Longest Path in Graphs: Parameterized Algorithms

Lecture I: Basics of Parameterized Algorithms, Long Path in 80's and Representative Sets

Saket Saurabh

The Institute of Mathematical Sciences, India
and University of Bergen, Norway,

RAA 2017, St. Petersburg, May 22–26, 2017

Problems we would be interested in...

Vertex Cover

Input: A graph $G = (V, E)$ and a positive integer k.

Parameter: k

Question: Does there exist a subset $V' \subseteq V$ of size at most k such that for every edge $(u, v) \in E$ either $u \in V'$ or $v \in V'$?

Hamiltonian Path

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

Question: Does there exist a path P in G that spans all the vertices?

Longest Path

Input: A graph $G = (V, E)$ and a positive integer k.

Parameter: k

Question: Does there exist a path P in G of length at least k?

Introduction and Kernelization

Fixed Parameter Tractable (FPT) Algorithms

For decision problems with input size n, and a parameter k, (which typically is the solution size), the goal here is to design an algorithm with running time $f(k) \cdot n^{\mathcal{O}(1)}$, where f is a function of k alone.

Problems that have such an algorithm are said to be fixed parameter tractable (FPT).

A Few Examples

Vertex Cover

Input: A graph $G = (V, E)$ and a positive integer k.

Parameter: k

Question: Does there exist a subset $V' \subseteq V$ of size at most k such that for every edge $(u, v) \in E$ either $u \in V'$ or $v \in V'$?

Longest Path

Input: A graph $G = (V, E)$ and a positive integer k.

Parameter: k

Question: Does there exist a path P in G of length at least k?

Kernelization: A Method for Everyone

Informally: A kernelization algorithm is a polynomial-time transformation that transforms any given parameterized instance to an equivalent instance of the same problem, with size and parameter bounded by a function of the parameter.

Kernel: Formally

Formally: A kernelization algorithm, or in short, a kernel for a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that given $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs in $p(|x| + k)$ time a pair $(x', k') \in \Sigma^* \times \mathbb{N}$ such that

Kernel: Formally

Formally: A kernelization algorithm, or in short, a kernel for a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that given $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs in $p(|x| + k)$ time a pair $(x', k') \in \Sigma^* \times \mathbb{N}$ such that

- $(x, k) \in L \iff (x', k') \in L$,
- $|x'|, k' \leqslant f(k)$,

where f is an arbitrary computable function, and p a polynomial. Any function f as above is referred to as the size of the kernel.

Kernel: Formally

Formally: A kernelization algorithm, or in short, a kernel for a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that given $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs in $p(|x| + k)$ time a pair $(x', k') \in \Sigma^* \times \mathbb{N}$ such that

- $(x, k) \in L \iff (x', k') \in L$,
- $|x'|, k' \leqslant f(k)$,

where f is an arbitrary computable function, and p a polynomial. Any function f as above is referred to as the size of the kernel.

Polynomial kernel $\implies$ f is polynomial.

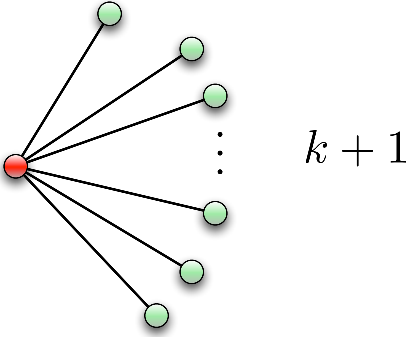
Example 1: Vertex Cover

Rule 1: Remove any isolated vertices.

Example 1: Vertex Cover

Rule 1: Remove any isolated vertices.

Rule 2: If there is a vertex v of degree at least $k + 1$ then include v in solution and $(G - \{v\}, k - 1)$



Example 1: Vertex Cover

Rule 1: Remove any isolated vertices.

Rule 2: If there is a vertex v of degree at least $k+1$ then include v in solution and $(G - \{v\}, k-1)$

Apply these rules until no longer possible.

Example 1: Vertex Cover

Rule 1: Remove any isolated vertices.

Rule 2: If there is a vertex v of degree at least $k+1$ then include v in solution and $(G - \{v\}, k-1)$

Apply these rules until no longer possible.

What conclusions can we draw ?

Example 1: Vertex Cover

Rule 1: Remove any isolated vertices.

Rule 2: If there is a vertex v of degree at least $k + 1$ then include v in solution and $(G - \{v\}, k - 1)$

Apply these rules until no longer possible.
What conclusions can we draw ?

Outcome 1: If G is not empty and k drops to 0 – the answer is No.

Example 1: Vertex Cover

Rule 1: Remove any isolated vertices.

Rule 2: If there is a vertex v of degree at least $k+1$ then include v in solution and $(G-\{v\}, k-1)$

Apply these rules until no longer possible.

What conclusions can we draw ?

Outcome 1: If G is not empty and k drops to 0 – the answer is No.

Observation: Every vertex has degree at most k – number of edges they can cover is at most k^2.

Example 1: Vertex Cover

Rule 1: Remove any isolated vertices.

Rule 2: If there is a vertex v of degree at least $k+1$ then include v in solution and $(G - \{v\}, k-1)$

Apply these rules until no longer possible.

