7 May 2021

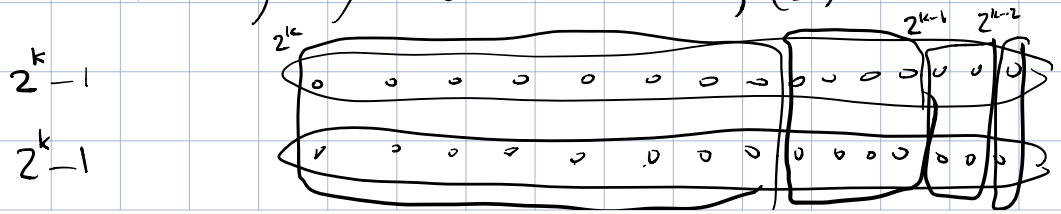
Set Cover Analysis
Linear Programming

Announcement. Trying to arrange alternate final exam date. I've been told now I need to get approval from the University Scheduler. Stay tuned for more info.

Given sets $S_1, \ldots, S_m \subseteq U$, $|U| = n = \bigcup_{i=1}^{m} S_i$.
Want to find minimum k such that U = the union of k of the given sets.

GREEDY SET COVER: repeatedly choose S_i that covers the greatest number of elements not yet covered, until all elements are covered.

Recall. Example where the opt # of sets, k^*, is 2 but greedy chooses $\approx \log\left(\frac{n}{2}\right)$ sets.



Fact Greedy never does worse than $O(\log n) \cdot k^*$.

Observation. If $k^* = 2$ than greedy never chooses more than $\log(n)$ sets.

Reason: in any iteration of the greedy main loop, we have a set T of not yet covered elements, and we know there exist two sets that cover all of T. $(k^* = 2)$

At least one of those two sets must cover half of T.

$\Rightarrow$ The set chosen by greedy covers at least half of T.

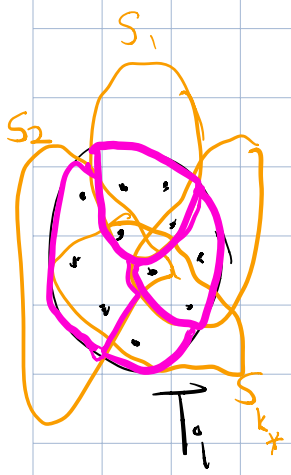
Conclusion: the # of not yet covered elements drops by a factor of at least 2 in every iteration.

After $\log(n)$ iterations, the number must be < 1, hence $\emptyset$.

Theorem: For any instance of SET COVER if k^* is the optimum value, then Greedy Algorithm chooses at most $k^* \cdot \ln(n)$ sets.

Proof: Let n_i = # elements left uncovered after i iterations of greedy.

$n_0 = n$.



(n_i elements)

As above, if T_i denotes the elements not yet covered after i iterations then we know $\exists$ k^* sets whose union covers T_i

$\Rightarrow$ at least one of the sets covers at least n_i/k^* elements of T_i

$\Rightarrow$ Greedy picks a set containing $\geq n_i/k^*$ elements of T_i

$\Rightarrow$

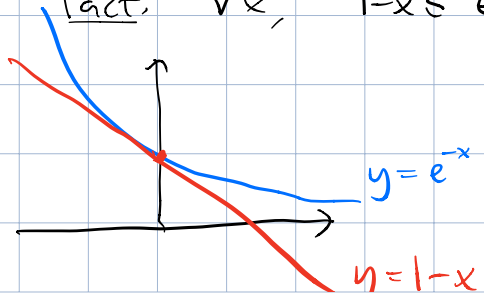
$$n_{i+1} \leq \left(1 - \frac{1}{k^*}\right) \cdot n_i$$

($\frac{1}{k^*} n_i$ or more of the elements were covered, so $\left(1-\frac{1}{k^*}\right) n_i$ or fewer remain uncovered)

After k iterations, $n_k \leq \left(1 - \frac{1}{k^*}\right)^k \cdot n_0$.

$\leq e^{-(1/k^*)\cdot k} \cdot n$

Fact. $\forall x$, $1 - x \leq e^{-x}$.



