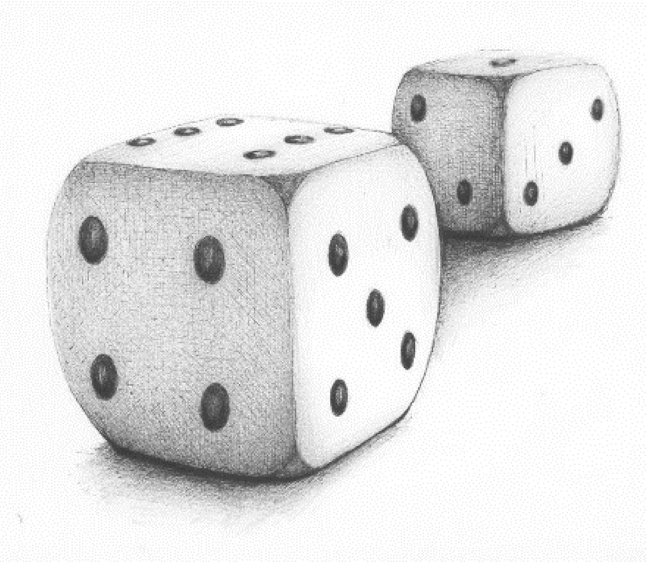


Randomized Algorithms

Yannic Maus

Outline

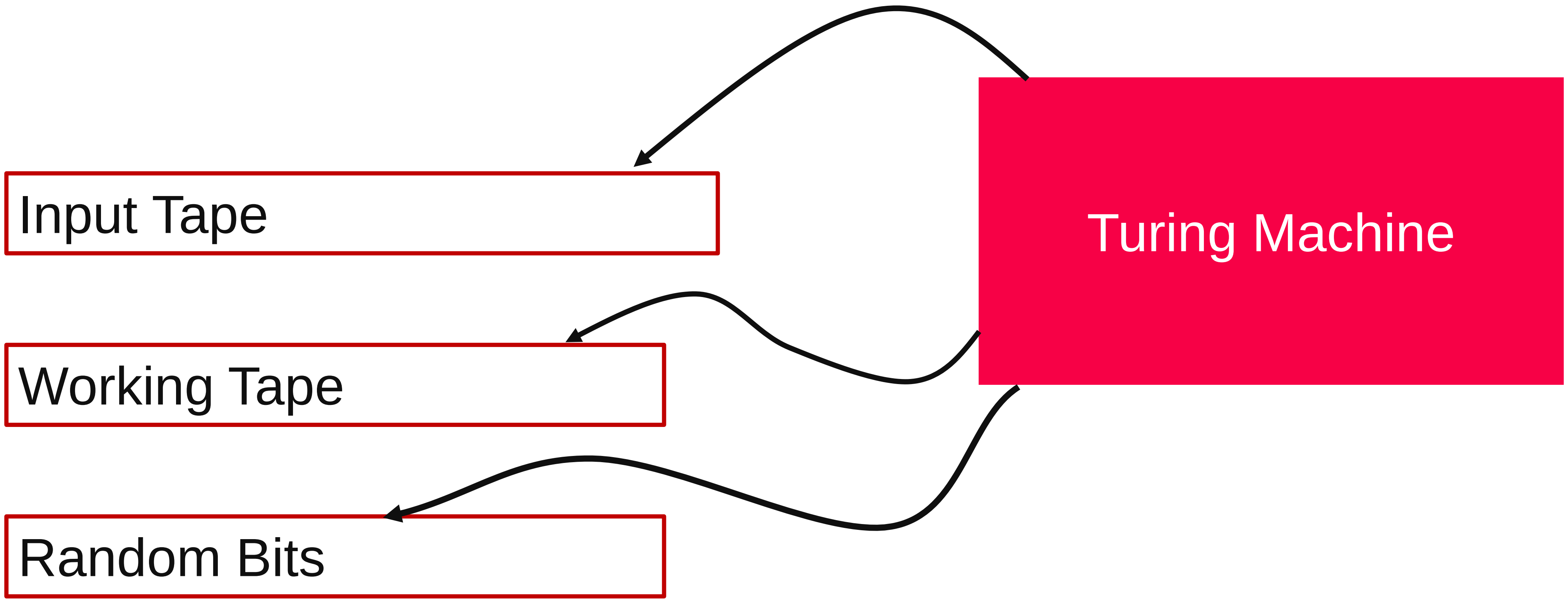
Part I: Randomized algorithms:
Las Vegas algorithms (LV), Monte Carlo algorithms (MC)

Part II: Karger’s min-cut algorithm

Part III: Small toolbox:
Probability boosting, Turn MC to LV
Linearity of expectation, Markov’s inequality, with high probability

Part IV: Randomized Approximation algorithm for max-cut

A glimpse at the computational model



Det. Algorithm = Function(Input)

Rand. Algorithm = Function (Input, Random Bits)

Computational model (randomized algorithm)

- **High level:** Your algorithm can flip coins
- **Example Quicksort:**

  The algorithm flips a coin to decide which element to take as the pivot element.

