Color-Coding

Speaker: David Wajc

Talk given as part of: Technion CS Department FPT Algorithms Seminar (236804)

Talk Outline

- Finding Paths/Cycles of Length k
    - Random Orientations
    - Random Colorings

- Derandomization

- Counting Paths of Length k

- Finding Cycles in Minor-Closed Families of Graphs

Finding Paths of Length k

Input: Directed or Undirected Graph $G = (V, E)$, integer k.

Output: A simple path $p \in G$ of length $|p| \geq k$, if one exists.

First Attempt: Using G's adjacency matrix, A_G:

$$\boldsymbol{A_{G_{i,j}}} = \begin{cases} \mathbf{1} & (\boldsymbol{i}, \boldsymbol{j}) \in \boldsymbol{E} \\ \mathbf{0} & \boldsymbol{else} \end{cases}$$

Claim: $A^k_{G_{i,j}}$ is exactly the number of paths $i \dashrightarrow j$ of length k in G.

Proof: By induction.

Algorithm: 1. Compute A^k_G .

2. Check if any entry is non-zero.

Problems with Adjacency Matrix Multiplication Approach

1. Not immediately obvious how to get the path from A_G^k.

2. The paths “counted” by ${A_G^k}_{i,j}$ are not necessarily simple!

Example:

G:

Clearly no simple paths of length greater than 1; however…

$$A_G = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$A_G^2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Finding Paths of Length k

Input: Directed or Undirected Graph $G = (V, E)$, integer k.

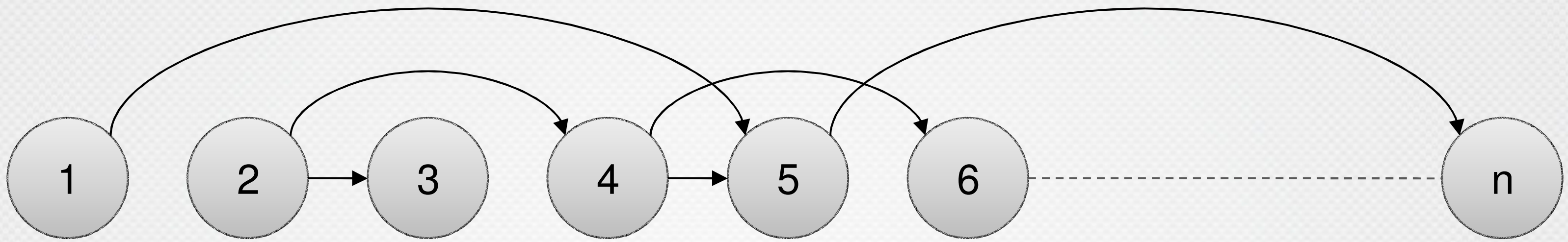
Output: A simple path $p \in G$ of length $|p| \geq k$, if one exists.

Second Attempt: Consider easy cases of the problem and use them.

Example: if G is a DAG (Directed Acyclic Graph) – linear-time Algorithm.

Finding Longest Path in a DAG

1. Topologically sort G. Now WLOG, all edges (i, j) have $i < j$.



2. For each vertex v compute $L[v]$, the length of a longest path starting at v, computing from the last to first vertex, using the formula: $L[v] = \begin{cases} 0 & d_{out}(v) = 0 \\ 1 + \max\{L[u]: (v, u) \in E\} & else \end{cases}$

3. To compute a path of length k, find a vertex with $L[v] = k$ and continue to a neighbor u with $L[u] = k - 1$, and so on.

Finding Paths of Length k

<u>As observed:</u> if G is a DAG – linear-time Algorithm.

<u>Idea:</u> Let's turn G into a DAG!

For an undirected graph, we direct edges in the following manner:

1. Choose some random permutation of the vertices, $\pi \in S_n$.
2. Direct edge $\{u, v\}$ from u to v if and only if $\pi(u) < \pi(v)$.

For directed graphs, remove all edges (u, v) with $\pi(u) > \pi(v)$.

(this just generalizes the undirected case)

Random Orientations

<u>Random Acyclic Orientation:</u>

1. Choose some random permutation of the vertices, $\pi \in S_n$.
2. Direct/leave edge uv from u to v if and only if $\pi(u) < \pi(v)$.

Denote the resulting graph by $\vec{G}$.

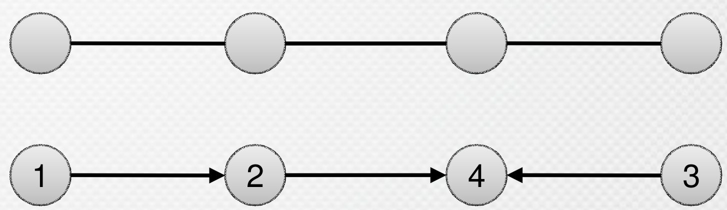
- $\vec{G}$ is a DAG.
- If there exists a path of length k in $\vec{G}$, the same path exists in G.

<u>Problem:</u>

If there exists a path of length k in G, there might be no (directed) path of length k in $\vec{G}$.

<u>Example:</u> G

$\vec{G}$



Random Orientations: Probability of Success

Lemma:
Let G be a directed graph containing a simple path of length k, denoted by p. Let $\vec{G}$ be as described above. Then $Pr\left[p \in \vec{G}\right] = \frac{1}{(k+1)!}$.

Proof: Fix $\pi_{|v \notin p}$.

- There exist $(k+1)!$ permutations that agree with $\pi_{|v \notin p}$.
- Given $\pi_{|v \notin p}$, all of these permutations have the same probability.
- Only one of these permutations leaves p in $\vec{G}$.
- Therefore $Pr\left[p \in \vec{G} \mid \pi_{|v \notin p} = \pi'\right] = \frac{1}{(k+1)!}$
- From the law of total probability $Pr\left[p \in \vec{G}\right] = \frac{1}{(k+1)!}$

Lemma 2: Similarly, for undirected G, $Pr\left[p \in \vec{G}\right] = \frac{2}{(k+1)!}$

Random Orientations: Probability of Success Alternative Proof

Lemma:
Let G be a directed graph containing a simple path of length k, denoted by p. Let $\vec{G}$ be as described above. Then $Pr[p \in \vec{G}] = \frac{1}{(k+1)!}$.

