Lecture 7: “Randomized Algorithms”

Prof. Sotiris Nikoletseas

University of Patras
and CTI

ΥΔΑ ΜΔΕ, Patras
2020 - 2021

1. Randomized Algorithms - Introduction

- Randomized is an algorithm whose evolution depends on random choices, in contrast to deterministic algorithms which decide how to evolve based on their input only.
- Two types of randomized algorithms:
  - Las Vegas: always correct output, running time is a random variable.
  - Monte Carlo: they may produce wrong output (but the error probability can be made appropriately small, actually negligible).

Advantages of randomized algorithms

- Because they decide randomly, they are simple, much simpler than corresponding deterministic algorithms which have to evaluate some (potentially complex) function of the input and accordingly evolve.
- They are usually very fast, much faster than their deterministic counterparts, since they actually introduce a trade-off between efficiency and correctness; this trade-off can be appropriately adjusted and yield a small error probability (even zero in the limit)

Basic paradigms for randomized algorithms (I)

Despite their rich diversity and wide applicability, their success is actually based on the use of some key underlying paradigms, techniques and principles, such as:

a. Foiling an adversary:
   - lower bounds for deterministic algorithms (d.a.) are actually derived by adversarial selection of hard input on which they behave poorly.
   - for each d.a. such adversarial inputs differ.
   - a randomized algorithm (r.a.) can be viewed as probability distribution on deterministic algorithms
   - thus, an adversary may pick hard input that foils one (or few) deterministic algorithms, but it is highly improbable to foil all (or most) of then; so, it can not "trap" a randomized algorithm into bad performance.

Basic paradigms for randomized algorithms (II)

b. Random sampling:
   - the r.a. performs random choices
   - this correspond to "randomly sampling" the input
   - a random sample quite often is representative of the entire (potentially very large) input space
   - thus, the simplicity of the random choice does not hurt correctness much

c. Random re-ordering:
   - A deterministic algorithm usually behaves poorly on few pathological inputs.
   - A recurrent idea (mainly in data structures): first randomly re-order the input (this is unlikely to produce the bad input); then apply a standard algorithm.

Basic paradigms for randomized algorithms (III)

d. Load balancing.
   Especially in problems of antagonistic sharing of limited resources (such as communication links), a random "spreading" of the global work load tends to produce more or less even load distribution to the resources avoiding bottleneck effects as well as under-utilization of some resources

e. Symmetry breaking.
   In distributed computing in particular, randomization can be used to distributively, locally make a collection of autonomous processors reach a global consensus (e.g. select a leader, break a deadlock etc.)

Basic paradigms for randomized algorithms (IV)

f. Probabilistic existence proofs.
The probabilistic method proves in a non-constructive way (i.e. without finding them) the existence of combinatorial structures with some desired property, by showing positive (i.e. non-zero) probability of the property in an appropriate probability space. Similarly, we can prove (without finding one) existence of an efficient algorithm for solving some problem.

2. The Min Cut Monte Carlo Algorithm - cut definition

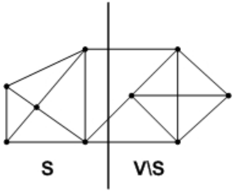
Let $G = (V, E)$ an undirected graph with $n = |V|$ vertices and $m = |E|$ edges.

- Definition. A cut in G is a partition of the vertices of V into two (disjoint) sets S and $V \backslash S$ where the edges of the cut are:

$$(S, V \backslash S) = \{uv | u \in S, v \in V \backslash S, uv \in E\}$$

  where $S \neq \varnothing$ and $V \backslash S \neq \varnothing$. We call the number of edges in the $(S, V \backslash S)$ cut the size of the cut.

- Example. A cut of size 3, $|S| = 5, |V \backslash S| = 4$:



The minimum cut problem

- We are interested in the problem of computing the minimum cut, that is the cut in the graph whose cardinality (number of edges) is minimum. In other words, find $S \subseteq V$ such that the cut $(S, V \backslash S)$ is as small as possible, and neither S nor $V \backslash S$ are empty.
- Complexity. The fastest known deterministic algorithm takes $O\left(n^2 \cdot m \cdot \log \frac{n^2}{m}\right)$ time which for dense graphs is $O(n^3)$.
- We will here present the fastest known minimum cut algorithm (by Karger) which is randomized (Monte Carlo) and takes $O(m \log^3 n)$ time, i.e. $O(n^2 \log^3 n)$ time, with high probability (i.e. with probability tending to 1 as some independent parameter tends to ∞).

