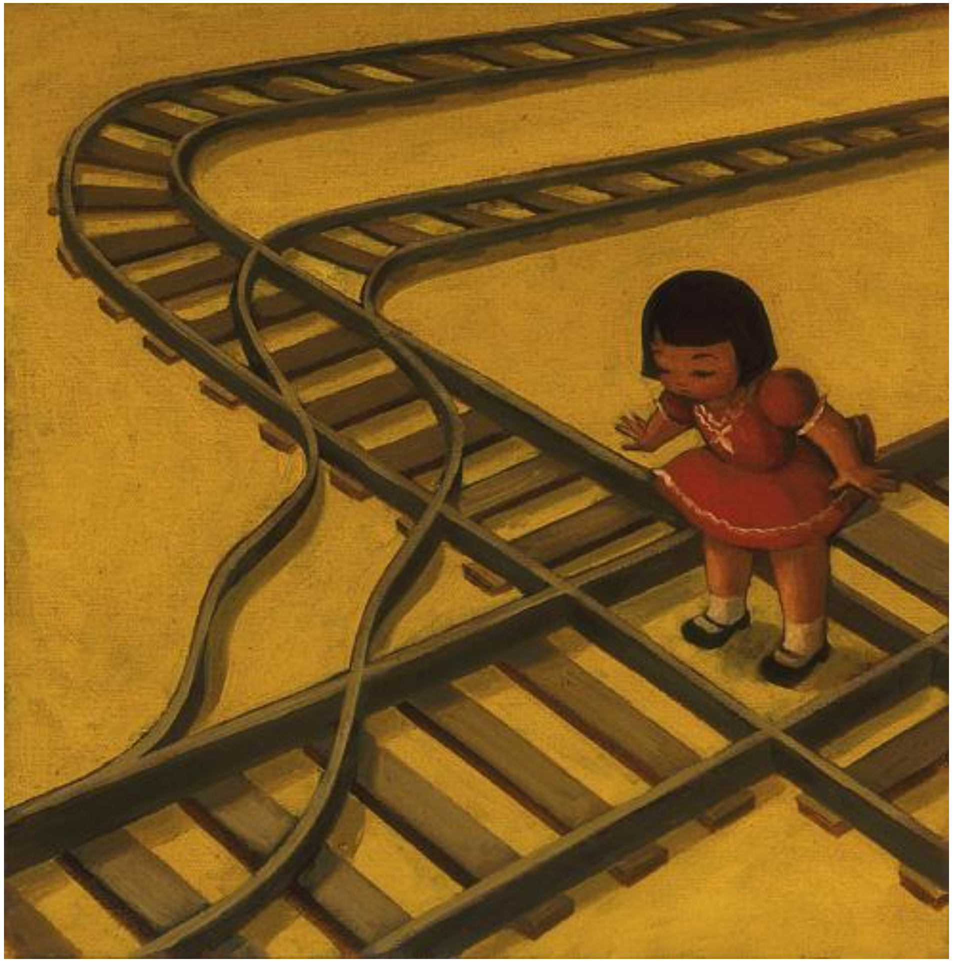
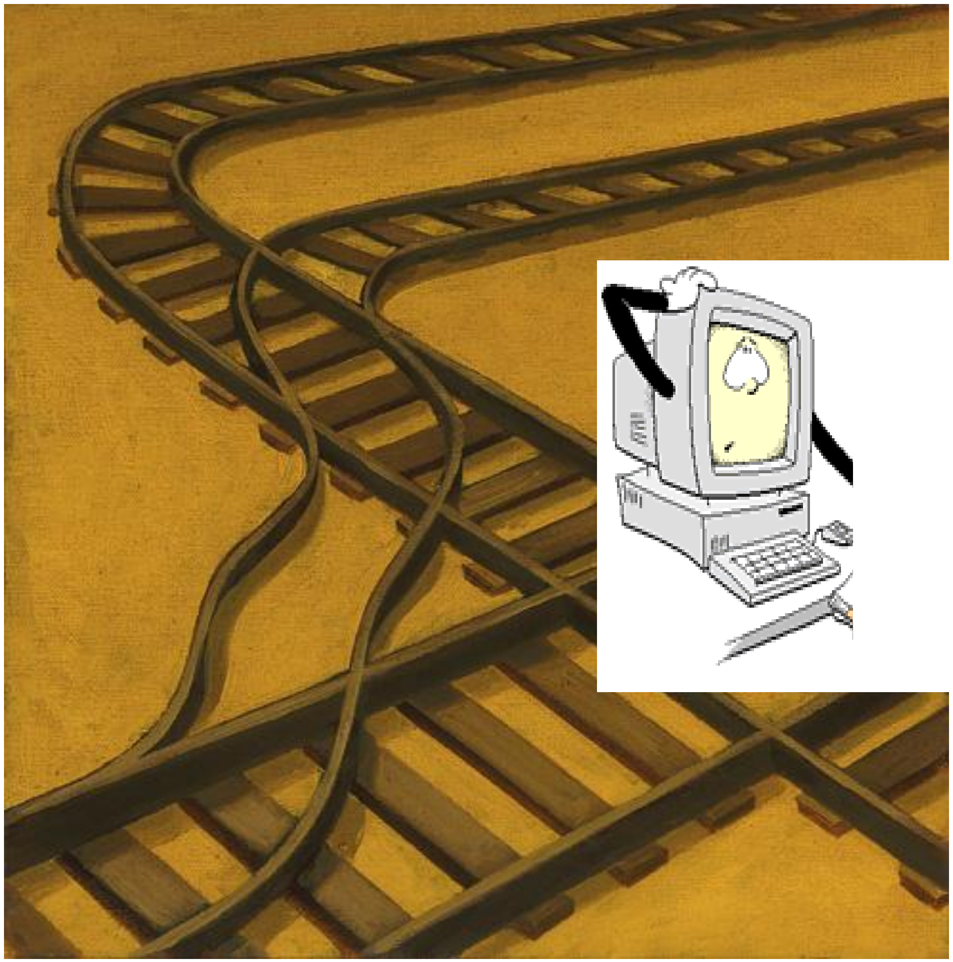
Introduction to Randomized Algorithms and the Probabilistic Method

Making Decision



Flip a coin.

