INTRODUCTION TO APX – HW2

greedy algs, SAT, LP and more

Exercise one

On the input for Maximum k-cut we get an undirected graph G and weights on the edges $w\colon E(G) \to \mathbb{R}^+$. Our goal is to partition the vertices into k disjoint sets $V_1, V_2, \ldots, V_k$ so that we maximize the sum of all "multicolored" edges (those that go from any one set to any other set).

Suggest and analyze a probabilistic $\frac{k-1}{k}$-approximation algorithm for Maximum k-cut.

Solution. The algorithm is easy enough: for every vertex we throw a k-sided die and put it into the partition assigned by the die.

Our expected value is then

$$E[value] = \sum_{uv\in E} w(uv)P[uv \text{ is multicolored }] = \sum_{uv\in E} w(uv)P[u\text{in different bin than } v] =$$

$$= \sum_{uv\in E} w(uv)(1-1/k) = \sum_{uv\in E} w(uv)\left(\frac{k-1}{k}\right).$$

Clearly $\sum_{uv\in E} w(uv) \geq OPT$ and we are done.

Exercise two

You might recall the algorithm LP-Sat for the problem Max Sat from the lecture. After combining with the naive Rand-Sat, we have produced a 3/4-approximation algorithm.

We try to avoid the combining part and apply another trick instead: we start with the optimal solution (y^*, z^*) for the exact same LP as in the algorithm LP-Sat but instead setting $P[x_i = 1] = y_i^*$ we set the probability to be $P[x_i = 1] = f(y_i^*)$, where the function $f\colon [0,1] \to [0,1]$ is defined as follows:

$$f(p) = \begin{cases} \frac{3p}{4} + \frac{1}{4} & \text{for } 0 \leq p \leq \frac{1}{3} \\ \frac{1}{2} & \text{for } \frac{1}{3} \leq p \leq \frac{2}{3} \\ \frac{3p}{4} & \text{for } \frac{2}{3} \leq p \leq 1 \end{cases}$$

Prove that this choice (without any sort of combination) leads to a $\frac{3}{4}$-approximation algorithm for Max Sat.

Solution. The key to solving this exercise is to simplify as much as possible and then think about what we want to prove.

First we recall the LP in LP-Sat:

$$\begin{aligned} &\max \sum_{c \text{ clause}} y_c \\ \forall c \text{ clause}\colon &\sum_{x_i \text{ appears positive in } c} x_i + \sum_{x_i \text{appears negative in } c} (1-x_i) \geq y_c \\ \forall c \text{ clause}\colon &y_c \leq 1 \\ \forall i\colon &x_i \geq 0 \end{aligned}$$

As is customary in randomized algorithms, we will argue clause by clause, as their expectation of being true is independent by linearity of expectation.) Formally:

$$E[\text{algorithm}] = \sum_{c \text{ clause}} E[c] = \sum_{c \text{ clause}} P[c \text{ satisfied}].$$

What can we simplify next? Well, we notice that our function f is symmetric in terms of switching between x_i and $\neg x_i$. Therefore, as in the lecture on LP-Sat, we can assume without loss of generality that our clause contains only positive literals.

We also move from $P[c \text{ satisfied}]$ to $(1 - P[c \text{ unsatisfied}])$, as the second probability is just a big product depending on the optimum vector x^*, as we see below:

$$P[c \text{ unsat}] = \prod_{x_i^* \le \frac{1}{3}} (1 - f(x_i^*)) \prod_{\frac{1}{3} \le x_i \le \frac{2}{3}} \frac{1}{2} \prod_{x_i^* \ge \frac{2}{3}} (1 - f(x_i^*)) = \prod_{x_i^* \le \frac{1}{3}} (\frac{3}{4} - \frac{3x_i^*}{4}) \prod_{\frac{1}{3} \le x_i \le \frac{2}{3}} \frac{1}{2} \prod_{x_i^* \ge \frac{2}{3}} (1 - \frac{3x_i^*}{4}).$$

The product on the right is the probability that a clause c is unsatisfied. Our goal is a 3/4 approximation, so we would like the probability to be always less than 1/4, but this is too ambitious. Indeed, if we were able to prove that immediately for all clauses, we would not use the LP bound on the optimum – that would be highly suspicious!

To use the LP bound, we would like to prove the following:

$$\prod_{x_i^* \le \frac{1}{3}} (\frac{3}{4} - \frac{3x_i^*}{4}) \prod_{\frac{1}{3} \le x_i \le \frac{2}{3}} \frac{1}{2} \prod_{x_i^* \ge \frac{2}{3}} (1 - \frac{3x_i^*}{4}) \le 1 - \sum_{i | x_i \text{ appears in } c} \frac{3x_i^*}{4}.$$

Indeed, if we can prove this, we can use it the following calculation, where we also use the LP inequality $\sum_{i|x_i \text{appears in } c} x_i \ge y_c$ and the standard fact that LP optimum is a lower bound on OPT:

$$E[\text{algorithm}] = \sum_{c \text{ clause}} (1 - P[c \text{ unsatisfied}]) \ge \sum_{c \text{ clause}} (\sum_{i|x_i \text{ appears in } c} \frac{3x_i^*}{4}.) \ge \sum_{c \text{ clause}} \frac{3y_c}{4} \ge \frac{3OPT}{4}.$$

However, if we *can* prove the product to be below 1/4 as described above, we have that probability of satisfying is at least 3/4, which is certainly more than $\frac{3y_c}{4}$ for any one clause.