Probability preliminaries

Let X, Y random variables.

- Definition. The <u>conditional probability</u> of X given Y is:
  $Pr\{X = x | Y = y\} = \frac{Pr\{X=x \cap Y=y\}}{Pr\{Y=y\}}$
- Important equivalent.
  $Pr\{X = x \cap Y = y\} = Pr\{X = x | Y = y\} \cdot Pr\{Y = y\}$
- Definition. We call r.v. X, Y <u>stochastically independent</u> iff
  $\forall x, y \qquad Pr\{X = x | Y = y\} = Pr\{X = x\}$

Equivalently, $Pr\{X = x \cap Y = y\} = Pr\{X = x\} \cdot Pr\{Y = y\}$
Similarly, <u>two events E_1, E_2 are independent</u> iff

$$Pr\{E_1 \cap E_2\} = Pr\{E_1\} \cdot Pr\{E_2\}$$

In general, $Pr\{E_1 \cap E_2\} = Pr\{E_1\} \cdot Pr\{E_2 | E_1\}$ and,
by induction,
$Pr\{\cap_{i=1}^{n} E_i\} = Pr\{E_1\} \cdot Pr\{E_2 | E_1\} \cdot Pr\{E_3 | E_1 E_2\} \cdots Pr\{E_n | E_1 E_2 \cdots E_{n-1}\}$

Karger's Algorithm - edge contraction

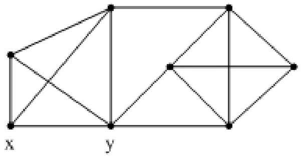
The basic operation of the algorithm is called edge contraction, i.e. we take an edge $e = xy$ and merge its two vertices into a single vertex. The resulting graph is called G/xy.

Note 1: We remove any resulting self-loops.

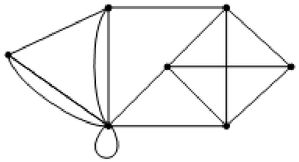
Note 2: The resulting graph is no longer a "simple" graph since it has "parallel" edges (i.e. more than one edges joining two vertices). In other words, edge contractions lead to multi-graphs.

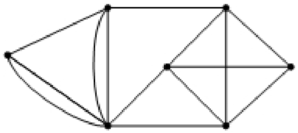
Note 3: We present multi-graphs as simple graphs with multiplicities on the edges.

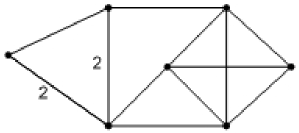
Example of edge contractions



the original graph G


contraction of edge xy


removing the self-loop



multiplicities on edges

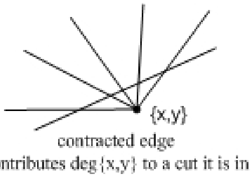
Note: In the initial graph G: $deg(x) = 3, deg(y) = 5$
In the contracted graph G/xy :$\underline{deg\{x, y\} = deg(x) + deg(y)}$, where self-loops contribute 2 in the degree. If we do not count self-loops:
$deg\{x, y\} = deg(x) + deg(y) - 2$

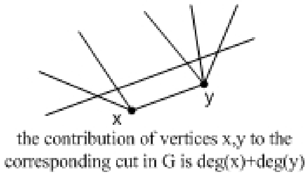
Features of the edge contraction operation

- The edge contraction operation obviously takes $O(n)$ time, where n is the number of vertices. It is actually done by merging the adjacency lists of the two contracted vertices, and then fixing the adjacency list of the vertices connected to the contracted vertices.
- Note: The cut is now computed counting multiplicities, i.e., if an edge is in the cut, and it has weight (multiplicity) w, then we add w to the total weight of the cut.