Proof: There exist $n!$ permutations. How many have $p \in \vec{G}$?

- WLOG, $p = 1 - 2 - \cdots - k - (k+1)$.
- To choose a permutation for which $p \in \vec{G}$, we have n options for $\pi(n)$, for which we have $n-1$ options for $\pi(n-1)$, …, for which we have $k+2$ options for $\pi(k+2)$, for which we have exactly *one* choice for $\pi(1), \pi(2), \dots, \pi(k+1)$.
- All in all, $\Pr[p \in \vec{G}] = \frac{n \cdot (n-1) \cdot \ldots \cdot (k+2) \cdot 1}{n!} = \frac{1}{(k+1)!}$

Lemma 2: Similarly, for undirected G, $Pr[p \in \vec{G}] = \frac{2}{(k+1)!}$

Random Algorithms

Random Algorithms come in two main flavors:

1. Las Vegas Algorithms: The algorithm always outputs correct solution, but the running time is a random variable.
   Example: QuickSort.

2. Monte Carlo Algorithms: The algorithm's running time is bounded, but it has a probability of error.
   Examples:
   The algorithm we are devising.
   Many Primality Testing Algorithms, etc'…

Monte-Carlo Algorithms Amplification

A Monte-Carlo algorithm which is always correct when it outputs "true", as in our case, is said to be *true-biased*. This is a particular case of algorithms with *one-sided errors*.

In such a case, if the algorithm answers "false", there is a chance of at most $(1 - 1/t)$ that the answer is incorrect, for some t.

Repeating the algorithm t times (independently) and answering "true" if one of the runs output "true" guarantees a probability of a false negative at most

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

In fact, repeating the algorithm $100 \cdot t$ times guarantees a probability of an incorrect answer is less than $1/2^{100}$.

Random Orientation: An Algorithm

<u>Algorithm:</u>

1. Repeat $(k+1)!$ times:
    1. Choose a random acyclic orientation of G, $\vec{G}$.
    2. Compute the longest path in $\vec{G}$, p.
    3. If $|p| \geq k$, output it and terminate.

<u>Running Time:</u>

$(k+1)!$ iterations of $O(E)$-time algorithm. Total: $O((k+1)! \cdot E)$.

<u>Correctness:</u> From all the above discussion, our algorithm has a one-sided error, with a probability of a false negative at most

$$1 - \frac{1}{(k+1)!}$$

Thus, repeating the algorithm $(k+1)!$ times we have a probability of at most $1/e$ of getting an incorrect answer.

Talk So Far

- Finding Paths/Cycles of Length k
    - Random Orientations
    - Random Colorings

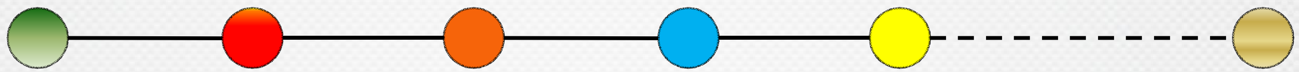
- Derandomization

- Counting Paths of Length k

- Finding Cycles in Minor-Closed Families of Graphs

Random Colorings

Assume G has its vertices colored with k colors, $c: V \rightarrow \{1,2, \ldots, k\}$.
<u>Definition:</u> We call a path a <u>*colorful path*</u> if each of its vertices is colored with a distinct color.



Note that a colorful path is also a simple path.

<u>Question:</u> What is the probability of a path p of length $k-1$ becoming colorful under a *random* coloring $c: V \rightarrow \{1,2, \ldots, k\}$?

<u>Answer:</u> Fix the colors of vertices $v \notin p$. For every such coloring, there exist k^k different colorings of the vertices $v \in p$.
$k!$ of these colorings make p colorful.
Therefore, $\Pr[p\ becomes\ colorful] = \frac{k!}{k^k} > \left(\frac{k}{e}\right)^k / k^k = e^{-k}$.

Finding Colorful Paths

Input: A graph $G = (V, E)$ and a coloring $c: V \to \{1,2, \dots, k\}$

Output: A colorful path of length $k - 1$ in G, if one exists.

Alon et al.'s solution: Dynamic Programming.

We'll give another formulation of their algorithm, which will hopefully give us more insight.

Idea: Again, build a DAG. But which one?

Finding Colorful Paths: A Reduction

Idea: keep $2^k - 1$ copies of V, each “recalling” what colors have been observed “so far”.

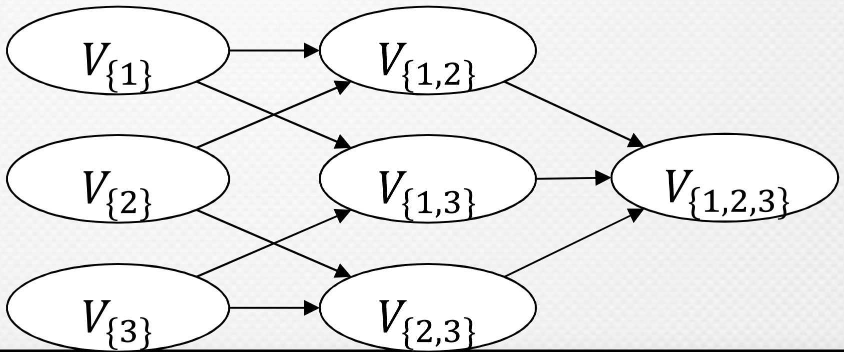
Formally: Build the following graph: $G' = (V', E')$, with

$$V' = \bigcup_{\phi \neq S \subseteq [k]} V_S$$

with $V_S = \begin{cases} \{v_S | v \in V\} & \text{if } |S| > 1 \\ \{v_S | v \in V, c(v) \in S\} & \text{if } |S| = 1 \end{cases}$

and $E' = \left\{(u_S, v_{S \cup \{c(v)\}}) \; | uv \in E, c(v) \notin S\right\}$

Example, for $k = 3$:



Finding Colorful Paths: The Reduction Graph

What is the size of the graph we built?

