Minicourse on parameterized algorithms and complexity

Part 3: Randomized techniques

Dániel Marx

November 3, 2016

Why randomized?

- A guaranteed error probability of 10^{-100} is as good as a deterministic algorithm.
  (Probability of hardware failure is larger!)
- Randomized algorithms can be more efficient and/or conceptually simpler.
- Can be the first step towards a deterministic algorithm.

Polynomial-time vs. FPT randomization

Polynomial-time randomized algorithms

- Randomized selection to pick a **typical**, **unproblematic**, **average** element/subset.
- Success probability is constant or at most polynomially small.

Randomized FPT algorithms

- Randomized selection to satisfy a **bounded number** of (unknown) constraints.
- Success probability might be exponentially small.

Randomization

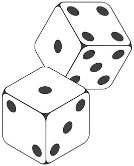
There are two main ways randomization appears:

- Algebraic techniques
  - Schwartz-Zippel Lemma
  - Linear matroids
- This lecture: combinatorial techniques.

Randomization as reduction

Problem A
(what we want to solve)

Randomized magic



Problem B
(what we can solve)

Color Coding

k-Path

Input: A graph G, integer k.
Find: A simple path of length k.

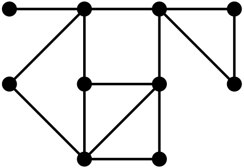
Note: The problem is clearly NP-hard, as it contains the Hamiltonian Path problem.

Theorem [Alon, Yuster, Zwick 1994]

k-Path can be solved in time $2^{O(k)} \cdot n^{O(1)}$.

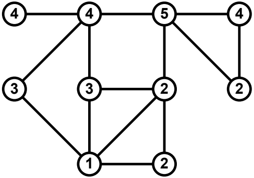
Color Coding

- Assign colors from $[k]$ to vertices $V(G)$ uniformly and independently at random.



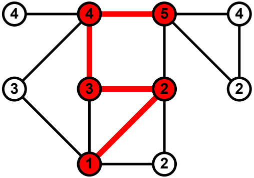
Color Coding

- Assign colors from $[k]$ to vertices $V(G)$ uniformly and independently at random.



Color Coding

- Assign colors from $[k]$ to vertices $V(G)$ uniformly and independently at random.



- Check if there is a path colored $1 - 2 - \cdots - k$; output "YES" or "NO".
  - If there is no k-path: no path colored $1 - 2 - \cdots - k$ exists $\Rightarrow$ "NO".
  - If there is a k-path: the probability that such a path is colored $1 - 2 - \cdots - k$ is k^{-k} thus the algorithm outputs "YES" with at least that probability.

Error probability

Useful fact

If the probability of success is at least p, then the probability that the algorithm **does not** say "YES" after $1/p$ repetitions is at most

$$(1-p)^{1/p} < \left(e^{-p}\right)^{1/p} = 1/e \approx 0.38$$

Error probability

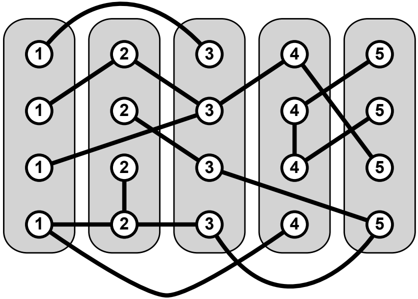
Useful fact

If the probability of success is at least p, then the probability that the algorithm **does not** say "YES" after $1/p$ repetitions is at most

$$(1-p)^{1/p} < \left(e^{-p}\right)^{1/p} = 1/e \approx 0.38$$

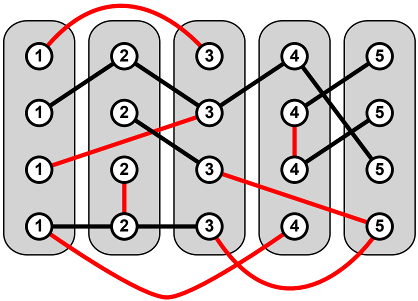
- Thus if $p > k^{-k}$, then error probability is at most $1/e$ after k^k repetitions.
- Repeating the whole algorithm a constant number of times can make the error probability an arbitrary small constant.
- For example, by trying $100 \cdot k^k$ random colorings, the probability of a wrong answer is at most $1/e^{100}$.

