COMS 6998: Randomness in Computing — Sep 17, 2012

Fast Min-Cut and Random Variables

Instructor: *Xi Chen* — Scribes: *David Solimano*

1 Faster Min-Cut

1.1 Review

Last time we studied a randomized algorithm that merged edges of the graph G until 2 vertices remained, and found that the algorithm worked with $\frac{1}{n^2}$ probability. To get a reasonable chance of success, say, $1 - \frac{1}{n}$, we needed to run $O(n^2 \log n)$ times, for a total runtme of $O(n^4)$. We can do better.

The Naive Min-Cut Algorithm

- Graph G is given
- Ramdonly pick an edge $uv \in G$
- Merge the vertices u, v to generate G' with one less vertex
- Stop when there are 2 vertices left

Last time, we proved that if C is a min-cut of G, then this algorithm finds C on one run with probability $\geq \frac{2}{n(n-1)}$ (The proof can be found in the Lecture 1 notes, or pages 7-9 of RA). In other words, $\Pr[Success] \approx \frac{2}{n^2}$.

1.2 Ideas for improvement

Why is this so bad? At the start of the algorithm, this is pretty good. Consider that the error probability of the first edge is about $\frac{1}{n}$. But, it is bad at the end of the algorithm. Perhaps we can stop earlier in the algorithm and try something else at the end, when the probability of success starts to decrease.
The main idea of FasterMC is that running our whole algorithm n^2 times to reduce our error probability is foolish, because we most likely picked a good edge to start, and can trust this pick in the remaining runs. But when we are getting towards the end of the algorithm, our chance of making a mistake increases as n decreases, so we should run the algorithm multiple times to reduce the probability of errors. This sounds a bit like a tree, which has few nodes on the top and more towards the bottom.

1.3 Preliminaries

1.3.1 Run until t vertices

We are operating on a graph G and hoping to get a particular min-cut C. If we run our algorithm until there are t vertices remaining, what is the probability that we didn't choose a bad edge? The analysis is

very similar to when we were calculating F_{n-1} previously.

$$\begin{aligned}\Pr[F_{n-t}] &= \Pr[E_{n-t}|F_{n-t-1}] \dots \Pr[E_2|F_1] \Pr[E_1] \\ &= \frac{t-1}{t+1}\frac{t}{t+2} \cdots \frac{n-3}{n-1}\frac{n-2}{n} = \frac{t(t-1)}{n(n-1)}\end{aligned}$$

So if we run the naive algorithm until the graph is of size t, then with probability of approximately $\frac{t^2}{n^2}$, our min cut C is in the graph of size t.

1.3.2 Merge cost

How much time does the $Merge$ step of the naive algorithm cost for one merge?
If we are storing the graph as an adjacency matrix, it takes time $O(n)$. To see this, consider the $n \times n$ matrix of G

$$\begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix}$$

When we merge two vertices, we take two rows and columns and sum them together, yielding a $(n-1) \times (n-1)$ matrix. In the below matrix, the starred entries represent the changed entries.

$$\begin{bmatrix} p_{11*} & p_{12*} & \cdots & p_{1n*} \\ p_{21*} & p_{22} & \cdots & p_{2(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ p_{(n-1)1*} & p_{(n-1)2} & \cdots & p_{(n-1)(n-1)} \end{bmatrix}$$

Significantly, the remaining rows and columns are unaffected. So our total cost is linear.

1.4 A Faster Min-Cut Algorithm

The two points we investigated let us develop a faster algorithm. Let's start by restating the problem.

Min-Cut

- Input: G, $|v| = n$
- Output: A min-cut C

We want to show the new faster algorithm and analyze its running time and probability of success. As an aside, we haven't discussed the cost of random sampling to pick edges to merge. Assume for now that we can perform a uniform random sampling in linear time.

FasterMC

The definition of the problem is the same as the standard Min-Cut algorithm

- Input: G, $|v| = n$
- Output: A min-cut C

The steps of the algorithm are

1. If $n \leq 6$, brute-force the solution
2. Else, calculate $t = \frac{n}{\sqrt{2}}$ and fork two independent copies of SlowMC (The reason for picking $\sqrt{2}$ as a constant will become clear later)
    - Run SlowMC on G to reduce its size to t, yielding a graph G_1 of size t
    - Run SlowMC on G to reduce its size to t, yielding a graph G_2 of size t
3. Make two recursive calls to FasterMC on G_1 and G_2, yielding cuts C_1 and C_2. Return the smaller cut.

