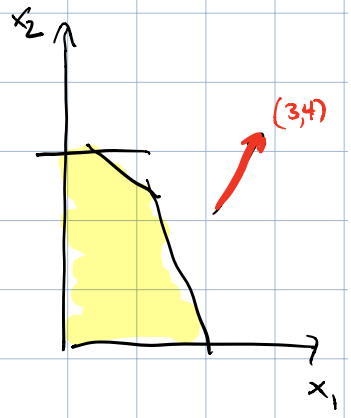
10 May 2021

Approximation via LP Rounding (§11.6)
Randomized Divide and Conquer (§13.5)

Recap. Linear programming (LP) is the problem of maximizing / minimizing a linear function over an n-dimensional set in $\mathbb{R}^n$ defined by linear equations and inequalities.

Ex.

$$\begin{aligned} \max. \quad & 3x_1 + 4x_2 \\ \text{s.t.} \quad & 2x_1 + x_2 \le 10 \\ & x_1 + x_2 \le 7 \\ & x_2 \le 6 \\ & x_1, x_2 \ge 0 \end{aligned}$$



FACT. There exist poly-time algorithms to solve linear programs.

Reducing Approximate Minimum Weight Vertex Cover to LP

Given undirected $G = (V, E)$ and weight $w(v) \ge 0$ for each vertex v. Find a set C that:

- is a vertex cover: each edge has endpt in C
- has minimum total weight

Method for reducing to LP involves using "decision variables" to encode vertex covers as $\{0,1\}$ vectors.

Idea: vertex set C correspond to vector $x \in \mathbb{R}^n$ (n = # vertices) such that

$$x_i = \begin{cases} 1 & \text{if } v_i \in C \\ 0 & \text{if } v_i \notin C. \end{cases}$$

"decision variable for v_i"

LP Relaxation of Weighted Vertex Cover

Linear function of $x_1, \ldots, x_n$

$$\text{minimize} \quad \sum_{i=1}^{n} w(v_i) \cdot x_i$$

For $\vec{x} \in \{0,1\}^n$ this sum evaluates to $\sum_{i : x_i = 1} w(v_i)$

$$\text{s.t.} \quad x_i + x_j \geq 1 \qquad \forall \text{ edge } (v_i, v_j) \in E$$

$$x_i \geq 0 \qquad \forall\, i = 1, 2, \ldots, n.$$

Set must include at least one endpoint of each edge.

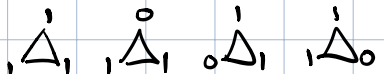
This means that the set of LP solutions is bigger than the set of vertex covers.

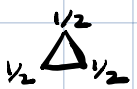
Ideally would restrict $x_i \in \{0,1\}$.
Next best thing is to restrict $x_i \in [0,1]$ using two linear inequalities

$$x_i \geq 0,$$
$$x_i \leq 1.$$

The second of these is unnecessary because the minimization objective takes care of it.

Example. $G =$ 

vertex covers: 

another LP solution: 

Suppose we solve the LP relaxation to obtain a vector $\vec{x}$ that satisfies the constraints and minimizes $\sum w(v_i) \cdot x_i$.

How to transform $\vec{x}$ back to an approx optimal set of vertices?

Let

$$y_i = \begin{cases} 1 & \text{if } x_i \geq \frac{1}{2} \\ \emptyset & \text{if } x_i < \frac{1}{2}. \end{cases}$$

and let

$$C = \{ v_i \mid y_i = 1 \}.$$

Then C is a vertex cover because

$\forall$ edge (v_i, v_j), $x_i + x_j \geq 1$

$\Longrightarrow$ at least one of x_i, x_j is $\geq \frac{1}{2}$

$\Longrightarrow$ at least one of y_i, y_j is $= 1$

$\Longrightarrow$ " " " " v_i, v_j belongs to C.