Making Decision



Flip a coin!

An algorithm which flip coins is called a **randomized algorithm**.

Why Randomness?

Making decisions could be complicated.

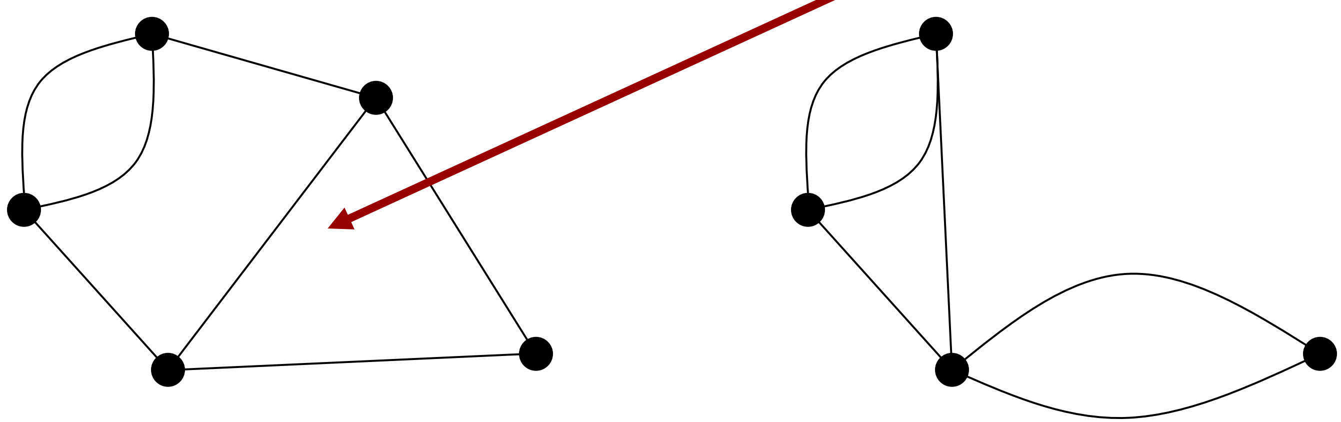
A randomized algorithm is **simpler**.

Consider the minimum cut problem

Can be solved by max flow.

Randomized algorithm?

Pick a random edge and contract.



And repeat until two vertices left.

Why Randomness?

Making good decisions could be expensive.

A randomized algorithm is **faster**.

Consider a sorting procedure.

5 9 13 8 11 6 7 10

5 6 7 8 9 13 11 10

Picking an element in the middle makes the procedure very efficient,
but it is expensive (i.e. linear time) to find such an element.

Picking a random element will do.

Why Randomness?

Making good decisions could be expensive.

A randomized algorithm is **faster**.

- Minimum spanning trees

A linear time randomized algorithm,

but no known linear time deterministic algorithm.

- Primality testing

A randomized polynomial time algorithm,

but it takes thirty years to find a deterministic one.

- Volume estimation of a convex body

A randomized polynomial time **approximation** algorithm,

but no known deterministic polynomial time approximation algorithm.

Minimum Cut

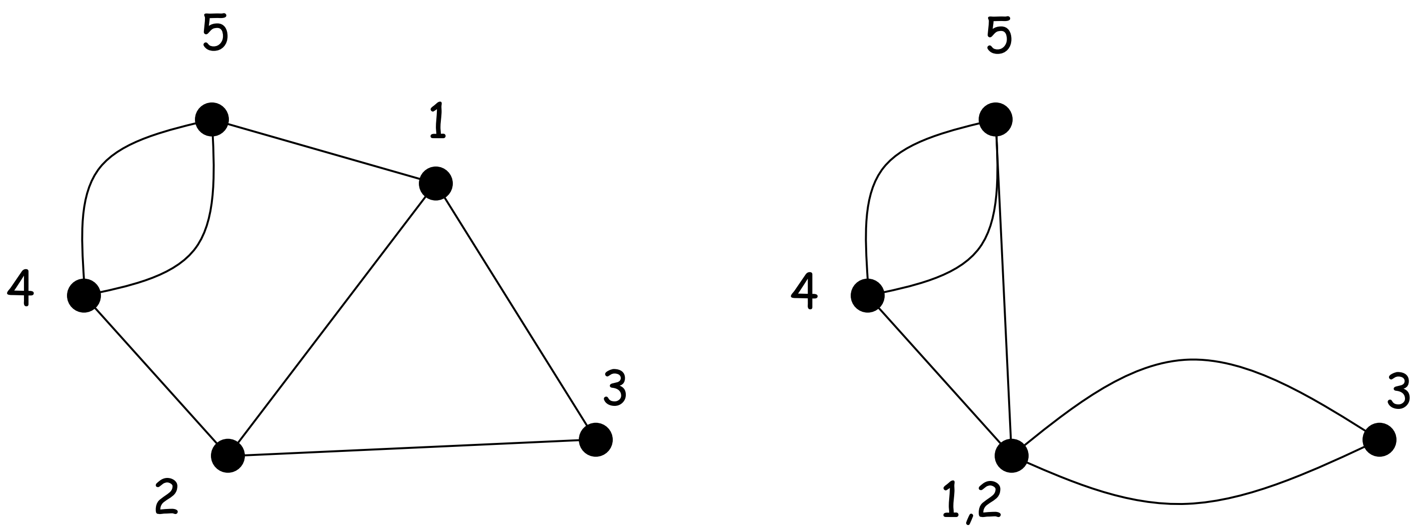
A cut is a set of edges whose removal disconnects the graph.

Minimum cut: Given an undirected multigraph G with n vertices,
find a cut of minimum cardinality (a min-cut).

This problem can be solved in polynomial by the max-flow algorithm.

However, using randomness, we can design a really simple algorithm.

