Lec30-approx-algorithms

KT Ch. 11

Sunday, November 24, 2024 8:54 PM

How do we deal with NP-complete (or other hard) problems?

Heuristics – but can we guarantee we'll at least get close to the right answer

What should we do when we don't have an efficient alg.?

Heuristics? Can we guarantee that we aren't that wrong?

Approximation algorithms

Def. Let $A_p(I)$ be the approximate solution value to a min. problem
- I: instance
- p: problem (often dropped in notation)
- A: alg

Let $OPT_p(I)$ be the optimal (minimum) solution for I.

Ex. goal: $\forall I,\ A(I) \le \alpha(|I|)\, OPT(I)$ (for any instance I)

Some function of problem size $|I|$, want to be small

e.g. $\alpha(n) = 2$ or $\alpha(n) \in \log n$

Note: $\alpha \ge 1$.

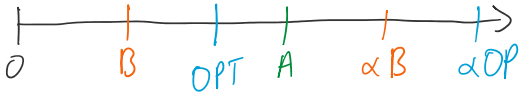
$\alpha(n) = 2$: our algorithm never returns a sol more than twice as large as the optimal

Lower Bounds

How do we analyze how far off from optimal we are without knowing the optimum?

A lower bound is a function $B(I) \le OPT(I)$ $\forall$ instances I.

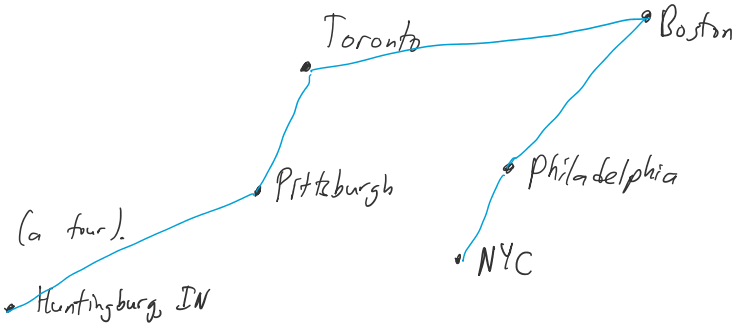
$\Rightarrow$ If $A(I) \le \alpha B(I)$, then $A(I) \le \alpha\, OPT(I)$.



Euclidean Traveling Salesman Problem (TSP)

Given n cities in $\mathbb{R}^2$ and the Euclidean metric, find the shortest path visiting all of them once (a tour).

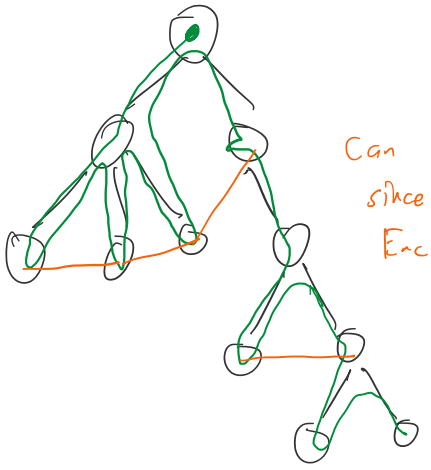
Complete graph with $\binom{n}{2}$ edges



NP-hard problem

Approximation A:

- Compute a MST T.
- Visit the cities in a pre-order traversal = visit a node, then the entire subtree of first child, then subtree of second child, etc.
  (similar to DFS in graph)



Can short-circuit since we are Euclidean

Let A be the edges visited on the tour found by our alg. (counting multiplicity)
Let A^* be edges visited in optimal tour.
Let cost(S) be the total length of edges in multiset S.

Thm $\text{cost}(A) \le 2\,\text{cost}(A^*)$

proof. $\text{cost}(T) \le \text{cost}(A^*)$ Why? A^* is a tree.

A walk W that traces the MST has length $2\,\text{cost}(T)$ as every edge is crossed twice.

$\Rightarrow \text{cost}(W) = 2\,\text{cost}(T) \le 2\,\text{cost}(A^*)$

W isn't a tour since it repeats cities.

Short-circuit later visits through the city.

By triangle inequality, this only reduces distance.

$\Rightarrow \text{cost}(A) \le 2\,\text{cost}(T) \le 2\,\text{cost}(A^*)$.

We now have a constant-factor approx. alg. for Euclidean TSP.

3SAT → Independent Set → Vertex Cover

Recall: Vertex cover is NP-complete.

Given undirected graph G, find smallest set S of vertices so every edge has an endpoint in S.

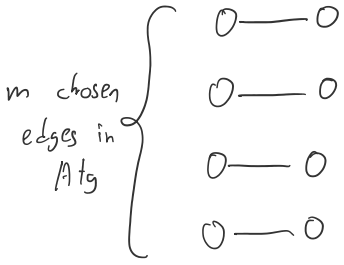
Greedy Attempt 1: Repeatedly pick a node that covers a new edge.

Fails: You might pick all but the last node if you choose badly.

Better greedy alg 2: Repeatedly pick an uncovered edge (u, v), & add both u & v to S. (and cover edges incident to either u, v)

Thm. Alg 2 is a 2-approximation.
i.e. it returns a vertex cover of size ≤ twice the optimal.

proof.