  Expected Runtime: $O(n \cdot \log n)$

  Worst case runtime: $O(n^2)$

- The output of a randomized algorithm is a random variable
- The execution path of a randomized algorithm is a random variable

Computational model (randomized algorithm)

- The output of a randomized algorithm is a random variable
- The execution path of a randomized algorithm is a random variable

Think of input x as fixed:

1. Flip coins $\mathrm{r}_1, \mathrm{r}_2, \ldots \in \{Heads, Tails\}$
2. Do some computation
3. Output $Alg(x, r_1, r_2, \ldots)$

Possible Statements:

For all inputs x:

$$\mathrm{E}\left[Running\ Time\left(Alg(x, r_1, r_2, \ldots)\right)\right] \leq 10|x| \quad \text{(expected running time)}$$

For all inputs x:

$$\mathbf{Pr}(Alg(x, r_1, r_2, \ldots) \text{ is correct}) \geq 0.3 \qquad \textbf{(error probability)}$$

Las Vegas and Monte Carlo Algorithms

Las Vegas (LV): Always correct, but may be slow

- output always correct
- running time is a random variable (one demands $E[\text{runtime}] < \infty$)

Monte Carlo (MC): Always fast, but may be incorrect

- output is a random variable, may be false
- runtime is bounded by something deterministically

memory aid: *MC = Mostly correct*

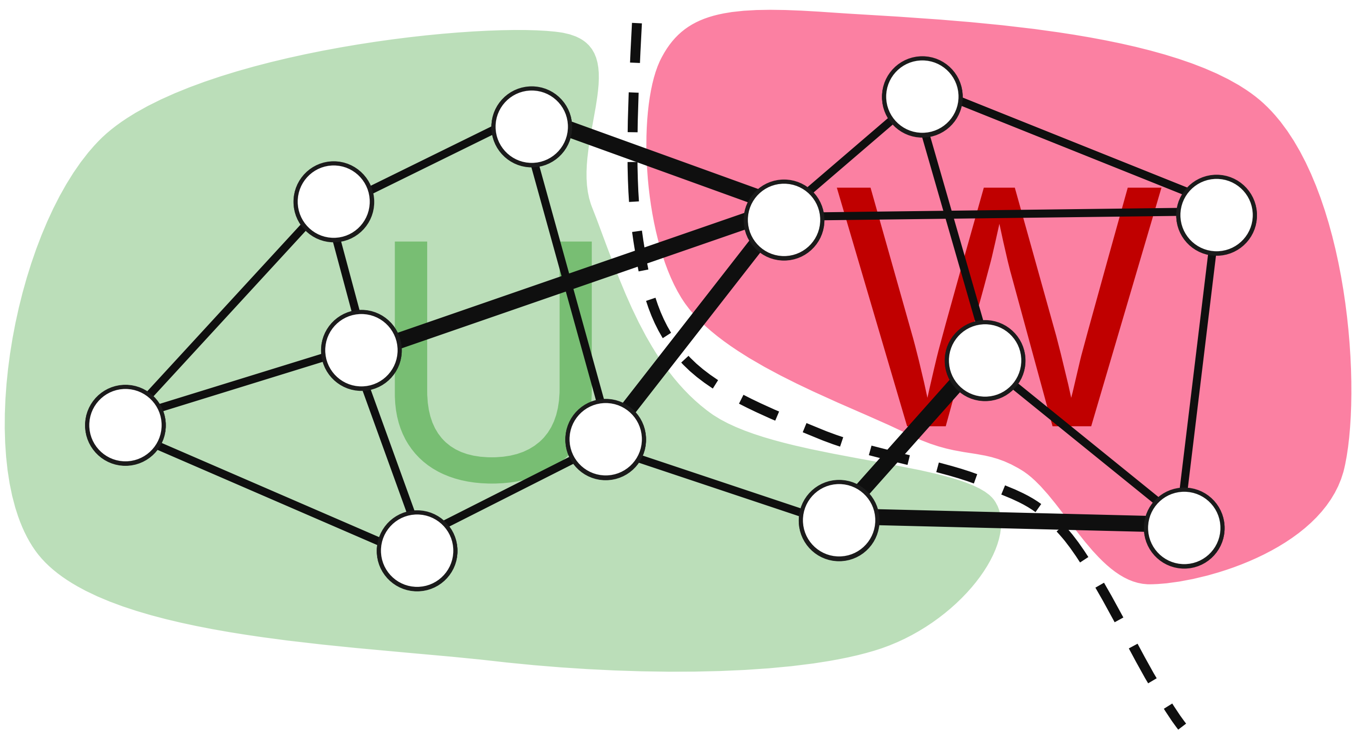
Main focus of this lecture: Monte Carlo Algorithms (MC)

The question that we're asking: $\forall$ inputs x:

- Fix runtime upper bound deterministically as asymptotic function $f(|x|)$
- Provide a lower bound for $\Pr(\text{Alg}(x, r_1, r_2, \ldots) = \text{correct}\ \mathit{output}\ \text{for}\ x))$

Typical statement on a MC algorithm:
The algorithm has runtime $O(n^3)$ and its output is correct with probability 0.9.