Edge Contraction

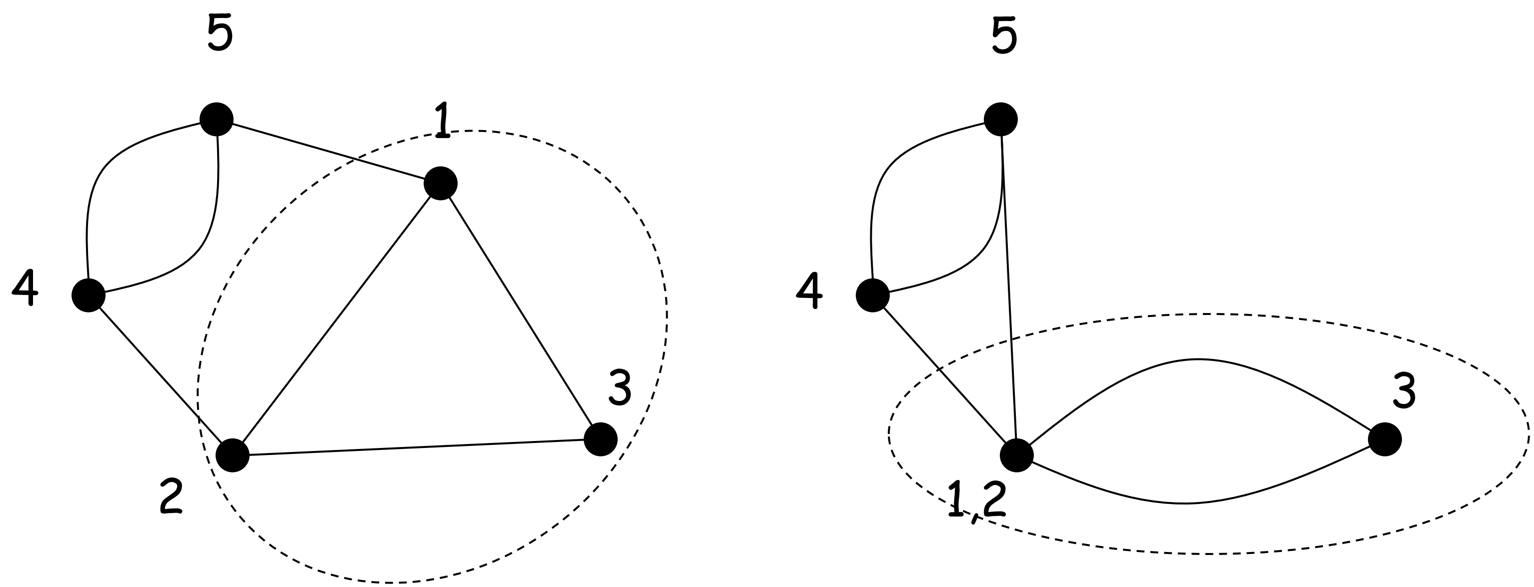


Contraction of an edge e = (u,v): Merge u and v into a single vertex "uv".

For any edge (w,u) or (w,v), this edge becomes (w,"uv") in the resulting graph.

Delete any edge which becomes a "self-loop", i.e. an edge of the form (v,v).

Edge Contraction



Observation: an edge contraction would not decrease the min-cut size.

This is because every cut in the graph at any intermediate stage is a cut in the original graph.

A Randomized Algorithm

Running time:

Each iteration takes O(n) time.

Each iteration has one fewer vertices.

So, there are at most n iteration.

Total running time $O(n^2)$.

Algorithm Contract:

Input: A multigraph G=(V,E)

Output: A cut C

1. H <- G.
2. while H has more than 2 vertices do

   2.1 choose an edge (x,y) uniformly at random from the edges in H.

   2.2 H <- H/(x,y).   (contract the edge (x,y))
3. C <- the set of vertices corresponding to one of the two meta-vertices in H.

Analysis

Obviously, this algorithm will not always work.
How should we analyze the success probability?

Let C be a minimum cut, and $E(C)$ be the edges crossing C.

Claim 1: C is produced as output if and only if none of the edges in $E(C)$ is contracted by the algorithm.

So, we want to estimate the probability that an edge in $E(C)$ is picked.

What is this probability?

Analysis

Let k be the number of edges in E(C).

How many edges are there in the initial graph?

Claim 2: The graph has at least nk/2 edges.

Because each vertex has degree at least k.

So, the probability that an edge in E(C) is picked at the first iteration is at most 2/n.

Analysis

In the i-th iteration, there are n-i+1 vertices in the graph.

Observation: an edge contraction would not decrease the min-cut size.

So the min-cut still has at least k edges.

Hence there are at least (n-i+1)k/2 edges in the i-th iteration.

The probability that an edge in E(C) is picked is at most 2/(n-i+1).

Analysis

$$
\begin{aligned}
&\mathbf{Pr}[C \text{ is output by the Algorithm Contract}]\\
= &\prod_{i=1}^{n-2} \mathbf{Pr}[\text{Edges in } E(C) \text{ are not picked at the i} - \text{th iteration}]\\
\geq &\prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right)\\
= &\prod_{i=1}^{n-2} \left(\frac{n-i-1}{n-i+1}\right)\\
= &\prod_{j=n}^{3} \left(\frac{j-2}{j}\right)\\
= &\frac{2}{n(n-1)} = \Omega(n^{-2})
\end{aligned}
$$

So, a particular min-cut is output by Algorithm Contract with probability at least $\Omega(n^{-2})$.

Boosting the Probability

To boost the probability, we repeat the algorithm for $n^2\log(n)$ times.

What is the probability that it still fails to find a min-cut?

$$
\begin{aligned}
&\mathbf{Pr}[C \text{ is } \mathbf{not} \text{ output by the algorithm}] \\
&\quad \le \left(1 - \frac{1}{n^2}\right)^{n^2 \log(n)} \\
&\quad < \left(\frac{1}{e}\right)^{\log(n)} \\
&\quad = \frac{1}{n}
\end{aligned}
$$

So, high probability to find a min-cut in $O(n^4\log(n))$ time.

