Lecture 1: Introduction, TSP, ATSP

16th January, 2015

Welcome to EO 249, "Approximation Algorithms". What are we going to learn in this course? A very short answer: algorithms with provable guarantees. "But wait, I have already done that in my design and analysis of algorithms course (which is a prerequisite for this course, I believe). Are you going to go over all of it again?", I hear you shout. Notwithstanding the not-so-obvious fact the repeating all that stuff may be worthwhile, the answer is no. We will look at new kinds of algorithms and proofs than what you probably did in the DAA course – although there will be some intersection. The main difference is that in this course we will *primarily* be looking at problems which are **NP-hard**, while in the former you mostly studied problems which are in **P**. Nevertheless, we will be designing polynomial time algorithms, and since we won't be cashing in a million dollars, our algorithms will not solve all instances of the problems exactly. But rather **approximately**, and that explains the name of the course.

1 Optimization Problems and Approximation Algorithms.

Formally, an optimization problem Π consists of instances $\mathcal{I}$ and *feasible solutions*, $\mathcal{S}$, to these instances. Each solution $S \in \mathcal{S}$ is associated a cost $c(S)$. To work with an example, the optimization problem *minimum spanning tree (MST)* has instances described by undirected graphs and costs (positive rationals) on the edges. Solutions are the spanning trees of the graph, and the cost of each solution is the sum of costs of the various edges. An optimization problem Π is called a *minimization* problem if one wishes to find minimum cost solutions for every instance; it is called a *maximization* problem, if one wishes to find maximum cost (in which case cost is often called profit) solution.

An algorithm $\mathcal{A}$ for an optimization problem Π, maps every instance $\mathcal{I}$ to a feasible solution $S \in \mathcal{S}$. So an MST algorithm takes an undirected graphs with edge costs and returns a spanning tree. Algorithm $\mathcal{A}$ is called the **optimal** algorithm for a minimization (respectively, maximization) problem Π if for each instance $\mathcal{I}$, the algorithm returns the solution $S \in \mathcal{S}$ of the *minimum* (respectively, *maximum*) cost. As mentioned above, for many optimization problems which are NP-hard, we do not expect to find optimal algorithms. This motivates the following definition.

Definition 1. *An algorithm $\mathcal{A}$ for a minimization problem Π is an α-factor approximation algorithm for Π, for some $\alpha \geq 1$, if for every instance $\mathcal{I}$, the solution $S_{\mathcal{A}}$ returned by $\mathcal{A}$ satisfies*

$$c(S_{\mathcal{A}}) \leq \alpha \cdot \min_{S \in \mathcal{S}} c(S)$$

An algorithm $\mathcal{A}$ for a maximization problem Π is an α-factor approximation algorithm for Π, for some $\alpha \leq 1$, if for every instance $\mathcal{I}$, the solution $S_{\mathcal{A}}$ returned by $\mathcal{A}$ satisfies

$$c(S_{\mathcal{A}}) \geq \alpha \cdot \max_{S \in \mathcal{S}} c(S)$$

It should be clear that for any problem Π, we would like to design *polynomial time* algorithms with α as close to 1 as possible. Can this be done? Throughout this course we will study this question, and see the extremely rich field which has arisen in the last 30 years or so!

2 The Travelling Salesman Problem (TSP)

Here is a famous NP-hard, optimization problem. The input is a set of n points denoted $\{1, 2, \ldots, n\}$. For each pair of distinct points i, j we have a cost $c(i, j)$. This defines an instance of the problem. The solution to the problem is a 'tour' of these n-points. A tour is nothing but a permutation σ of $\{1, 2, \ldots, n\}$ – the order in which a salesman is supposed to visit these points. The cost of a tour σ is defined to be

$$c(\sigma) = \sum_{i=1}^{n} c(\sigma(i), \sigma(i+1))$$

where $\sigma(n+1)$ is just (bad but convenient) notation for $\sigma(1)$. Henceforth, we stop being formal about instances and solutions and rely on the 'informal-to-formal' compiler in our students' heads.

This problem is NP-hard. You may remember the Hamiltonian cycle problem which was (and still is, and will be) NP-complete; the TSP definitely smells Hamiltonian. In fact, as your first exercise, we will ask you to show that it is NP-hard to find a polynomial time α-approximation algorithm for *any* α! Rather, we will now make an assumption on the instances of the problem and design an algorithm for those. We assume the costs $c(i, j)$ form what is called a *metric*. For any three points i, j, k, we assume

$$c(i, k) \leq c(i, j) + c(j, k)$$

We will see that this assumption will lead to very good approximation algorithms.

Eulerian Walks of Graphs. Given a (multi)-graph $G = (V, E)$, an Eulerian walk of the graph is a walk π where each edge of the graph is traversed **exactly** once and we return to the starting vertex. Eulerian walks again seem related to tours and Hamiltonian cycles, and they will be the main ingredient of our algorithms today. In one crucual respect they differ: it is extremely easy to figure out if a graph G has an Eulerian walk or not.

Theorem 1. *A multigraph G has an Eulerian walk iff it is connected and every vertex of G has even degree.*

Shortcutting: from walks to tours Any walk can be converted to a tour by just 'refusing' to go to a vertex twice (except the last vertex). Formally, given π, we obtain σ by deleting all repetitions of any vertex except $\pi(\text{last})$. We claim that $c(\sigma) \leq c(\pi)$ which is nothing but $c(E)$. This follows from the triangle inequality property. Consider we delete v from u, v, w in π. We claim that the new walk has only less cost – then we can inductively proceed. But this is equivalent to saying $c(u, w) \leq c(u, v) + c(v, w)$.