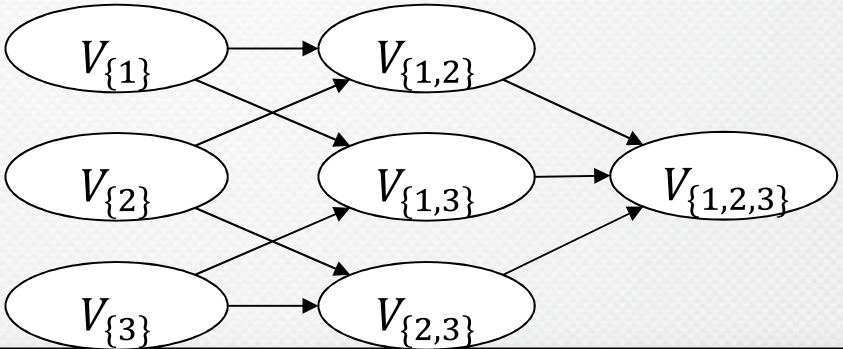
$$|V'| \leq 2^k |V|$$
$$|E'| \leq 2^k |E|$$

Claim: The graph used in our reduction is a DAG.

Proof: Every edge goes from a copy of V tagged with some $S \subseteq [k]$ to a vertex tagged with a *larger* subset of $[k]$.

Claim: we can find longest paths in this graph in $O(2^k(V+E))$ time.

Proof: corollary of the above.

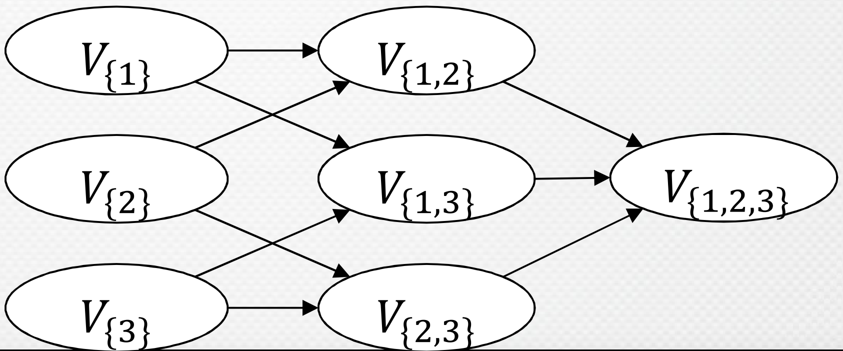


Finding Colorful Paths: A Reduction (Correctness)

Finally, we claim that all paths in G' correspond to colorful paths in G, and every colorful path in G corresponds to a path in G'.

Claim: G has a colorful path of length $k \Leftrightarrow G'$ has a path of length k.
Furthermore, given a path of length k in G', a colorful path of length k in G can be computed in linear time.
Proof: Next slide.

Corollary: We can find the longest colorful path in G in $O(2^k \cdot E)$ time.



Finding Colorful Paths: A Reduction (Correctness)

<u>Claim:</u> G has a colorful path of length k $\Leftrightarrow$ G' has a path of length k.

<u>Proof:</u> $\Rightarrow$ Let $p = v_1 v_2 \ldots v_k$ be the colorful path of length k in G.

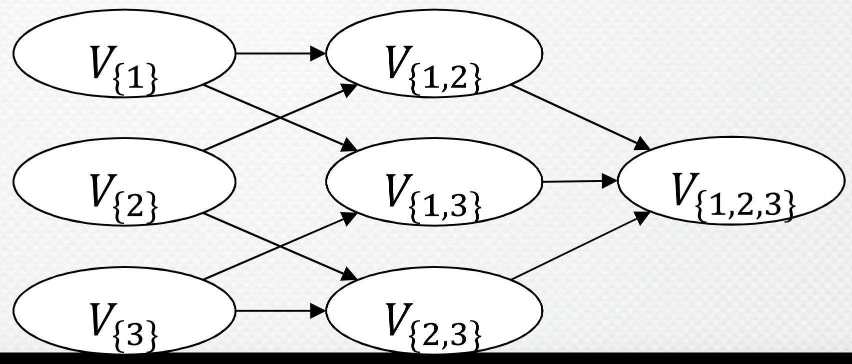
Then, if we define $S_i = \{c(v_j) : j \leq i\}$, we notice that

$$p' = v_{1_{S_1}} v_{2_{S_2}} \ldots v_{k_{S_k}} \in G'$$

$\Leftarrow$ Let $p' = v_{1_{C_1}} v_{2_{C_2}} \ldots v_{k_{C_k}}$ be a path in G'. Then:

$C_i = \{c(v_j) : j \leq i\}$ (by induction)

and the path $p = v_1 v_2 \ldots v_k$ exists in G and is colorful.



Random Coloring: An Algorithm

<u>Algorithm:</u>

1. Repeat e^k times:
   1. Choose a random coloring $c: V \to [k]$
   2. Compute the longest colorful path in G, p.
   3. If $|p| \geq k - 1$, output it and terminate.

<u>Running Time:</u>

e^k iterations of $O(2^k \cdot E)$-time algorithm. Total: $O((2e)^k \cdot E)$.

