COMPSCI 532: Design and Analysis of Algorithms	September 30, 2015
Lecture 10	
Lecturer: Debmalya Panigrahi	*Scribe: Allen Xiao*

1 Overview

In this lecture, we will define *Monte Carlo* and *Las Vegas* algorithms. Then, we will give examples of each, including a simple randomized algorithm for the *global minimum cut* problem.

2 Monte Carlo and Las Vegas Algorithms

Definition 1. *A randomized algorithm is called a **Monte Carlo** algorithm if it may fail or return incorrect answers, but has runtime independent of the randomness.*

Typically, for Monte Carlo algorithms, we look to prove guarantees of the algorithm "succeeding" with high probability (often something like $(1-1/n^d)$). Intuitively, we know how long Monte Carlo algorithms take to run, but their correctness is left to randomness. On the opposite side, we have Las Vegas algorithms.

Definition 2. *A randomized algorithm is called a **Las Vegas** algorithm if it always returns the correct answer, but its runtime bounds hold only in expectation.*

In Las Vegas algorithms, runtime is at the mercy of randomness, but the algorithm always succeeds in giving a correct answer.

3 Global Minimum Cut

Definition 3. *Let $G = (v,w)$ be an undirected, unit-capacity graph. The **global minimum cut** problem is to find the smallest non-trivial cut in G. In other words, a complete partitioning of V into (S,T) which minimizes:*

$$|\{(v,w) \mid (v,w) \in E \cap (S \times T)\}|$$

where both S and T are nonempty.

If we have an algorithm $\mathscr{A}$ to solve the *s-t* minimum cut problem, we can solve global min-cut within $n-1$ invocations of $\mathscr{A}$. This strategy uses the fact that the minimum cut between any $s \in S$ and $t \in T$ is always the global minimum cut; we merely have to find one such pair. In general, $\tilde{O}(mn)$ time deterministic algorithms are known for global min-cut ($\tilde{O}(\cdot)$ means to ignore logarithmic factors).

3.1 Karger 1993

A Monte Carlo algorithm by Karger [Kar93] gives the global min-cut, with high probability, in $\tilde{O}(n^2)$ time. This algorithm is based around *edge contractions*:

Definition 4. *Let $G = (V,E)$ be a (multi)graph, and $(u,v) \in E$. **Contracting** edge (u,v) produces a new multigraph where:*

1. *Both u and v are replaced by a single new supervertex* $w = \{u, v\}$.
2. *Each edge between u and v is replaced by a self-loop* (w, w). *Edges which have u or v as an endpoint now have w as that endpoint.*

The resulting graph may include multi-edges and self-loops.

Edge contractions can simplify the graph while preserving a cut (S, T).

Fact 1. *Let* (S, T) *be a cut in G. Let* (v, w) *be an edge where both* $v, w \in S$ *or both* $v, w \in T$. *Then if* G' *is the graph after contracting* (v, w), *then* $|(S, T)|$ *is the same in* G' *as in G.*

Lemma 2. *Let* G' *be the graph after contracting e in G. Every cut in* G' *corresponds a cut of the same size in G, on the same supervertices.*

Proof. This follows directly from Fact 1. Note, however, that the opposite direction does not hold: not every cut in G' reappears as a cut in G. In particular, the cuts in G which included e disappear in G'. □

Contraction preserves the size for cuts which don't use the contracted edge. Intuitively, contraction doesn't *produce* smaller cuts in G', which means the minimum cut remains the minimum cut.

Corollary 3. *Let* (S, T) *be the global minimum cut of G, and let* G' *be the graph after contracting* $e \notin (S, T)$. *Then* (S, T) *is the global minimum cut of* G', *and has the same size.*

The algorithm is as follows:

Algorithm 1 (Karger 1993)

```
G_n ← G
for i = (n − 1) to 2 do
    G_i ← G_{i+1}
    Contract e ∈ E_i, picked uniformly at random, and remove all self-loops.
end for
return Cut represented by two remaining supervertices of G_2.
```

3.2 Success Probability

First, we relate the size minimum cut to the number of edges:

Lemma 4. *Let* λ *be the size of the minimum cut. On a graph with i vertices,* $|E| \geq (i\lambda)/2$.

