseteq E$ so that the graph $G' = (V, E')$ has all the terminals in one connected component. We aim to minimize the cost, i.e. $\sum_{e \in E'} c(e)$. Your task is to design a 2-approximation algorithm.

Hint: The graph does not need to satisfy the triangle inequality. First, think about the case when it does (it should be easy then). To solve the general case, try to use some of the techniques from the TSP approximation.

Solution. We first observe that an optimum Steiner tree is always a tree, as there is no reason to consider cycles or other graphs other than trees.

After learning this, our first idea should be some sort of spanning tree – either on all vertices or just on the terminal vertices. Clearly, the former idea is a bad one, as there can be a very distant vertex that is not part of the terminals and any optimum avoids it.

We now claim that the minimum spanning tree on just the terminal vertices is a 2-approximation in the metric case.

Okay, suppose that we have some optimal tree OPT that also includes some other vertices. We wish to bound our cost by $2c(OPT)$, but we will do it in an indirect way. Consider a DFS traversal of the tree OPT. From the DFS traversal, we create a list L of terminals by adding every terminal when we first encounter it.

Now, we think of L as a path P_L on the terminals – if the list is $[t_1, t_2, t_3]$, we go from t_1 to t_2 and from t_2 to t_3. We claim that $c(P_L) \leq 2c(OPT)$. This follows from the fact that a DFS traversal visits each edge of the tree OPT at most twice, and our shortcuts $t_1 - t_2$ and $t_2 - t_3$ are shorter than the paths in OPT by triangle inequality.

To finish the metric case, we just note that P_L is a valid spanning tree on the terminals, and so the minimum spanning tree has to be even shorter.

The general case works basically the same way, we just need to work with the *shortest path metric*.

EXERCISE FOUR Consider a cubic 2-edge-connected graph G. The word *cubic* means that every degree of the graph is equal to 3. The word *2-edge-connected* means that the graph does not contain a bridge, which is an edge whose removal disconnects the graph. The graph is not weighted, so all the edges have distance one.

1. Show that any such graph has a TSP tour of length at most $4|E|/3$.
2. Prove that the point $(1/3, 1/3, 1/3, \ldots, 1/3)$ lies always in the perfect matching polytope of G.
3. Prove that for G there exists a set of perfect matchings $M_1, \ldots M_k$ of G and a corresponding set of constants $\lambda_1 \geq 0, \lambda_2 \geq 0, \lambda_3 \geq 0, \ldots, \lambda_k \geq 0, \sum_{i=1}^{k} \lambda_i = 1$ having the following property: if we take any perfect matching M_i randomly with probability λ_i, then for *every edge e in the whole* $E(G)$ it holds that $P[e \in M_i] = 1/3$.

The perfect matching polytope is this one:

$$\begin{aligned}
&\forall v \in V: && \sum_{e=vx} x_e = 1\\
&\forall S \subsetneq V, S \neq \emptyset, |S| \text{ odd}: && \sum_{e \in E(S, V\setminus S)} x_e \geq 1\\
&\forall e \in E: && x_e \geq 0
\end{aligned}$$

Solution.

1. If we take every edge once plus a perfect matching, we take $|E| + |E|/3 = 4|E|/3$ edges and the graph is now connected and has all degrees even. Why does this graph always have a perfect matching? It is more or less an exercise from graph theory, but let me answer with a trick: because the perfect matching polytope is non-empty! If it is nonempty, then there is at least one vertex of the polytope, which is a perfect matching. And why is it nonempty? Because $(1/3, 1/3, \ldots, 1/3)$ lies inside, as we claim in our next section.
2. We verify the conditions of the matching polytope. The first equality is clear: around every vertex we total $1/3 + 1/3 + 1/3 = 1$, as the graph has degree 3.
   The second inequality needs a bit more thought. We look at any set S of odd size and the edges going out of it. First of all, the sum on the edges cannot be equal to $1/3$, because then there would be a bridge – which is forbidden by 2-edge-connectedness. So why not $2/3$? It turns out that we have $|S|$ odd, and the sum of the degrees in this set is $3|S|$, which is an odd number – so there has to be an odd number of edges going out of S. Therefore at least three edges have to be going out, and the total sum is at least 1.

3. This is just an exercise in equivalent definitions. We know that $P = (1/3, 1/3, \ldots, 1/3)$ is a point in the perfect matching polytope of our graph. From a linear optimization course, we know this is equivalent to P being a convex combination of some (finite) subset of vertices of the perfect matching polytope.
   Now, the vertices of the perfect matching polytope are just perfect matchings (we know this again from a linear optimization course). And what does being a convex combination mean? It means that there are $\lambda_1, \lambda_2, \ldots, \lambda_n \geq 0$ with $\sum_i \lambda_i = 1$ such that $P = \sum \lambda_i V_i$ for the aforementioned vertices/perfect matchings.
   We therefore have that there is a set of perfect matchings and associated lambdas so that P can be thought of as a convex combination of those. But those positive λ_i that sum up to 1 can be also used to define a probability distribution – indeed, if we take any V_i (a perfect matching) with probability λ_i, we get the distribution/point P. Finally, we observe that a probability of an edge belonging to this distribution P is precisely its coordinate – and all those are equal to $1/3$, as we wanted.

Notes. This exercise is interesting because of the three following reasons:

- The arguments presented here is a starting step that leads to an improved approximation algorithms that return a tour of length at most $4n/3 - 2/3$ on subcubic bridgeless graphs (there are more steps, though).
- You may have considered finding 3 perfect matchings so that every edge is covered exactly once; this would indeed prove the task 3.3. However, you cannot find them – in fact, there is a large family of cubic 2-edge-connected graphs (called *snarks*) which cannot be partitioned this way.
  After learning of snarks, you could have considered finding 6 perfect matchings so that every edge is covered exactly *twice*. However, whether this is possible is an important open problem called the *Berge-Fulkerson conjecture.*
- We have proven the equivalence of probability distributions of points and convex combination of points. Our use of this was very simplistic, but this equivalence is a very useful fact which is used as a basis of *Lasserre hierarchies of semidefinite programs*, which is a cutting-edge tool in the theory of approximation algorithms.
  To define it in very broad terms, a hierarchy of level r creates a spectrum of *pseudodistributions* which behave like distributions only if you look at probability events of size at most r. A hierarchy of level n is equivalent to a full probability distribution on the vertices of the polytope, but it is often exponentially hard to compute – whereas $O(1)$ levels can be computed in polynomial time and they often suffice to compute some sort of approximation.