<u>Correctness:</u> As the probability of a path of length $k - 1$ becoming colorful is at least $1/e^k$, the probability of a false negative is at most

$$1 - 1/e^k$$

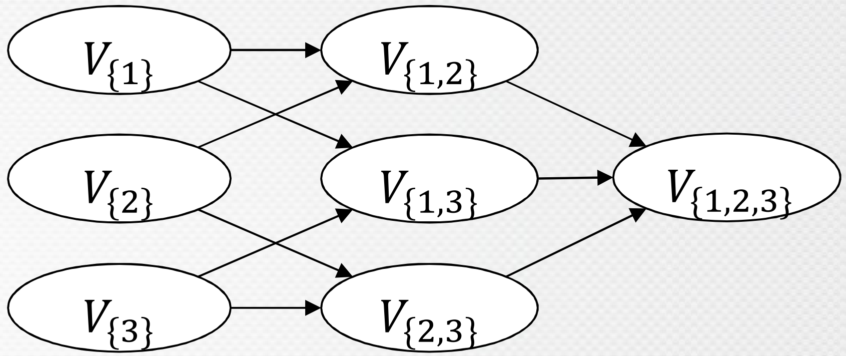
Repeating the process e^k times yields a probability of error at most $1/e$.

Finding Cycles of Length k

Input: Directed or Undirected Graph $G = (V, E)$, integer k.

Output: A simple cycle $C \in G$ of length k, if one exists.

Observation: Our reduction can be modified to allow us to find all vertices at the end of paths of length $k - 1$ starting at a specific vertex $s \in V$.



A cycle is a path of length $k - 1$ from some $s \in V$, to another vertex v such that $(v, s) \in E$.

This immediately yields a $2^{O(k)} \cdot E \cdot V$ time algorithm.

Talk So Far

- Finding Paths/Cycles of Length k
    - Random Orientations
    - Random Colorings
- Derandomization
- Counting Paths of Length k
- Finding Cycles in Minor-Closed Families of Graphs

Derandomization

All the algorithms we've shown so far have a certain (arbitrarily small) probability of giving a false positive. Can we do better?

Consider the Random Coloring Algorithm. How can we guarantee that a path p of length k is found?

First Attempt: Let's go over all possible colorings of $c: V \to [k]$.
Every path becomes colorful in at least one of these colorings.
For each coloring, search for a colorful path and return any length-k path found.

Problem: k^n possible colorings $\Rightarrow$ not $O(f(k) \cdot poly(n))$

Derandomization
Second Attempt

All we need is <u>some</u> subset of the possible colorings, F, which guarantees for every subset of vertices $S \subseteq V$ of size $|S| = k$ that all vertices of S get distinct colors for <u>some</u> coloring in F.

Such a family is called a <u>k-perfect family of hash-functions</u>.

<u>Theorem:</u> There exists a k-perfect family of hash functions from V to $[k]$ of size $2^{O(k)} \log V$, computable in $2^{O(k)} V \log V$ time.

<u>Corollary:</u> Can find path of length k in time $2^{O(k)} \cdot E \cdot \log V$, if exist.

Random Coloring: Deterministic Algorithm

Algorithm:

1. Compute a k-perfect family of hash functions, F.
2. For each coloring $c \in F$:
   1. Compute the longest colorful path in G, p.
   2. If $|p| \geq k - 1$, output it and terminate.
3. Return "no path of length $\geq k - 1$".

Running Time:

Step 1: $2^{O(k)} V \log V$ time.

Step 2: $2^{O(k)} \log V$ iterations of $O(2^k \cdot E)$-time algorithm.

Total: $O(2^{O(k)} \cdot E \cdot \log V)$ time.

Talk So Far

- Finding Paths/Cycles of Length k
  - Random Orientations
  - Random Colorings
- Derandomization
- Counting Paths of Length k
- Finding Cycles in Minor-Closed Families of Graphs

Counting Paths of Length k

Input: Directed or Undirected Graph $G = (V, E)$, integer k.

Output: The number of simple paths in G of length k.

Bad News: This problem is not only NP-Hard, but even $W[1]$-Hard, so we (probably) cannot hope to find an efficient FPT algorithm for it.

Definition: We say an algorithm A *approximates a counting problem* by a multiplicative factor $\delta > 1$ if for every input x, the algorithm's output, $A(x)$, satisfies

$$N(x)/\delta \leq A(x) \leq \delta \cdot N(x),$$

where $N(x)$ is the exact output of the counting problem for x.

Counting Colorful Paths

We will again want to color G with k colors and then try to solve our problem. Let us begin again by assuming that G is colored with k colors.

Input: Directed or Undirected Graph $G = (V, E)$, with function $c: V \to [k]$.

Output: The number of colorful paths in G of length k.

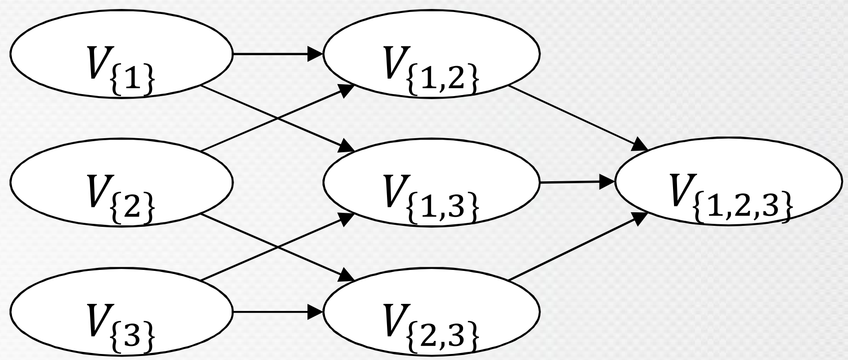
Recall our first attempt using adjacency matrix exponentiation. This failed due to the possible existence of non-simple paths of length k.

But what if G is a DAG?
Answer: In that case, all paths are simple!

Counting Colorful Paths (1)

<u>First Method:</u>

1. build the DAG G' from our previous algorithms, which had at most $2^k|V|$ vertices. Let A be its adjacency matrix.
2. Compute A^k.
3. Sum all entries of A^k.



Step 1 takes $O(2^k \cdot E)$ time.
Step 3 takes $O(4^k \cdot V^2)$ time.

