CS 761: Randomized Algorithms **University of Waterloo: Winter 2025**

Lecture 2

January 09, 2025

Instructor: Sepehr Assadi

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

Topics of this Lecture

In today's lecture, we will see two other simple applications of randomized algorithms: finding minimum cuts and finding (approximate) maximum cuts.

1 Cuts in Undirected Graphs

We start by reviewing some basic definitions.

Given an undirected graph $G = (V, E)$, a **cut** is any partition of the vertices into two (disjoint) non-empty sets $(S, V \setminus S)$ – note that the partitions $(S, V \setminus S)$ and $(V \setminus S, S)$ refer to the same cut and so we can simply denote a cut by a set S without loss of generality. Any cut $\emptyset \subset S \subset V$ defines a set of **cut edges**, denoted by $\delta(S)$, as the edges that have one endpoint in each side of the partition $(S, V \setminus S)$. Formally,

$$\delta(S) := \{e = (u, v) \in E \mid u \in S \ \wedge \ v \in V \setminus S\}.$$

We sometimes say that these edges **cross** the cut S. Figure 1 gives an illustration of these definitions.

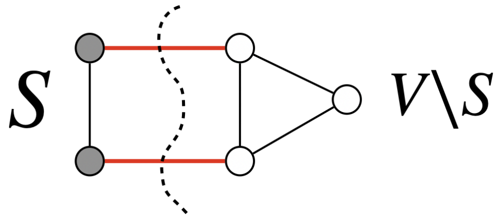


Figure 1: An illustration of graph cuts and cut-edges. The vertices in S-side of the cut are marked gray and the $(V \setminus S)$-side are marked white. The cut-edges are drawn with thick red lines.

Two interesting quantities (among several others) when talking about cuts are **minimum cuts** and **maximum cuts**; the former is any cut with the smallest number of cut edges and the latter is with the largest number instead. In this lecture, we see an algorithm for each problem.

2 Karger's Contraction Algorithm for Minimum Cuts

Throughout, for a graph $G = (V, E)$, we use $\lambda(G)$ to denote the size of a minimum cut in G. Our goal in this section is to find an algorithm for finding any cut of size $\lambda(G)$ in a given graph G.

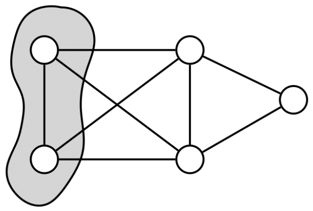
You may have previously seen algorithms for finding minimum cuts using $n-1$ iterations of maximum flow[1]. However, we are now going to see an elegant randomized algorithm due to Karger in [**?**] using more elementary tools with an extremely simple analysis.

We start by introducing a basic graph theoretic operation at the core of Karger's algorithm and then get to the basic version of the algorithm, and finally, the complete algorithm itself.

2.1 Contraction in Graphs

Given an undirected graph $G = (V, E)$ and a set $A \subseteq V$ of vertices, the **contraction** of A in G is a *multi-graph*[2], denoted by $G_{|A}$, with vertex-set $V \setminus A \cup \{a\}$ (where a is a newly defined vertex), and edge-set including all edges in G with both-endpoints in $V \setminus A$, and new edges (a, v) for each edge $(u, v) \in E$ with $u \in A$ and $v \in V \setminus A$.

Informally speaking, contraction of A in G corresponds to replacing all vertices in A into a single vertex a, removing all edges that were inside A, replacing all edges between A and $V \setminus A$ by edges between a and $V \setminus A$, and keeping all the other edges intact. See Figure 2 for an example.


(a) The contraction set in the original graph.

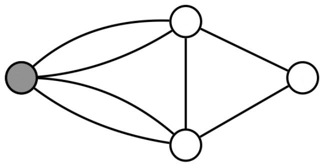

(b) The resulting graph.

Figure 2: An example of a graph contraction.

We will use the following properties of contractions with respect to minimum cuts.

Claim 1. *For any graph $G = (V, E)$ and any set $A \subseteq V$, $\lambda(G_{|A}) \geqslant \lambda(G)$.*

Proof. Let $V_A := V \setminus A \cup \{a\}$ denote the vertex-set of $G_{|A}$. Consider any cut $(S, V_A \setminus S)$ in $G_{|A}$ and without loss of generality assume $a \in S$. Then, the cut $S \setminus \{a\} \cup A$ in G has exactly the same number of cut-edges in G as S does in $G_{|A}$.