Proof. Let $\deg(v)$ be the degree of v. Summing over all vertices double counts every edge.

$$\sum_{v \in V} \deg(v) = 2|E|$$

The degree cut of any vertex gives an upper bound on the size of the minimum cut.

$$\sum_{v \in V} \deg(v) \geq i\lambda$$

Rearranging:

$$|E| \geq \frac{i\lambda}{2}$$

□

Theorem 5. *For any given global mincut $(S, T = V \setminus S)$, Karger's algorithm outputs (S,T) with probability $\Omega\left(1/n^2\right)$.*

Proof. For any cut to be output by the algorithm, none of its edges must be contracted before the end (G_2). Let λ be the number of edges in global min-cut (S,T). Let $G_i = (V_i, E_i)$ be the graph at i vertices.

(S,T) disappears in G_i if one of its edges are contracted. Let $\mathscr{E}_i$ be the event that no (S,T) edge is contracted in the contraction on G_i to form G_{i-1}.

$$\begin{aligned}\Pr\left((S,T) \text{ successfully output}\right) &= \Pr(\mathscr{E}_n \cap \mathscr{E}_{n-1} \cap \cdots \cap \mathscr{E}_2) \\ &= \Pr(\mathscr{E}_n)\Pr(\mathscr{E}_{n-1} \mid \mathscr{E}_n)\Pr(\mathscr{E}_{n-2} \mid \mathscr{E}_n, \mathscr{E}_{n-1})\cdots\end{aligned}$$

Lemma 4 tells us that:

$$|E_i| \geq \frac{i\lambda}{2}$$

Since the edges are chosen uniformly from E_i:

$$\begin{aligned}\Pr\left(\mathscr{E}_i \mid \cap_{k=n}^{i+1}\mathscr{E}_k\right) &= 1 - \frac{\lambda}{|E_i|} \\ &\geq 1 - \frac{\lambda}{(i\lambda)/2} \\ &= 1 - \frac{2}{i}\end{aligned}$$

Replacing:

$$\begin{aligned}\Pr\left((S,T) \text{ successfully output}\right) &= \Pr(\mathscr{E}_n)\Pr(\mathscr{E}_{n-1} \mid \mathscr{E}_n)\Pr(\mathscr{E}_{n-1} \cap \mathscr{E}_n\mathscr{E}_{n-1})\cdots \\ &\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\cdots\left(1 - \frac{2}{3}\right) \\ &= \frac{n-2}{n} \times \frac{n-3}{n-1} \times \frac{n-4}{n-2} \times \cdots \times \frac{3}{5} \times \frac{2}{4} \times \frac{1}{3} \\ &= \frac{2 \cdot 1}{n(n-1)} \\ &= \binom{n}{2}^{-1} = \Theta(1/n^2)\end{aligned}$$

□

Corollary 6. *There are at most $\binom{n}{2}$ unique minimum cuts in a graph.*

How can we obtain a high probability bound? Repeat the algorithm $O(n^2 \log n)$ times and output the smallest cut.

$$\begin{aligned}\Pr\left(\text{None of the outputs were global minimum cut}\right) &\leq \prod_{t=1}^{n^2 c \log n}\left(1 - \frac{1}{n^2}\right) \\ &\leq \left(e^{-1/n^2}\right)^{n^2 c \log n} \\ &= O(1/n^c) \\ \Pr\left(\text{At least one output was global minimum cut}\right) &= \Omega(1 - 1/\operatorname{poly}(n))\end{aligned}$$

3.3 Running Time

The contraction algorithm we described can be implemented in $\tilde{O}(n^4)$ time. There are $O(n^2$ repetitions of the base algorithm, using $n-1$ contractions (which are $O(n)$ time on most data structures).