Finding a path colored $1 - 2 - \cdots - k$



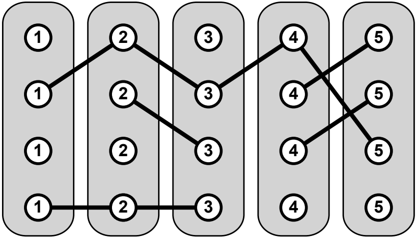
- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

Finding a path colored $1 - 2 - \cdots - k$



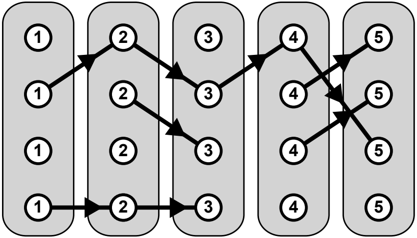
- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

Finding a path colored $1 - 2 - \cdots - k$



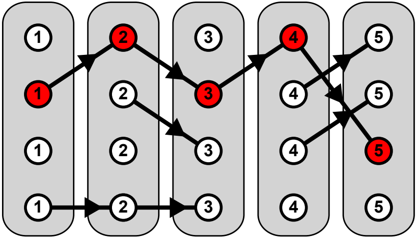
- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

Finding a path colored $1 - 2 - \cdots - k$



- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

Finding a path colored $1 - 2 - \cdots - k$

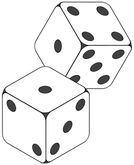


- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

Color Coding

k-PATH

Color Coding
success probability:
k^{-k}

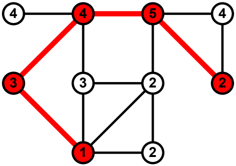


Finding a
$1 - 2 - \cdots - k$
colored path

polynomial-time
solvable

Improved Color Coding

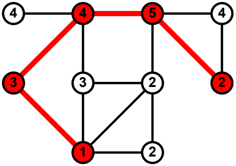
- Assign colors from $[k]$ to vertices $V(G)$ uniformly and independently at random.



- Check if there is a **colorful** path where each color appears exactly once on the vertices; output "YES" or "NO".

Improved Color Coding

- Assign colors from $[k]$ to vertices $V(G)$ uniformly and independently at random.



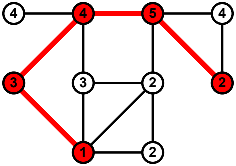
- Check if there is a **colorful** path where each color appears exactly once on the vertices; output "YES" or "NO".
  - If there is no k-path: no **colorful** path exists $\Rightarrow$ "NO".
  - If there is a k-path: the probability that it is **colorful** is

$$\frac{k!}{k^k} > \frac{\left(\frac{k}{e}\right)^k}{k^k} = e^{-k},$$

thus the algorithm outputs "YES" with at least that probability.

Improved Color Coding

- Assign colors from $[k]$ to vertices $V(G)$ uniformly and independently at random.



- Repeating the algorithm $100e^k$ times decreases the error probability to e^{-100}.

How to find a colorful path?

- Try all permutations ($k! \cdot n^{O(1)}$ time)
- Dynamic programming ($2^k \cdot n^{O(1)}$ time)

Finding a colorful path

Subproblems:

We introduce $2^k \cdot |V(G)|$ Boolean variables:

$$\boxed{\begin{array}{c} x(v, C) = \text{TRUE for some } v \in V(G) \text{ and } C \subseteq [k] \\ \Updownarrow \\ \text{There is a path } P \text{ ending at } v \text{ such that each color in} \\ C \text{ appears on } P \text{ exactly once and no other color} \\ \text{appears.} \end{array}}$$

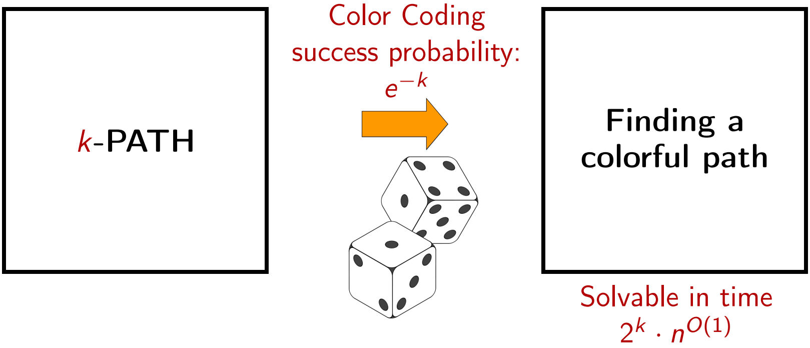
Answer:

There is a colorful path $\iff$ $x(v, [k]) = \text{TRUE}$ for some vertex v.