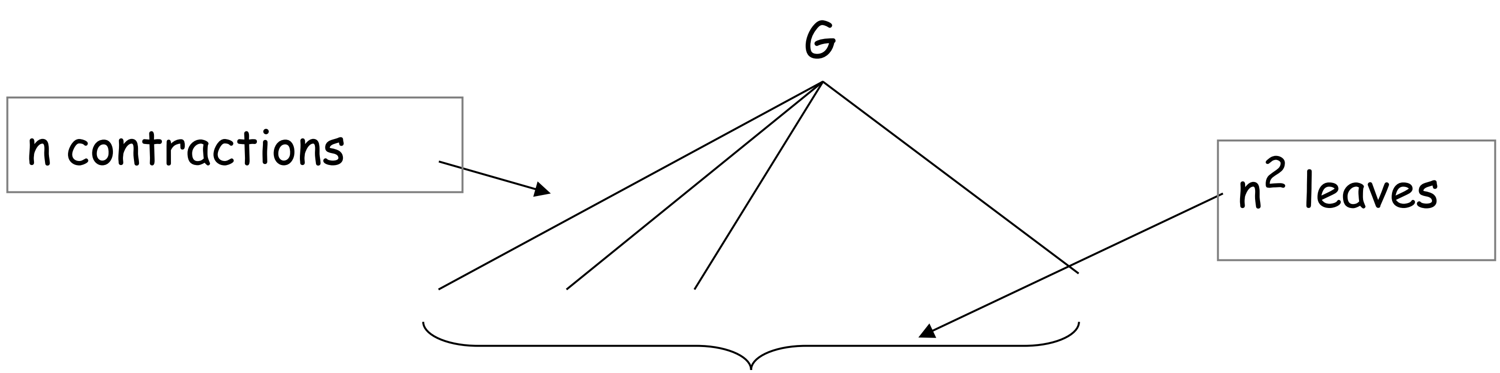
Ideas to an Improved Algorithm

Key: The probability that an edge in E(C) is picked is at most 2/(n-i+1).

Observation: at early iterations, it is not very likely that the algorithm fails.

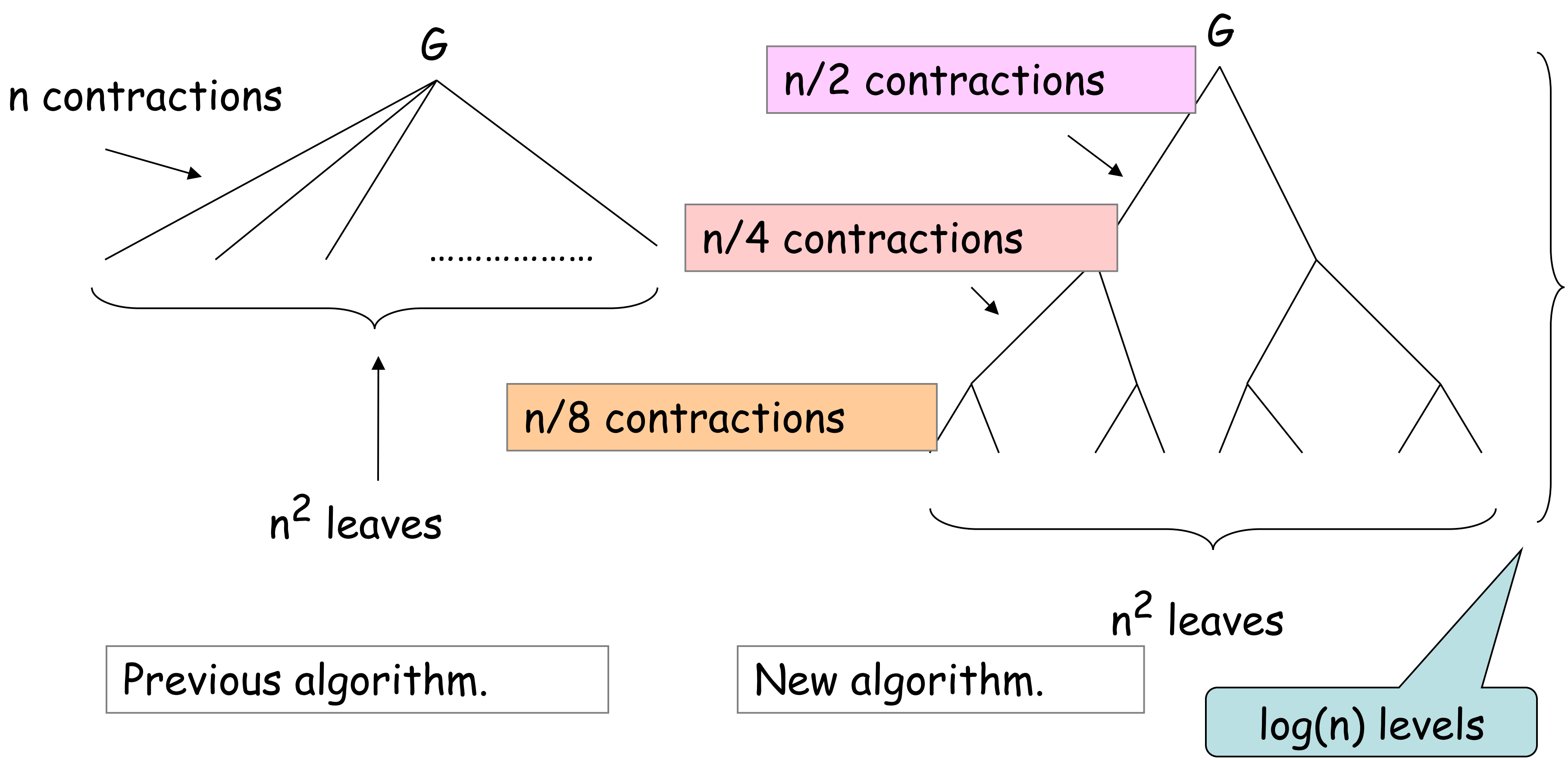
Idea: "share" the random choices at the beginning!

In the previous algorithm, we boost the probability by running
the Algorithm Contract (from scratch) independently many times.



Ideas to an Improved Algorithm

Idea: “share” the random choices at the beginning!



A *Fast* Randomized Algorithm

Algorithm FastCut:

Input: A multigraph $G=(V,E)$

Output: A cut C

1. n <- |V|.
2. **if** n <= 6, **then** compute min-cut by brute-force enumeration **else**

   2.1 t <- (1 + n/√2).

   2.2 Using **Algorithm Contract**, perform two independent contraction sequences to obtain graphs H1 and H2 each with t vertices.