This implies that for any cut in $G_{|A}$, there is a cut of the same size in G. Since $\lambda(G)$ is the size of *minimum* cuts, this means $\lambda(G)$ can only become smaller than (or equal to) $\lambda(G_{|A})$. $\square$

Claim 2. *Fix any graph $G = (V, E)$ and let S be a minimum cut in G. Then, for any set $A \subseteq V$ such that $A \cap S = \emptyset$ or $A \cap S = A$, we have $\lambda(G_{|A}) = \lambda(G)$.*

[1]Throughout this course, we use n to denote the number of vertices in the underlying graph and may not specify it each time explicitly.

[2]Recall that a multi-graph is the same as a graph, except that we allow more than one edge between each pair of vertices, namely, we allow *parallel* edges.

Proof. We only prove the claim for when $A \cap S = \emptyset$; the other case is symmetric by noting that if $A \cap S = A$, then $A \cap (V \setminus S) = \emptyset$ and the sets S and $(V \setminus S)$ correspond to the same cut.

Since $A \cap S = \emptyset$, we have that $\delta_{G_{|A}}(S)$ is exactly the same as $\delta_G(S)$ (where $\delta_{(\cdot)}(S)$ is the cut-edges of S in the corresponding graph '$(\cdot)$'), by the definition of a contraction. This implies that

$$\lambda(G_{|A}) \leqslant |\delta_{G_{|A}}(S)| = |\delta_G(S)| = \lambda(G).$$

□

Claim 2 now gives us a recipe for solving the minimum cut problem: find a set A of vertices that does not intersect with the minimum cut, contract this set to get a smaller graph $G_{|A}$, and recurse on the smaller graph. The non-intersecting property of A together with Claim 2 allows us to recover a minimum cut of G from that of $G_{|A}$ (by simply replacing the vertex a in the minimum cut with the set A). This is all well and good except for an obvious reason: how can find a set that does not intersect a minimum cut without knowing the minimum cut in the first place?

This is where the true brilliance of the algorithm and its use of randomization lies: There is a very simple way of picking A randomly which ensures that with some"good" probability, we are not going to "destroy" the minimum cut.

2.2 The Basic Contraction Algorithm

We can now present the basic version of the contraction algorithm.

Algorithm 1. `basic-contraction`(G): given an undirected multi-graph $G = (V, E)$,

(i) Sample an edge $e = (u, v)$ uniformly at random from G;

(ii) Contract the set $\{u, v\}$ in G to obtain $G_{\{u,v\}}$;

(iii) Recurse on $G_{\{u,v\}}$ and continue until only two vertices are remained; then, return the sets corresponding to these two (possibly) contracted vertices.

Figure 3 gives an illustration of this algorithm.

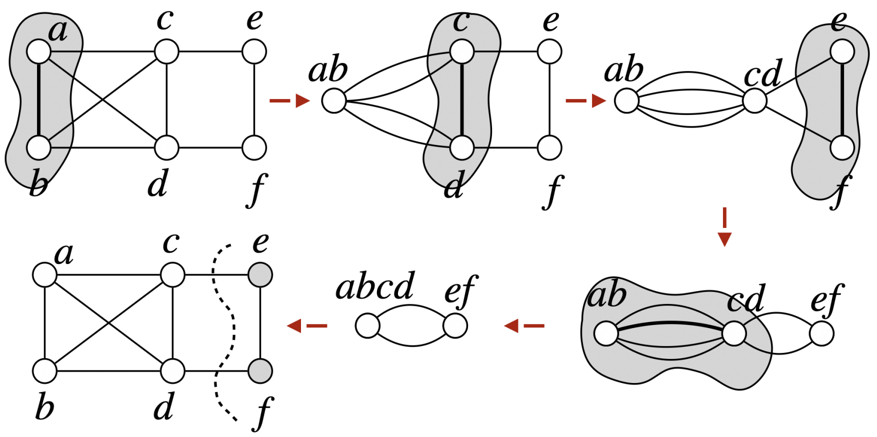


Figure 3: An illustration of a successful run of the `basic-contraction`.