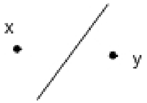
Edge contraction - important property

Note: The size of the minimum cut in G/xy is at least as large as the minimum cut in G.
This is so because any cut in G/xy has a corresponding cut of same cardinality in G.




x
y
the contribution of vertices x,y to the
corresponding cut in G is deg(x)+deg(y)

but, as said, $deg\{x,y\} = deg\{x\} + deg\{y\}$. Note: the opposite is not necessarily true because of cuts like this one, for which the degrees do not add.

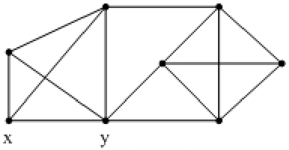


Main idea of Karger's algorithm

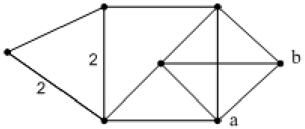
Remark: Because of the property in the previous slide
the minimum cut can not decrease with the edge contractions!

Basic idea: Repeatedly, pick an edge and contract it, shrinking the graph all the time until only 2 vertices remain. The multiplicity of the edge joining these last 2 vertices is an upper bound on the minimum cut in the original graph.

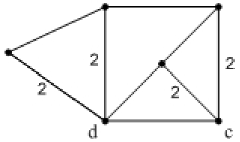
Example of repetitive edge contractions



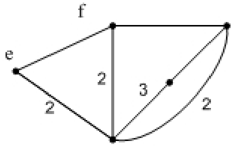
original graph G



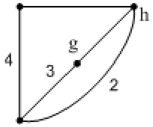
$G_1 = G / xy$



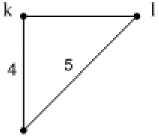
$G_2 = G_1 / ab$



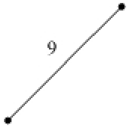
$G_3 = G_2 / cd$



$G_4 = G_3 / ef$



$G_5 = G_4 / gh$



$G_6 = G_5 / kl$

When the final cut is not minimum

- we showed that

  min cut original $\leq$ min cut contracted

- if we do not contract any edge of the minimum cut then equality holds and the output is correct.
- the output may be wrong when we contract an edge of the minimum cut.
- it may be correct if we contact an edge of a minimum cut but there are more than one minimum cuts and at least one survives contractions.

Cutting the Gordian Knot with randomness

- we showed that if we do not contract a minimum cut edge then the output is correct
- but, from an algorithmic design point of view, this argument is circular, since we do not know yet the minimum cut.
- we solve the Gordian Knot via randomness, picking the next edge for contraction randomly, and hoping to, appropriately, bound the error probability.

The pseudo code of the algorithm

```
**Algorithm** $MINCUT(G)$
    $G_0 \leftarrow G$
    $i = 0$
    **while** $G_i$ has more than two vertices **do**
        Pick randomly an edge $e_i$ from $G_i$
        $G_{i+1} \leftarrow G_i / e_i$
        $i \leftarrow i + 1$
    Let $(S, V/S)$ the cut in the original graph
      corresponding to the single edge in $G_i$
```

Estimating the error probability

Lemma 1

If a graph of n vertices has a minimum cut of size K, then $|E(G)| \geq \frac{Kn}{2}$.

Proof: Obviously each vertex has degree at least K (otherwise the vertex itself would lead to a cut smaller than K)

$$\Rightarrow \sum_{v \in V(G)} deg(v) \geq nK$$

But $\sum_{v \in V(G)} deg(v) = 2 \cdot |E(G)|$, thus $|E(G)| \geq \frac{nK}{2}$ □

Lemma 2

If we pick a random edge, then the probability that it belongs to the minimum cut is at most $\frac{2}{n}$.

Proof: Let e the random edge picked. Then

$$Pr\{e \text{ in the minimum cut}\} = \frac{K}{|E(G)|} \leq \frac{K}{\frac{K_n}{2}} = \frac{2}{n}$$ □

Time complexity and correctness

Remark: MINCUT runs in $O(n^2)$ time, since it performs $n-2$ contractions taking time $O(n)$ each.

Lemma 3

MINCUT outputs correctly the minimum cut with probability at least $\frac{2}{n(n-1)}$.