In summation, our setting and goal is as follows:

For a CNF clause c with no negative literals, we have the expression:

$$\prod_{x_i^* \le \frac{1}{3}} (\frac{3}{4} - \frac{3x_i^*}{4}) \prod_{\frac{1}{3} \le x_i \le \frac{2}{3}} \frac{1}{2} \prod_{x_i^* \ge \frac{2}{3}} (1 - \frac{3x_i^*}{4}).$$

We "win" (and complete our argument) if:

- we can prove that the expression is less than or equal 1/4;
- or if we can prove that the expression is less than or equal $1 - \sum_{i|x_i \text{ appears in } c} \frac{3x_i^*}{4}$.

What helped me personally is to imagine the proof of the inequality

$$\prod_{x_i^* \le \frac{1}{3}} (\frac{3}{4} - \frac{3x_i^*}{4}) \prod_{\frac{1}{3} \le x_i \le \frac{2}{3}} \frac{1}{2} \prod_{x_i^* \ge \frac{2}{3}} (1 - \frac{3x_i^*}{4}) \le 1 - \sum_{i | x_i \text{ appears in } c} \frac{3x_i^*}{4}.$$

as a game between the left-hand side and the right-hand side. In the beginning, we have no variables on either side, so there is an empty product (1) on the left-hand side and $1 - 0$ on the right-hand side.

We continue the game by multiplying a number on the left-hand side (corresponding to one x_i^*) and subtracting a value on the right-hand side. If we can prove in every step of the game that the inequality is preserved, we "win". We can also win immediately by a "wild card" if the left-hand side drops below 1/4.

We start our argument by noting how the value of x_i^* influences the left-hand and right-hand side:

- $x_i^* > \frac{2}{3}$: LHS multiplied by a number in $[\frac{1}{4}, \frac{1}{2})$, RHS subtracts a number in $(\frac{1}{2}, \frac{3}{4}]$.
- $\frac{2}{3} \geq x_i^* \geq \frac{1}{3}$: LHS multiplied by exactly $\frac{1}{2}$, RHS subtracts a number in $[\frac{1}{4}, \frac{1}{2}]$.
- $x_i^* < \frac{1}{3}$: LHS multiplied by a number in $(\frac{1}{2}, \frac{3}{4}]$, RHS subtracts a number in $[0, \frac{1}{4})$.

From these we can easily deduce the following observations, which we state without proof:

1. One step of the game with $x_i^* > \frac{2}{3}$ or $\frac{2}{3} \geq x_i^* \geq \frac{1}{3}$ preserves the inequality.
2. There cannot be two variables with $x_i^* \geq \frac{1}{3}$ or LHS drops below $\frac{1}{4}$.

The remaining case is to deal with several variables of type $x_i^* < \frac{1}{3}$ which decrease both sides only slightly. Let us rephrase the final situation in a more general way:

- We have LHS A and RHS B, with $\frac{1}{4} < A \leq B \leq 1$. We have a value $p < \frac{1}{3}$.
- We multiply A by $(\frac{3}{4} - \frac{3p}{4})$ and we subtract $\frac{3p}{4}$ from RHS.
- We know that the new $A' := (\frac{3}{4} - \frac{3p}{4})A$ satisfies $A' > \frac{1}{4}$.
- The new $B' := B - \frac{3p}{4}$.
- We want to prove that $A' \leq B'$ – for contradiction we can assume that $A' > B'$.

There are several ways to finish this argument. Personally, I prefer the "lazy" way – just plug these inequalities into Wolfram Alpha, which concludes there is no solution – and so $A' \leq B'$ after every step of the game with $x_i^* < \frac{1}{3}$.

Therefore, after all steps of the game, either the LHS is below $\frac{1}{4}$ or the inequality

$$\prod_{x_i^* \leq \frac{1}{3}} (\frac{3}{4} - \frac{3p}{4}) \prod_{\frac{1}{3} \leq x_i \leq \frac{2}{3}} \frac{1}{2} \prod_{x_i^* \geq \frac{2}{3}} (1 - \frac{3p}{4}) \leq 1 - \sum_{i | x_i \text{appears in } c} \frac{3x_i^*}{4}$$

holds. Knowing this, we prove the $\frac{3}{4}$-approximation using the LP bound as above.