What conclusions can we draw ?

Outcome 1: If G is not empty and k drops to 0 – the answer is No.

Observation: Every vertex has degree at most k – number of edges they can cover is at most k^2.

Outcome 2: If $|E| > k^2$ – the answer is No. Else $|E| \leqslant k^2$, $|V| \leqslant 2k^2$ and we have polynomial sized kernel of $\mathcal{O}(k^2)$.

Historical Development of Longest Path

Naive Algorithm for Longest Path

Naive Algorithm for Longest Path

$$\binom{n}{k} k!$$

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- Implies we can test whether there is a path of $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ in polynomial time.

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- Implies we can test whether there is a path of $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ in polynomial time.
- Papadimitriou and Yannakakis [Structures 1993] conjectured that testing whether there is a path of $\mathcal{O}(\log n)$ is in polynomial time.

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- Implies we can test whether there is a path of $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ in polynomial time.
- Papadimitriou and Yannakakis [Structures 1993] conjectured that testing whether there is a path of $\mathcal{O}(\log n)$ is in polynomial time.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm.

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- Implies we can test whether there is a path of $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ in polynomial time.
- Papadimitriou and Yannakakis [Structures 1993] conjectured that testing whether there is a path of $\mathcal{O}(\log n)$ is in polynomial time.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm.

Technique Invented – COLOR-CODING

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm.
- 2007– Chen, Kneis, Lu, Molle, Richter, Rossmanith, Sze and Zhang $4^k n^c$ time algorithm.

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- 1995–Alon, Yuster and Zwick– $2^{O(k)}n^c$ time algorithm.
- 2007– Chen, Kneis, Lu, Molle, Richter, Rossmanith, Sze and Zhang $4^k n^c$ time algorithm.

Technique Invented – Divide and COLOR

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm.
- 2007– Chen, Kneis, Lu, Molle, Richter, Rossmanith, Sze and Zhang $4^k n^c$ time algorithm.

Technique Invented – Divide and COLOR
Still the fastest deterministic polynomial space algorithm.

Open Problem: Design a deterministic polynomial space algorithm for Longest-Path running in time $(4-\epsilon)^k n^c$ for some fixed $\epsilon > 0$.

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm.
- 2007– Chen, Kneis, Lu, Molle, Richter, Rossmanith, Sze and Zhang $4^k n^c$ time algorithm.
- 2008– Koutis $2.83^k n^c$ time randomized algorithm.

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm.
- 2007– Chen, Kneis, Lu, Molle, Richter, Rossmanith, Sze and Zhang $4^k n^c$ time algorithm.
- 2008– Koutis $2.83^k n^c$ time randomized algorithm.

Technique Invented – Algebraic Methods

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm
- 2007– Chen, Kneis, Lu, Molle, Richter, Rossmanith, Sze and Zhang $4^k n^c$ time algorithm.
- 2008– Koutis $2.83^k n^c$ time randomized algorithm.
- 2009–Williams $2^k n^c$ time randomized algorithm.
- 2013– Björklund, Husfeldt, Kaski and Koivisto $1.657^k n^c$ time randomized algorithm.

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm
- 2007– Chen, Kneis, Lu, Molle, Richter, Rossmanith, Sze and Zhang $4^k n^c$ time algorithm.
- 2008– Koutis $2.83^k n^c$ time randomized algorithm.
- 2009–Williams $2^k n^c$ time randomized algorithm.
- 2013– Björklund, Husfeldt, Kaski and Koivisto $1.657^k n^c$ time randomized algorithm.

Technique Invented – Narrow Sieve

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm
- 2007– Chen, Kneis, Lu, Molle, Richter, Rossmanith, Sze and Zhang $4^k n^c$ time algorithm.
- 2008– Koutis $2.83^k n^c$ time randomized algorithm.
- 2009–Williams $2^k n^c$ time randomized algorithm.
- 2013– Björklund, Husfeldt, Kaski and Koivisto $1.657^k n^c$ time randomized algorithm.

Technique Invented – Narrow Sieve
Still the fastest known algorithm (though randomized and works only for undirected graphs)

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm
- 2007– Chen, Kneis, Lu, Molle, Richter, Rossmanith, Sze and Zhang $4^k n^c$ time algorithm.
- 2008– Koutis $2.83^k n^c$ time randomized algorithm.
- 2009–Williams $2^k n^c$ time randomized algorithm.
- 2013– Björklund, Husfeldt, Kaski and Koivisto $1.657^k n^c$ time randomized algorithm.

Technique Invented – Narrow Sieve
Still the fastest known algorithm (though randomized and works only for undirected graphs)

Open Problem: Design an algorithm for Longest-Path running in time $(1.657-\epsilon)^k n^c$ for some fixed $\epsilon > 0$.

More Open Problems

Open Problem: Design an algorithm for Longest-Path running in time $(2-\epsilon)^k n^c$ for some fixed $\epsilon > 0$ on directed graphs.

More Open Problems

Open Problem: Design an algorithm for Longest-Path running in time $(2-\epsilon)^k n^c$ for some fixed $\epsilon > 0$ on directed graphs.