Lemma: If Alg 2 chooses m edges, then the optimal set cover S^* must have $|S^*| \geq m$. (lower bound)

pf. The m chosen edges share no endpts, so to cover just these edges, we need at least m nodes. ⇒ any set cover must have $\geq m$ nodes.
⇒ $|S^*| \geq m$. □

Notice that Alg 2 returns a set S with $|S| = 2m$ by construction.

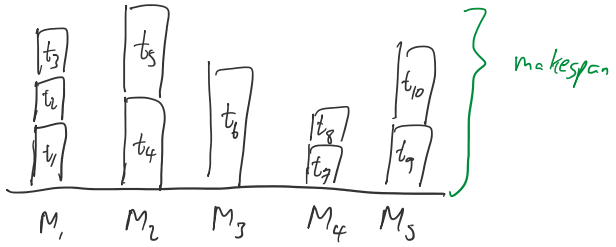
⇒ $|S| \leq 2|S^*|$.

⇒ Alg 2 is a 2-approximation ▨

Note that we don't know what size the actual vertex cover is, just that our greedy edge selection alg gets close.

Problem (Minimum Makespan) Given n jobs of length $t_1, t_2, \ldots, t_n$, and m machines $M_1, \ldots, M_m$, schedule the jobs on the machines to minimize the makespan.

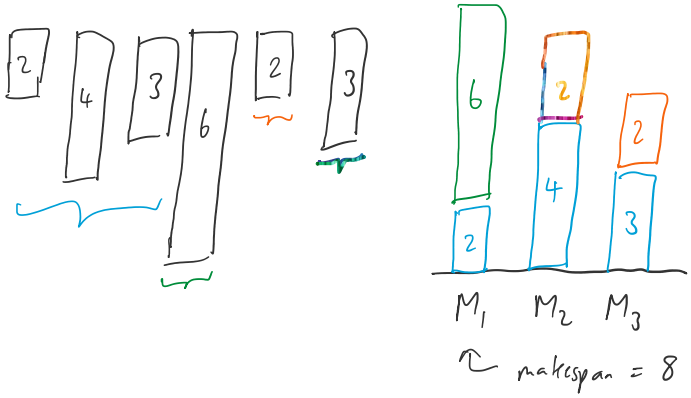
Def. The **makespan** is the total time to complete all jobs, running in parallel!



Minimizing Makespan is NP-complete (proof left to student)

Greedy Alg 1: Given items $t_1, \ldots, t_n$, schedule items in list order on least used machine, so far.

Example:



Initially all machines are unused.

Then schedule by whichever machine is least occupied

Let $G(I)$ be the makespan obtained using Alg 1 for instance I.
Let $OPT(I)$ be smallest possible makespan.

Goal: Prove $\forall$ instance I, $G(I) \leq \alpha\, OPT(I)$ for some $\alpha \geq 1$.

Can't be better than optimal!

α-approximation

Lower bounds:

- Some machine must do more work than t_j $\forall j$

$$\Rightarrow OPT(I) \geq \max_j t_j$$

- Some machine must do more work than average

$$\Rightarrow OPT(I) \geq \frac{1}{m} \sum_j t_j .$$

Thm: $G(I) \leq 2\, OPT(I)$.

pf. Let M be the machine with max load in greedy sol.
Let t_j be last job assigned to M.
Let L_i be the load the greedy alg puts on machine i
($G(I) = L_M$)

When t_j was placed on M, M had lightest load. Why?

Every machine had load $\geq G(I) - t_j$

$$\Rightarrow \underbrace{m(G(I) - t_j)}_{\text{less than load at item } j \text{ being placed}} \leq \underbrace{\sum_k L_k}_{\text{load on machines at end}} \leq \underbrace{\sum_j t_j}_{\text{total load from items}}$$

$$\Rightarrow G(I) - t_j \leq \frac{1}{m}\sum_j t_j \leq OPT(I).$$

not counting the last item placed on M, it had less than average load.

Also $t_j \leq OPT(I)$. (other lower bound)

$$\Rightarrow G(I) \leq OPT(I) + t_j \leq 2OPT(I).$$

This 2-approx proof was more involved, since we had to split up the solution to use two different lower bounds.

It turns out our analysis is close to tight in that greedy will sometimes give sol that are nearly twice the optimal.

Greedy (sorted) Alg 2: First sort jobs by decreasing length, then apply greedy alg 1.

Intuition: place big jobs first, then fit in little ones where-ever.

Thm. Sorted greedy alg 2 is a $\frac{3}{2}$-approximation alg.
i.e. it gives $G(I) \leq \frac{3}{2} OPT(I)$

proof Let $t_1, \ldots, t_n$ be the jobs sorted by length in decreasing order.

proof. Let $t_1, \dots, t_n$ be the jobs sorted by length in decreasing order.

Notice: When t_{m+1} is placed on a machine, there is already another longer job on that machine.

$$\Rightarrow OPT(I) \geq 2t_{m+1}$$

Now if $n \geq m+1$, then the last item t_j placed on M has $t_j \leq t_{m+1} \leq \frac{1}{2} OPT(I)$.

$$\Rightarrow G(I) \leq OPT(I) + t_j \leq \frac{3}{2} OPT(I) \quad \text{by new lower bound.}$$

If you are placing tasks, important to schedule bigger ones first.