3.4 Karger-Stein 1996

The algorithm by Karger and Stein [KS96] modifies the basic contraction algorithm to run in $\tilde{O}(n^2)$ time. The intuition behind this method is that the success probability ((S,T) not contracted) decreases when the number of vertices ($i = |V|$) is small. Let us try to quantify this: when does the probability of success drop below $1/2$?

$$\begin{aligned}
\frac{1}{2} &\geq \Pr(\text{Contracting to } G_t \text{ preserves } (S,T)) \\
&= \prod_{i=(n-1)}^{t} \Pr\left((S,T) \text{ is a cut in } G_i \mid (S,T) \text{ is a cut in } G_{i+1}\right) \\
&\leq \prod_{i=(n-1)}^{t} \frac{i-1}{i+1} \\
&= \frac{t(t-1)}{n(n-1)} \\
&= \binom{t}{2}\binom{n}{2}^{-1} = \Theta\left(\frac{t^2}{n^2}\right)
\end{aligned}$$

Rearranging, the threshold $i = t$ is:

$$t \leq n/\sqrt{2}$$

We will choose a threshold t where $\Pr(\text{Contracting to } G_t \text{ preserves } (S,T)) = 1/2$. In Karger-Stein, each time we reach this threshold, we will make sure the number of independent evaluations of the current graph is doubled. Finally, we return the best min-cut reported by any of the recursive evaluations we spawned. When the graph is small enough we can just solve for the minimum cut manually.

Algorithm 2 (Karger-Stein 1996)

```
function MIN-CUT(G_i)
    n ← |V|
    if n ≤ 6 then
        Solve for the minimum cut manually as (S, T).
        return (S, T)
    end if
    t ← (n/√2) + 1
    A ← CONTRACT-UNTIL(G, t)
    B ← CONTRACT-UNTIL(G, t)
    return min{ MIN-CUT(A), MIN-CUT(B) }
end function
```

```
function CONTRACT-UNTIL(G, t)
    while |V| > t do
        Contract e ∈ E, picked uniformly at random.
        Remove all self-loops in G.
    end while
    return G
end function
```

3.5 Karger-Stein Success Probability

Consider the computation tree for Karger-Stein. This takes the form of a bipartite tree, where each node is a recursive call.

Lemma 7. *Karger-Stein does not contract min-cut (S,T) with probability $\Omega(1/\log n)$.*

Proof. Let (S,T) be a minimum cut of the the starting graph. First, we define the following parameterized events:

- $\mathscr{E}_0(G') :=$ G' does not have (S,T) contracted.
- $\mathscr{E}_1(G') :=$ Given $\mathscr{E}_0(G')$, MIN-CUT(G') returned (S,T).
- $\mathscr{E}_2(G',t) :=$ Given $\mathscr{E}_0(G')$, an independent call of CONTRACT-UNTIL(G', t) did not contract (S,T).

We will also refer to the output of CONTRACT-UNTIL(G, t) as G_t, following the G_i notation we used previously. The probability of success for the overall algorithm (given starting graph G) is $\Pr(\mathscr{E}_1(G))$.

$$\Pr(\mathscr{E}_1(G)) = 1 - \Pr(\overline{\mathscr{E}_1(G)})$$

How might MIN-CUT(G) have contracted (S,T)? Both MIN-CUT(A) and MIN-CUT(B) must have (S,T) contracted. For that to occur, either the CONTRACT-UNTIL() producing A (resp. B) contracted (S,T), or one of the later contractions in the recursive call on A (resp. B) contracted (S,T).

$$\begin{aligned}
\Pr(\mathscr{E}_1(G)) &= 1 - \Pr(\overline{\mathscr{E}_1(G)}) \\
&= 1 - \left[\Pr(\overline{\mathscr{E}_2(G,t)}) + \Pr(\mathscr{E}_2(G,t))\Pr(\overline{\mathscr{E}_1(A)})\right]\left[\Pr(\overline{\mathscr{E}_2(G,t)}) + \Pr(\mathscr{E}_2(G,t))\Pr(\overline{\mathscr{E}_1(B)})\right] \\
&= 1 - \left[\Pr(\overline{\mathscr{E}_2(G,t)}) + \Pr(\mathscr{E}_2(G,t))\Pr(\overline{\mathscr{E}_1(G_t)})\right]^2
\end{aligned}$$