Open Problem: Design an algorithm for Longest-Path running in time $(2-\epsilon)^n n^c$ for some fixed $\epsilon > 0$ on directed graphs. Here, n is the number of vertices.

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm.
- 2007– Chen, Kneis, Lu, Molle, Richter, Rossmanith, Sze and Zhang $4^k n^c$ time algorithm.
- 2008– Koutis $2.83^k n^c$ time randomized algorithm.
- 2009–Williams $2^k n^c$ time randomized algorithm.
- 2013–Björklund, Husfeldt, Kaski and Koivisto $1.657^k n^c$ time randomized algorithm.
- 2014– Fomin, Lokshtanov and Saurabh $2.83^k n^c$ time deterministic algorithm.

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm.
- 2007– Chen, Kneis, Lu, Molle, Richter, Rossmanith, Sze and Zhang $4^k n^c$ time algorithm.
- 2008– Koutis $2.83^k n^c$ time randomized algorithm.
- 2009–Williams $2^k n^c$ time randomized algorithm.
- 2013–Björklund, Husfeldt, Kaski and Koivisto $1.657^k n^c$ time randomized algorithm.
- 2014– Fomin, Lokshtanov and Saurabh $2.83^k n^c$ time deterministic algorithm.

Technique Invented — Fast Computation of Representative Families

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm.
- 2007– Chen, Kneis, Lu, Molle, Richter, Rossmanith, Sze and Zhang $4^k n^c$ time algorithm.
- 2008– Koutis $2.83^k n^c$ time randomized algorithm.
- 2009–Williams $2^k n^c$ time randomized algorithm.
- 2013–Björklund, Husfeldt, Kaski and Koivisto $1.657^k n^c$ time randomized algorithm.
- 2014– Fomin, Lokshtanov and Saurabh $2.83^k n^c$ time deterministic algorithm.
- 2014 –Fomin, Lokshtanov, Panolan and Saurabh and Shachnai and Zehavi gave $2.618^k n^c$ time deterministic algorithm.
- 2014–Zehavi gave $2.597^k n^c$ time deterministic algorithm.

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm.
- 2007– Chen, Kneis, Lu, Molle, Richter, Rossmanith, Sze and Zhang $4^k n^c$ time algorithm.
- 2008– Koutis $2.83^k n^c$ time randomized algorithm.
- 2009–Williams $2^k n^c$ time randomized algorithm.
- 2013–Björklund, Husfeldt, Kaski and Koivisto $1.657^k n^c$ time randomized algorithm.
- 2014– Fomin, Lokshtanov and Saurabh $2.83^k n^c$ time deterministic algorithm.
- 2014 –Fomin, Lokshtanov, Panolan and Saurabh and Shachnai and Zehavi gave $2.618^k n^c$ time deterministic algorithm.
- 2014–Zehavi gave $2.597^k n^c$ time deterministic algorithm.

Technique Invented – Fast Computation of Representative Families combined with Color Coding

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.
- 1995–Alon, Yuster and Zwick– $2^{\mathcal{O}(k)}n^c$ time algorithm.
- 2007– Chen, Kneis, Lu, Molle, Richter, Rossmanith, Sze and Zhang $4^k n^c$ time algorithm.
- 2008– Koutis $2.83^k n^c$ time randomized algorithm.
- 2009–Williams $2^k n^c$ time randomized algorithm.
- 2013–Björklund, Husfeldt, Kaski and Koivisto $1.657^k n^c$ time randomized algorithm.
- 2014– Fomin, Lokshtanov and Saurabh $2.83^k n^c$ time deterministic algorithm.
- 2014 –Fomin, Lokshtanov, Panolan and Saurabh and Shachnai and Zehavi gave $2.618^k n^c$ time deterministic algorithm.
- 2014–Zehavi gave $2.597^k n^c$ time deterministic algorithm.

Technique Invented – Fast Computation of Representative Families combined with Color Coding

Still the fastest known deterministic algorithm (though takes exponential space)

Open Problem: Design a deterministic algorithm for Longest-Path running in time $2.45^k n^c$.

The list is not comprehensive and I have left out algorithms based on treewidth. Will speak about it if time permits.

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.

Longest-Path

- 1985–Monien – $k!nm$ time algorithm.

Technique we will see – Representative Families/Sets

REPRESENTATIVE SETS

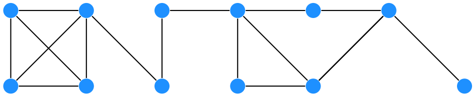
Why, What and How.

REPRESENTATIVE SETS

Why, What and How.