A note on step 1 - we perform the brute-force search for two reasons. The first is that our algorithm is recursive and so we need a base case somewhere. The second is that it avoids issues we saw previously with large errors on small graphs. The cost of this step is constant.

This is a fairly intelligent algorithm, because it increases the number of search threads exponentially as the node count decreases and the difficulty of the problem increases. It starts with just one thread, splits to two subcalls on graphs of size $\frac{n}{\sqrt{2}}$, each of which does the same thing, until towards the bottom of our call graph we stop recursing and brute-force the solutions.

The effort on the top levels of the graph is much less than on the bottom levels, which is reasonable because the probability of making a mkstake goes up as the size of the graph goes down.

1.5 Analysis of FasterMC

Running time

Let's look at the recusrive form of our time cost function. We have a cost of $O(n^2)$ to reduce the graph to size $\frac{n}{\sqrt{2}}$, and then we have two recursive calls on the reduced graph. So

$$T(n) = O(n^2) + 2T\left(\frac{n}{\sqrt{2}}\right) = O(n^2 \log n)$$

This runtime is quite good, only a little slower than an iteration of the SlowMC algorithm.

Theorem 1. *$\forall G$ of size n, FasterMC succeeds with probability $\Omega(\frac{1}{\log n})$*

This is much better than the SlowMC success probability of $\frac{1}{n^2}$! We only need to iterate FasterMC for $\log^2 n$ iterations in order to reduce our error probability to approximately $1 - \frac{1}{n}$, so our total runtime is about $n^2 \log^3 n$, and the n^2 dominates. This is much better than the known best flow algorithms which are $O(mn)$, and also much easier to implement.

Proof. Write down the recursive forumla.
$P(n)$ is the minimum probability of success across all graphs of size n. We need to show a lower bound of at least $\frac{1}{\log n}$ for $P(n)$, while expressing it in the form of

$$P(n) \geq f\left(P\left(\frac{n}{\sqrt{2}}\right)\right)$$

We know that

$$P(n) = 1 - \Pr[G_1\, fails \cap G_2\, fails]$$

That is, our algorithm only fails if both recursive calls fail. Remember that the recursive calls are independent. So we can write this as

$$P(n) = 1 - \Pr[G_1\, fails]\Pr[G_2\, fails]$$

Now, the probability that a single recursive call fails, $\Pr[Gfails]$, is $1 - \Pr[Gsucceeds]$. The recursive call succeeds when the bad edges are avoided, and its own recursive call succeeds. So

$$1 - \Pr[G\, succeeds] = 1 - \left(1 - \frac{t^2}{n^2} P\left(\frac{n}{\sqrt{2}}\right)\right)$$

Solving this recursive formula, we find that

$$P(n) = \Omega\left(\frac{1}{\log n}\right)$$

□

While this algorithm looks like it will yield a big tree and thus be slow, in fact it runs quickly. More details on this algorithm, which is called the Karger-Stein algorithm, can be found on page 292 of RA.

2 Random Variables and Expectation

2.1 Basic definitions

For our purposes, we will always assume that our sample space Ω is either finite or countably infinite.

Definition 2. *A random variable is a map from the sample space to the reals*

$$X : \Omega \to \mathbb{R}$$

Example 3 (Flip n coins)**.** *For flips $b_1, b_2, \ldots, b_n$, the number of coins that come up heads or 1 is a random variable denoted by X.*

We can also talk about radom variables with regards to events. If e are elementary events, then

Definition 4. *$X = a$: contains all $e \in \Omega$ s.t. $X(e) = a$*

We define independence of random variables as

Definition 5 (Independence)**.** *X, Y are independent if*

$$\Pr[X = x \cap Y = y] = \Pr[X = x] \Pr[Y = y]$$

Definition 6 (Expectation)**.** *The expectation of a random variable is defined as*

$$E[X] = \sum_{a \in \Omega} a \Pr[X = a]$$

Definition 7 (Sum of Random Variables)**.** *If X and Y are random variables, we use $X + Y$ to denote the following random variable*

$$(X + Y)(e) = X(e) + Y(e)$$

2.2 Linearity of expectations

Linearity of expectationts is a very important property because it always holds without any conditions on the variables being summed.