Initialization & Recurrence:

Exercise.

Improved Color Coding



Derandomization

Definition

A family $\mathcal{H}$ of functions $[n] \to [k]$ is a k-**perfect** family of hash functions if for every $S \subseteq [n]$ with $|S| = k$, there is an $h \in \mathcal{H}$ such that $h(x) \neq h(y)$ for any $x, y \in S$, $x \neq y$.

Theorem [Alon, Yuster, Zwick 1994]

There is a k-perfect family of functions $[n] \to [k]$ having size $2^{O(k)} \log n$ (and can be constructed in time polynomial in the size of the family).

Derandomization

Definition

A family $\mathcal{H}$ of functions $[n] \to [k]$ is a k-**perfect** family of hash functions if for every $S \subseteq [n]$ with $|S| = k$, there is an $h \in \mathcal{H}$ such that $h(x) \neq h(y)$ for any $x, y \in S$, $x \neq y$.

Theorem [Alon, Yuster, Zwick 1994]

There is a k-perfect family of functions $[n] \to [k]$ having size $2^{O(k)} \log n$ (and can be constructed in time polynomial in the size of the family).

Instead of trying $O(e^k)$ **random colorings,** we go through a k-**perfect family** $\mathcal{H}$ of functions $V(G) \to [k]$.

If there is a solution S

$\Rightarrow$ The vertices of S are colorful for at least one $h \in \mathcal{H}$

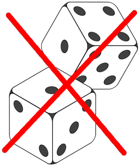
$\Rightarrow$ Algorithm outputs "YES".

$\Rightarrow$ k-Path can be solved in **deterministic** time $2^{O(k)} \cdot n^{O(1)}$.

Derandomized Color Coding

k-PATH

k-perfect family
$2^{O(k)} \log n$ functions



Finding a colorful path

Solvable in time
$2^k \cdot n^{O(1)}$

Bounded-degree graphs

Meta theorems exist for bounded-degree graphs, but randomization is usually simpler.

DENSE k-VERTEX SUBGRAPH

Input: A graph G, integers k, m.
Find: A set of k vertices inducing $\geq m$ edges.

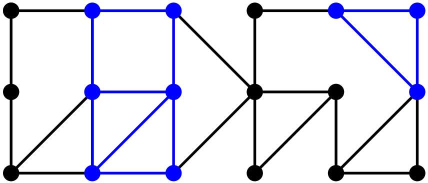
Note: on general graphs, the problem is W[1]-hard parameterized by k, as it contains k-CLIQUE.

Theorem

DENSE k-VERTEX SUBGRAPH can be solved in randomized time $2^{k(d+1)} \cdot n^{O(1)}$ on graphs with maximum degree d.

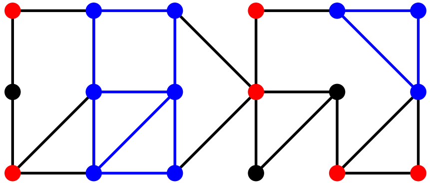
Dense k-vertex Subgraph

- Remove each vertex with probability $1/2$ independently.



Dense k-vertex Subgraph

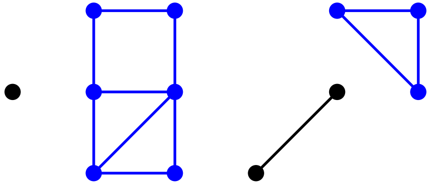
- Remove each vertex with probability $1/2$ independently.



- With probability 2^{-k} no vertex of the solution is removed.
- With probability 2^{-kd} every neighbor of the solution is removed.
- $\Rightarrow$ We have to find a solution that is the union of connected components!