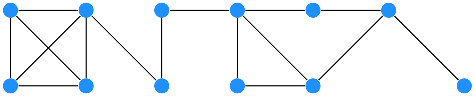
Dynamic Programming for Hamiltonian Path

Ham-Path



$1 \quad 2 \quad 3 \quad \cdots \quad i \quad \cdots \quad n-1 \quad n$

∘ Ham-Path



$1 \quad 2 \quad 3 \quad \cdots \quad i \quad \cdots \quad n-1 \quad n$

v_1

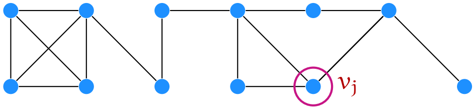
$\vdots$

v_j

$\vdots$

v_n

Ham-Path



$1 \quad 2 \quad 3 \quad \cdots \quad i \quad \cdots \quad n-1 \quad n$

v_1

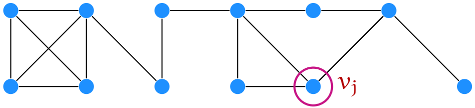
$\vdots$

v_j

$\vdots$

v_n

∘ HAM-PATH



$1 \quad 2 \quad 3 \quad \cdots \quad \boxed{i} \quad \cdots \quad n-1 \quad n$

v_1

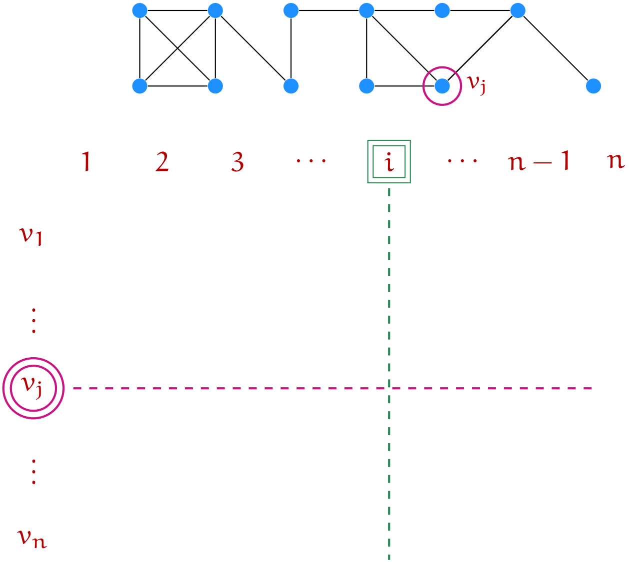
$\vdots$

v_j

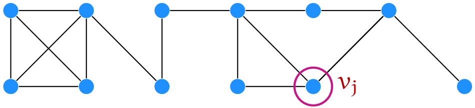
$\vdots$

v_n

Ham-Path



Ham-Path



$1 \quad 2 \quad 3 \quad \cdots \quad \boxed{i} \quad \cdots \quad n-1 \quad n$

v_1

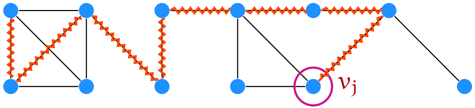
$\vdots$

v_j $\qquad V[\text{Paths of length } i \text{ ending at } v_j]$

$\vdots$

v_n

◦ HAM-PATH

Example: 

$1 \quad 2 \quad 3 \quad \cdots \quad \boxed{i} \quad \cdots \quad n-1 \quad n$

v_1

$\vdots$

v_j

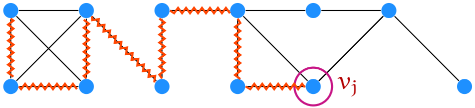
V[Paths of length i ending at v_j]

$\vdots$

v_n

∘ Ham-Path

Example:



$1 \quad 2 \quad 3 \quad \cdots \quad \boxed{i} \quad \cdots \quad n-1 \quad n$

v_1

$\vdots$

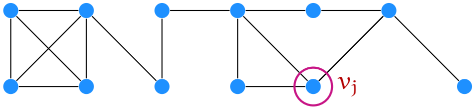
v_j

V[Paths of length i ending at v_j]

$\vdots$

v_n

∘ HAM-PATH



$1 \quad 2 \quad 3 \quad \cdots \quad \boxed{i} \quad \cdots \quad n-1 \quad n$

v_1

$\vdots$

v_j

$\vdots$

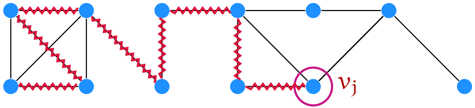
v_n

SETS, NOT SEQUENCES.

V[Paths of length i ending at v_j]

Ham-Path

Example:



$1 \quad 2 \quad 3 \quad \cdots \quad \boxed{i} \quad \cdots \quad n-1 \quad n$

v_1

$\vdots$

v_j

$\vdots$

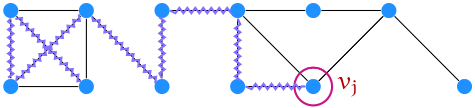
v_n

SETS, NOT SEQUENCES.

V[Paths of length i ending at v_j]

Ham-Path

Example:



$1 \quad 2 \quad 3 \quad \cdots \quad \boxed{i} \quad \cdots \quad n-1 \quad n$

v_1

$\vdots$

v_j

$\vdots$

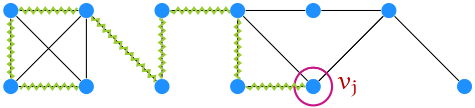
v_n

SETS, NOT SEQUENCES.

V[Paths of length i ending at v_j]

HAM-PATH

Example:



$1 \quad 2 \quad 3 \quad \cdots \quad \boxed{i} \quad \cdots \quad n-1 \quad n$

v_1

$\vdots$

v_j

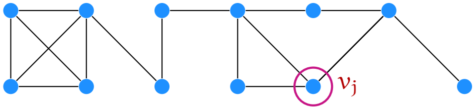
$\vdots$

v_n

SETS, NOT SEQUENCES.