The final inequality holds because the generation of and recursive calls on A and B are two independent but equivalent processes. Recall that we chose t so that: $\Pr(\mathscr{E}_2(G,t)) = \Pr(\overline{\mathscr{E}_2(G,t)}) = 1/2$. The expression for $\Pr(\mathscr{E}_1(G))$ therefore follows a recurrence relation.

$$\begin{aligned}
\Pr(\mathscr{E}_1(G)) &= 1 - \left[\Pr(\overline{\mathscr{E}_2(G,t)}) + \Pr(\mathscr{E}_2(G,t))\Pr(\overline{\mathscr{E}_1(G_t)})\right]^2 \\
&= 1 - \left[\frac{1}{2} + \frac{1}{2}(1 - \Pr(\mathscr{E}_1(G_t)))\right]^2 \\
&= 1 - \left[1 - \frac{1}{2}\Pr(\mathscr{E}_1(G_t))\right]^2 \\
&= \Pr(\mathscr{E}_1(G_t)) - \frac{1}{4}\Pr(\mathscr{E}_1(G_t))^2
\end{aligned}$$

Let d be the height of G in the computation tree (G_t has height $d-1$). We claim that $\Pr(\mathscr{E}_1(G)) > 1/d$ is a solution to the recurrence. Assume $\Pr(\mathscr{E}_1(G_t)) > 1/(d-1)$, then:

$$\begin{aligned}
\Pr(\mathscr{E}_1(G)) &= \Pr(\mathscr{E}_1(G_t)) - \frac{1}{4}\Pr(\mathscr{E}_1(G_t))^2 \\
&> \frac{1}{d-1} - \frac{1}{4}\left(\frac{1}{d-1}\right)^2 \\
&> \frac{1}{d-1} - \frac{1}{d(d-1)} \\
&= \frac{1}{d}
\end{aligned}$$

The height of the computation tree is $d = \log n$, since G_t reduces the number of vertices fractionally ($n \to n/\sqrt{2}$). We conclude that the success probability is $\Pr(\mathscr{E}_1(G)) > 1/\log n$. □

3.6 Karger-Stein Running Time

We can apply Master Theorem to find the running time. Each time the function makes exactly two recursive calls, on subproblems of size at most $t = n/\sqrt{2}$. Within a call, $O(n^2)$ work is done during the contraction steps.

$$\begin{aligned}
T(n) &= 2T(n/\sqrt{2}) + O(n^2) \\
&= O(n^2 \log n)
\end{aligned}$$

Since we know the success probability is $\Omega(1/\log n)$, we can repeat the whole algorithm $\log^2 n$ times to get high probability. This brings the total time to $O(n^2 \log^3 n)$, still in $\tilde{O}(n^2)$ time.

4 Randomized Quicksort

Let $S = s_0, s_1, \ldots, s_n$ be an unsorted sequence, and let $S \mid a$ be the concatenation of a onto the end of S ("append a to S"). The quicksort algorithm recursively breaks the unsorted sequence into two halves split about an arbitrary pivot, $p \in S$. One half is strictly less than p, the other greater. The algorithm is called recursively on these halves, and the concatenated results form the output.