We now analyze the main properties of this algorithm. The first step is to lower bound the probability that each fixed step of the algorithm "preserves" the minimum cut.

Lemma 3. *Let $G = (V, E)$ be a multi-graph and (u, v) be a uniformly random edge chosen in the first step of* `basic-contraction`(G)*. Then,*

$$\Pr\left(\lambda(G) = \lambda(G_{|\{u,v\}})\right) \geqslant 1 - \frac{2}{n},$$

where n is the number of vertices in G.

Proof. Notice that for any vertex $v \in V$, the singleton cut $\{v\}$ has size equal to the degree of v in G, denoted by $\deg_G(v)$. This implies that for all $v \in V$, $\deg_G(v) \geqslant \lambda(G)$. On the other hand, we know that $\sum_v \deg_G(v) = 2|E|$ (this is often called the handshaking lemma); thus, $|E| \geqslant n \cdot \lambda(G)/2$.

Fix any minimum cut S of G. If the edge (u, v) is not a cut-edge of S, then, $\{u, v\}$ does not intersect with one side of the cut S, and thus by Claim 2, we will have $\lambda(G) = \lambda(G_{|\{u,v\}})$. This implies that,

$$\begin{aligned}
\Pr\left(\lambda(G) = \lambda(G_{|\{u,v\}})\right) &\geqslant \Pr\left((u, v) \notin \delta(S)\right) && \text{(by Claim 2)} \\
&= 1 - \frac{|\delta(S)|}{|E|} && \text{(as we pick } (u, v) \text{ uniformly at random from edges in } E) \\
&\geqslant 1 - \frac{\lambda(G)}{n \cdot \lambda(G)/2} && \text{(as } |\delta(S)| = \lambda(G) \text{ and } |E| \geqslant n \cdot \lambda(G)/2 \text{ as calculated above)} \\
&= 1 - \frac{2}{n}.
\end{aligned}$$

□

We can now use this lemma to bound the probability that `basic-contraction` outputs a minimum cut.

Lemma 4. *For any multi-graph $G = (V, E)$,*

$$\Pr\left(\texttt{basic-contraction}(G) \text{ outputs a minimum cut of } G\right) \geqslant \frac{2}{n \cdot (n-1)}.$$

Proof. The proof is by a repeated application of Lemma 3. Notice that each contraction step reduces the number of vertices and the algorithm stops when only two vertices are left. Thus, there are $n-2$ contraction steps. For each $i \in [n-2]$, let G_i denote the resulting graph after the i-the contraction step. We thus have,

$$\Pr\left(\texttt{basic-contraction}(G) \text{ outputs a minimum cut of } G\right) = \Pr\left(\lambda(G_{n-2}) = \lambda(G)\right).$$

We are now going to calculate the RHS above. Notice that by Claim 1, we always have,

$$\lambda(G_{n-2}) \geqslant \lambda(G_{n-3}) \cdots \geqslant \lambda(G_1) \geqslant \lambda(G).$$

Thus, to bound the RHS, we can write,

$$\begin{aligned}
\Pr\left(\lambda(G_{n-2}) = \lambda(G)\right) &= \Pr\left(\lambda(G_{n-2}) = \lambda(G_{n-3}) \wedge \lambda(G_{n-3}) = \lambda(G_{n-4}) \wedge \cdots \wedge \lambda(G_1) = \lambda(G)\right) \\
&= \prod_{i=1}^{n-2} \Pr\left(\lambda(G_i) = \lambda(G_{i-1}) \mid \lambda(G_{i-2}) = \cdots = \lambda(G_1) = \lambda(G)\right). \\
&\qquad \text{(by the definition of conditional probability: } \Pr(A \wedge B) = \Pr(B) \cdot \Pr(A \mid B))
\end{aligned}$$

But each term in the RHS of the above equation corresponds to the probability that a single step of `basic-contraction` preserves the minimum cut value in its contraction, which is precisely the quantity lower bounded in Lemma 3. Thus, we have,

$$\prod_{i=1}^{n-2} \Pr\left(\lambda(G_i) = \lambda(G_{i-1}) \mid \lambda(G_{i-2}) = \cdots = \lambda(G_1) = \lambda(G)\right) \geqslant \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right)$$