V[Paths of length i ending at v_j]

○ Ham-Path



$1 \quad 2 \quad 3 \quad \cdots \quad \boxed{i} \quad \cdots \quad n-1 \quad n$

v_1

$\vdots$

v_j

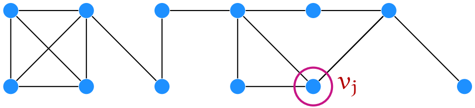
$\vdots$

v_n

SETS, NOT SEQUENCES.

V[Paths of length i ending at v_j]

Two paths that use the same set of vertices but visit them in different orders are equivalent.



$1 \quad 2 \quad 3 \quad \cdots \quad \boxed{i} \quad \cdots \quad n-1 \quad n$

v_1

$\vdots$

v_j

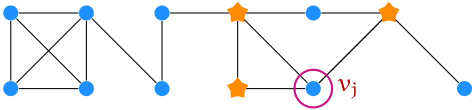
$\vdots$

v_n

V[Paths of length i ending at v_j]

= V[Paths of length $(i-1)$ ending at u, avoiding v_j.]

Ham-Path



$1 \quad 2 \quad 3 \quad \cdots \quad \boxed{i} \quad \cdots \quad n-1 \quad n$

v_1

$\vdots$

v_j

$\vdots$

v_n

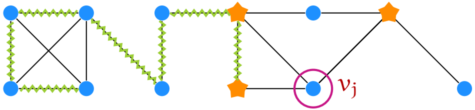
V[Paths of length i ending at v_j]

= V[Paths of length $(i-1)$ ending at u, avoiding v_j.]

$u \in N(v_j)$

∘ Ham-Path

Valid:



$1 \quad 2 \quad 3 \quad \cdots \quad \boxed{i} \quad \cdots \quad n-1 \quad n$

v_1

$\vdots$

v_j

$\vdots$

v_n

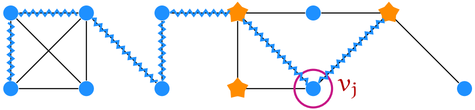
V[Paths of length i ending at v_j]

= V[Paths of length $(i-1)$ ending at u, avoiding v_j.]

$u \in N(v_j)$

∘ Ham-Path

Invalid:



$1 \quad 2 \quad 3 \quad \cdots \quad \boxed{i} \quad \cdots \quad n-1 \quad n$

v_1

$\vdots$

v_j

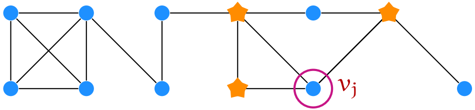
V[Paths of length i ending at v_j]

$\vdots$

= V[Paths of length $(i-1)$ ending at u, avoiding v_j.]

$u \in N(v_j)$

v_n



$1 \quad 2 \quad 3 \quad \cdots \quad \boxed{i} \quad \cdots \quad n-1 \quad n$

v_1

$\vdots$

Potentially storing $\binom{n}{i}$ sets.

v_j

V[Paths of length i ending at v_j]

$\vdots$

= V[Paths of length $(i-1)$ ending at u, avoiding v_j.]

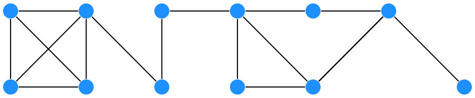
$u \in N(v_j)$

v_n

Let us now turn to k-Path.

To find paths of length at least k,
we may simply use the DP table for Hamiltonian Path
restricted to the first k columns.

K-PATH



$1 \quad 2 \quad 3 \quad \cdots \quad i \quad \cdots \quad k-1 \quad k$

v_1

$\vdots$

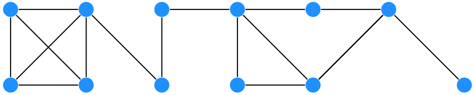
v_j

Worst case running time: $\mathcal{O}^\star\left(\binom{n}{k}\right)$

$\vdots$

v_n

K-Path



$1 \quad 2 \quad 3 \quad \cdots \quad i \quad \cdots \quad k-1 \quad k$

v_1

$\vdots$

v_j

Worst case running time: $\mathcal{O}^\star(n^k)$

$\vdots$

v_n

Do we really need to store all these sets?

Do we really need to store all these sets?

In the i^{th} column, we are storing paths of length i.

Do we really need to store all these sets?

In the i^{th} column, we are storing paths of length i.

Let P be a path of length k.

Do we really need to store all these sets?

In the i^{th} column, we are storing paths of length i.

Let P be a path of length k.

There may be several paths of length i that "latch on" to the last $(k - i)$ vertices of P.

Do we really need to store all these sets?

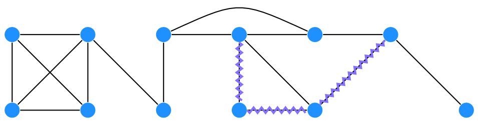
In the i^{th} column, we are storing paths of length i.

Let P be a path of length k.

