CS 170 Spring 2016 — Efficient Algorithms and Intractable Problems — Alessandro Chiesa and Umesh Vazirani

Final Review

1 Approximation Algorithms

Suppose we have an approximation algorithm A that takes in instance I and returns a solution with value $A(I)$. Then the approximation ratio is:

Maximization Problems	Minimization Problems
$\min_I \frac{A(I)}{\text{OPT}(I)} \leq 1$	$\max_I \frac{A(I)}{\text{OPT}(I)} \geq 1$

A general strategy for finding approximation algorithms (for minimization problems) is:

1. Solve another problem that gives a lower-bound on the optimal solution. (This problem is usually a *relaxed*, less constrained version of the original problem, but as a result its solution is not feasible in the original problem)
2. Find a way to turn this non-feasible solution into a feasible solution, inevitably increasing its value. The maximum amount this value can increase by is the approximation ratio.

2 Conceptual Questions

1. Which is better (for a minimization problem)? An approximation ratio of $\log(n)$ or 2?

   Solutions: 2 is better, since we want the approximation ratio to be as low as possible, and a $\log n$ approximation ratio grows with the instance size.

2. True or False, does reductions preserve approximation algorithms? Suppose problem X reduces to problem Y, and $\mathscr{A}$ is an algorithm that gives an approximation ratio of δ for Y. Will $\mathscr{A}$ give the same approximation ratio (δ) for X after it is reduced to an instance of Y?

   Solutions: False, there are NP-complete problems such as (non-metric) TSP that are hard to approximate but there are also NP-complete problems such as vertex cover that have a constant approximation factor. For example, consider a complete bipartite graph (each vertex from one side is connected to all vertices on the other side). Recall that the vertices not selected in a minimum vertex cover is the maximum independent set. Choosing all but one vertex gives a factor $\Theta(1)$ approximation since the minimum vertex cover is $n/2$. However taking one vertex not selected in the approximation gives a factor $\Theta(1/n)$ approximation to maximum independent set since the optimal value is $n/2$.

3 Practice Questions

1. Approximation algorithms are useful for tackling computationally hard problems. Sometimes the task at hand is not computationally hard, but we still want approximation guarantees. Consider the following scenario:

   Your friends have convinced you to go skiing with them for the first time in your life. You are not sure whether you will like skiing or not, and it might even take you several sessions to find out whether

you like it. Skiing equipment is expensive. If you want to buy a set, it will cost you x dollars, but you can also rent a set for one session for y dollars. Should you invest in a set or just rent?

Friend A suggests you should simply buy a set, since renting is like throwing money away.

Friend B suggests you should always rent a set, since you never know when you might quit skiing.

(a) If you were able to predict the future and know in advance that you would go skiing for n sessions, what would be the optimal strategy? Let the money you spend using this strategy be called OPT.

Solutions:

It is never optimal to rent before buying, because we could have bought earlier. So the only two strategies are renting for all sessions, or buying the first time. If n is the number of sessions, the first strategy spends ny dollars and the second one spends x. So clearly OPT $= \min(x, ny)$.

(b) How bad can listening to friend A be compared to OPT, in terms of an approximation factor?

Solutions:

By listening to A we will be just paying x. So the approximation ratio would be $\frac{x}{\min(x,ny)}$ which is worst when $n = 1$, at which point it becomes $\frac{x}{\min(x,y)}$. If x and y were not given, this could be arbitrarily large.

(c) How bad can listening to friend B be compared to OPT?

Solutions:

Listening to B would cost us ny. So the approximation ratio would be $\frac{ny}{\min(x,ny)}$. Clearly as $n \to \infty$ this fraction grows arbitrarily large (since the numerator grows but after some point the denominator stays constant). So the approximation ratio can be arbitrarily large.

Friend C suggests you follow this strategy: start by renting and keep track of how much you have spent on rentals. The first time that this sum is about to go over x, instead of renting, just buy a set. How bad can following C be compared to OPT?

Solutions:

First of all, the approximation ratio will never be worse than 2. If we never get to the buying point, then the money we spend is actually equal to OPT. If we get to the buying point then it must be that $ny \geq x$. So OPT $= x$. But using this strategy we do not pay more than $2x$, because we make sure we never spend more on renting than buying (so x is at most the money spent on rentals, and another x for the buy). Now if we follow this strategy and quit exactly right after we buy the item, we achieve the worst approximation factor, which for suitable values of x and y, can become arbitrarily close to 2.

2. Give a factor $1/2$ approximation algorithm for the following problem: given a directed graph $G = (V, E)$, pick a maximum-size set of edges from E so that the resulting subgraph is acyclic.

Solutions:

Arrange the vertices in a line, and group the edges into two sets, the ones going forward and the ones going backward. Each set alone forms a acyclic subgraph, and we pick the larger of the two. Hence we pick at least half of the edges, giving an approximation ratio of $1/2$.

3. Let $\pi = \pi_1 \pi_2 \ldots \pi_n$ be a permutation of $1\ 2 \ldots n$. A reversal $\rho(i, j)$ reverses the contiguous subsequence between i and j i.e. transforms

$$\pi_1 \ldots \pi_{i-1} \pi_i \pi_{i+1} \ldots \pi_{j-1} \pi_j \pi_{j+1} \ldots \pi_n$$

into

$$\pi_1 \dots \pi_{i-1} \pi_j \pi_{j-1} \dots \pi_{i+1} \pi_i \pi_{j+1} \dots \pi_n$$

The *sorting by reversals* problem is to find the minimum number of reversals that transforms a given permutation to the identity permutation. For example, the permutation [3,4,2,1] can be sorted by flipping [3,4] to get [4,3,2,1] and then flipping this entire sequence.

(a) Lets extend the permutation by adding $\pi_0 = 0$ and $\pi_{n+1} = n+1$ to the ends (π_0 and π_{n+1} are not moved during sorting). We say there is a breakpoint between π_i and π_{i+1} if they are not consecutive (in either order). For example, the permutation [3,4,2,1] is converted to [0,3,4,2,1,5] and there are three breakpoints: between (0,3), (4,2), and (1,5). Note there are not breakpoints between (3,4) or between (2,1) as each adjacent pair consists of consecutive elements.

Give a lower bound on the number of reversals needed to sort a permutation, π, if π has $b(\pi)$ breakpoints.

Solutions:

Let the number of breakpoints in a permutation π be $b(\pi)$. A reversal can eliminate at most two breakpoints at its two ends. Therefore, the number of reversals needed to sort a permutation π, is at least $b(\pi)/2$.

(b) Give an algorithm for the sorting by reversals problem with approximation ratio of at most 4 with respect to the minimal number of necessary reversals. (Hint: Any maximal run of contiguously increasing and contiguously decreasing elements should be kept together.)

Solutions:

We use the term *strip* for maximal runs of contiguously increasing and contiguously decreasing elements and consider single-element strips to be decreasing.

Some prefix of the extended permutation is sorted, say up to i. If the element $i+1$ is part of a decreasing strip which has index j, we apply the reversal $\rho(i+1, j)$ which will remove at least one breakpoint between i and the next element and will not introduce any new breakpoint.

If $i+1$ is part of an increasing strip, applying $\rho(i+1, j)$ will not decrease the number of breakpoints. However, we can reverse the increasing strip $i+1$ is part of and then $i+1$ will be part of a decreasing strip. We can then apply the reversal described above to decrease number of breakpoints.

Therefore, we can remove at least one breakpoint for every two reversals. So, we will need $2b(\pi)$ reversals to sort, giving us a 4-approximation algorithm.

4. Give a $1 - \frac{1}{k}$ approximation algorithm for Max k-Cut: given an undirected graph $G = (V, E)$ with non-negative edge costs, and an integer k, find a partition of V into sets $S_1, \cdots, S_k$ so that the total cost of edges running between these sets is maximized.

Solutions:

As with the Max 2-Cut approximation algorithm, keep track of k sets, adding each vertex to the set it has the least number of edges incident on. At most $\frac{1}{k}$ edges are "lost" (does not run between the final k sets chosen), so in total the fraction of edges running between the k sets chosen is at least $1 - \frac{1}{k}$.