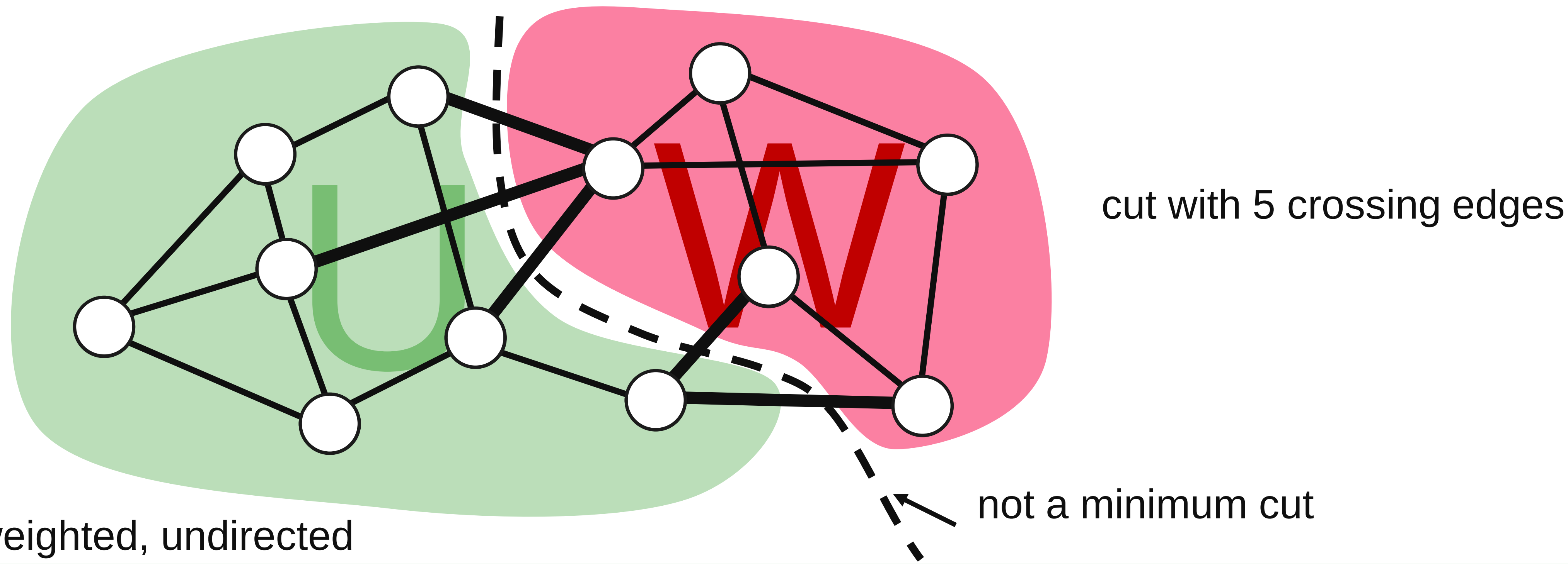
Karger's min-cut algorithm



Karger's Min-Cut algorithm

Definition: A **cut** of a graph $G = (V, E)$ is a partition of its vertices into two disjoint sets $U, W = V \setminus U \subseteq V$. $E(U, W)$ are the edges crossing the cut.

A **minimum cut (min-cut)** is a cut that **minimizes** the number of edges crossing the cut among all cuts.



*graphs are unweighted, undirected

Karger's Min-Cut algorithm

Definition: A **cut** of a graph $G = (V, E)$ is a partition of its vertices into two disjoint sets $U, W = V \setminus U \subseteq V$. $E(U, W)$ are the edges crossing the cut.

A **minimum cut (min-cut)** is a cut that **minimizes** the number of edges crossing the cut among all cuts.

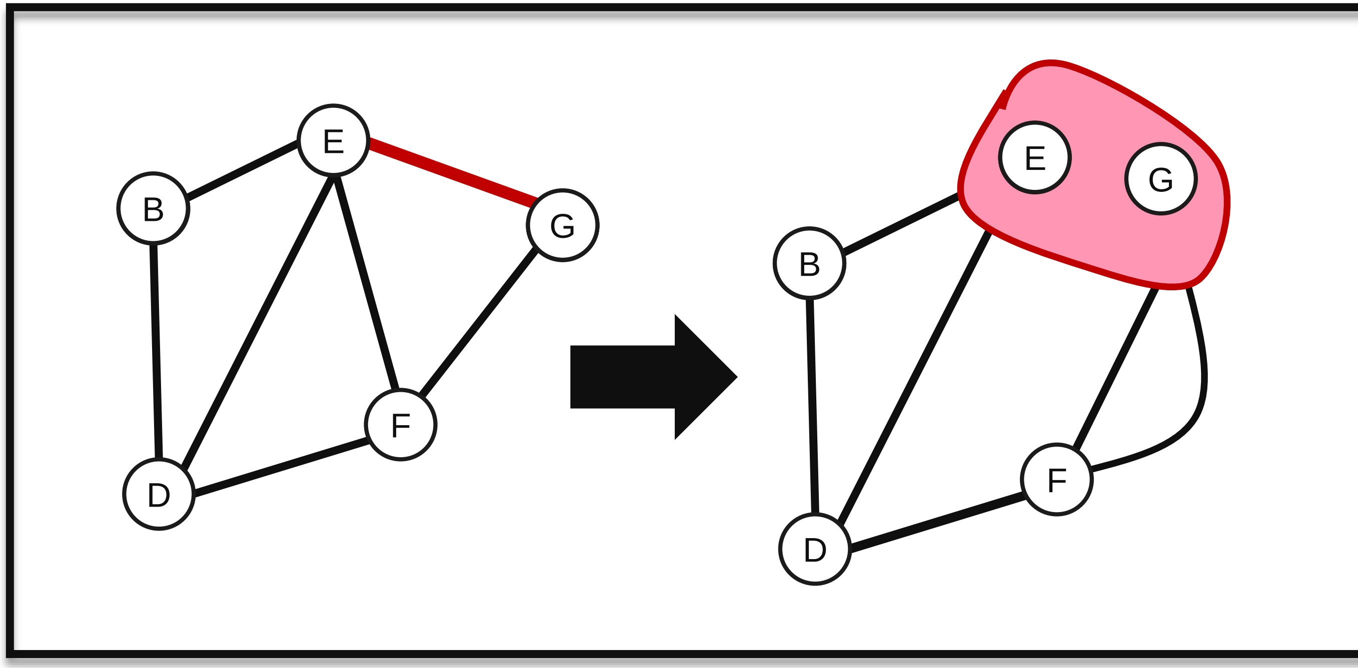
- There may be several minimum cuts.

Remark:
The **max-flow min-s-t-cut theorem** yields a deterministic (involved) algorithm to compute a min-s-t-cut. E.g., in $O(|E|^2|V|) = O(n^5)$, via the Edmonds-Karp Algorithm.