Step 2 can be done naïvely in $O\left(\left(2^k \cdot V\right)^3 \cdot k\right) = O(8^k \cdot V^3 \cdot k)$ time, by using naïve matrix multiplication $k-1$ times.

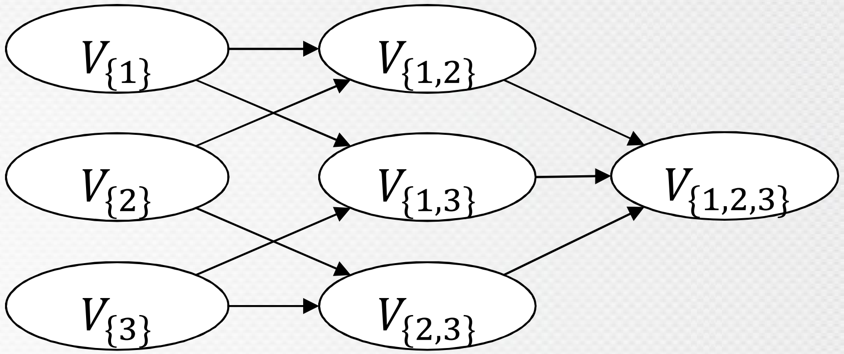
<u>Possible Improvements:</u>

1. use fast matrix multiplication ⇒ total time $O\left(\left(2^k \cdot V\right)^\omega \cdot k\right)$
2. replace the iterative multiplications by repeated squaring, thus replacing the factor of k by $\log k$.

Counting Colorful Paths

<u>Second (Faster) Method:</u>

1. Build the DAG G' from our previous algorithms.
2. Compute *number* of paths of length k using dynamic programming, similar to algorithm computing paths of length k in G'. *



Both steps take $O(2^k \cdot E)$ time.

* Every vertex has a pair of values $L[v], \#[v]$, with $L[v]$ the length of the longest path starting at v and $\#[v]$ the number of paths of length $L[v]$ starting at v.

(Approximately) Counting Paths of Length k

Definition: We say a family of functions from $[n]$ to $[k]$ is a δ-balanced (n, k)-family of hash functions if for every subset $S \subseteq [n]$ of size $|S| = k$, the number of functions that are 1-1 on S is between T/δ and $\delta \cdot T$ for some constant T.

Theorem: There exists such a family of size $O(e^{k+O(\log^3 k)} \log n)$, computable in $O(e^{k+O(\log^3 k)} n \log n)$ time.

Algorithm:
1. Compute a δ-balanced $(|V|, k)$-family, F.
2. For every coloring $c \in F$, count the number of colorful paths of length k in G, c.
3. Divide the sum of these values by T.

Correctness: Every path becomes colorful anywhere between T/δ and $\delta \cdot T$ times, so $A(x)$ holds $N(x)/\delta \leq A(x) \leq \delta \cdot N(x)$.

Talk So Far

- Finding Paths/Cycles of Length k
    - Random Orientations
    - Random Colorings
- Derandomization
- Approximately Counting Paths of Length k
- Finding Cycles in Minor-Closed Families of Graphs

Finding Cycles of Length k (Revisited)

Input: Undirected Graph $G = (V, E)$, integer k.

Output: A copy of C_k (a simple cycle of length k) in G, if one exists.

Recall that we have shown FPT algorithms for this problem on general graphs. We will therefore look for faster algorithms for this problem for graphs G from a minor-closed family of graphs.

Minors

<u>Definition:</u> We say a family of graphs F is <u>*minor-closed*</u> if for every graph $G \in F$, and for every sequence of the following operations

1. Vertex Removals
2. Edge Removals
3. Edge Contractions

the resulting graph G' holds $G' \in F$.

<u>Example of contraction:</u>

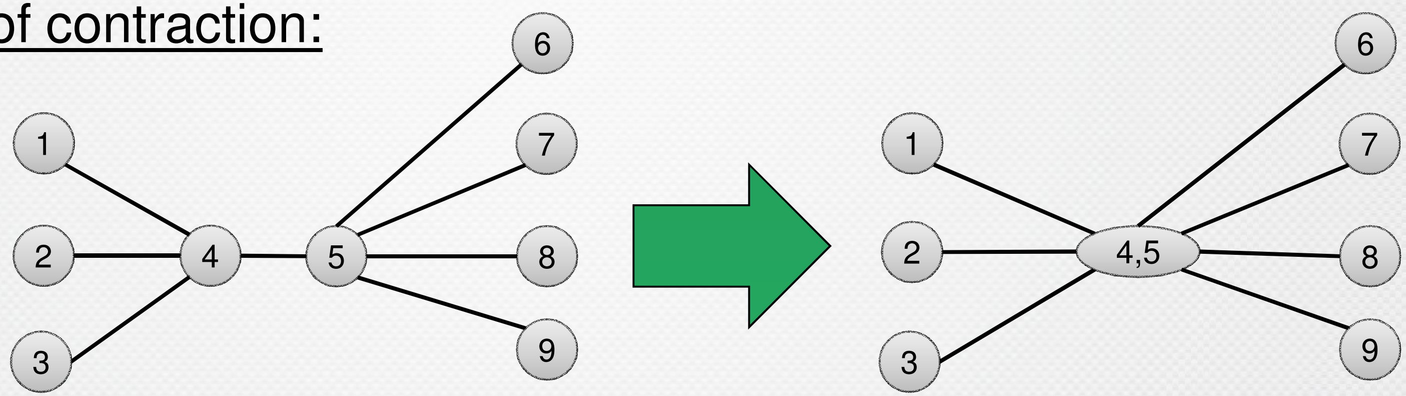
Minors

<u>Definition:</u> We say a family of graphs F is *<u>minor-closed</u>* if for every graph $G \in F$, and for every sequence of the following operations

1. Vertex Removals
2. Edge Removals
3. Edge Contractions

the resulting graph G' holds $G' \in F$.