There may be several paths of length i that “latch on” to the last $(k-i)$ vertices of P.

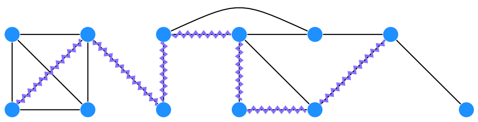
We need to store just one of them.

Example.



Example.

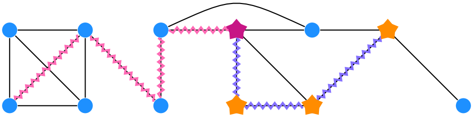
Suppose we have a path P on seven edges.

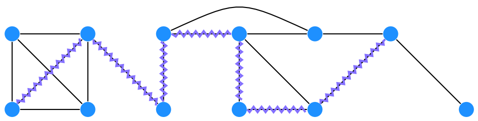


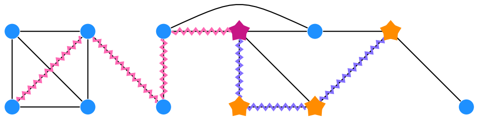
Example.

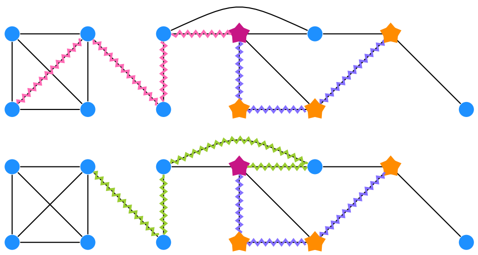
Suppose we have a path P on seven edges.

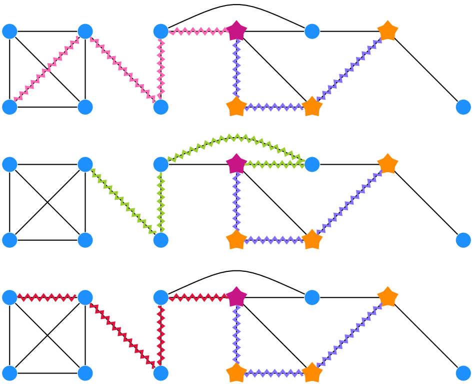
Consider it broken up into the first four and the last three edges.

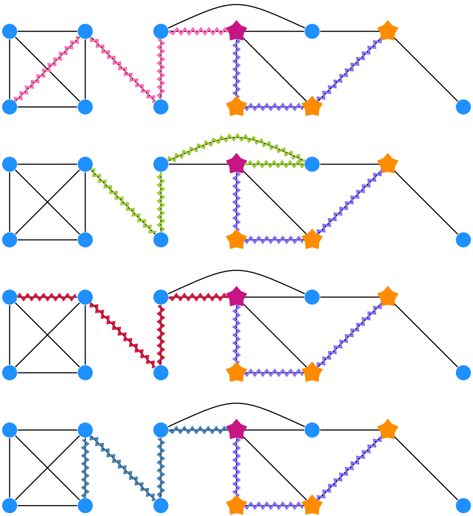


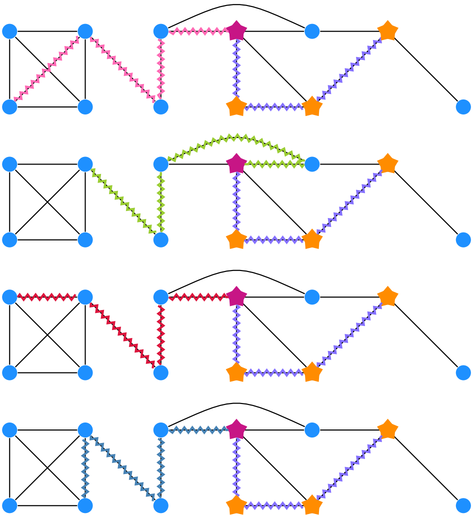






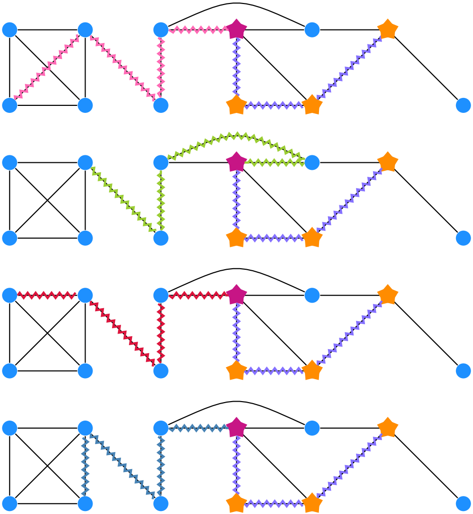






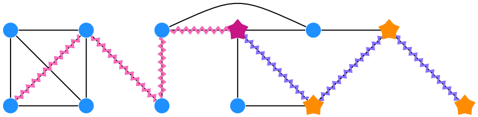
A Fixed Future ($v_{i+1} - \cdots - v_k$).

The Possibilities for Partial Solutions Compatible with $v_{i+1} - \cdots - v_k$.