EXERCISE THREE Consider the problem of GRAPH BALANCING, where you get an undirected graph G with weights on the edges $p\colon E(G) \to \mathbb{R}^+$. Our goal is to make the graph directed so that the most loaded vertex (the vertex with the most weight directed towards it) has minimum possible load. Formally we seek an orientation of the edges which minimizes the goal function $u = \max_{v \in V} \sum_{e \in E; e \text{ directed to } v}$.

Suggest and analyze a 2-approximation algorithm for the problem.

Tip: Linear programming might come in handy.

Solution.

We first design an ILP relaxation for this problem, using $[0,1]$-variables $x_{e \to u}$ and $x_{e \to v}$ to denote whether e is assigned to u or v. The program is as follows:

$$\begin{aligned} & \min p \\ \forall v \in V\colon & \sum_{uv \in E} p(uv) x_{uv \to v} && \leq p \\ \forall uv \in E\colon & x_{uv \to u} + x_{uv \to v} && = 1 \\ \forall v \in V, \forall uv \in E\colon & x_{uv \to v} && \geq 0 \end{aligned}$$

After solving the LP and getting the optimal vector x^* and the value p^*, we aim to round it so we get an integral approximate solution. Here we can use that $x_{uv\to u} + x_{uv\to v} = 1$ and use the idea from 2-approximation of VERTEX COVER – just round the number that is at least 0.5 to 1. A quick check of the inequalities reveals that indeed, this solution increases p to at most $2p$, and from the fact that $p^* \leq OPT_{ILP} = OPT$ (a linear program must have a lower value than an integer program) we have our proof of 2-approximation.

EXERCISE FOUR In the metric k-SUPPLIER PROBLEM we get $m + n$ points on input, where m of them are (in advance) marked as *suppliers* and the rest are *consumers*. Between all those points is a metric (with a triangle inequality as usual). Our task in this case is to select k suppliers so that we minimize the longest distance between a customer and its closest supplier.

Suggest and analyze a 3-approximation algorithm for the k-SUPPLIER PROBLEM.

Tip: This problem is related to the k-CENTER PROBLEM, which is going to be presented at the exercise session and if you miss it, you can also read about it in the Williamson, Shmoys book, chapter 2.2.

Solution. We will proceed in an analytical fashion to show the process of figuring out the solution.

Recalling the k-CENTER PROBLEM, we consider the optimum placement of the k suppliers and balls of radius OPT around each optimal supplier s_{OPT} (by *ball* of radius r we just mean the set of all suppliers and consumers within distance r, as defined by the metric of the problem).

As in k-CENTER, we know that (by optimality) all consumers are located within these k balls. In the analysis of k-center, we claimed that all centers selected by our algorithm are also within these k balls, and in fact every ball contains at least one of those.

We will use to our advantage that we have a different approximation ratio to prove – here we can use 3-approximation instead of just 2. This means we can increase the balls. How much can we enlarge them?

Setting them to $2OPT$ is a natural choice, and a right one – if we can prove that every ball of radius $2OPT$ contains at least one supply center s_{ALG} selected by the algorithm, then $ALG \leq 3OPT$. This will be true because the distance from s_{ALG} to s_{OPT} is at most $2OPT$, and the distance from s_{OPT} to any consumer point c covered by s_{OPT} is OPT by the assumption of optimality. In total, we get $d(s_{ALG}, c) \leq 3OPT$.

A snag in the k-SUPPLIER PROBLEM is that there *can* be possible supplier centers completely outside every ball, and the algorithm could have erroneously selected those instead of the suppliers within the balls. You can imagine one (very bad) supply center being $100cm$ away while OPT is just $1cm$.

To eliminate the bad supply centers, we look at one possible consumer (located in at least one ball of radius $2OPT$). We observe that all points within distance OPT to this consumer (including the optimal supply center and possibly other supply centers) are within the ball of radius $2OPT$ as well.

We cannot teach the algorithm to ignore all supply centers further than OPT from any customer – the algorithm does not know OPT in advance. We need to use a lower bound on OPT and ignore all supply centers further away than this lower bound.

A natural lower bound for one consumer c is just $d(c, s)$ for the *nearest* supply point s. Any other supply point chosen by the algorithm or optimum has to be in distance at least $d(c, s)$.

We therefore separate supply points into two groups: A supply point s is *interesting* if there exists a consumer which has s as the closest supply point to it. The rest are *uninteresting*.

Looking back, we see that our choice is a good one – for any supply point s_{OPT} of the optimum, the ball of radius $2OPT$ contains all interesting points for every consumer c with $d(s_{OPT}, c) \leq OPT$. In other words, if our algorithm chooses only interesting points, it will never select a point that is outside every ball.

What remains is to give an algorithm and show that indeed, our algorithm does not select two points from the same ball while leaving one ball completely empty.

A possible algorithm is the following:

1. Choose one interesting point as a supply center arbitrarily.
2. Then, choose every next interesting point that minimizes the current maximum the most. Proceed until you place all k points.

The rest of the analysis of this algorithm is the same as for the k-CENTER PROBLEM.