(by Lemma 3 and since the number of vertices in G_{i-1} is $n-i+1$)

$$= \prod_{i=1}^{n-2} \left(\frac{n-i-1}{n-i+1}\right)$$

$$= \left(\frac{n-2}{n}\right) \cdot \left(\frac{n-3}{n-1}\right) \cdot \left(\frac{n-4}{n-2}\right) \cdot \cdots \cdot \left(\frac{2}{4}\right) \cdot \left(\frac{1}{3}\right)$$

$$= \frac{2}{n \cdot (n-1)}.$$

(as the terms telescopically cancel each other except for first two denominators and last two nominators)

This implies the `basic-contraction`(G) outputs a minimum cut of G with probability at least $2/n \cdot (n-1)$, concluding the proof. ☐

Before continuing, let us mention an important corollary of these lemmas in bounding the total number of different minimum cuts in a graph (notice that minimum cuts may not be unique).

Corollary 5. *Any undirected graph $G = (V, E)$ contains at most $\binom{n}{2}$ different minimum cuts.*

Proof. It can be easily verified that the conclusions of Lemma 3 and Lemma 4 is actually stronger than stated and is the following: for any fixed minimum cut S of G,

$$\Pr\left(\texttt{basic-contraction}(G) \text{ outputs } S\right) \geqslant \frac{2}{n \cdot (n-1)} = \binom{n}{2}^{-1}.$$

But now note we can write

$$\Pr\left(\texttt{basic-contraction}(G) \text{ outputs some minimum cut}\right) = \sum_{S \in \text{minimum cuts}} \Pr\left(\texttt{basic-contraction}(G) \text{ outputs } S\right)$$

(as the events for S partition the event in LHS)

$$\geqslant \# \text{ of minimum cuts} \cdot \binom{n}{2}^{-1}.$$

(by the bounds stated above)

Given that the LHS above can be upper bounded by 1 (as it is a probability), we get the desired bound on the number of minimum cuts. ☐

2.3 The Final Algorithm

We are now almost done. Algorithm 1 already solves the problem for us except that its probability of success is too low (and tends to zero when size of the graph goes to infinity). On the other hand, whenever we work with randomized algorithms, we either require them to be always true and only risk on their resources, namely, we bound their expected running time, or require them to be correct with probability at least some large constant, typically 2/3. [3]

Thus, the last step of our approach is to *boost* the probability of success of Algorithm 1 to be some large constant. This can be done quite easily by simply running the algorithm multiple times and returning the best answer. Formally,

[3] The convention is to call the former family of algorithms Las Vegas algorithms and the latter ones Monte Carlo algorithms.

Algorithm 2. Given an undirected multi-graph $G = (V, E)$,

(*i*) Run `basic-contraction`(G) for n^2 times *independently* to find the cuts $S_1, \ldots, S_{n^2}$;

(*ii*) Return the smallest of these cuts as a minimum cut of the graph G.

Lemma 6. *For any undirected multi-graph $G = (V, E)$,*

$$\Pr(\text{Algorithm 2 outputs a minimum cut of } G) \geqslant 2/3.$$

Proof. By definition, Algorithm 2 fails in outputting a minimum cut only if every single one of its n^2 independent runs of `basic-contraction`(G) fails in doing so. By Lemma 4, the probability of failure for each of these runs is upper bounded by $1 - 2/n^2$ (we simply upper bounded $n \cdot (n-1)$ by n^2). Each of these events happen independently also and thus we have,

$$\Pr(\text{Algorithm 2 fails}) = \Pr\big(\text{every one of } n^2 \text{ runs of } \texttt{basic-contraction}(G) \text{ fail}\big) \leqslant \left(1 - \frac{2}{n^2}\right)^{n^2}$$

(by Lemma 4 and independence of the n^2 runs)

$$\leqslant \exp(-\frac{2}{n^2} \cdot n^2) = \exp(-2) < 1/3;$$

in the first inequality of the second line, we used the fact that $1 - x \leqslant e^{-x}$ (we use e^{-x} and $\exp(-x)$ interchangeably) for every $x \in (0, 1)$.[4] This concludes the proof. □

We did not put any emphasize on the runtime of this algorithm. However, it is easy to see that each contraction step of Algorithm 1 can be implemented in $O(n)$ time by modifying the at most $O(n)$ edges incident on the contracted vertices. This means that Algorithm 1 itself can be run in $O(n^2)$ time. This in turn implies that Algorithm 2 is an $O(n^4)$ time algorithm. This is not a particularly efficient algorithm for the problem but it makes up for it by being extremely simple. We will also see later in this course how essentially the same ideas can lead to a much faster algorithm as well.