How to get a good G?: Lower bounds on OPT. So now our attack is clear – we need to find a graph G which is Eulerian, that is, all degrees are even, and furthermore the sum of cost of all edges in G is $\leq \alpha \cdot OPT$, where OPT is the cost of the minimum cost tour. Then we would get an α-approximate algorithm which is eulerian walk + shortcut.

But how do we get such a graph? We don't know OPT – after all that is what NP-hard means, right? To by pass this, what one does is instead of working with OPT, one works with **lower bounds** on OPT. Suppose there is a quantity L which we can indeed compute in polynomial time **and** we can **prove** that $L \leq OPT$, then we would like to actually ue L as our benchmark rather than OPT. Sounds crazy, right? Let's see how we can do this for TSP.

Lower bounds on OPT for TSP. We will show use such two lower bounds. First, consider the minimium spanning tree on the complete graph formed by n points with the costs $c(i,j)$. We claim that $MST \leq OPT$. Why?

Second lower bound seems even weaker. Let O be a subset of the points such that $|O|$ is even. Consider the complete graph on O with the costs $c(i,j)$. Let M be the minimum cost perfect matching of O. We claim that $c(M) \leq OPT/2$. Why?

Approximation algorithms for TSP. Now we are armed with everything we need. First we show a 2-approximation.

- Let T be the minimum spanning tree on the complete graph with costs $c(i,j)$. Note $c(T) \leq OPT$.
- Consider the *multi*-graph $2T$ where we **duplicate** every edge of T.
- **Note:** $2T$ is Eulerian. Let π be an Eulerian walk and σ be the shortcutted tour. Return σ.

We claim $c(\sigma) \leq 2OPT$. This is almost immediate – $c(\sigma) \leq c(E(2T)) = 2c(T) \leq 2OPT$.

Now we show a slightly cleverer way to make the MST Eulerian than just duplicating edges. Let O be the set of odd-degree vertices of T. Claim: $|O|$ is even. Why? Let M_O be the minimum cost perfect matching in the graph induced by O. Note that $T + M_O$ is Eulerian. Now in bullet point, replace $2T$ by $T + M_O$. We claim that the σ we get is a 1.5 approximation. This is because

$$c(\sigma) \leq c(T + M_O) = c(T) + c(M_O) \leq OPT + OPT/2 = 1.5OPT$$

(Big-Kahuna) Open Problem #1. Find an 1.49-approximation algorithm for metric TSP.

3 Asymmetric TSP

Till now we have implicitly assumed that we are dealing with an undirected graph. That is, we assumed $c(i,j) = c(j,i)$ for all pairs i,j. (Can you find where we used this?) In the asymmetric TSP, one doesn't have this assumption. One still assumes the metric assumption.

Instead of talking of graphs, we now talk of directed graphs. Eulerian walks still make sense except we need to traverse every directed edge exactly once. The corresponding theorem regarding Eulerian walks in directed graphs is this.

Theorem 2. *A multi-digraph G has an Eulerian walk iff it is strongly connected and every vertex of G has in-degree equal to out-degree.*

Our knowledge regarding ATSP is considerably less compared to that of TSP. In particular, no constant factor approximation is known for ATSP and neither has it been ruled out. We describe a $\log n$-factor approximation algorithm. To do this, first we need to describe a lower bound.

Lower bound on OPT: Cycle cover Given a directed graph, a cycle cover is a collection of non-trivial **vertex disjoint** cycles $\{C_1, \ldots, C_k\}$ which span all the vertices. That is, each vertex is in exactly one cycle. The cost of a cycle cover is the cost of all the directed edges in the cycles. Let's use $CYC(G)$ to denote the cost of the optimal cycle cover of a digraph G. Obviously, $CYC(G) \leq OPT$, since the ATSP tour is a valid cycle cover. In fact, for any subset $O \subseteq \{1, \ldots, n\}$, if G_O denotes the complete digraph induced by these vertices, then $CYC(G_O) \leq OPT$ as well, because of the metric property. The crux of the ATSP algorithm is the following theorem.

Theorem 3. *There is a polynomial time algorithm to find the minimum cost cycle cover in any digraph.*

Now we are ready to describe the approximation algorithm for ATSP.

- Initially $F = \emptyset$.
- Repeat till $|G| = 1$:
  - Find the minimum cycle cover $\{C_1^*, \ldots, C_k^*\}$ of G.
  - Add the edges of the cycles to F.
  - Pick an **arbitrary** representative from each cycle to form the set O.
  - Let $G = G_O$.
- Claim: F is an Eulerian digraph. Let π be the Eulerian walk and σ the shortcutted tour.
- Return σ.

First let's establish the claim in the fourth bullet. Since we only add cycles to F, the in-degree equalling out-degree property holds. To establish connectivity, pick some arbitrary pair of vertices u and v. If u and v lie in the same cycle of the first cycle cover, then there is a path from u to v on the cycle. Otherwise, let r_u and r_v be the representative vertices picked in O. Note there is a path from u to r_u and a path from r_v to v in F. Since we repeat the same procedure on G_O, by induction , there is a path from r_u to r_v in F. All together, we have a path from u to v.

Theorem 4. *The above algorithm is a $\log_2 n$-factor approximation algorithm for ATSP.*

Proof. Let r be the number of times the inner cycle is run. Since $CYC(G_O) \leq OPT$ for every O, and so $|F| \leq r \cdot OPT$. At the end of each loop, the size of $|G|$ decreases by at least a factor of 2 (since each cycle is of length ≥ 2), $r \leq \log_2 n$. □