Definition 8 (General formula for linearity)**.**

$$E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E\left[X_i\right]$$

For just two variables, we can write this as $E[X + Y] = E[X] + E[Y]$.

2.2.1 Examples with coin flipping

Let's look at an example involving flipping an unfair coin with probability p of coming up heads, and thus $1 - p$ of coming up tails, where we want to determine how many heads to expect.

Example 9 (Coin flipping with explicit calculation)**.** *Let X be the random variable for the numer of heads that we get in n flips, and we want to know $E[X]$. To calculate this explicitly would require calculating $\Pr[X = 0], \ldots, \Pr[X = n]$, which requires the binomial formula*

$$Pr[X = i] = \binom{n}{i} p^i (1 - p)^{n-i}$$

and then summing

$$E[X] = \sum_{i=0}^{n} i \binom{n}{i} p^i (1 - p)^{n-i}$$

Which is rather challenging to calculate.

We can solve the previous problem much more easily with the proper choice of random variables. Let's introduce the concept of an indicator random variable.

Definition 10 (Indicator RVs)**.** *X_i is an indicator random variable if it takes on the value 1 when a certain property holds, and 0 otherwise.*

Now let's try redoing the example with these new random variables.

Example 11 (Coin flipping with indicators)**.** *Introduce n random variables X_i with the property that*

$$X_i = \begin{cases} 1 & \textit{if flip } i = 1 \\ 0 & \textit{otherwise} \end{cases}$$

Now we define X as the total number of heads, and let $X = X_1 + X_2 \ldots + X_n$. Now apply linearity of expectations.

$$E[X_i] = 1 \Pr[X_i = 1] + 0 \Pr[X_i] = 0 = p$$

$$E[X] = E[X_1] + E[X_2] + \ldots + E[X_n] = p + p + \ldots + p = np$$

This is much easier to reason about. One thing to be careful about is to remember what we're really doing. We take any outcome and look at X or the sum of the X_is, and we're showing that either function always yields the same result for any elementary event.

2.2.2 Why it works

Let's now show why linearity of expectations works from what we already know about probability. Remember that we defined $E[X] = \sum_{e \in \Omega} X(e) \Pr(e)$.

Proof.

$$E[X + Y] = \sum_{a,b} (a + b) \Pr[X = a \cap Y = b] = \sum_{a,b} a \Pr[X = a \cap Y = b] + \sum_{a,b} b \Pr[X = a \cap Y = b]$$

If we take the summations in two pieces, we find that

$$= \sum_a a \sum_b \Pr[X = a \cap Y = b] + \sum_b b \sum_a \Pr[X = a \cap Y = b]$$

Now, since the new inner summations are pairwise disjoint, we can find that

$$= \sum_a a \Pr[X = a] + \sum_b b \Pr[Y = b]$$

□

2.3 Inequalities

We usually care about probabilities, not expectations, but there are many useful tools in the realm of expectations that will help us with probabilities.

2.3.1 Markov's Inequality

Definition 12. *For a non-negative random variable X,*

$$\Pr[X \geq a] \leq \frac{E[X]}{a}$$

Proof.

$$E[X] = \sum_b b \Pr[X = b] = \sum_{b<a} b \Pr[X = b] + \sum_{b \geq a} b \Pr[X = b]$$

Taking advantage of the non-negativity of X,

$$\geq 0 + \sum_{b \geq a} a \Pr[X = b] = a \Pr[X \geq a]$$

At this point we can divide through by a and the inequality is proved. □

So we can always use $E[X]$ to get an upper bound, though it usually won't be very tight. Its main benefit is that it has only one restriction on its calculation, which is that X is nonnegative. $\Pr[X \geq a]$ is often called the tail probability.