If C^* is a minimum weight vertex cover, and $\vec{z}$ is the corresponding $\{0,1\}$-vector,

$$OPT = \sum_{v_i \in C^*} w(v_i) = \sum_{i=1}^{n} w(v_i) \cdot z_i \geq \sum_{i=1}^{n} w(v_i) \cdot x_i$$

$$= LP\text{-}OPT$$

The solution C computed by our LP rounding algorithm satisfies

because $y_i \le 2x_i \quad \forall i$.

$$w(C) = \sum_{v_i \in C} w(v_i) = \sum_{i=1}^{n} w(v_i) \cdot y_i \le 2 \sum_{i=1}^{n} w(v_i) \cdot x_i$$
$$= 2 \cdot \text{LP-OPT}.$$

Combining these two lines...

$$w(C) \le 2 \cdot (\text{LP-OPT}) \le 2 \cdot \text{OPT}$$

... justifies that the algorithm is a 2-approximation.

Using randomness to improve the running time of an algorithm....
e.g. randomized quicksort.

Randomized median finding.
More generally, randomized selection:
given an array of n numbers, and $k \in \{1, 2, \ldots, n\}$, find the k^{th}-largest element in the array.
Sort and select k^{th} element of sorted array:
runs in time $O(n \log n)$.
We'll see a randomized algorithm with $O(n)$ running time.

Idea: choose a random pivot,
use divide and conquer.
gain running time savings
by recursing on only one piece.

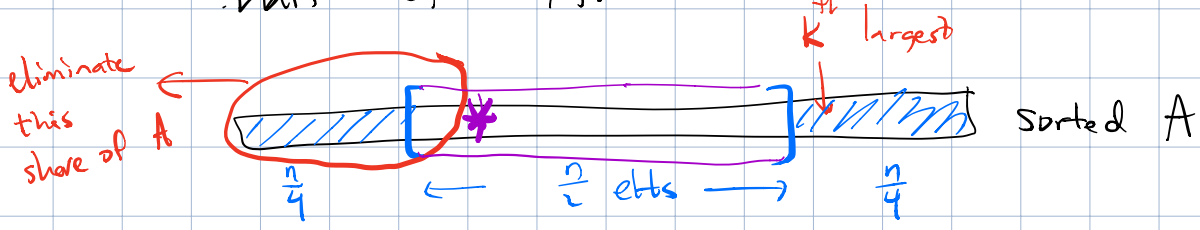
SELECT(A, k):

// A is an array of length n, assume distinct elements
// $1 \le k \le n$
// output k^{th} largest element of A.

Sample $i \in \{0, 1, \ldots, n-1\}$ uniformly random
Compare $A[i]$ with $A[j]$ $\forall j \neq i$
Form arrays B, C such that
B contains all $A[j] < A[i]$
C contains all $A[j] > A[i]$
let n_B = length(B), n_C = length(C).
if $n_C = k-1$:
output $A[i]$
if $n_C > k-1$:
output SELECT(C, k)
else:
output SELECT$(B, k - n_C - 1)$

Running time analysis. Let's find an upper bound on <u>expected</u> running time.

Intuition: with probability $\frac{1}{2}$, the random pivot in the middle half of A.



When that happens, the recursive call runs on an array of size $\frac{3n}{4}$ or smaller.

Let $T(n,k) = \mathbb{E}[\text{running time of SELECT}(A,k)]$.

Let

$$T(n) = \max_{1 \le k \le n} \{T(n,k)\}.$$

Then

$n-1$ comparisons to partition array

$\frac{1}{2}$ probability of recursing on array of size $\le 3n/4$

$$T(n) \le n + \frac{1}{2} \cdot T\left(\frac{3n}{4}\right) + \frac{1}{2} \cdot T(n)$$

$$\frac{1}{2} T(n) \le n + \frac{1}{2} T\left(\frac{3n}{4}\right)$$

$$T(n) \le 2n + T\left(\frac{3n}{4}\right).$$

$\frac{1}{2}$ probability of selecting a pivot outside middle half. Recurse on array of size $\le n$.

Unroll that recurrence:

$$T(n) \leq 2n + 2\left(\frac{3}{4}\right)n + 2\left(\frac{3}{4}\right)^2 n + \cdots$$

$$\leq 2\left(\sum_{j=0}^{\infty} \left(\frac{3}{4}\right)^j\right) n$$

$$= 8n$$

The algorithm never performs more than $8n$ comparisons in expectation.

Remark. More complicated algorithms can deterministically find k^{th} largest element using $O(n)$ comparisons.