<u>Example of contraction:</u>



Example: Planar Graphs

<u>Definition:</u> A graph which can be drawn in the plane with no edges crossing is called a *planar graph.*

Planar graphs, are a minor-closed family of graphs.

<u>Some Useful Properties of Planar Graph:</u>

1. $|E| \leq 3|V| - 6$.
2. $O(V)$-time algorithm to compute an embedding in the plane.
3. Many basic problems solvable in linear-time on planar graphs.

Planar Graph: Example

This map of Manhattan proves that Manhattan’s road network is a planar graph.

Perhaps not Planar?

Back to Cycle-Finding

Theorem: For every non-trivial minor-closed family of graphs, F, there exists some constant d_F such that every $G \in F$ has some vertex v with $d(v) \leq d_F$.

For Example: For $F = planar\ graphs$, $d_F \leq 5$. (why?)

Definition: We say a graph G is d-*degenerate* if there exists an acyclic orientation of G such that for every $v \in V$, $d_{out}(v) \leq d$.

Theorem: For every non-trivial minor-closed family of graphs, F, there exists some constant d_F such that every $G \in F$ is d_F-degenerate.

d-degenerate graphs

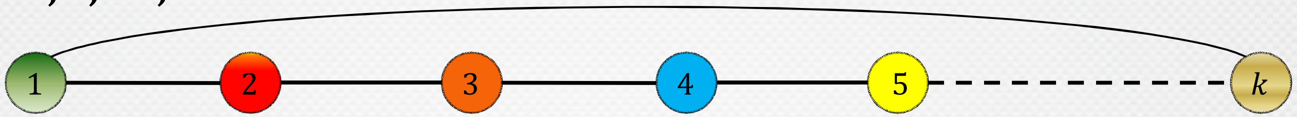
Theorem: There exists a linear-time algorithm that given a d-degenerate graph G finds an acyclic orientation of G such that $v \in V$, $d_{out}(v) \leq d$.

Proof: Very similar to algorithm for topological sorting. We will illustrate it for minor-closed families.

1. For $i = 1, \ldots, n$
   1. Let v be a vertex with $d(v) \leq d$ (guaranteed to exist)
   2. $N(v) \leftarrow i$. (v is the i-th vertex in the ordering)
   3. Remove v from G.
2. Direct all edges uv from u to v if and only if $N(u) < N(v)$.

Random Colorings and Cycles

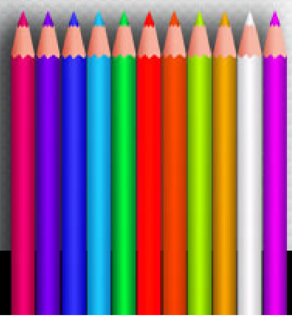
Assume G has its vertices colored with k colors, $c: V \rightarrow \{1,2,\ldots,k\}$.
We say a cycle is <u>well-colored</u> if its vertices are consecutively colored $1,2,\ldots,k$.



Note that a well-colored cycle is a simple cycle.

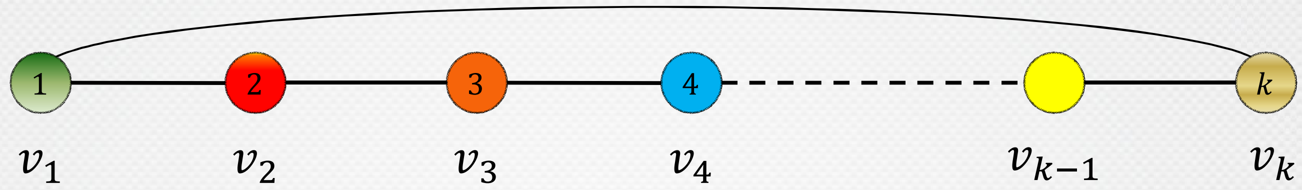
<u>Question:</u> What is the probability of a cycle C of length k becoming well-colored under a random coloring $c: V \rightarrow \{1,2,\ldots,k\}$?

<u>Answer:</u> Fix the colors of vertices $v \notin C$. For every such coloring, there exist k^k different colorings of the vertices $v \in C$, and for $2k$ of these colorings C is well-colored.
Therefore $\Pr[C \; is \; well - colored] = 2k/k^k = 2/k^{k-1}$



Finding a Well-Colored Cycle

Let $v_1 v_2 \dots v_k$ be the well-colored cycle's vertices, with $c(v_i) = i$.



As we are only concerned with well-colored cycles, we drop edges not colored by consecutive colors (modulo k). Next, we do the following:

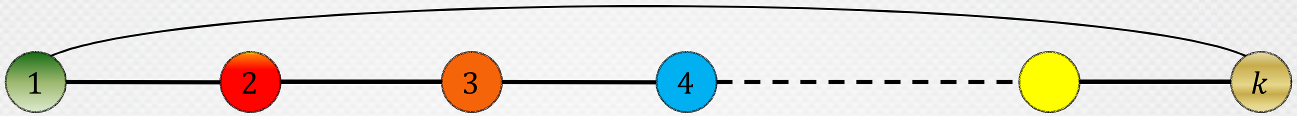
I. Compute an acyclic orientation of G such that $v \in V$, $d_{out}(v) \le d$.