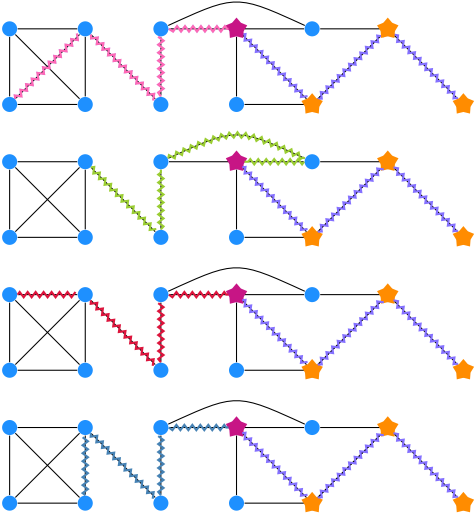


A Fixed Future ($v_{i+1} - \cdots - v_k$).

Let’s try a different example.

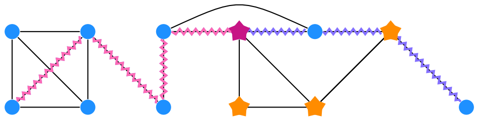


The Possibilities for Partial Solutions Compatible with $v_{i+1} - \cdots - v_k$.

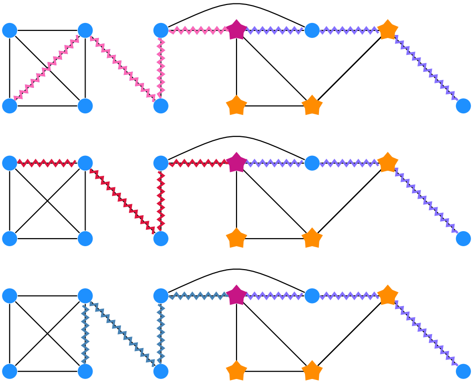


A Fixed Future ($v_{i+1} - \cdots - v_k$).

Here's one more example:



The Possibilities for Partial Solutions Compatible with $v_{i+1} - \cdots - v_k$.



A Fixed Future ($v_{i+1} - \cdots - v_k$).

For any possible ending of length $(k-i)$, we want to be sure that we store at least one among the possibly many “prefixes”.

For any possible ending of length $(k-i)$, we want to be sure that we store at least one among the possibly many "prefixes".

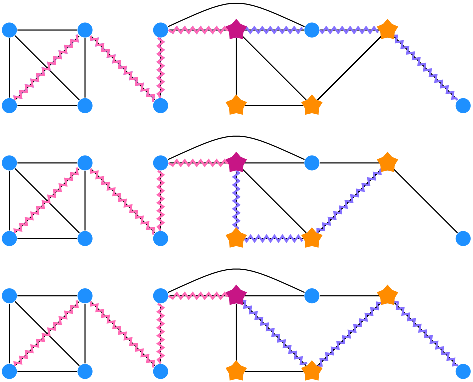
This could also be $\binom{n}{k-i}$.

For any possible ending of length $(k - i)$, we want to be sure that we store at least one among the possibly many "prefixes".

This could also be $\binom{n}{k-i}$.

The hope for "saving" comes from the fact that a single path of length i is potentially capable of being a prefix to several distinct endings.

For example...



REPRESENTATIVE SETS

Why, What and How.

Partial solutions: paths of length j ending at v_i

Partial solutions: paths of length j ending at v_i

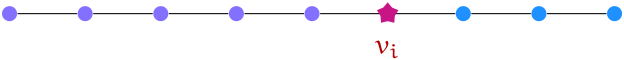
A "small" representative family.

If:

Partial solutions: paths of length j ending at v_i

A “small” representative family.

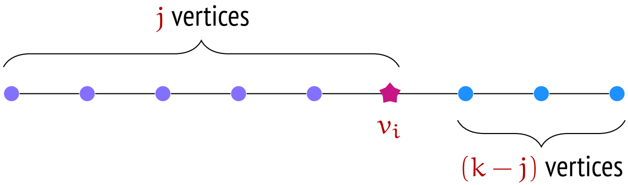
If:



Partial solutions: paths of length j ending at v_i

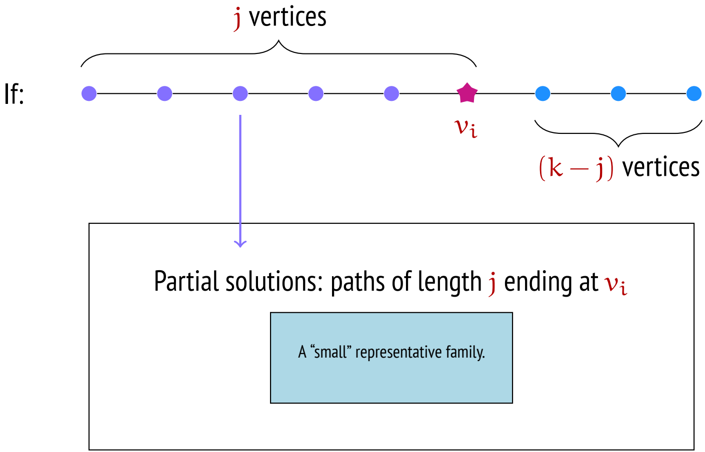
A “small” representative family.