   2.3 Recursively compute min-cuts in each of H1 and H2.

   2.4 **return** the smaller of the two min-cuts.

A Surprise

Theorem 1: Algorithm **Fastcut** runs in $O(n^2 \log(n))$ time.

Theorem 2: Algorithm **Fastcut** succeeds with probability $\Omega(1/\log(n))$.

By a similar boosting argument, repeating Fastcut for $\log^2(n)$ times would succeed with high probability.

Total running time = $O(n^2 \log^3(n))$!
It is much faster than the best known deterministic algorithm, which has running time $O(n^3)$.

Min-cut is faster than Max-Flow!

Complexity

Theorem 1: Algorithm **Fastcut** runs in $O(n^2 \log(n))$ time.

Formal Proof:

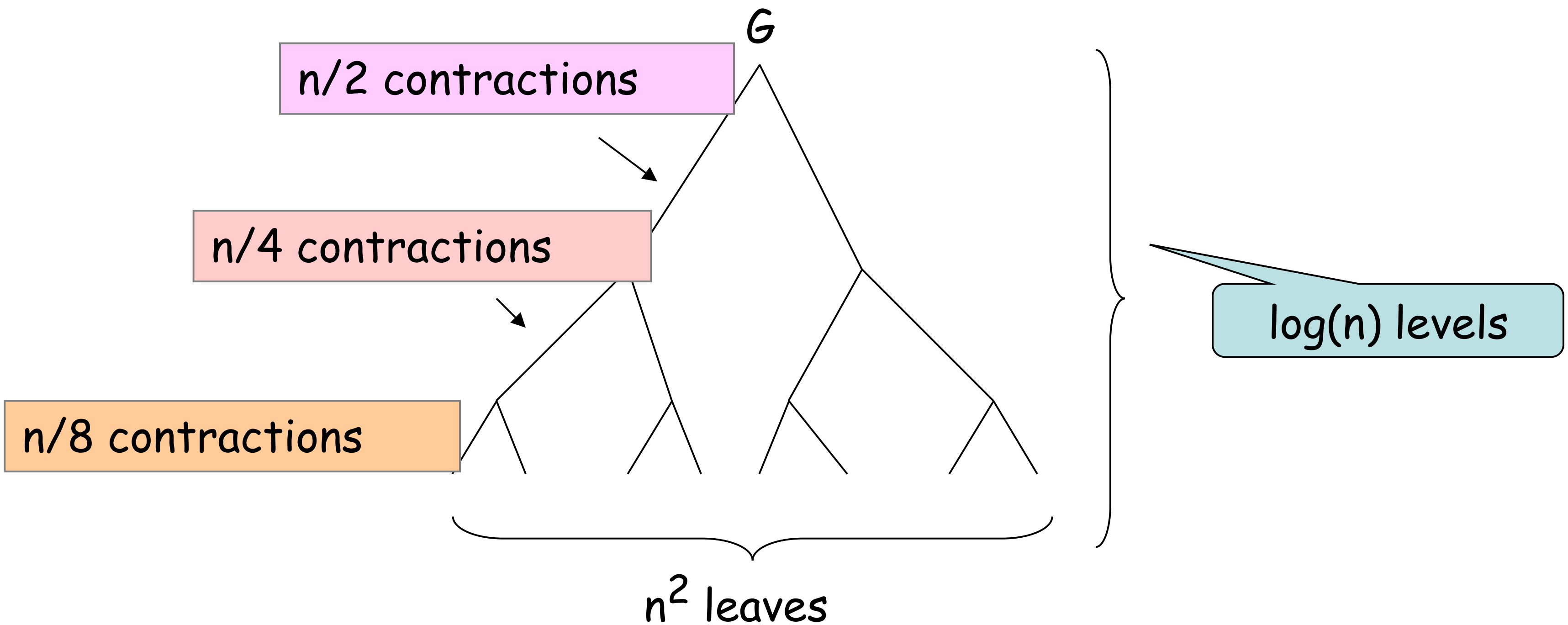
Let $T(n)$ be the running time of **Fastcut** when given an n-vertex graph.

$$T(n) = 2T\left(1 + \frac{n}{\sqrt{2}}\right) + O(n^2)$$

And the solution to this recurrence is:

$$T(n) = O(n^2 \log n)$$

Complexity



First level complexity = $2 \times (n^2/2) = n^2$.

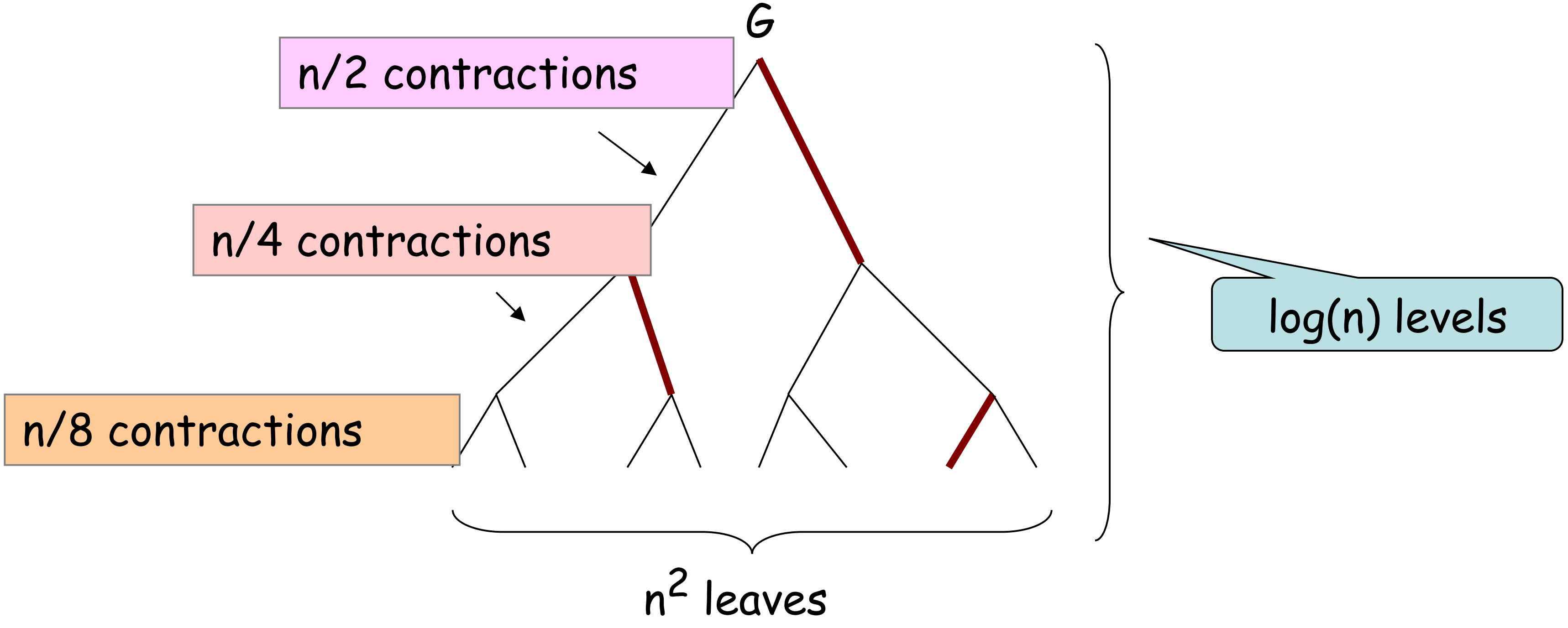
Second level complexity = $4 \times (n^2/4) = n^2$.