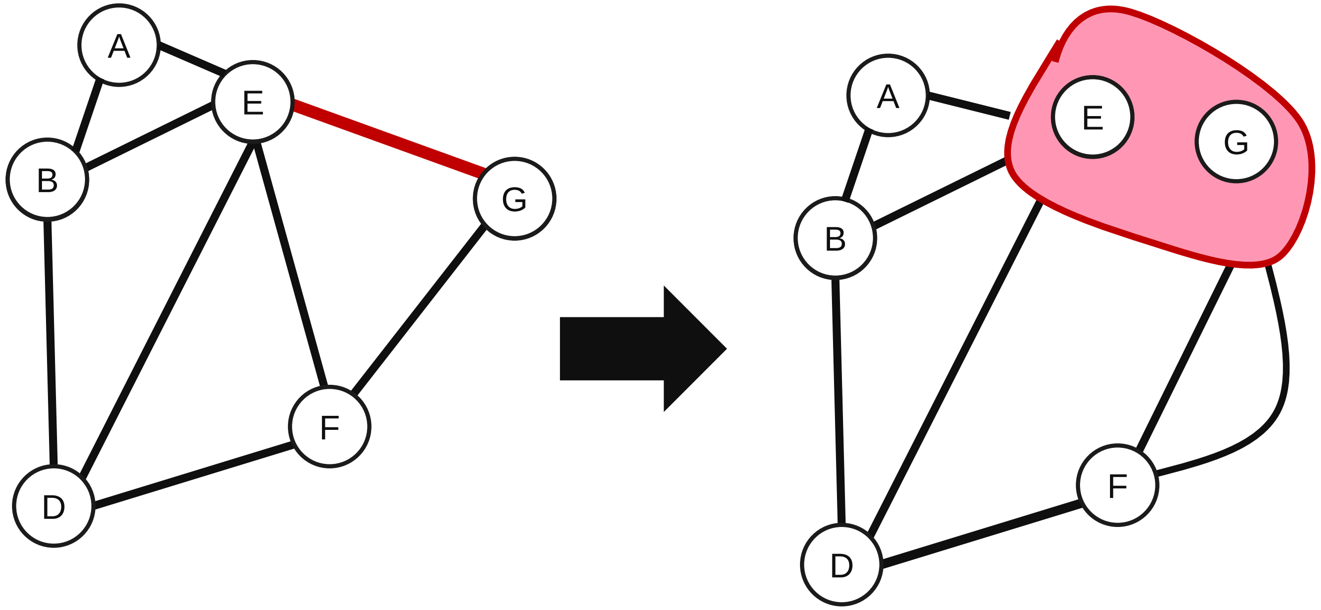
But this is an min-s-t-cut … not a min-cut. What's the difference?

