DD2440 Lecture 9
NP-hard optimization problems
2014-11-19

Lecturer: Johan Håstad
Scribe: Sandra Liljeqvist

Some examples of NP-hard problems

Traveling Salesperson (TSP)

We have n cities with distances d_{ij}. Find shortest route that visits all cities once and returns to the start.

$$\text{Two cases: } \begin{cases} d_{ij} = d_{ji}, & \text{symmetric.} \\ d_{ij} \neq d_{ji}, & \text{asymmetric.} \end{cases}$$

Triangle inequality: $d_{ij} \leq d_{ik} + d_{kj}$

Special case: Euclidean distances in plane.

Max-Clique

Given a graph with n nodes, find maximal number that are all connected pairwise.

Max-Cut

For a graph with n nodes, split the graph into two pieces so that as many edges as possible are cut.

Max-3 SAT

Given a 3-CNF formula
$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge ...$
find an assignment that satisfies the largest number of clauses.

Remember that 3Sat, the question of whether one can satisfy all the clauses is one of the first NP-complete problems defined by Cook. The maximization problem is at least as hard and hence NP-hard. The other problems are in fact also NP-hard.

Conditions for all NP-complete problems

(Unless NP=P, which is unlikely)

We can not have an algorithm that:

1. always runs in polynomial time
2. always finds the best solution

Exact algorithms

Time for exact algorithms

Even if we cannot get polynomial time we can get good exponential time: 2^n is better than 3^n that is better than $n!$.

Max-Clique: 2^n possible subsets (not best possible)
Max-Cut: 2^n possible subsets
Max-3SAT: 2^n variable assignments (not the best possible)
TSP: simple: $(n-1)!$, with dynamic programming: $n^2 2^n$

Fast heuristics

You may or may not be able to analyze

1. Running time
2. Quality of solution

Local optimization

Usually stuck in local optimum.

Get out of it by:

- Simulated annealing
- Tabu search
- Genetic algorithms

Theses sometimes works well, but there is no real theory.

Approximation algorithms

Runs in polynomial time and gives an answer within a factor c of optimal, where c might be a function of n.

TSP

There is a simple algorithms that achieves factor $c = 2$ and a more complicated one that gives $c = 1.5$. It is unknown whether 1.5 is the best possible.

Max-Clique

Within factor n (n: number of nodes) is obvious as the clique cannot contain more than all vertices and it easy to find a clique of size 1. This trivial bound can be improved slightly but the best ratio in polynomial time is $O(\frac{n(\log\log n)^2}{(logn)^3})$ (this was stated incorrectly in lecture and corrected in notes).

Max-Cut

Think random or greedy: cuts half of the edges on average.
Approximation ratio $\geq 1/2$ (to be improved in later lecture).

Max-3SAT

Each clause satisfied with probability 7/8 by random assignment.
Approximation ratio: 7/8 (NP-hard to do better)

Approximation symmetric TSP (with triangle-inequality)

1. Find a min-cost spanning tree (very simple)
2. Take each edge both ways
   (Now we have a directed graph with in degree=out degree at each node. Such graph has an Euler-tour[1])
3. Find Euler tour (very simple).
4. Make shortcuts.

How well will this perform?

We claim that the cost of found tour ≤ 2(cost of spanning tree).

The cost of Euler-tour is twice the cost of the spanning tree and the cost of the TSP-tour is a most this (could be shorter by shortcuts and the use of the triangle-inequality).

The cost of spanning tree is at most the cost of optimal tour as removing any edge of the tour gives a spanning tree and we have found the cheapest spanning tree.

We conclude that the cost of found tour is at most twice the cost of the optimal tour.

There is a better algorithms by Christofides that finds the cheapest matching of the vertices of odd degree in the spanning tree. It turns out that the cost of this matching is at most 1/2 OPT, but we discuss this in more detail in next lecture.

[1] We remind you that an Euler tour is a tour that uses each edge exactly once.