Algorithm 3 (Quicksort)

```
1:  function QUICKSORT(S)
2:      if |S| ≤ 1 then
3:          return S
4:      else
5:          Pick an index r into S.
6:          p ← s_r
7:          L ← ∅
8:          R ← ∅
9:          for i = 1, ..., |S|; i ≠ r do
10:             if s_i < p then
11:                 L ← L | s_i
12:             else
13:                 R ← R | s_i
14:             end if
15:         end for
16:         return QUICKSORT(L) | p | QUICKSORT(R)
17:     end if
18: end function
```

Quicksort uses the following recurrence for its running time: each recursive call chooses a pivot which partitions S into two sets of size k and $n-k$ respectively.

$$T(n) = T(k) + T(n-k) + O(n)$$

In the best case, every pivot splits S evenly ($|L| = |R| = n/2$). There is a deterministic algorithm (median finding) which finds this pivot in $O(n)$ time, but the constant factor is worse than our random choice.

$$T(n) = 2T(n/2) + O(n) = \Theta(n \log n)$$

In the worst case, quicksort is a quadratic time algorithm. Suppose that, on every choice of pivot p, one of L or R is completely empty (i.e. p is the first or last element of a sorted S). The running time for quicksort would be:

$$T(n) = T(n-1) + O(n) = \Theta(n^2)$$

In *randomized* quicksort, the choice of p is a uniform random element of subproblem's S (above, r would be a random index into S). We will show that this is sufficient to give an expected runtime of $O(n \log n)$.

4.1 Expected Runtime

Let s_i be the ith element of S, and $s_{(i)}$ be the ith *sorted* element of S. The only "work" done in quicksort is through comparisons. First, notice that:

Lemma 8. *Let s_i, s_j be two different elements of S. s_i and s_j are compared no more than once during quicksort.*

Proof. s_i and s_j are only compared if one of them is selected as a pivot, and only once during that recursive call. The pivot is not included in L or R, so no comparisons occur between them in any of the following recursive calls. □

We will bound the expected number of comparisons. Let X_{ij} be an indicator random variable which takes value 1 if $s_{(i)}$ was compared to $s_{(j)}$, for $i < j$. Then the total time spent is $X = \sum_{i<j} X_{ij}$.

$$\mathbb{E}[X] = \sum_{i<j} \mathbb{E}[X_{ij}]$$

What is the probability that $X_{ij} = 1$? Remember, $s_{(i)}$ and $s_{(j)}$ are compared when one of them is selected as the pivot while they are in the same subsequence S. They will *not* be compared if they are split into separate subsequences before either one is selected as a pivot.

Lemma 9. *$s_{(i)}$ and $s_{(j)}$ are compared if and only if no $s_{(k)}$ is selected as a pivot before i or j, for $k \in (i, j)$.*

What is the probability that no $s_{(k)}$ is selected before either $s_{(i)}$ or $s_{(j)}$?

$$\Pr(X_{ij} = 1) = \Pr(i \text{ or } j \text{ is first out of } [i, j]) = \frac{2}{j - i + 1}$$

Now we can compute the expected runtime:

$$\begin{aligned} \mathbb{E}[X] &= \sum_{i<j} \mathbb{E}[X_{ij}] \\ &= \sum_{i<j} \Pr(X_{ij} = 1) \\ &= \sum_{i<j} \frac{2}{j - i + 1} \end{aligned}$$

We can do a change of variables, with $k = j - i + 1$:

$$\begin{aligned} \mathbb{E}[X] &= \sum_{i<j} \frac{2}{j - i + 1} \\ &= \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \frac{2}{k} \\ &= \Theta(n \log n) \end{aligned}$$

4.2 Backward Analysis

This is an alternative analysis for the $O(n \log n)$ runtime. We can think of the execution of quicksort as a sequence of pivot selections which splits S into more buckets. On step k, the number of pivots increases from $k - 1$ to k, and the number of buckets increases from $(k - 1) + 1$ to $k + 1$. The amount of work (time complexity) of this step is proportional to the size of the split bucket, since we compare each element against the new pivot.

Backward analysis is a strategy for analyzing algorithms by observing their execution in reverse. Here, we will start with n sorted pivots and reduce the number to 0 (likewise, $n + 1$ empty buckets to 1 of size n). For convenience, we will index these backward steps the same way as the forward steps: on step k there are k pivots remaining. However, on backward step k, the number of pivots *decreases* from k to $k - 1$. Two adjacent buckets are merged when we remove a pivot, and the work is proportional to their combined size. Note that we are not changing the algorithm – we are only adding each step's work in reverse order. As it turns out, this gives us a good analysis for the expected work of a single (forward or backward) step.