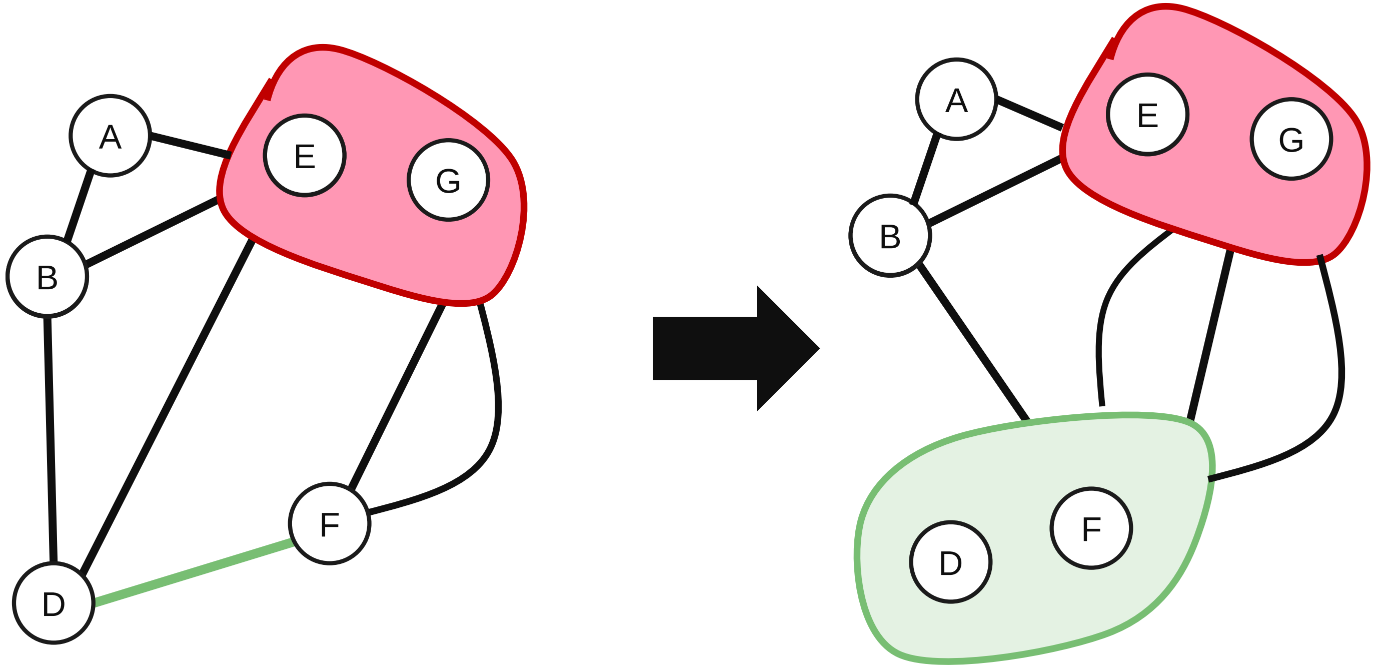
Karger's contraction algorithm

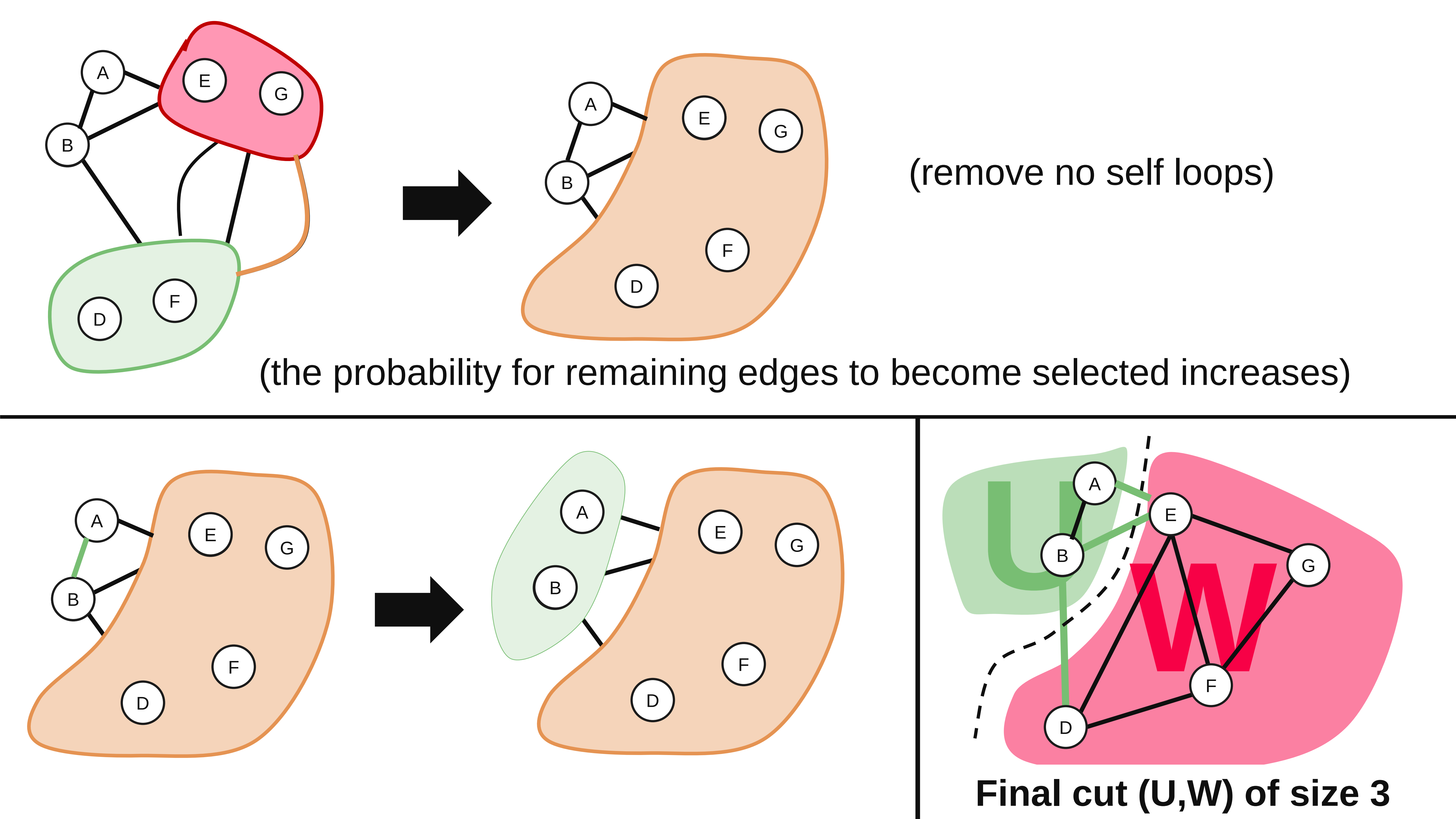
```
Input: Graph G=(V,E)
Output: Cut (U,W) of G

While |V|>2
    pick a random edge e in E
    contract e
    remove self-loops
```




```
Output the cut induced by the two remaining vertices
```


A
E
B
G
F
D
A
E
G
B
F
D


A
E
G
B
F
D
A
E
G
B
D
F


A
B
E
G
D
F
(remove no self loops)
(the probability for remaining edges to become selected increases)
U
W
Final cut (U,W) of size 3

Karger's algorithm: Intuition

- After i steps, we have $n - i$ vertices remaining
- We repeat this for $n - 2$ steps, until we have exactly 2 vertices remaining
- The remaining 2 "super" vertices induce a cut

We want to show the following seemingly weak lemma:

Lemma: Karger contraction algorithm outputs a min-cut with probability at least $2/((n-1)n)$.

Remark: *This seems horrible, but indeed it is pretty good as we will see. It is much better than picking a random cut. There are exponentially* ($2^{|V|} = 2^n$) *many different cuts.*
Intuitively *Karger's algorithm is better than picking a random cut, because it is unlikely that we contract an edge of a minimum cut, simply because there are few such edges.*

Karger's contraction algorithm: Proof

Lemma: Karger contraction algorithm outputs a min-cut with probability at least $2/((n-1)n)$.

Proof:

Consider an arbitrary min cut $(U, W = V \setminus U)$ with $C = E(U,W)$

- $e_1, e_2, e_3, \ldots, e_{n-2}$: the edges contracted by Karger's algorithm
- E_i: event that e_i does not cross the cut C.

$$\Pr(\text{Karger returns cut } C) = \Pr(E_1 \wedge E_2 \wedge E_3, \ldots, \wedge E_{n-2})$$

$$= \Pr(E_1) \cdot \Pr(E_2 | E_1) \cdot \ \ldots \ \Pr(E_{n-2} \mid E_1 \wedge \ldots \wedge E_{n-3})$$

$$\ldots (\text{we will show}) \ldots \geq \frac{2}{n(n-1)}$$

Bounding $\Pr(E_i \mid E_1 \wedge \cdots \wedge E_{i-1})$

$$\Pr(\bar{E}_i | E_1 \wedge .. \wedge E_{i-1}) \leq \frac{\#edges\ in\ cut\ C}{\#remaining\ edges\ after\ i-1\ contractions} \leq \frac{2}{n-i+1}$$