The i-th level complexity = $2^i \times (n^2/2^i) = n^2$.

Total time = $n^2 \log(n)$.

Analysis

Theorem 2: Algorithm **Fastcut** succeeds with probability $\Omega(1/\log(n))$.



Claim: for each "branch", the survive probability is at least $\frac{1}{2}$.

Analysis

Claim: for each "branch", the survive probability is at least $\frac{1}{2}$.

$$\mathbf{Pr}[C \text{ is survived in a branch at level k}]$$

$$= \prod_{i=n/\sqrt{2}^k}^{n/\sqrt{2}^{k+1}} \mathbf{Pr}[\text{ survive when the graph has i vertices}]$$

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

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

$$= \frac{n/\sqrt{2}^{k+1} - 2}{n/\sqrt{2}^k - 1} \cdot \frac{n/\sqrt{2}^{k+1} - 1}{n/\sqrt{2}^k} \approx \frac{1}{2}$$

Analysis

Theorem 2: Algorithm **Fastcut** succeeds with probability $\Omega(1/\log(n))$.

Proof:

Let k denote the depth of recursion,

and p(k) be a lower bound on the success probability.

$$p(k+1) \geq 1 - \left(1 - \frac{1}{2}p(k)\right)^2$$

For convenience, set it to be equality:

$$p(k+1) = p(k) - \frac{p(k)^2}{4}$$

Analysis

Theorem 2: Algorithm **Fastcut** succeeds with probability $\Omega(1/\log(n))$.

$$p(k+1) = p(k) - \frac{p(k)^2}{4}$$

We want to prove that $p(k) = \Omega(1/k)$, and then substituting $k=\log(n)$ would do.

It is more convenient to replace $p(k)$ by $q(k) = 4/p(k)-1$, and we have:

$$q(k+1) = q(k) + 1 + \frac{1}{q(k)}$$

A simple inductive argument now establishes that

$$k < q(k) < k + H_{k-1} + 4$$

where $$H_k = 1 + \frac{1}{2} + \ldots + \frac{1}{k} = \Theta(\log k)$$

So $q(k) = O(k)$

Set Cover

Set cover problem:

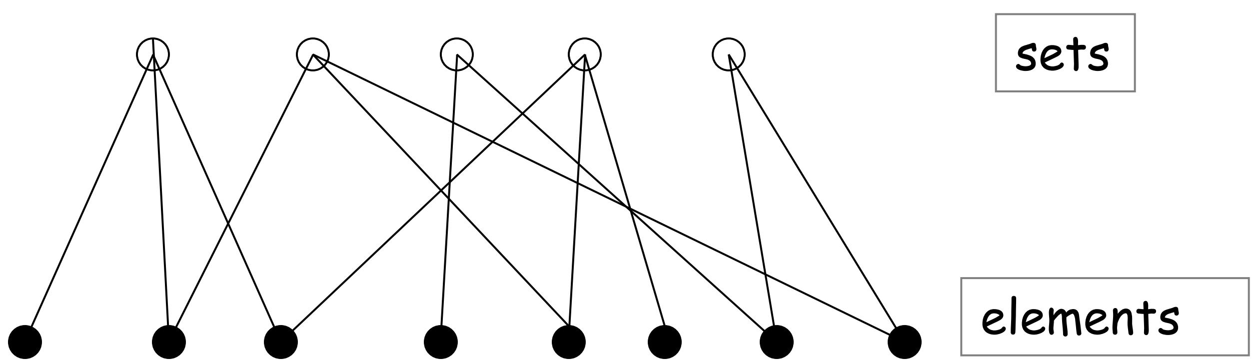
Given a ground set U of n elements, a collection of subsets of U,

S* = {S1,S2,…,Sk}, where each subset has a cost c(Si),

find a minimum cost subcollection of ***S**** that covers all elements of U.

Vertex cover is a special case, why?

A convenient interpretation:



Choose a min-cost set of white vertices to “cover” all black vertices.