Claim 10. *On backward step k, there are k pivots remaining. The pivot selected and removed that step is uniformly random distributed among those k pivots.*

Proof. In the forward step k, the k pivots were selected in some sequence. For any two elements in this k, their relative order in the sequence is uniform, by symmetry. It follows that the kth pivot is also uniform from the set of k. □

Lemma 11. *The expected work in backward step k is $2n/k$.*

Proof. Consider this fencepost diagram of buckets and pivots in step k. Pictorially, each pivot is a '|', each bucket a (possibly empty) sequence of '×'.

$$\times\times\times|\times\times||\times\times\times\times\times\times\times|\times\times|\times|\times\times\times\times|\times\times\times\times$$

The amount of work to remove one of the pivots is proportional to the size of the two buckets surrounding it. The circled elements below are in the buckets which would merge if the fourth pivot is removed.

$$\times\times\times|\times\times||\otimes\otimes\otimes\otimes\otimes\otimes\otimes|\otimes\otimes|\times|\times\times\times\times|\times\times\times\times$$

Order the k pivots, then let w_i be the work of removing pivot i this step. Let b_i be the size of the ith bucket (precedes pivot i). The buckets cover all the non-pivot elements.

$$\sum_{i=1}^{k+1} b_i = n-k \leq n$$

As we highlighted in the diagram, the work w_i is:

$$w_i = b_i + b_{i+1}$$

From the claim before, we proved that the next pivot is selected uniformly at random from the k. Let W_k be the work of this backward step.

$$\begin{aligned}
\mathbb{E}[W_k] &= \sum_{i=1}^{k} w_i \Pr(i \text{ removed}) \\
&= \frac{1}{k}\sum_{i=1}^{k} b_i + b_{i+1} \\
&= \frac{1}{k}\left[(b_1+b_2)+(b_2+b_3)+\cdots+(b_k+b_{k+1})\right] \\
&\leq \frac{2}{k}\sum_{i=1}^{k+1} b_i \\
&\leq \frac{2n}{k}
\end{aligned}$$

□

Corollary 12. *Randomized quicksort runs in expected $\Theta(n \log n)$ time.*

Proof. Let $W = \sum_k W_k$ be the total work. The expected runtime is:

$$\begin{aligned}
\mathbb{E}[W] &= \mathbb{E}\left[\sum_{k=1}^{n} W_k\right] \\
&= \sum_{k=1}^{n} \mathbb{E}[W_k] \\
&\leq \sum_{k=1}^{n} \frac{2n}{k} \\
&= 2nH_n \\
&= \Theta(n \log n)
\end{aligned}$$

□

4.3 Concentration Bound

We give another analysis of randomized quicksort which shows that quicksort runs in $O(n \log n)$ time with *high probability*. Instead of computing the work for each pivot choice, we will use an amortized analysis where we charge 1 to each $s \in S$ for every subproblem (recursive call) s participates in. In other words, the number of calls before s is chosen as a pivot.

Let $h(s)$ be the number of subproblems s appears in. Then, the time complexity of the algorithm is:

$$n \max_{s \in S} h(s)$$

We will show that, with high probability, $\max h(s) = O(\log n)$ For this to occur, we need the recursive subsequences to be fractionally smaller (balanced) very often.