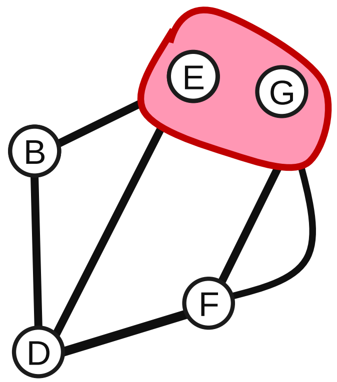
The event that we contract an edge of C in the i-th step, given that we have not contracted any edge of C before

$$\boxed{\Pr(E_i \mid E_1 \wedge \cdots \wedge E_{i-1}) = 1 - \Pr(\bar{E}_i \mid E_1 \wedge \cdots \wedge E_{i-1}1) \geq (n-i-1)/(n-i+1)}$$

$$\#remaining\ edges \geq \#remainingVertices \cdot \frac{minDegree}{2}$$

$$\geq (n-(i-1)) \cdot \frac{minDegree}{2}$$

$$\geq (n-i+1) \cdot (\#\ edges\ in\ cut\ C)/2$$



Karger contraction algorithm: Finalizing the proof

Lemma: Karger's contraction algorithm outputs a min-cut with probability at least $2/(n-1)n$.

Proof:

$$\boxed{\Pr(E_i \mid E_1 \wedge \cdots \wedge E_{i-1}) = 1 - \Pr(\bar{E}_i \mid E_1 \wedge \cdots \wedge E_{i-1}1) \geq (n-i-1)/(n-i+1)}$$

$$\begin{aligned}\Pr(\text{Karger returns cut } C) &= \Pr(E_1 \wedge E_2 \wedge E_3, \dots, \wedge E_{n-2}) \\ &= \Pr(E_1) \cdot \Pr(E_2 \mid E_1) \cdot \dots \Pr(E_{n-2} \mid E_1 \wedge \dots \wedge E_{n-3}) \\ &= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \dots \cdot \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3} \geq \frac{2}{n(n-1)}\end{aligned}$$

end of proof

Theorem: Karger's algorithm

Theorem (Karger): $T = \frac{n(n-1)}{2} \cdot \log 1/\delta$ repetitions of **Karger's contraction algorithm** and returning the smallest cut you see during the process computes a min-cut with probability at least $1-\delta$.

```
min-cut=∞
Repeat for T times
    min-cut=min(min-cut, Karger-Contraction-Alg)
Return min-cut
```

One iteration correct with prob.

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

$$\Pr(\text{output is not a min}-\text{cut}) \leq (1-p)^T \overset{(e^{-p})^T}{\leq} e^{-T \cdot p} = \delta.$$

$\boxed{1-x \leq e^{-x}}$

Remark: *This algorithm only outputs the value of a min-cut. Of course we can also output a min-cut by remembering the best cut found.*

Karger’s algorithm: Implementation

There are many ways to actually implement Karger’s algorithm with varying influence on the complexity.

One Option: Interpret Karger’s algorithm as running Kruskal’s MST algorithm with random edge weights.

- Recall that Kruskal with a union-find data structure maintains connected components of nodes that have been merged by a spanning tree. These components form the role of a super node in Karger’s algorithm .

(the implementation is not the focus of this lecture)

A small Toolbox

For analyzing randomized algorithms

MC: Probability boosting (very important!)

Given: MC algorithm A, correct with probability $p > 0$
New MC algorithm B, correct with probability $\geq 1 - \delta$*>0*

Algorithm B: Repeat algorithm A for $\boldsymbol{p^{-1}\log\left(\frac{1}{\delta}\right)}$ *iterations*
Return “best solution”

Probability that none of the iterations is correct: $(1-p)^i \leq \; e^{-p \cdot i} \; = \delta$

$$\boxed{1 - x \leq e^{-x}, x \in \mathbb{R}}$$

MC: Probability boosting (very important!)

If you have an MC that is correct with probability 1%. Repeat it often enough and return the best solution, and you will have an MC algorithm that is correct with probability 99.9%.

Caveat: How to decide which solution is best?

In Karger's algorithm we saw an approach for probability boosting for maximization/minimization problems [return the largest/smallest solution].

From MC to LV

If you can check whether an output is correct, one can transfer an MC algorithm into an LV algorithm:

```
    Repeat MC algorithm until correct solution is found
```

This will always produces a correct solution (LV algorithm)

- Expected runtime depends on:
  - error probability of your MC (correct with probability p),
  - the runtime $f(n)$ of the MC algorithm, and
  - the runtime $h(n)$ of the checking procedure

If the correctness check is deterministic, **expected runtime** = $\mathrm{x} \cdot (f(n) + h(n))$,
where x is the expected number of p-biased coin flips until you see heads ($x = 1/p$)
(geometric random variable)

Linearity of Expectation

Linearity of expectation: Let $X_1, \dots, X_n$ be random variables and $a_1, \dots, a_n$ real values. Then we have:

$$E[\sum a_i X_i] = \sum a_i E[X_i]$$

- Extremely powerful and important tool
- It does not matter whether the random variables X_i are dependent or not

(should be known from probability theory)

Markov inequality

Markov inequality: If X is a nonnegative random variable and $a > 0$, then the probability that X is at least *a* is at most the expectation of X divided by a:

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

One prime application:

Consider an algorithm that should minimize some value X, and we have designed an algorithm that computes a small value for X, in expectation. Then, we obtain:

$$\Pr(X \geq 3\,E[X]) \leq E[X]/(3E[X]) = 1/3$$

We obtain: $\Pr(X < 3\,E[X]) = 1 - \boldsymbol{Pr(X \geq 3E[X])} \geq 2/3$

*of course this works with other values than 3 as well

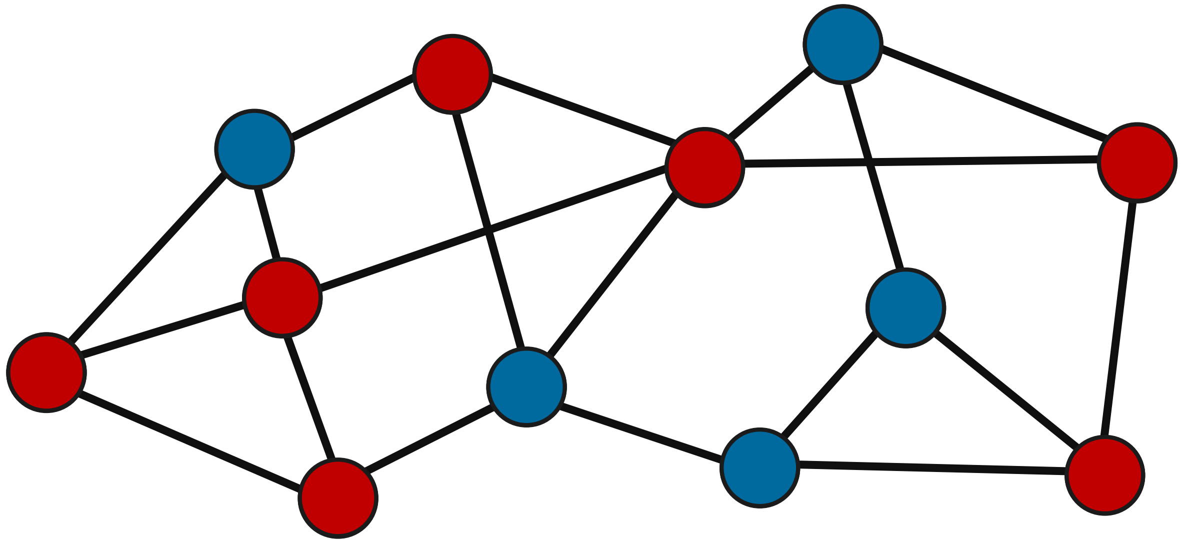
With high probability

Definition (with high probability): An algorithm is correct **with high probability** if its output on an instance of size n is correct with probability $\geq 1 - \frac{1}{n}$.

(Typically, we want that algorithms that are correct w.h.p.)

In other words, the probability that the output is incorrect is at most $1/n$. E.g., for an instance with 100 nodes we require that the input is false with probability at most 1%. On an input with 1000 nodes, we require that the input is false with probability at most 0.1%, etc.

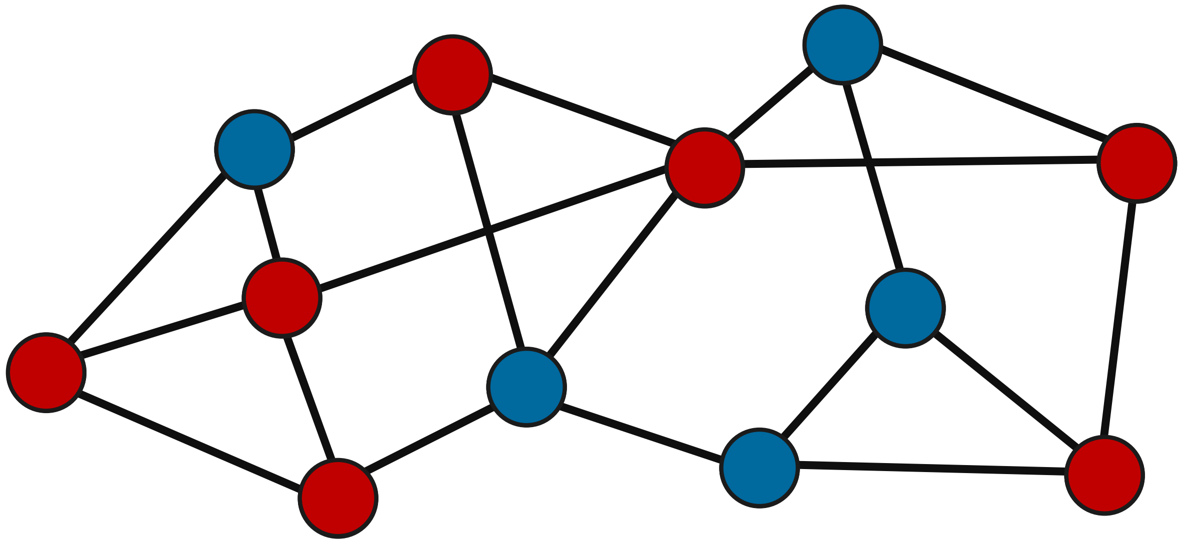
Max-Cut



colors, but not a proper graph coloring

> **Definition:** A **maximum cut (max-cut)** is a cut that **maximizes** the number of edges crossing the cut among all cuts.

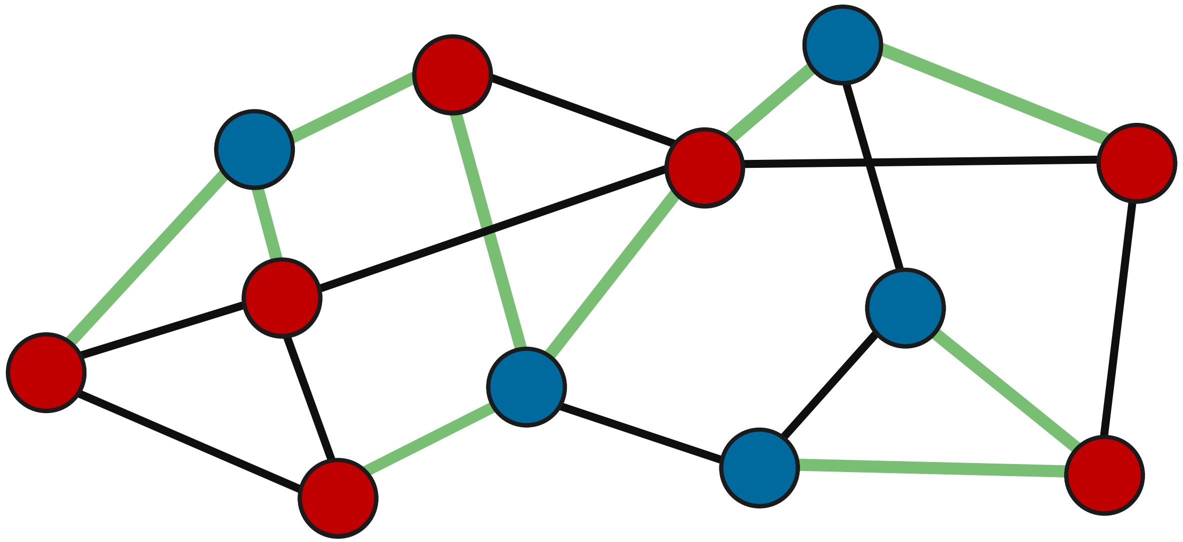
- Max-cut is NP-complete (in contrast to min-cut), not proven in this lecture



Size of the cut?

