Lecture 10

1 Writing a report

We had some discussion what should be in a good report. Some items.

- Recall algorithm briefly.
- Describe what you did.
- What algorithms are bottlenecks?
- What parameters did you choose?
- Did it produce factors at expected rate?
- How did it compare to others (say GMP for GCD e.g.)?

Instruction were not very clear but it is probably good to be ready to discuss a questions along this line when meeting with the TA.

2 Facts

We need the following facts for our algorithm on TSP. If you recall the algorithm discussed in the end of last class first finds a min cost spanning tree and then a min cost matching of the vertices of odd degree in this spanning tree.

- Min cost matching can be found in polynomial time.
- Cost of this matching is $\leq \frac{1}{2}OPT$.

These two facts imply efficient $\frac{3}{2}$-approximation algorithm.
Christofides: 1976, Still best known.

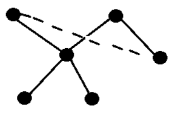

Figure 1:

3 Matching in graphs

Matching in graphs is polynomial time and let us recall some facts.

3.1 Bipartite graphs (fig.2)

A graph is bipartite if the vertices are divided into two sets and each edge has its endpoints in two different sets. In this case it easy to find a matching by augmenting paths.

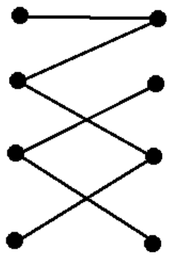

Figure 2:

3.2 General graphs

Matching on general graphs is a mess but still polynomial time. This implies that Christofides algorithm is nice in theory but not very practical.

Let me prove that cost of matching is $\leq \frac{1}{2}$ cost of tour. Consider the optimal tour in Figure 3 (we do not know it but we can use it in the analysis). The vertices colored are the odd degree vertices in min-cost spanning tree.

Look at two matchings where we match next to each other optimal tour. The sum of the costs of these matchings is at most the cost of the tour. This implies that the cost of the cheaper of these two matchings is at most half the cost of the tour and thus the cost of the min cost matching is at most half the cost of the optimal tour.

Let us analyze the cost of the found tour. It is at most the cost of the Euler tour (might be shorter due to shortcuts) and this cost is at most cost of the matching plus the cost of the min cost spanning tree. The latter is bounded by OPT and the former by $\frac{1}{2}OPT$ and thus we get the total bound $\frac{3}{2}OPT$. Done!

How about points in the plane and Euclidean distances? We have the following theorem.

Theorem 1 (Sanjeev Arora, Joseph Mitchell) $\forall \epsilon > 0 \exists$ *polytime algorithm that solves TSP in plane within a factor* $(1 + \varepsilon)$.

The running time of improved versions is $O(n(logn)^{O(\frac{1}{\varepsilon})})$ so it quasi linear which is great in theory. Unfortunately it seems hopeless in practice. The idea of the algorithm is to divide plane into small pieces and do dynamic programming. The interested student can look up the papers for details.

Note Arora and Mitchell found very similar algorithms independently in 1997 (with worse running times $O(n^{O(\frac{1}{\varepsilon})})$). The got the Gödel prize for this result. Let us recall a two items that made Kurt Gödel famous.

Gödel-numbering: Integers can code anything. This was used to prove that for any recursive[1] axiomatization of integers there are true statements that are not provable.

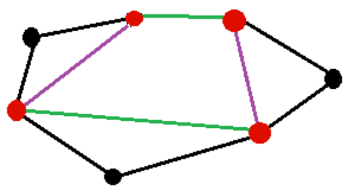

Figure 3:

3.3 Project 2

Task: Solve TSP as well as you can ≤ 1000 vertices. There are are a total of 50 instances to be solved in 2 seconds per instance in Kattis. Some ideas:

- Do something random to get a tour.
- Do more or less greedy local optimization.

[1]There is an algorithm that recognizes an axiom.

3.4 The bread and butter algorithm

The algorithm 2-OPT Find (i, j) and (k, l) in such that in current tour $i \Rightarrow j$, $k \Rightarrow l$, while it is cheaper to have $i \Rightarrow l$, $j \Rightarrow k$.