Proof: Before the $i-$th contraction $(1 \leq i \leq n-2)$ the graph has $n-i+1$ vertices. Let $\mathcal{E}_i$ the probability that the edge contracted at the $i-$th repetition is not in the minimum cut.

Then $Pr\{\mathcal{E}_i|\mathcal{E}_1,\ldots,\mathcal{E}_{i-1}\} \geq 1 - \frac{2}{n-i+1}$

Thus
$$Pr\{\text{correct output}\} = Pr\{\bigcap_{i=1}^{n-2} \mathcal{E}_i\} =$$
$$= Pr\{\mathcal{E}_1\}Pr\{\mathcal{E}_2|\mathcal{E}_1\}\cdots Pr\{\mathcal{E}_{n-2}|\mathcal{E}_1\cdots\mathcal{E}_{n-3}\} \geq$$
$$\geq (1-\tfrac{2}{n})(1-\tfrac{2}{n-1})(1-\tfrac{2}{n-2})\cdots(1-\tfrac{2}{3}) =$$
$$= \frac{n-2}{n}\frac{n-3}{n-1}\frac{n-4}{n-2}\cdots\frac{2}{4}\frac{1}{3} = \frac{2}{n(n-1)}$$
□

Probability amplification

Informal concept: Amplification is the process of running a random experiment again and again until the property we want happens with appropriately good probability.

A repetitive algorithm. Let MINCUTREP the algorithm that runs MINCUT $n(n-1)$ times and returns the minimum of all cuts computed in all those (independent) executions of MINCUT.

Lemma 4

The probability of MINCUTREP failing to return a minimum cut is less than 0.14.

Proof: The probability of failure is less than

$$\left(1 - \frac{2}{n(n-1)}\right)^{n(n-1)} \leq e^{-2} < 0.14$$

(since all $n(n-1)$ repetitions of MINCUT must fail) □

More amplification

Theorem 1

The minimum cut can be computed in $O(n^4 \log n)$ time, with high probability of correct output.

Proof: Similarly to Lemma 4, the error probability after $n(n-1)\log n$ repetitions of MINCUT is less than:

$$\left(1 - \frac{2}{n(n-1)}\right)^{n(n-1)\log n} \leq e^{-2\log n} = n^{-2}$$
$$\Rightarrow Pr\{\text{correct output}\} \geq 1 - n^{-2} \to 1 \text{ as } n \to \infty$$

The complexity is $O(n^2 \log n)$ times $O(n^2)$ $i.e.$ $O(n^4 \log n)$

Towards a faster algorithm

Questions: – Can we design a more complicated yet faster algorithm?

– Why does MINCUTREP need many repetitions?

Remark: The probability of success in the first l iterations is:

$$Pr\{\mathcal{E}_1 \cdots \mathcal{E}_l\} \geq (1-\frac{2}{n})(1-\frac{2}{n-1})(1-\frac{2}{n-2})\cdots(1-\frac{2}{n-l+1}) =$$
$$= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{n-l-1}{n-l+1} = \frac{(n-l)(n-l-1)}{n(n-1)}$$

At the start (when l is small) this probability is large but it deteriorates very quickly when l gets larger and the graph smaller.

Idea: As the graph gets smaller, the error probability (of contracting a minimum cut edge) increases. So, we will run the algorithm more times when the graph is smaller.

A size-dependent contraction process

We will thus use the following new version of repetitive contraction operations which depend on the graph size.

```
$CONTRACT(G,t)$
begin
    while $|V(G)| > t$ do
        Pick a random edge $e$ of the graph
        $G \leftarrow G/e$
return $G$
end
```

The new (recursive) algorithm FASTCUT

```
$FASTCUT(G = (V, E))$
    $G$ a multi graph
begin
    $n \leftarrow |V(G)|$
    if $n \leq 6$ then
        compute minimum cut of $G$
        via brute force and return it
    $t \leftarrow \left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil$
    $H_1 \leftarrow CONTRACT(G, t)$
    $H_2 \leftarrow CONTRACT(G, t)$
    $X_1 \leftarrow FASTCUT(H_1)$
    $X_2 \leftarrow FASTCUT(H_2)$
    return minimum cut of $X_1$ and $X_2$
end
```

Intuitive explanation of how FASTCUT works

- H_1 (and H_2) repetitively contract the graph as long as it is still quite large
- Since $CONTRACT(G, t)$ is randomized, H_1 and H_2 may lead to different results and we want two independent execution series for redundance.
- When the graph becomes small, we recursively run the FASTCUT algorithm, to ensure an appropriately large number of contractions.
- When the graph has less than 6 vertices we calculate the minimum cut by brute force.