3 A 1/2-Approximation Algorithm for Maximum Cuts

We now switch to solving the maximum cut problem. Unlike the minimum cut problem, finding a maximum cut is NP-hard and thus we should not expect to be able to solve this problem in polynomial time[5]. Thus, we are going to instead focus on obtaining an *approximate* solution, namely, outputting a cut which is provably at most a factor α smaller than the maximum cut of the graph (namely, is "large enough" depending on if we can get α to be large). Specifically, we will see a very simple randomized algorithm that achieves a 1/2-approximation in expectation. The algorithm is extremely simple:

Algorithm 3. Given an undirected graph $G = (V, E)$,

(*i*) Sample every vertex $v \in V$ to be in the set S independently and with probability half.

(*ii*) Return $(S, V \setminus S)$ as an approximate maximum cut of G.

[4]Despite being a very simple inequality to prove, it is also extremely useful and we will be using it very often in this course. You are strongly encouraged to prove its correctness once for yourself.

[5]Or maybe we should warm up to the idea that P can be equal to NP after all... see this brilliant piece by Emanuele Viola on why he believes P = NP:

https://www.ccs.neu.edu/home/viola/papers/moti.pdf

See Chapter 19, but while you are at it, you may want to check out lots of other cool stuff in there as well.

Lemma 7. *For any graph G with m edges, running Algorithm 3 returns a cut S satisfying:*

$$\mathbb{E}\,|\delta(S)| \geqslant m/2.$$

Given that maximum cut of G can have at most m edges clearly, the returned solution of Lemma 7 is a 1/2-approximation *in expectation*. Before getting to prove this lemma, a quick question: is getting a solution which is good in expectation only good enough for us? Maybe yes or maybe not depending on the situation; but see if you can find a way to use this algorithm as a blackbox to obtain an algorithm that can get arbitrary close to a 1/2-approximation with a probability that is arbitrarily close to 1.

Proof of Lemma 7. For any edge $e = (u, v)$ in G, define a random variable $X_e \in \{0, 1\}$ which is 1 iff the edge e is a cut edge of the cut returned by Algorithm 3. We have,

$$\mathbb{E}\,|\delta(S)| = \mathbb{E}\left[\sum_{e\in E} X_e\right] = \sum_{e\in E} \mathbb{E}\,[X_e] = \sum_{e=(u,v)\in E} \Pr\left(\text{exactly one of } u \text{ or } v \text{ is in } S\right) = |E|\cdot\frac{1}{2} = \frac{m}{2};$$

here, the second equality is by linearity of expectation, the third is by the definition of a cut edge, and the fourth is because u and v are assigned to inside or outside of S independently and with probability 1/2 each. This concludes the proof. $\square$

3.1 Detour: Derandomization?

Could we have obtained a deterministic algorithm instead? Yes, and there are multiple ways of doing that but we are going to do it through a general technique known as *derandomization*.

Let us start with a more modest goal. Notice that Algorithm 3 uses n random bits. Can we reduce the random bits used by Algorithm 3? A key observation is that in the proof of Lemma 7, the only property we need from our randomness is that for every edge (u, v), we want to have

$$\Pr\left(\text{exactly one of } u \text{ or } v \text{ is in } S\right) = \frac{1}{2}.$$

This can be achieved using much fewer random bits actually. Let us number vertices of G from 0 to $n-1$ and for any vertex $v \in V$, let $x(v) \in \{0,1\}^{\lceil \log n\rceil}$ denote the binary representation of the name of this vertex. Now, suppose we only sample $r \in \{0,1\}^{\lceil \log n\rceil}$ uniformly at random. This way, we have only used at most $\log n + 1$ random bits (instead of n in the original algorithm). To decide whether or not a vertex joins the set S we compute $\langle x(v)\,,\, r\rangle$ in $\mathbb{F}_2$, namely, we compute the inner product of $x(v)$ and r mod two.