When $k > k^* \ln(n)$,

$$e^{-(k/k^*)} < e^{-\ln(n)} = \frac{1}{n}$$

So $$n_k \leq e^{-k/k^*} \cdot n < \frac{1}{n} \cdot n = 1.$$

Therefore $n_k = 0$.

I've shown that after $k^* \ln(n)$ or fewer iterations, Greedy has covered all elements.

In Chapter 11.3 you'll see a different analysis with two advantages.

1. Applies to Weighted Set Cover where sets have costs and the goal is ~~to~~ find set cover with min total cost.
2. Shows the approximation factor is

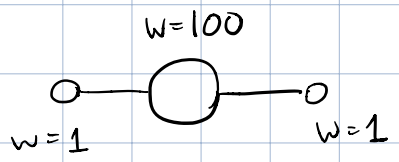
$$\leq \ln\left(\max_i |S_i|\right)$$

Next up.... Weighted Vertex Cover.

Given: $G = (V, E)$ undirected, weights $w(v) \geq 0$ $\forall v \in V$.

Find $S \subseteq V$ that covers every edge, minimize $\sum_{v \in S} w(v)$.

Recall. When all weights are 1, we saw a 2-approximation by simply picking uncovered edges and selecting both their endpoints, until all edges are covered.



The alg. above selects a set with combined weight 101.
Best vertex cover has weight 2.

I will present an alg. based on LINEAR PROGRAMMING. (LP)

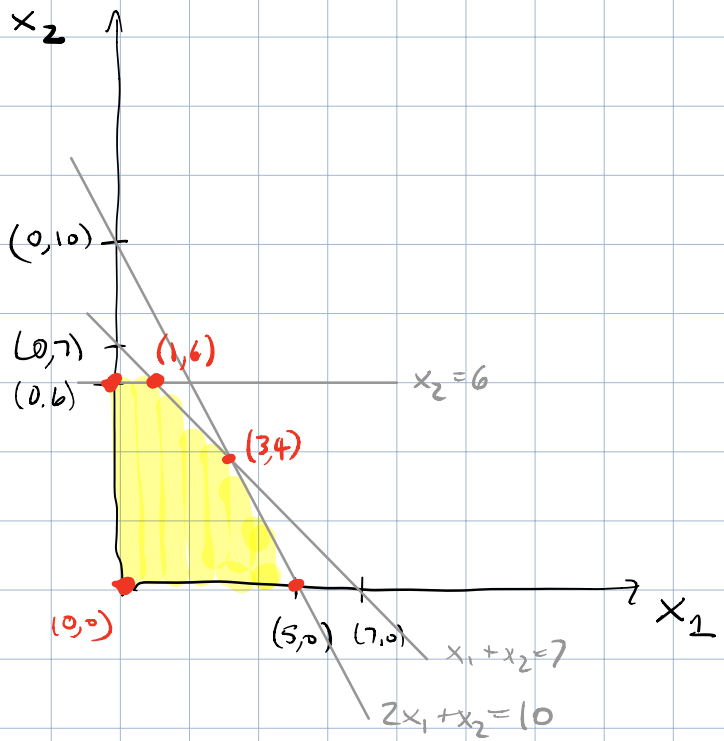
LP is one of the most powerful and general purpose problems we know how to solve in poly time.

LP: Given a system of linear equations and inequalities in n real-valued variables, find a solution that maximizes some linear objective function.

Example (2 variables)

$$\max \quad 3x_1 + 4x_2$$

$$\text{s.t.} \quad \begin{aligned} 2x_1 + x_2 &\leq 10 \\ x_1 + x_2 &\leq 7 \\ x_2 &\leq 6 \\ x_1, x_2 &\geq 0 \end{aligned}$$



In higher dimensions, a system of linear inequalities defines a polyhedron.

In dimension n, with m linear inequalities, a vertex of the polyhedron occurs when n linearly independent

inequalies become equations.

$\Rightarrow$ could be as many as $\binom{m}{n}$ vertices.

$\Rightarrow$ # of vertices can be exponential in dimension.

Must seek alternative algorithms for finding best vertex.

Poly-time algorithms have been known since 1970's–80's.

First such algorithm to be discovered, "ellipsoid algorithm", is a high-dimensional analogue of binary search.