Time complexity of FASTCUT

Lemma 5

FASTCUT takes $O(n^2 \log n)$ time, where $n = |V(G)|$

Proof: The CONTRACT operation obviously takes $O(n^2)$ time and FASTCUT calls CONTRACT twice, needing $O(n^2)$ time.

Then, two recursive calls follow, on the resulting H_1, H_2 graphs.

Let $T(n)$ the running time of FASTCUT. Then

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

The solution of this recurrency is $T(n) = O(n^2 \log n)$. $\square$

Correctness (I)

Lemma 6

The probability that CONTRACT $(G, \frac{n}{\sqrt{2}})$ had not contracted cut a minimum cut edge is at least $\frac{1}{2}$.

Proof: CONTRACT (G,t) performs $n-t$ contractions. As in the remark of slide 24, the probability of not contracting a min cut edge is

$$\frac{(n-l)(n-l-1)}{n(n-1)} \text{ where } l = n-t$$

i.e., $$\frac{t(t-1)}{n(n-1)} = \frac{\left\lceil 1+\frac{n}{\sqrt{2}} \right\rceil \left(\left\lceil 1+\frac{n}{\sqrt{2}} \right\rceil - 1 \right)}{n(n-1)} \geq \frac{1}{2} \qquad \square$$

Correctness (II)

Theorem 2

FASTCUT finds the minimum cut with probability larger than $\frac{c}{\log n}$, where c constant large enough.

Proof: Let $P(n)$ the probability that FASTCUT finds the correct cut size on a graph with n vertices.
The probability that the algorithm succeeds in the first call on H_1 is the probability that CONTRACT does not hit a min cut edge (by Lemma 6, this is $\geq \frac{1}{2}$) times the success probability of the recursive call; thus, it is

$$\frac{1}{2} P\left(\frac{n}{\sqrt{2}}\right)$$

.
The failure probability that both H_1 and H_2 fail is $\leq \left[1 - \frac{1}{2} P(\frac{n}{\sqrt{2}})\right]^2$. Thus the algorithm succeeds with probability at least:

$$P(n) \geq 1 - \left[1 - \tfrac{1}{2} P(\tfrac{n}{\sqrt{2}})\right]^2,$$

whose solution is $P(n) \geq \frac{c}{\log n}$, where c is constant. □

Conclusions for FASTCUT

Theorem 3

Running FASTCUT $c\log^2 n$ times guarantees finding the minimum cut correctly with probability at least $1-\frac{1}{n^2}$ (c a constant large enough).

Proof: $Pr\{\text{FASTCUT} fails\} \leq \left(1-\frac{c}{logn}\right)^{c\log^2 n} \leq e^{-c^2 logn} = n^{-c^2}$
$\Rightarrow Pr\{\text{FASTCUT succeeds}\} \geq 1-n^{-2}$
(via choosing c appropriately). □

Note: The total time FASTCUT takes is $O(n^2\log n)$ times
$c\cdot\log^2 n$, i.e. $O(n^2\log^3 n)$.

Comparison of MINCUTREP, FASTCUT

- MINCUTREP takes $O(n^2)$ time to succeed with probability $\geq \frac{2}{n^2}$ and its amplification needs $O(n^4 \log n)$ time to succeed with probability $\to 1$.
- FASTCUT takes more time $[O(n^2 \log n)$, i.e. more repetitions$]$ but succeeds with larger probability $\geq \frac{1}{\log n}$ and its amplification needs $O(n^2 \log^3 n)$ time to succeed with probability $\to 1$.
- In particular, FASTCUT takes $O(n^2)$ time at the start (via CONTRACT, when the graph is large) and $O(n^2 \log n)$ time later on at the recursive calls when the graph is smaller.