Dense k-vertex Subgraph

- Remove each vertex with probability $1/2$ independently.



- With probability 2^{-k} no vertex of the solution is removed.
- With probability 2^{-kd} every neighbor of the solution is removed.
- $\Rightarrow$ We have to find a solution that is the union of connected components!

Dense k-vertex Subgraph

- Remove each vertex with probability $1/2$ independently.

k_1 vertices	k_2 vertices	k_3 vertices	$\dots$	k_i vertices
m_1 edges	m_2 edges	m_3 edges		m_i edges

Select connected components with

- at most k vertices and
- at least m edges.

What problem is this?

DENSE k-VERTEX SUBGRAPH

- Remove each vertex with probability $1/2$ independently.

k_1 vertices	k_2 vertices	k_3 vertices		k_i vertices
m_1 edges	m_2 edges	m_3 edges	$\dots$	m_i edges

Select connected components with

- at most k vertices and
- at least m edges.

What problem is this?

KNAPSACK!

Dense k-vertex Subgraph

Select connected components with

- at most k vertices and
- at least m edges.

This is exactly KNAPSACK!
(I.e., pick objects of total weight at most S and value at least V.)

We can interpret

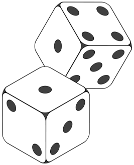
- number of vertices = weight of the items
- number of edges = value of the items

If the weights are integers, then DP solves the problem in time polynomial in the number of objects and the maximum weight.

Dense k-vertex Subgraph

DENSE k-VERTEX SUBGRAPH

Random deletions
success probability:
$2^{-k(d+1)}$



KNAPSACK

Polynomial time

Balanced Separation

Useful problem for recursion:

Balanced Separation

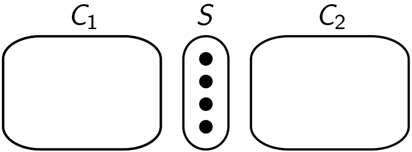
Input: A graph G, integers k, q.

Find: A set S of at most k vertices such that $G \setminus S$ has **at least two** components of size at least q each.

Theorem

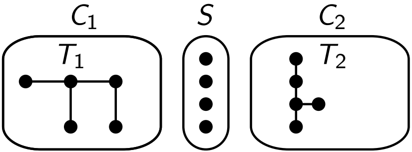
Balanced Separation can be solved in randomized time $2^{O(q+k)} \cdot n^{O(1)}$.

Balanced Separation



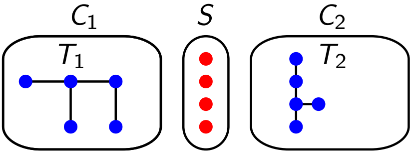
- Remove each vertex with probability $1/2$ independently.

Balanced Separation



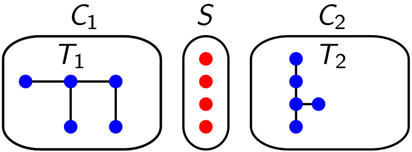
- Remove each vertex with probability $1/2$ independently.

Balanced Separation



- Remove each vertex with probability $1/2$ independently.
- With probability 2^{-k} every vertex of the solution is removed.
- With probability 2^{-q} no vertex of T_1 is removed.
- With probability 2^{-q} no vertex of T_2 is removed.

Balanced Separation

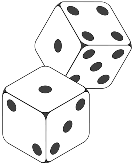


- Remove each vertex with probability $1/2$ independently.
- With probability 2^{-k} every vertex of the solution is removed.
- With probability 2^{-q} no vertex of T_1 is removed.
- With probability 2^{-q} no vertex of T_2 is removed.
- $\Rightarrow$ The reduced graph G' has two components of size $\geq q$ that can be separated in the original graph G by k vertices.
- For any pair of large components of G', we find a minimum $s - t$ cut in G.

Balanced Separation

BALANCED SEPARATION

Random deletions
success probability:
$2^{-(k+2q)}$



MINIMUM $s - t$ CUT

Polynomial time

Conclusions

- Randomization gives elegant solution to many problems.
- Derandomization is sometimes possible (but less elegant).
- Small (but $f(k)$) success probability is good for us.
- Reducing the problem we want to solve to a problem that is easier to solve.