II. For all $v \in V$, assign arbitrary (distinct) indices $1, 2, \dots, d$ to each edge leaving v.

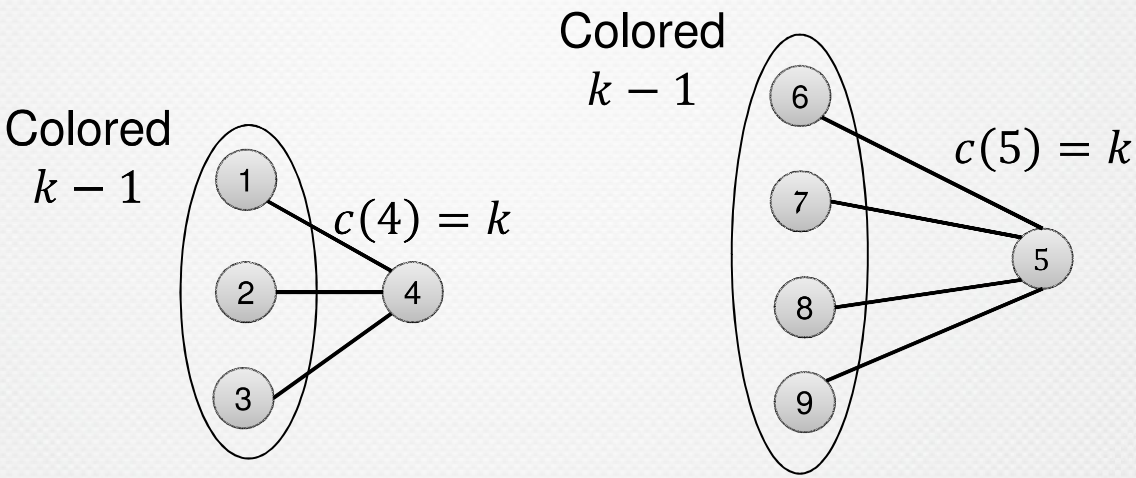
WLOG, the edge $v_{k-1} v_k$ edge was directed from v_{k-1} to v_k and has index i.

Finding a Well-Colored Cycle: Observations

The edge $v_{k-1}v_k$ was directed in step I from v_{k-1} to v_k and has index i.

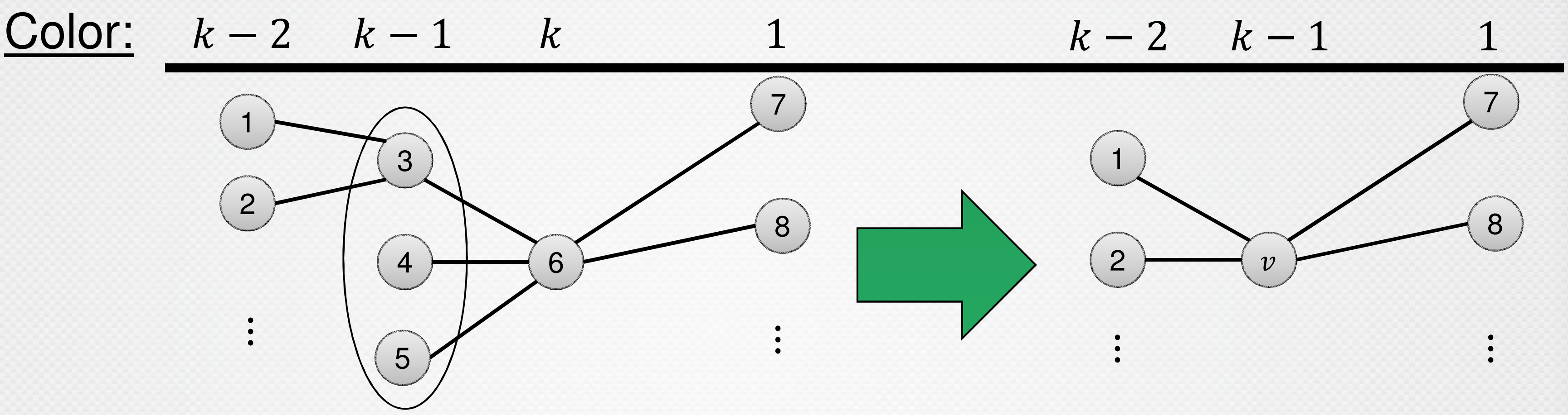


If we remove edges uv with $c(u) = c(v_{k-1}) = k - 1$ and $c(v) = c(v_k) = k$ that disagree with the orientation of $v_{k-1}v_k$ and/or its index, i, the graph of vertices colored $k - 1$ or k is made up of stars:



Finding a Well-Colored Cycle: Second Observation

What happens if we contract each star from previous observation to a single vertex with color $k-1$?



Call the resulting colored graph G'. We observe:

1. $G' \in F$.
2. G' has a well-colored C_{k-1}.
3. If G has no well-colored C_k, G' has no well-colored C_{k-1}.

Finding a Well-Colored Cycle Recursive Algorithm

I. Compute an acyclic orientation of G such that $v \in V$, $d_{out}(v) \leq d$.

II. Assign arbitrary indices to edges leaving every v: $1,2, \dots, d$.

III. Randomly guess a direction for edges uv, with $c(u) = k - 1$ and $c(v) = k$, and an index $i \in [d]$.

IV. Remove edges uv, with $c(u) = k - 1$, $c(v) = k$ which do not agree with guess.

V. Contract stars made of vertices colored $k - 1$ and k and give the new vertices color $k - 1$.

VI. Recursively search for a well-colored C_{k-1} in new colored graph.

VII. If found cycle $v_1 v_2 \dots v_{k-2} x v_1$ in G, output $v_1 v_2 \dots v_{k-2} v_{k-1} v_k v_1$, with v_{k-1} and v_k vertices that were contracted "into" x with neighbors v_{k-2} and v_1, repectively.

Finding a Well-Colored Cycle: Recursion Bottom + Running Time

Recursion Bottom:

Theorem: There exists an $O(V)$-time algorithm to find a copy of C_3 in $G \in F$ for any minor-closed family F.

Note that a C_3 is necessarily well-colored in G.

⇒ If we reach $k = 3$ we use an $O(V)$-time algorithm to find C_3 in G.

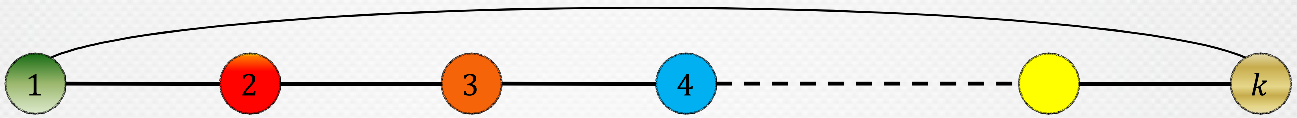
Running Time:

Every level of the recurrence we perform $O(E) = O(V)$ work. Therefore the total running time is $O(k \cdot V)$.