We will define a "good" (otherwise, "bad") subproblem involving s to be one where the subsequence $|L|, |R| \in (|S|/4, 3|S|/4)$. Let $X_{s,i}$ be an indicator random variable for when the ith subproblem involving s is bad.

$$X_{s,i} = \begin{cases} 1 & i\text{th subproblem of } s \text{ is good} \\ 0 & \text{otherwise} \end{cases}$$

Since the pivot is chosen uniformly from the elements of S:

$$\Pr(X_{s,i} = 1) = \frac{|S|/2}{|S|} = \frac{1}{2}$$

Let $X_s(N) = \sum_{i=1}^{N} X_{s,i}$, the number of good subproblems with s in the first N. Since each subproblem's good/bad status is independent:

$$\mathbb{E}[X_s(N)] = \sum_{i=1}^{N} \mathbb{E}[X_{s,i}] = \frac{N}{2}$$

We can apply the Chernoff bound, since $X_s(N)$ is a sum of independent 0-1 variables. For some $\varepsilon \in (0,1)$:

$$\Pr(X_s(N) < (1-\varepsilon)N/2) \leq \exp\left(-\frac{\varepsilon^2 N/2}{3}\right)$$

We use an upper bound on the number of good subproblems as a guide for choosing N and ε.

Lemma 13. *There are at most $O(\log n)$ good subproblems for any s.*

Proof. There are n vertices in the beginning. A good subproblem reduces the size of the subsequence containing s by at least $3/4$. The number of times this can occur before $n = 1$ is $\log_{4/3}(n)$. □

So, for some constant C, choose $N = 2C\ln n$. The Chernoff bound becomes:

$$\Pr(X_s(2C\ln n) < (1-\varepsilon)C\ln n) \leq \exp\left(-\frac{\varepsilon^2 C\ln n}{3}\right) \leq n^{-\frac{\varepsilon^2 C}{3}}$$

We want to make the right hand side look like $1/\text{poly}(n)$, so we should choose ε such that $\varepsilon^2 C/3$ is a constant. By tuning our choice of ε, we can get any constant. Here, we will aim for $1/n^2$.

$$\varepsilon^2 C/3 = 2 \implies \varepsilon = \sqrt{6/C}$$

Let $\varepsilon = \sqrt{6/C}$. Since we would like $\varepsilon \in (0,1)$, this also suggests a choice of $C > 6$.

$$\begin{aligned}\Pr(X_s(2C\ln n) < (1+\varepsilon)C\ln n) &\leq n^{-\frac{\varepsilon^2 C}{3}} \\ &= n^{-\frac{(6/C)C}{3}} \\ &= 1/n^2\end{aligned}$$

Applying a union bound over elements of S:

$$\Pr\left(\exists s \in S \mid X_s(2C\ln(n)) < (1-\sqrt{6/C})C\ln n\right) \leq \frac{1}{n}$$

Take the complement statement:

$$\Pr\left(\forall s \in S \mid X_s(2C\ln(n)) \geq (1-\sqrt{6/C})C\ln n\right) \geq 1 - \frac{1}{n}$$

This statement reads: "With high probability, every element sees $\omega(\log n)$ good subproblems within $O(\log n)$ subproblems." By Lemma 13, this is sufficient for termination, modulo some constant factors. With high probability, then, every element has logarithmic subproblem depth.

$$h(s) = O(\log n) \quad \forall s \in S$$

And the total time complexity is $O(n\log n)$.

5 Summary

In this lecture, we defined Monte Carlo and Las Vegas algorithms, then went into examples for each.

We presented the simple Monte-Carlo algorithm by Karger for finding the global minimum cut, and then the modified version by Karger and Stein which significantly raised the probability of success ($\Omega(1/n^2) \to \Omega(1/\log n)$), but only mildly raised the runtime of a single execution ($O(n^2) \to O(n^2\log n)$). Currently, the fastest algorithms for global minimum cut are variants of Karger's algorithm. These are Monte Carlo algorithms; it is still open if there are Las Vegas variants with the same runtime.

We analyzed randomized quicksort, a Las Vegas algorithm, in several ways (including backward analysis). Backward analysis is often useful for algorithms which make incremental randomized steps, like this one. Then we performed an alternative analysis which used a Chernoff bound to show that randomized quicksort has $O(n\log n)$ running time with high probability.

References

[Kar93] David R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-out algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 21–30, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[KS96] David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996.