Linear Programming Relaxation

$$\min \sum_{S \in S^*} c(S) x_S$$

$$\sum_{S : e \in S} x_S \geq 1 \qquad \text{for each element e.}$$

$$x_S \geq 0 \qquad \text{for each subset S.}$$

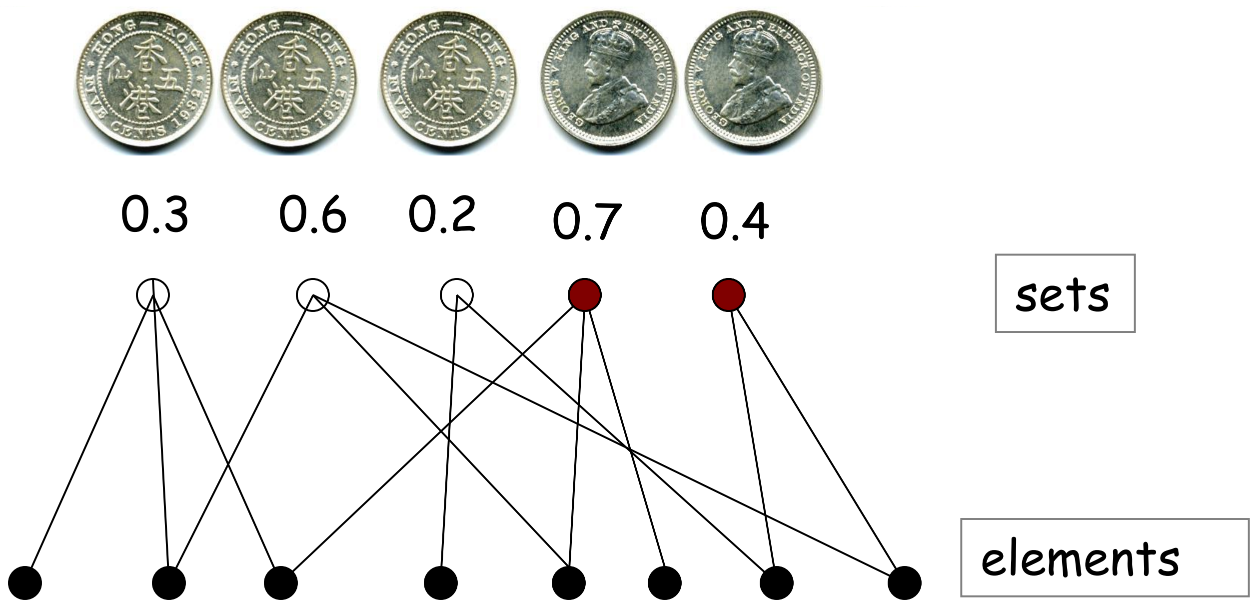
How to "round" the fractional solutions?

Idea: View the fractional values as probabilities, and do it randomly!

Algorithm

First solve the linear program to obtain the fractional values x^*.

Then flip a (biased) coin for each set with probability $x^*(S)$ being "head".



Add all the "head" vertices to the set cover.

Repeat log(n) rounds.

Performance

Theorem: The randomized rounding gives an $O(\log(n))$-approximation.

Claim 1: The sets picked in each round have an expected cost of at most LP.

Claim 2: Each element is covered with high probability after $O(\log(n))$ rounds.

So, after $O(\log(n))$ rounds, the expected total cost is at most $O(\log(n))$ LP, and every element is covered with high probability, and hence the theorem.

Remark: It is NP-hard to have a better than $O(\log(n))$-approximation!

Cost

Claim 1: The sets picked in each round have an expected cost of at most LP.

$$
\begin{aligned}
& \mathbf{E}[\text{total cost}] \\
= & \sum_{S \in S^*} \mathbf{E}[\text{cost of } S] \\
= & \sum_{S \in S^*} \mathbf{Pr}[S \text{ is picked}] \cdot c(S) \\
= & \sum_{S \in S^*} x_S \cdot c(S) \\
= & \; LP
\end{aligned}
$$

Feasibility

Claim 2: Each element is covered with high probability after O(log(n)) rounds.

First consider the probability that an element e is covered after one round.

Let say e is covered by S1, …, Sk which have values x1, …, xk.

By the linear program, $x1 + x2 + \ldots + xk \geq 1$.

Pr[e is not covered in one round] = $(1 - x1)(1 - x2)\ldots(1 - xk)$.

This is maximized when $x1=x2=\ldots=xk=1/k$, why?

Pr[e is not covered in one round] $\leq (1 - 1/k)^k$

Feasibility

Claim 2: Each element is covered with high probability after O(log(n)) rounds.

First consider the probability that an element e is covered after one round.

Pr[e is not covered in one round] <= $(1 - 1/k)^k$

So, $$\mathbf{Pr}[e \text{ is covered in one round}] \geq 1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - \frac{1}{e}$$

What about after O(log(n)) rounds?

$$\mathbf{Pr}[e \text{ is not covered}] \leq \left(\frac{1}{e}\right)^{O(\log n)} \leq \frac{1}{4n}$$

Feasibility

Claim 2: Each element is covered with high probability after O(log(n)) rounds.

$$\mathbf{Pr}[e \text{ is not covered}] \leq \left(\frac{1}{e}\right)^{O(\log n)} \leq \frac{1}{4n}$$

So,

$$\mathbf{Pr}[\text{some element is not covered}] \leq n \cdot \frac{1}{4n} \leq \frac{1}{4}$$

So,

$$\mathbf{Pr}[\text{a set cover is returned}] \geq \frac{3}{4}$$

Remark

Let say the sets picked have an expected total cost of at most clog(n) LP.

Claim: The total cost is greater than 4clog(n) LP with probability at most $\frac{1}{4}$.

This follows from the Markov inequality, which says that:

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

Proof of Markov inequality:

$$\mathbf{E}[X] = |X| \cdot Pr[X \geq t] + |X| \cdot Pr[X < t] \geq t \cdot Pr[X \geq t]$$

The claim follows by substituting E[X]=clog(n)LP and t=4clog(n)LP

Wrap Up

$$\mathbf{Pr}[\text{a set cover is returned}] \geq \frac{3}{4}$$

$$\mathbf{Pr}[\text{cost} \leq O(\log n)] \geq \frac{3}{4}$$

Theorem: The randomized rounding gives an O(log(n))-approximation.

This is the only known rounding method for set cover.

Randomized rounding has many other applications.

Why Randomness?

Probabilistic method:
Proving the existence of an object satisfying certain properties without actually constructing it.

Show that a random object satisfying those properties with positive probability!

That is, "create" the desired object by flipping coins!

Maximum Satisfiability

MAX-3-SAT: Given a Boolean formula with 3 literals in each clause, find a truth assignment which satisfies the maximum number of clauses.

$$(x_1 \vee x_2 \vee x_3) \wedge \ldots \wedge (\overline{x_1} \vee x_4 \vee \overline{x_3})$$

Theorem: there is a truth assignment which satisfies 7/8 fraction of the clauses.