Finding a Well-Colored Cycle: Probability of Success

If G has no well-colored cycle, none of the graphs in our algorithm will have a well-colored cycle, and the algorithm will not output a cycle.

Assume that G has a well-colored cycle $C = v_1 v_2 \ldots v_k v_1$.

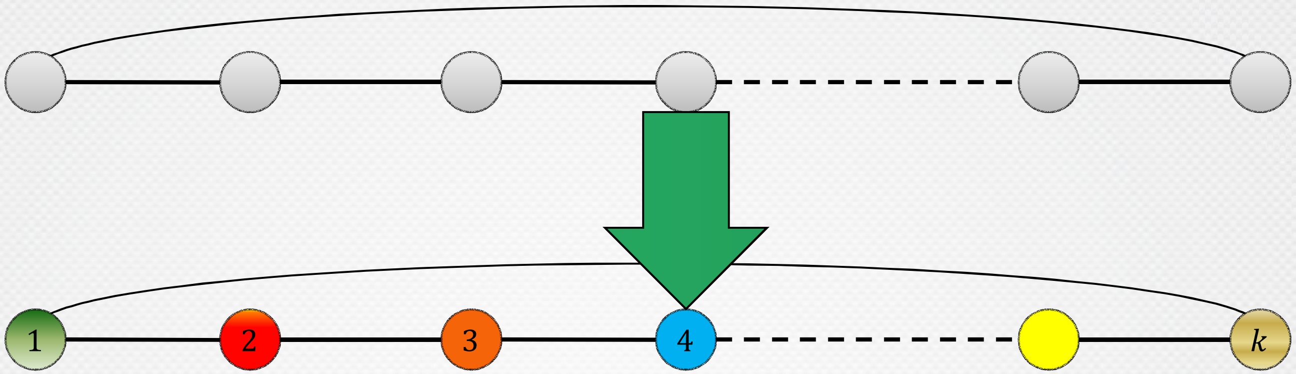


The probability that we guessed both the direction and the index of $v_{k-1}v_k$ correctly is at least $1/2d$. In such a case the colored graph in the next recursive call will have a well-colored cycle.

The probability of all recursive calls "succeeding" is at least $1/(2d)^k$.

Finding a Cycle: Probability of Success

The probability of a copy of C_k becoming well-colored is $2/k^{k-1}$.



The probability of finding a well-colored cycle is at least $1/(2d)^k$.

All in all, if G has a C_k, the probability of finding it is at least $1/(2d)^k k^{k-1}$.

Running the algorithm $(2d)^k k^{k-1}$ times give a probability of failure at most $1/e$.

Finding Cycles in Minor-Closed Families: Algorithm

Algorithm:

1. Repeat $(2d)^k k^{k-1}$ times:
    1. Choose a random coloring $c: V \to [k]$.
    2. Search for well-colored C_k in G. If found, output and halt.

Running Time:

$(2d)^k k^{k-1}$ iterations of $O(kV)$-time algorithm. Total: $O((2dk)^k \cdot V)$.

Cycles in Minor-Closed Families Derandomization

Given a coloring of G with some copy of C_k well-colored, we can derandomize the randomness due to our "guesses" of direction of edge (v_{k-1}, v_k) and index.
This increases the running time of finding a well-colored copy of C_k by a factor of $(2d)^k$, as in every one of the k levels of the recursion we consider all $2d$ options.

The randomness due to our choice of a random coloring can be replaced by exhausting a list of $k^{O(k)} \log V$ colorings for which every sequence $v_1, v_2, \dots, v_k \in V$ is consecutively colored by $1,2, \dots, k$.

All in all this yields a $(2d_F)^k k^{O(k)} V \log V$ time deterministic algorithm for finding cycles of length k in a graph $G \in F$.

Summary

- Finding Paths/Cycles of Length k
    - Random Orientations
    - Random Colorings
- Derandomization
- Approximately Counting Paths of length k
- Finding Cycles in Minor-Closed Families of Graphs

Color-Coding: A User's Guide

Given a graph G, we would like to find some induced subgraph of G isomorphic to some graph H of size $|H| = k$.

1. Randomly color the graph.
2. Show that $\Pr[\mathit{copy\ of\ H\ become\ "colorful"}] \geq 1/f(k)$
3. Devise FPT algorithm for finding "colorful" copy of H
4. Repeat algorithm of step 3 $O(f(k))$ times.
5. (Derandomize if necessary)

Some More Examples

- Finding (sub-)forests of size k
  - ❑ Can be done in $O(2^{O(k)}E)$ time with probability of error at most $1/2$.
  - ❑ $O(2^{O(k)}E \cdot \log V)$-time deterministic algorithm.

- Finding induced subgraphs of size k and treewidth t
  - ❑ Can be done in $O(2^{O(k)}V^{t+1})$ time with probability of error at most $1/2$.
  - ❑ $O(2^{O(k)}V^{t+1} \cdot \log V)$-time deterministic algorithm.

Questions?

Thank You.

Bibliography:

1) Alon, N., Yuster, R., and Zwick, U. 1994. “Color-coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs”. In Proceedings of the Twenty-Sixth Annual ACM Symposium on theory of Computing (Montreal, Quebec, Canada, May 23–25, 1994). STOC '94. ACM, New York, NY, 326–335.

2) Alon, N. and Gutner, S. 2007. “Balanced Families of perfect Hash Functions and Their Applications” In Proceedings of the Thirty-Fourth International Colloquium on Automata, Languages and Programming (Wrocław, Poland, July 9-13, 2007). ICALP 2007. EATCS.

3) Alon, N. and Gutner, S. 2009. “Balanced hashing, color coding and approximate counting”. In Journal of Parametrized and Exact Computation. Springer, 1-16.