2.3.2 Jensen's Inequality

Using linearity of expectations, we can show that

$$E[X^2] \geq (E[X])^2$$

This suggests a more general form of the above observation

Definition 13 (Jensen's inequality)**.** *If f is a convex function*

$$E[f(X)] \geq f(E[X])$$

Now let's define a convex function. Their general shape is similar to $y = x^2$.

Definition 14 (Convex function)**.** *For x, y on the function's curve, all points on the segment xy are above the curve. Algebraically,*

$$\forall x, y, \forall \lambda \in [0, 1]\, f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y)$$

Another definition is

Definition 15 (Convex function)**.** *A twice differentiable f is convex if all of its 2nd order derivatives are nonnegative.*

With that out of the way, we will prove Jensen's inequality

Proof. We will use the property of convex functions that if we take the tangent line at any point, all of the points on the function's curve are above the tangent line. That is,

$$\forall \mu\, f(X) \geq f(\mu) + f'(\mu)(X - \mu)$$

Take expectations of both sides,

$$E[f(X)] \geq E[f(\mu) + f'(\mu)(X - \mu)]$$

Apply linearity of expectations

$$E[f(X)] \geq E[f(\mu)] + E[f'(\mu)(X - \mu)]$$

Set $\mu = E[X]$. We substitute. Now μ is a fixed real equal to $E[X]$, so $E[f(\mu)] = f(E[X])$. At this point, we just need to prove that $E[f'(\mu)(X - \mu)]$ is equal to 0. Note that $f'(\mu)$ is a constant, and can be factored out. So on the right hand of the + sign we get.

$$E[f'(\mu)(x - \mu)] = f'(\mu)E[(x - \mu)] = f'(\mu)E[(X - E[X])] = f'(\mu) \times 0 = 0$$

□

Jensen's inequality is often useful to get probabilities of functions of random variables. For example, If I want to get an upper bound for $E[X]$ but for some reason can only get an upper bound for $E[X^2]$, then Jensen's inequality builds a bridge between them.

3 Applications of Random Variables

3.1 Coupon Collector Problem

- Input: A bin with n balls labelled from 1 to n.
- Action: At each step of the problem, I remove a ball, note the number, and put it back.
- Output: I want to know how many rounds I need to go through until I see all of the balls.

One way we might try to solve this problem is to define a random variable

Definition 16 (All balls RV)**.** *X is the number of rounds necessary to see all n balls.*

Let's look at a run of the problem and see what X comes out to

Example 17. *I have 4 balls, and I make the choices*

1 1 2 1 2 3 2 1 4

So $X = 9$. If I can calculate $E[X]$, the Markov's inequality can help us to bound probabilities. However, that seems rather difficult to calculate in this scenaro. So let's try a decomposition of X into simpler random variables.

Definition 18 (New ball RV)**.** *X_i is the number of rounds it takes to see a new ball, if I have already seen $i - 1$ balls. $X = \sum_{i=1}^{n} X_i$.*

So in the above example, $X_1 = 1$, $X_2 = 2$, $X_3 = 3$, and $X_4 = 3$, and $X = 9$ as we found by the direct calculation. Now we want to find $E[X]$, so we can use the linearity of expectations:

$$E[X] = \sum_{i=1}^{n} E[X_i]$$

It is clear that $E[X_1] = 1$ since we always see a new ball in the first round. But, what is $E[X_i]$ in general? To start, let's assume I have seen $i - 1$ balls, what is the probability that I see a ball that I haven't seen before on the first draw?

$$Pr[X_i = 1] = \frac{n - (i-1)}{n} = p$$

That is just the probability that I draw one of the $n - (i-1)$ balls I haven't seen from the pool of n balls, and we call that p_i. For $X_i = 2$, we must fail to find a new ball on the first draw, and succeed on the second.

$$Pr[X_i = 2] = (1-p)p$$

In general, the formula for $X_i = k$ involves us failing to find a new ball on the first $k-1$ draws, and succeeding on the kth. This is a well known probability distribution

Definition 19 (Geometric distribution)**.**

$$Pr[X_i = k] = (1-p)^{k-1}p$$

The expected value of a geometrically distributed random variable is well known.