Proof: find a random truth assignment –
each variable is set to TRUE or FALSE with equal probability.

The probability that a clause is satisfied is 7/8.

By linearity of expectation, we have proved the theorem.

Remark

Linearity of expectation is a simple but powerful technique.

Theorem: there is a truth assignment which satisfies 7/8 fraction of the clauses.

Corollary: there is a 7/8-approximation algorithm for MAX-3-SAT.

This randomized algorithm can be *derandomized*.

Theorem: It is NP-hard to obtain a better than 7/8-approximation algorithm!

An Aside

Consider the following random process of generating a 3-SAT instance.

There are n variables, each clause is picked uniformly randomly one by one.

There is a surprising phenomenon called **phase transition**.

If $m/n < 4.26$, then the formula is satisfiable with overwhelming probability.

If $m/n > 4.26$, then the formula is unsatisfiable with overwhelming probability.

Is it related to physics?

Why Randomness?

Probabilistic method:
Proving the existence of an object satisfying certain properties without actually constructing it.

Show that a random object satisfying those properties with positive probability!

Ramsey graph
Find a two edge-colouring of a complete graph of size n so that there is no monochromatic complete subgraph of size 2log(n)

A random 2 colouring will do.

Ramsey Number

Given a complete graph, we want to colour its edges by 2 colours.

Ramsey number R(k,l): is the smallest number n so that:
For every complete graph G with at least n vertices,
then any 2-colouring of G would either has a red complete subgraph of size k (a red K(k)) or a blue complete subgraph of size l (a blue K(l)).

There is a colouring with no red K(k) and no blue K(l).

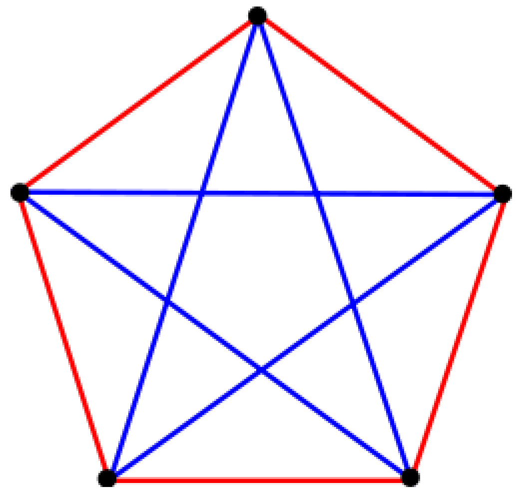
R(k,l)

Every colouring must have a red K(k) or a blue K(l).

Number of vertices

Ramsey Number R(3,3)

What is R(3,3)?



So R(3,3) > 5.

R(3,3)

There is a colouring with no red K(3) and no blue K(3).

Every colouring must have a red K(3) or a blue K(3).

Number of vertices

Ramsey Number R(3,3)

Claim: R(3,3)=6.

Proof: Suppose the edges of a complete graph on 6 vertices are coloured red and blue. Pick a vertex v. There are 5 edges incident to v and so at least 3 of them must be the same colour, say blue. If any of the edges (r, s), (r, t), (s, t) are also blue then we have an entirely blue triangle. If not, then those three edges are all red and we have with an entirely red triangle.

There is a colouring with no red K(3) and no blue K(3).

R(3,3)

Every colouring must have a red K(3) or a blue K(3).

Number of vertices

Ramsey Theorem

Ramsey Theorem: R(k,l) is finite for any k,l.

<u>Claim: $R(r, s) \le R(r-1, s) + R(r, s-1)$:</u>

Consider a complete graph on $R(r-1, s) + R(r, s-1)$ vertices.

Pick a vertex v from the graph and consider two subgraphs M and N where a vertex w is in M if and only if (v, w) is red and is in N otherwise.

Now, either $|M| \ge R(r-1, s)$ or $|N| \ge R(r, s-1)$. In the former case if M has a blue $K(s)$ then so does the original graph and we are finished. Otherwise M has a red $K(r-1)$ and so $M \cup \{v\}$ has a red $K(r)$ by definition of M. The latter case is analogous.

Ramsey Number R(k,k)

What is R(k,k)?

$$R(k,k) > \lfloor 2^{k/2} \rfloor$$

That is, for any graph with at most $\lfloor 2^{k/2} \rfloor$ vertices,
there is a 2-colouring with no red K(k) and no blue K(k).

How to find such a colouring?

Probabilistic method!

There is a colouring with
no red K(k) and no blue K(k).

R(k,k)

Every colouring must have
a red K(k) or a blue K(k).

Number of vertices

Ramsey Number R(k,k)

Consider a graph G on n vertices.
Colour each edge independently with either red or blue.

For any fixed set R of k vertices, let A_R be the event that the induced subgraph of K_n on R is *monochromatic*.

$$\mathbf{Pr}(A_R) = 2^{1-\binom{k}{2}}$$

Since there are at most $\binom{n}{k}$ possible choices for R, the probability that at least one of the events occurs is at most

$$\binom{n}{k}2^{1-\binom{k}{2}}$$

Ramsey Number R(k,k)

suppose $$\binom{n}{k} 2^{1-\binom{k}{2}} < 1$$

Then with positive probability this is a good colouring.

take $$n = \lfloor 2^{k/2} \rfloor$$

$$\binom{n}{k} 2^{1-\binom{k}{2}} < \frac{2^{1+\frac{k}{2}}}{k!} \cdot \frac{n^k}{2^{\frac{k^2}{2}}} < 1$$

$\boxed{\text{And this implies the theorem.}}$

Ramsey Number R(k,k)

Theorem: for any graph with at most $\lfloor 2^{k/2} \rfloor$ vertices, there is a 2-colouring with no red K(k) and no blue K(k).

Corollary A complete graph of size n has a 2-colouring so that there is no monochromatic complete subgraph of size 2log(n).

However, we do not know how to construct such a colouring if we don't flip coins!

On the other hand, say n=1024, then a random colouring does not satisfy this property with probability at most

$$\frac{2^{11}}{20!}$$