Consider any edge $e = (u, v) \in E$. We have,

$$\begin{aligned}\Pr\left(\text{exactly one of } u \text{ or } v \text{ is in } S\right) &= \Pr\left(\langle x(u)\,,\, r\rangle \neq \langle x(v)\,,\, r\rangle\right)\\ &\qquad\text{(by the rule of the algorithm for placing } u \text{ and } v \text{ in or out of } S\text{)}\\ &= \Pr\left(\langle x(u) - x(v)\,,\, r\rangle \neq 0\right).\\ &\qquad\text{(given inner product is a bi-linear operator)}\end{aligned}$$

Notice that $x(u) - x(v)$ is a binary vector in $\{0,1\}^{\lceil \log n\rceil}$ which is not all-zero given u and v are different. We can thus use the following lemma to conclude the proof.

Lemma 8. *Fix any integer $t \geqslant 1$. For any binary vector $z \in \{0,1\}^t$ with $z \neq 0$ and a randomly chosen vector $r \in \{0,1\}^t$,*

$$\Pr_r\left(\langle z\,,\, r\rangle \neq 0\right) = \frac{1}{2},$$

where the computation is done in $\mathbb{F}_2$.

Proof. Consider some index $i \in [t]$ such that $z_i = 1$ which exists by our assumption that $z \neq 0$. We can write

$$\langle z\,,\, r\rangle = \langle z_{-i}\,,\, r_{-i}\rangle + z_i \cdot r_i = \langle z_{-i}\,,\, r_{-i}\rangle + r_i,$$

where z_{-i} (resp. r_{-i}) is interpreted as the $t-1$ dimensional vector obtained from z (resp. r) by omitting its i-th coordinate. Now notice that coordinates of r are still chosen independently. Thus, we first condition on the choice of r_{-i} which fixes $\langle z_{-i}\,,\, r_{-i}\rangle = b$ for some $b \in \{0,1\}$. We now have

$$\begin{aligned}
\Pr\left(\langle z\,,\, r\rangle \neq 0\right) &= \sum_{b \in \{0,1\}} \Pr\left(\langle z_{-i}\,,\, r_{-i}\rangle = b\right) \cdot \Pr\left(\langle z\,,\, r\rangle \neq 0 \mid \langle z_{-i}\,,\, r_{-i}\rangle = b\right) \\
&\qquad \text{(by the law of conditional probabilities)} \\
&= \sum_{b \in \{0,1\}} \Pr\left(\langle z_{-i}\,,\, r_{-i}\rangle = b\right) \cdot \Pr\left(r_i \neq b \mid \langle z_{-i}\,,\, r_{-i}\rangle = b\right) \qquad \text{(by the equation above)} \\
&= \sum_{b \in \{0,1\}} \Pr\left(\langle z_{-i}\,,\, r_{-i}\rangle = b\right) \cdot \frac{1}{2} \qquad \text{(as } r_i \text{ is uniform over } \{0,1\} \text{ conditioned on } r_{-i}\text{)} \\
&= \frac{1}{2}.
\end{aligned}$$

(as sum of probabilities is equal to the probability that the given inner product gets some value which is 1)

This concludes the proof. □

We thus obtained an algorithm for maximum cut that finds a 1/2-approximate cut in expectation and uses only $\lceil \log n \rceil$ random bits!

A deterministic algorithm? Recall that our initial goal was to have an algorithm which is deterministic and so not even uses the "few" random bits of the previous algorithm. But now there is a simple solution. The total possible choices for $\lceil \log n \rceil$ random bits in the previous algorithm is obviously $2^{\lceil \log n \rceil} = O(n)$.

Therefore, instead of sampling random bits r, we are simply going to enumerate over all $O(n)$ possible choices of them and for each one run the algorithm using these "random bits" (which are actually picked deterministically!) and return the best answer. Given that previously, in expectation, we would have obtained a 1/2-approximation, there must exists at least once choice of random bits that gives us such a good solution also – our deterministic algorithm thus finds these random bits and outputs a solution which is at least a 1/2-approximation.

References

[Kar93] David R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 21–30. ACM/SIAM, 1993. 2