PACT Example Lecture: Approximation Algorithms

Neha Nishikant

October 2017

These are notes for a lecture of Approximation Algorithms, in particular LP-Rounding, given by Prof. Sanjeev Khanna at the PACT Princeton summer program. These notes are only to be used for academic purposes.

α-approximation algorithm: Given a maximization problem **P**, an α-approximation algorithm A, for **P** is a poly-time algorithm that on any input I, outputs a solution satisfying the following:

$$A[I] \geq \frac{OPT(I)}{\alpha}$$

where $A[I]$ is the value of the approximation and OPT(I) is the optimal value
*for minimization, $A[I] \leq \frac{OPT(I)}{\alpha}$

See a problem with this?:
How can you guarantee this without knowing what the optimal value is?
$\downarrow$
example: **Max SAT**
Input: m clauses $c_1, c_2, c_3, ..., c_m$ with some number of boolean literals
Goal: assignment of literals to maximize the number satisfyable clauses
2-aproximation algorithm:

- randomly assign each literal a value of True of False
- take the complement
- determine which of these two assignments maximizes the number of satisfyable clauses. By the pigeon-hole principle, this can guarantee that the winning assignment satisfies at leas $\frac{m}{2}$ literals

As shown by the example above, the solution to this problem is to **upperbound the optimal solution** and take this as OPT(I). In this case, we upperbounded OPT(I) with m

Linear Programming
Input:

- n variables $x_1, x_2, x_3, ..., x_n$
- expression of these variables
- m conditions

Goal: assignment of variables to maximize the expression
e.g: Maximize $3x_1 + 7x_2 - 2x_3$ given

$$\begin{aligned} x_1 + 8x_2 &\leq 100 \\ 2x_1 - x_2 + 7x_3 &\leq 200 \end{aligned}$$

Linear Programming is poly-time solveable
However, **Integer Linear Programming** is *NP-hard*. This problem is the same as linear programming, but the variable assignment must be all integers

$\downarrow$
Approximation procedure:

- formulate your NP-hard problem as an integer linear programming problem
- drop the integer constraint and solve the resulting linear program
- **LP-round**: correct the solution to integers that satisfy the constraint
- This conversion process shifts your solution by α

-

example: Minimum Vertex Cover $\rightarrow$ *ILP*
Minimum Vertex Cover: Input: graph $G = \{V, E\}$ with n vertices $v_1, v_2, ...v_n$
Goal: find minimum *vertex cover C Step 1*: Convert to ILP:
Input: Let each vertex v_i correspond to a variable x_i Let

$$x_i = 1 \mid v_i \exists C$$
$$x_i = 0 \mid v_i \exists C$$

constraints:

$$\forall \{u, v\} \exists E, x_u + x_v \geq 1$$
$$\forall x_i, x_i = 0 \vee x_i = 1$$

Goal: Minimize $\sum_i^n x_i$
Step 2: drop integer constraint $\rightarrow 0 \leq x_i \leq 1$
Step 3: LP-rounding $\forall i$ if $x_i > \dfrac{1}{2}$ then set $x_i = 1 \rightarrow$ else $x_i = 0$
(intuition: choose the higher x_i of $\{u, v\}$ and set it to 1, the other to 0)
$\downarrow$
Integrality Gap: worst case ratio between OPT integral solution and OPT fractional solution for a certain linear program
prove that for our algorithm of Minimum Vertex Cover $\rightarrow$ *ILP*, the integrality gap is 2:
Take a complete graph $G : \{V, E\}$
$|V| = \dfrac{n(n-1)}{2}$
By pigeonhole principle, $n - 1$ is the cardinality of the OPT vertex cover
Worst case: let x_i be $\dfrac{1}{2} \forall i \rightarrow \sum_i^n x_i = \dfrac{n}{2}$
As $n \rightarrow \infty, n - 1 = 2(\dfrac{n}{2}$
Thus our algorithm has an integrality gap of 2

example: Minimum Set cover $\rightarrow$ *ILP*r
Minimum Set Cover:
Input: collection of m sets $s_1, s_2, ..., s_m$ where each $s_i \subset 1, 2, ..., n =$ Universe u
Goal: find minimum subcollection= S of sets whose union is U
Step 1: Let each set s_i correspond to a variable x_i
constraints:

$$x_i = 1 \mid s_i \; exists \; S, 0 \; else$$
$$\forall j \; \exists \{1, 2, ..., n\} x_{i_1} + x_{i_2} \geq 1$$

where $x_{i_1} + x_{i_2}, ... =$ variables of sets in which j appears
Step 2: drop integrality constraint $\rightarrow 0 \leq x_i \leq 1$
Step 3: LP Rounding
Let K be the max number of times that one element appears in S

Thus if:

$$x_i > \frac{1}{k} \rightarrow x_i = 1$$
$$x_i < \frac{1}{k} \rightarrow x_i = 0$$

This works, but the integrality gap will be k, following the same logic of the previous example. In the worse case, $k = m$. In that case, why not just output all sets?
$\downarrow$
randomized LP-rounding:
Choose set s_i in your solution with probability x_i
Then set a new variable $z_i \forall i$:

$$z_i = 1 \mid s_i \; exists \; S, 0 \; else$$

$E[\sum_i^n E_i] = \sum_i^n x_i =$ LP OPT
$\downarrow$
Check:
Pr[element j is covered]=1-Pr[element j is not in by any included set]= $1 - \pi_{j=1}^{k}(1 - x_{ij}$
By identity $1 + x \leq e^x$, so $1 - e^{-\sum x_i j}/leq 1 - \frac{1}{e}$
How can we make this probability exactly 1?
Instead of including s_i with probability x_i, take with probability $2x_i ln(n)$
Pr[element j is covered]=$1 - e^{-2ln(n)\sum x_i j} = 1 - \frac{1}{n^2}$
Pr[at least one element is not covered]=$\frac{1}{n^2}n = \frac{1}{n} \rightarrow 0$ as $n \rightarrow \infty$