Definition 20 (Expectation of geometric random variable)**.**

$$E[X] = \frac{1}{p}$$

Proof.

$$E[X] = \sum_{k \geq 1} (1-p)^{k-1} pk = \sum_{k \geq 1} \Pr[X_i \geq k]$$

$$= \frac{1}{1-(1-p)} = \frac{1}{p}$$

Note that this proof uses a lemma that we did not prove in class, but can be found in section 2.4 of the PC book.

□

Going back to our main problem, we can use this to find the expected number of rounds until we have seen all of the balls:

$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \frac{1}{p_i} = \sum_{i=1}^{n} \frac{n}{n-(i-1)}$$

$$= n\left(\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{1}\right)$$

The fractional part of the equation is the harmonic series, so we can approximate as

$$\approx n \ln n$$

Observation 21. *The main challenge in bounding our expectations and probability is finding a good decomposition of our problem into simple random variables that we can focus on.*

3.2 Perfect Hashing

Definition 22 (U)**.** *U is the universe, for example, all strings of a fixed length. It is all possible inputs into our function.*

Definition 23 (Universal hashing family)**.** *H is a t-universal hashing family if it is a set of functions*

$$h : U \to [t] = \{1, 2, \ldots, t\}$$

It must also satisfy the property that if we pick $h \in H$ uniformly at random and have two fixed keys $k, k' \in U$

$$\Pr[h(k) = h(k')] \leq \frac{1}{t}$$

So a universal hashing family is sort of like a random mapping. We will assume that we have such an H and that $\forall h \in H$ the cost to evaluate it is $O(1)$. This is unlike the function in homework 1.

Goal

The goal is to look at a set of, for example, meaningful English words - say that U is the set of all strings of length n, and S is a small subset of U. We want a permanent copy of S and we want to be able to support a quick set lookup into S when $|S| = n$. The best tree that we can build takes $\log n$ time to do a lookup. Using perfect hashing, we can do an $O(1)$ lookup. We can then fix the data structure that we built to some permanent form, such as a CD.

A problem

Let's say we pick a large enough t and then randomly pick a $h \in H$. Then we look at the $k_1, k_2, \ldots, k_n = S$ elements in our input, and calculate $h(k_i)$ for each. We then place each k_i in the slot determined by $h(k_i)$. We have a potential problem here, because some of these can collide, and that results in slowdowns, becuase the usual way of handling a collisions is to turn the buckets into linked lists, which have lookup time that is linear in the bucket size, not constant. However, if I am lucky, there will be no collisions and all of my lookups are constant costs. So the question becomes, how large must I make t to reduce or eliminate collisions?

Lemma 24. *if $t = 2\binom{n}{2}$, then $h \in H$ has no collisions with probability $\geq \frac{1}{2}$*

Proof. To prove this, we will define a set of random variables $X_{ij}, 1 \leq i \leq j \leq n$ as follows

$$X_{ij} = \begin{cases} 1 & \text{if } h(k_i) = h(k_j) \\ 0 & \text{otherwise} \end{cases}$$

$$E[X_{ij}] = \Pr[X_{ij} = 1] \leq \frac{1}{t}$$

We have no collisions only if

$$\forall X_{ij}, \Pr[X_{ij}] = 0$$

Markov's inequality will help us to bound these probabilities, but we have an issue in that the probabilities are correlated.

$$E\left[\sum X_{ij}\right] = \sum E\left[X_{ij}\right] = \binom{n}{2}\frac{1}{t} = \frac{1}{2}$$

If we set $t = \frac{n}{2}$ then we can get $E[X] = \Pr[X] = \frac{1}{2}$

$$Pr\left[\sum X_{ij} > 0\right] = \Pr\left[\sum X_{ij} \geq 1\right] \leq \frac{E\left[\sum X_{ij}\right]}{1} = \frac{1}{2}$$

Going from the first to the second steps, we used the fact that the X_{ij} have integer values only, and in the last step we used Markov's inequality. So, our algorithm succeeds with probability at least $\frac{1}{2}$ as we set out to show.

□

Analysis

At least half of the time, a random hash function will be collision free, which is good. However, we may end up building a table that is mostly empty of size $\approx n^2$, which is costly. Next week, we will look into building two-level hash tables to reduce the wasted space.