Definition: A **maximum cut (max-cut)** is a cut that **maximizes** the number of edges crossing the cut among all cuts.

- Max-cut is NP-complete (in contrast to min-cut), not proven in this lecture



Size of the cut?

10 cut edges

Randomized Max-Cut

```
Randomized Algorithm: Color each vertex randomly red/blue
```

How many cut-edges do we expect?

What is the probability for an edge to be a cut-edge?

$\Pr(v = \text{blue} \wedge u = \text{red}) = \Pr(v = \text{blue}) \cdot \Pr(u = \text{red}) = 1/4$

$\Pr(v = \text{red} \wedge u = \text{blue}) = \Pr(v = \text{red}) \cdot \Pr(u = \text{blue}) = 1/4$

$$\Pr(\text{edge } \{u, v\} \text{ is cut edge}) = 1/4 + 1/4 = 1/2$$

Randomized Max-Cut

```
Randomized Algorithm: Color each vertex randomly red/blue
```

For each edge $e \in E$: random variable $X_e = 1$, iff e is cut edge, $X_e = 0$, otherwise

$$E[X_e] = 1 \cdot \Pr(X_e = 1) + 0 \cdot \Pr(X_e = 0) = 1 \cdot \frac{1}{2} = \frac{1}{2}$$

$$X = \sum_{e \in E} X_e$$ Total number of cut edges

$$\boldsymbol{E[X] = E[\sum X_e] = \sum E[X_e] = |E|/2}$$

Very good cut in expectation, w/ const. prob. still good

Lemma: Randomly assigning nodes to the partitions of a cut, in expectation produces $|E|/2$ cut edges.

How to produce a Monte Carlo algorithm?
(for which problem do we get a MC algorithm)

Lemma: Randomly assigning nodes to the partitions of a cut produces **at least $|E|/4$ cut edges**, with probability at least 1/3.

Proof:
Let $Y = |E| - X$ be the number of monochromatic (non-cut edges).
$\boldsymbol{E[Y] = |E| - E[X] = |E|/2}$**.**

$$\boldsymbol{Pr}\left(\boldsymbol{X \leq \frac{|E|}{4}}\right) = \Pr\left(\boldsymbol{Y \geq \frac{3|E|}{4}}\right) \leq \frac{\boldsymbol{E[Y]}}{\frac{\boldsymbol{3|E|}}{\boldsymbol{4}}} = \frac{\boldsymbol{2}}{\boldsymbol{3}}.$$ **(Markov inequality)**

Probability boosting

Theorem: For $\delta > 0$, there is a randomized MC algorithm that outputs a cut with at least $|E|/4$ cut edges with probability at least $1 - \delta$ in $O((|V| + |E|) \cdot \log_{\frac{3}{2}} 1/\delta)$ time.

Proof:
We repeat the previous algorithm $T = \log_{\frac{3}{2}}(1/\delta)$ times and output the largest cut that we see throughout. We obtain

$$\Pr\left(output\ cut < \frac{|E|}{4}\right) = \left(\frac{2}{3}\right)^T \leq \delta.$$

Runtime: The randomized flipping takes O(|V|) steps. Checking the size of the cut in one iteration takes O(|E|) steps.

W.h.p. algorithm for large cuts

Corollary: There is a randomized algorithm that w.h.p. outputs a cut with $|E|/4$ cut edges and has runtime $O((|V| + |E|) \cdot \log n)$.

Proof:
Use the previous theorem and set $\delta = 1/n$ to obtain that the error probability is at most $1/n$.

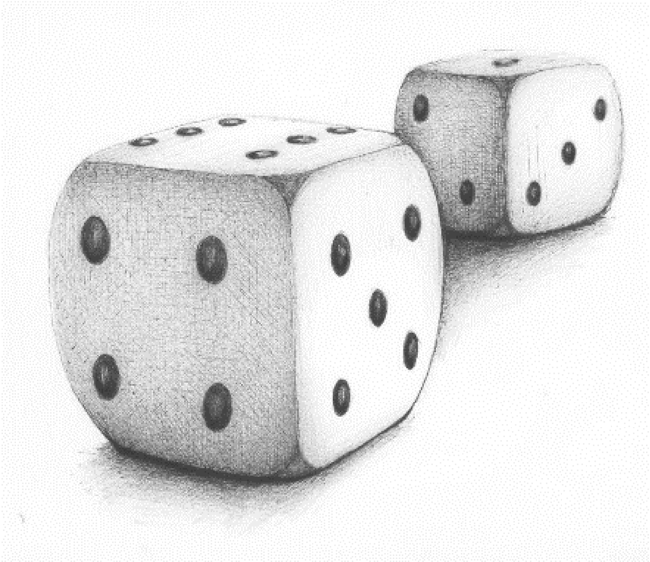
Remark on the approximation guarantee

- Max-cut is NP-complete
- Our algorithm usually does not output an optimal solution
- Still, we get a constant approximation
- There is no PTAS for max-cut unless P=NP

Exercise: Show that the presented algorithm provides a 4-approximation.

Concluding remarks

Randomization is (a) great (tool)



- often simple algorithms
- often difficult analysis