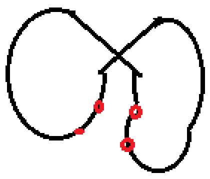

Figure 4:

Try not to use n^2 time to look for good swaps. This happens if you do it naively as there are n choices for each edge. The simple observation that for a switch to be profitable it must be that l is closer to i than j or that k is closer to j than to i. Already this limits the number of choices.

3.5 A great general tool

Linear programming

$$max \sum_{i=0}^{n-1} c_i x_i$$

under $Ax \leq b$,i. e.

$$\sum_{i=0}^{n-1} a_{ij} x_i \leq b_j, j = 0.....m-1.$$

This is to optimize a linear function under linear constraints.

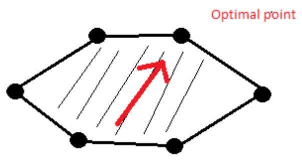

Figure 5:

This is an example with $n = 2, m = 6$, $Ax \leq b$
Feasible region "A polytope in $\mathbf{R}^n$".
Comes in variants min instead of max. This is the same as

$$max \sum c_i x_i$$

is equivalent to

$$min - \sum c_i x_i$$

Requiring $x_i \geq 0$. Just special type of inequalities

Only allowing $x_i \geq 0$ and only equalities

$$\sum_{i=0}^{n-1} a_{ij} x_i = b_j, j = 0....m-1$$

This also gives the same set of problems as we can, for each general inequality introduce a new variable and replace

$$\sum a_{ij} x_i \leq b_j$$

by

$$\sum a_{ij} x_i + y_j = b_j$$

together with $y_j \geq 0$.

3.6 LP = Linear Program

Facts:

- In theory solving an LP in polynomial time.
- In practice it is quite fast. (Nice ones with 10^5 variables are ok)

3.6.1 First algorithm

Simplex: Dantzig $\sim$ 1947
Follow the corners of the polytope.

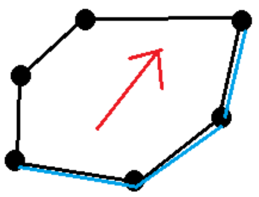

Figure 6:

Fast in practice, slow in theory. Might take 2^n time.

3.6.2 Ellipsoid algorithm

Provable polynomial time, hopeless in practice.

3.6.3 Interesting information

Karmarkar: Interior point method mid 1980s.
Combination of "Follow gradient avoid boundary". Provable polynomial time.

Why is LP easy?

- No local optima.
- Any feasible point if you look towards to optimal point you improve all the way.

Using LPs for hard combinatorial problems.
General recipe:
Introduce variables $x_i \in \{0,1\}$ to code problem, relax to $x_i \in [0,1]$ and write down as many inequalities as possible. In the case of TSP we can have a variable x_{ij} that takes the value 1 if the tour goes from i to j and $x_{ij} = 0$ otherwise. This implies that we want to minimize $\sum_{i,j} x_{ij}d_{ij}$, where d_{ij} is the distance from i to j. Note that this is a nice linear function. We relax $x_{ij} \in \{0,1\}$ to $x_{ij} \in [0,1]$ which is a linear constraint as this is $(0 \leq x_{ij} \leq 1)$.

The condition that the tour leaves i once is $\sum_j x_{ij} = 1$ and that it enters once is $\sum_j x_{ji} = 1$, also nice linear constraints.

More inequalities? Does this give a tour even if $x_{ij} \in \{0,1\}$. No as we also need the tour to be connected. To this end we can introduce $\sum_{i \epsilon S, \neg j \epsilon S} x_{ij} \geq 1$ for any subset, $S \subseteq [n]$. One problem is that these are exponentially many constraints but we will discuss that next time.

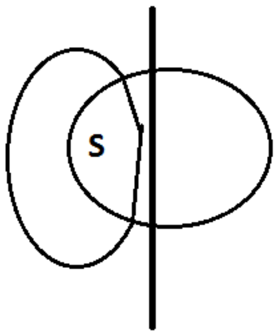

Figure 7: