Lecture 1: Introduction and Review

We begin with a short introduction to the course, and logistics. We then survey some basics about approximation algorithms and probability. We also introduce some of the problems that we will encounter over the course of the semester.

Disclaimer: These lecture notes are informal in nature and are not thoroughly proofread. In case you find a serious error, please send email to the instructor pointing it out.

Introduction and Background

The goal of the course is to introduce some key ideas in modern algorithm design. We will cover topics such as spectral algorithms, convex relaxations, random walks, local search, the multiplicative weight update method, and so on. To illustrate the techniques, we will use graph problems and variants of clustering.

Course Logistics

Office hours will be Wednesday 11AM-Noon. For information about grading, references and course material, see:

http://www.cs.utah.edu/∼bhaskara/courses/x968/

Approximation Algorithms

As we discussed in class, many interesting optimization problems we encounter turn out to be NP hard. Can we still give *interesting* algorithms for these problems?

One natural idea is to come up with algorithms that produce a solution whose objective value is *close* to that of the optimal solution. An algorithm that does so for every input is called an approximation algorithm.

Formally, suppose we have a problem $\mathcal{P}$ in which the goal is to *minimize* an objective function, and let ALG be an algorithm for the problem. For an input I, we denote by $\mathrm{OPT}(I)$ the optimal value (of the objective function), and by $\mathrm{ALG}(I)$ the objective value for the solution produced by the algorithm.

The *approximation ratio* of an algorithm ALG is defined to be

$$\max_{\text{inputs } I} \frac{\mathrm{ALG}(I)}{\mathrm{OPT}(I)}.$$

Remark 1. *Note that the approximation ratio is a* worst case *quantity – i.e., an algorithm has a large approximation ratio even if it does badly on* one input*.*

Remark 2. *The definition above is tailored towards* minimization *problems. For maximization problems, we consider the max value of the ratio* $\frac{OPT(I)}{ALG(I)}$*.*

Remark 3. *From our definition (and the remark above), an approximation ratio is always* ≥ 1 *– the closer it is to* 1 *the better. Many algorithms have approximation ratios that are functions of the input size. In these cases, the* $\max_{inputs}$ *means max over inputs of a certain size.*

Let us now see simple examples of approximation algorithms. First, some basic notations about graphs:

- A graph G is defined by a pair (V, E), where V is the set of vertices and E the set of edges (could be weighted, or directed).
- The neighborhood of a vertex v is the set of vertices that have an edge to v, and is denoted $\Gamma(v)$. The size of $\Gamma(v)$ is called the degree of v, denoted $\deg(v)$.
- For a set of vertices S, we write $\Gamma(S)$ to be $\cup_{v \in S}\Gamma(v)$ (union of the neighborhoods of vertices in S).
- For two disjoint sets of vertices $V_1, V_2 \subseteq V$, denote by $E(V_1, V_2)$ the set of edges with one endpoint in V_1 and the other in V_2.

Maximum Cut

Definition of the problem:

> Given an undirected, unweighted graph $G = (V, E)$, partition V into V_1 and V_2 so as to maximize $|E(V_1, V_2)|$.

The objective value is thus $|E(V_1, V_2)|$. We call the edges in $E(V_1, V_2)$ the edges that are *cut* by the partition. Let us consider the following natural algorithm, which can be thought of as "local search".

1. Start with an arbitrary partition V_1, V_2 of V (so every vertex in V is in one of the parts).
2. If there exists a vertex $u \in V$ that has $> (1/2)\deg(u)$ edges going to vertices in the same part as u, then move u to the other part. Else, terminate.
3. Repeat step 2 as long as there exists such a vertex u.

Efficiency. Note that whenever step 2 executes (without termination), the number of edges in the cut strictly increases – this is because the number of edges incident to u that are *cut* is $< (1/2)\deg(u)$ in the beginning, and $> (1/2)\deg(u)$ at the end, and all the other edges remain unchanged. Thus the algorithm terminates in at most $|E|$ steps.

Approximation ratio. What can we say about the quality of the solution at the end of the algorithm?

Let C_u denote the number of edges incident to u that are *cut* when the algorithm terminates. Thus we have

$$E(V_1, V_2) = \frac{1}{2} \cdot \sum_u C_u. \tag{1}$$

The factor $1/2$ is because the sum ends up counting every edge precisely twice (once for each end-point). What do we know about C_u?

By design, when the algorithm terminates, we have $C_u \geq (1/2)\deg(u)$ (else we would have moved u to the other part). Thus we have $\sum_u C_u \geq (1/2) \cdot \sum_u \deg(u) = |E|$ (here we used the fact that in any undirected graph, $\sum_u \deg(u) = 2|E|$, which is true because the summation counts every edge precisely twice).

Combined with (1), we obtain

$$E(V_1, V_2) \geq \frac{1}{2} \cdot |E|.$$

Maximum Cut continued on next page...

This proves that for any input I, $\text{ALG}(I) \geq (1/2)\text{OPT}(I)$, because $\text{OPT}(I)$ is (trivially) at most $|E|$. Thus the approximation ratio is at most 2.

In the Homework, we will see that the approximation ratio is actually *equal to* 2 (asymptotically), which means there are inputs for which the ratio between OPT and ALG can be arbitrarily close to 2. This shows that the analysis above is *tight*.

In the analysis above, it was easy to compare OPT and ALG, because we had an obvious upper bound on OPT – the total number of edges in the graph. For many problems, such an easy bound might not exist. The next example illustrates this. Thus the way in which we analyze the algorithm is significantly different.

Set Cover

Let $\mathcal{T}$ be a set of topics (e.g., sports, weather, finance, math, ...). Formally, let us denote them by $T_1, T_2, \ldots, T_n$. Suppose we have m people, denoted $P_1, P_2, \ldots, P_m$, and suppose that each person is an expert on a subset of the topics (e.g., P_1 could be an expert on T_2, T_5 and T_6, P_2 an expert on T_1 and T_5, and so on).

The aim is to pick the smallest set of people s.t. among them, there exists an expert on *each* of the n topics. I.e., we want the smallest set of people who *cover* all the topics.

The objective function here is the size of the set of people picked.

Greedy algorithm: A natural idea is to start by picking the person who is an expert on the most number of topics (break ties arbitrarily). This leaves us with some topics *uncovered*. The natural thing now is to pick a person who is an expert on the most number of *these* (uncovered) topics. This leaves us with a smaller set of uncovered topics. Now we pick the person who is an expert on the most number of these topics, and continue this process until there are no uncovered topics.[1]

How can we bound the number of iterations of this algorithm? One way is to try to quantify the *progress* we are making, i.e., can we show that the number of uncovered topics keeps reducing at every step? And if so, at what rate?

To answer this, let us define k to be the optimal value of the objective, i.e., there exists a set of k people who cover all the topics. Let us denote them by $P_{i_1}, P_{i_2}, \ldots, P_{i_k}$. Further, let us denote by U_j the set of uncovered topics after j steps of the algorithm. Clearly U_0 is the entire set of topics (intially, none of the topics are covered).

The key claim is that for every $j \geq 0$, step $(j+1)$ in fact covers at least $|U_j|/k$ of the uncovered elements U_j. Let us first see this for $j = 0$, in which case step $(j+1)$ is simply the first step:

We know that $P_{i_1}, \ldots, P_{i_k}$ cover the entire set of topics, thus at least one of them must cover at least n/k topics (by averaging). Our algorithm picks the person who covers the *maximum* number of uncovered topics, thus this person covers at least n/k topics.

The exact same argument applies for $j \geq 1$, because $P_{i_1}, P_{i_2}, \ldots, P_{i_k}$ cover all the topics, and in particular all of U_j. Thus the person picked at step $(j+1)$ must cover at least $|U_j|/k$ uncovered topics. The upshot of

[1] As long as it is *possible* to cover all topics, our algorithm will terminate. The approximation problem does not make sense if it is not possible to cover all topics (it is infeasible).

Set Cover continued on next page...

this is that the number of uncovered topics *after* step $(j+1)$ is $\leq |U_j| - \frac{|U_j|}{k}$. More precisely:

$$|U_{j+1}| \leq \left(1 - \frac{1}{k}\right)|U_j|.$$

Now suppose the algorithm runs for T iterations. We have

$$|U_T| \leq (1-1/k)|U_{T-1}| \leq (1-1/k)^2|U_{T-2}| \leq \cdots \leq (1-1/k)^T|U_0| = (1-1/k)^T n.$$

Thus if we set $T = k \log n$,[2] we have (using the inequality $(1-x) < e^{-x}$ for $x > 0$):

$$|U_T| \leq n \cdot (1-1/k)^T < n \cdot e^{-T/k} = n \cdot e^{-\log n} = 1$$

Thus the number of uncovered elements is < 1, meaning it is zero. Thus the number of iterations is at most $k \log n$. Recall that k is the optimal value of the objective. Thus the algorithm is a $(\log n)$ factor approximation.

Remark 4. *This is an example of a problem in which the approximation factor depends on the input size n. In fact, it is known (a paper of Uriel Feige) that approximating set cover to a factor $(1-\epsilon)\log n$ is NP-hard, for any constant ϵ, so this greedy algorithm is essentially the best possible, in terms of approximation ratio!*

Review of Basic Probability

Let us begin by seeing a couple of examples of the so-called *Union bound*. For any two events A and B in a probability space, we have

$$\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B].$$

Here $A \vee B$ is the event A "or" B, and $A \wedge B$ denotes A "and" B. This immediately implies that $\Pr[A \vee B] \leq \Pr[A] + \Pr[B]$ (simply because the other term is non-negative).

We can extend this to a collection of events $A_1, A_2, \ldots, A_n$. We have

$$\Pr[A_1 \vee A_2 \vee \cdots \vee A_n] \leq \sum_i \Pr[A_i].$$

While very simple, the inequality is rather powerful, and will be useful in many places in the course.

Collecting coupons

Suppose we have n stores, each of which gives out coupons in a newspaper. When a person buys a copy of the newspaper, he gets the coupon of a random store. The question is, how many newspapers should he buy to be 99% sure that he gets the coupons of *all* the stores?

Intuitively, this is how the process should go: initially, the person ends up seeing a lot of distinct coupons. But once the number of "unseen" coupons becomes small, so does his likelihood of seeing them. As an extreme

[2] All logarithms we consider will natural logs, i.e., to base e.

Collecting coupons continued on next page...

case, suppose he has seen all but one of the coupons; now the probability that he sees this coupon when he buys a newspaper is only $1/n$, so in expectation, he needs to buy n newspapers to see this one coupon.

One can make this observation semi-formal. Suppose the person has i coupons remaining. Then the next time he buys a newspaper, he has a probability of i/n of seeing an unseen coupon. Thus the expected number of newspapers he needs to buy to see an unseen coupon is n/i. After this many newspapers, the number of unseen coupons drops to $i-1$. We can then use this argument again replacing i with $(i-1)$.

Thus the expected number of newspapers he needs to buy for i to drop from n to 0 is

$$\frac{n}{n} + \frac{n}{n-1} + \ldots \frac{n}{1} = n\left(1 + \frac{1}{2} + \cdots + \frac{1}{n}\right) = \Theta(n \log n).$$

How can we make the argument more formal? Furthermore, we want a "success probability" as well (99%). For this, suppose the person buys T newspapers. Let us analyze the probabilty that he has *not* seen all the coupons. If this probability is $< 1\%$, then the success probability is $> 99\%$. Now, not seeing all the coupons is equivalent to saying that at least one of the coupons was not seen.

Let us denote by A_i the event that coupon i was not seen (after buying T papers). Then we are interested in getting an upper bound for $\Pr[A_1 \vee A_2 \vee \cdots \vee A_n]$. We can apply the union bound, to say that it is $\leq \sum_i \Pr[A_i]$. But what is $\Pr[A_i]$?

It is precisely the probability that none of the T newspapers ended up giving coupon i. This is exactly $\left(1 - \frac{1}{n}\right)^T$ (for any i). Thus we have

$$\Pr[A_1 \vee A_2 \vee \cdots \vee A_n] \leq n \cdot \left(1 - \frac{1}{n}\right)^T.$$

Once again, we can use the inequality $1 - x < e^{-x}$ to say that $(1 - 1/n)^T < e^{-T/n}$. Now to make the RHS above $< 1/100$, we simply need

$$e^{-T/n} < \frac{1}{100n}, \text{ or equivalently, } T \geq n \log(100n).$$

This is, of course, $n \log n + \Theta(n)$, which is essentially the bound from the heuristic argument.

The main advantage with a union bound is that we reduce the question of computing the probability of a 'complicated' event to that of a bunch of 'simple' events, which we can directly estimate.

Balls and Bins

Suppose we have n bins, labelled $B_1, B_2, \ldots, B_n$. Now consider throwing n balls randomly into the bins (i.e., for each ball, we throw it randomly into one of the bins, and proceed with the next ball). A natural question is, how "unbalanced" is the resulting assignment? For instance, could one bin end up getting a lot of balls?

This sort of a question is very useful in applications – imagine a large number of search queries, and we forward each query to a randomly chosen machine. We do not want any machine to be *overloaded* with too many queries.

Formally, if $S_1, S_2, \ldots, S_n$ denote the number of balls that landed in the bins (in order), we are interested

Balls and Bins continued on next page...

in the quantity $\max_i S_i$. For what T can we say that $\max_i S_i < T$ with probability 99%?

Once again, we can use a union bound to argue. If X_i denotes the event $S_i > T$, then we are interested in bounding $\Pr[X_1 \vee X_2 \vee \cdots \vee X_n]$, which we can analyze as before.

Exercise. Prove that for $T = C \log n / \log \log n$ for some C, we have $\max_i S_i < T$ with probability $\geq 99\%$.

This bound actually happens to be *tight*! Specifically, with probability at least 99%, there *exists* a bin j that gets at least $c \log n / \log \log n$ balls (for an absolute constant c). This quantity could be quite large for large n, meaning that random assignment typically produces a fairly unbalanced solution. There are very elegant ways of dealing with this. The reader is encouraged to look up: *the power of two choices*.