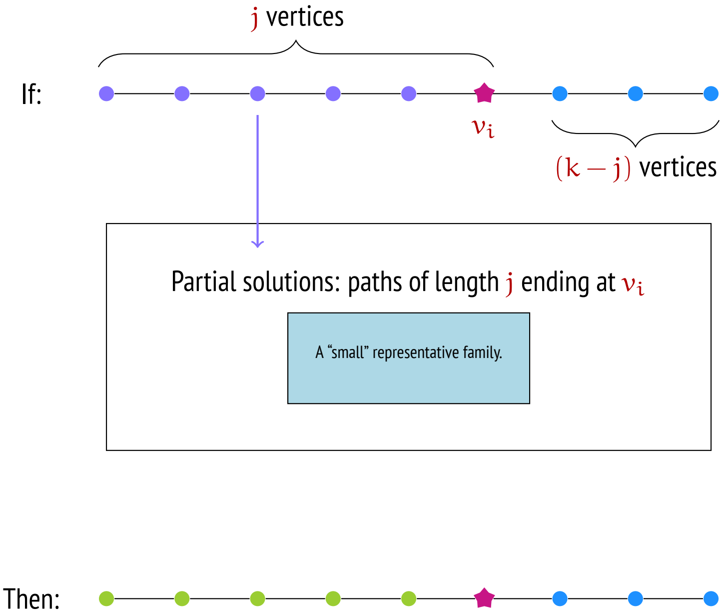
If:

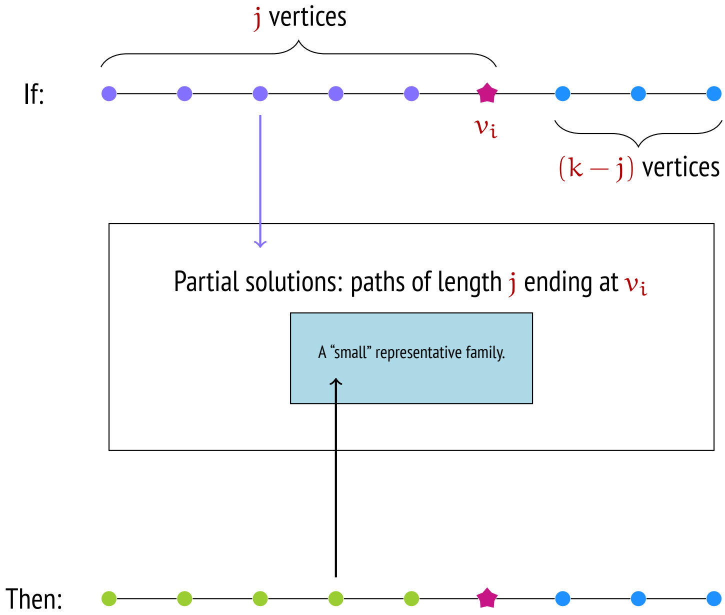


Partial solutions: paths of length j ending at v_i

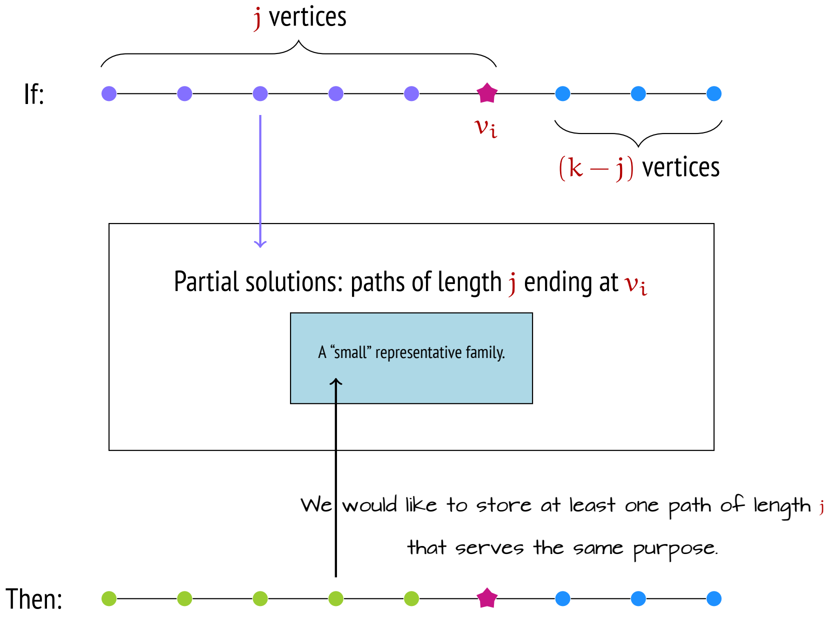
A “small” representative family.


If:
j vertices
v_i
$(k-j)$ vertices
Partial solutions: paths of length j ending at v_i
A "small" representative family.


j vertices
If:
v_i
$(k - j)$ vertices
Partial solutions: paths of length j ending at v_i
A "small" representative family.
Then:



j vertices
If:
v_i
(k − j) vertices
Partial solutions: paths of length j ending at v_i
A “small” representative family.
Then:


j vertices
If:
v_i
(k − j) vertices
Partial solutions: paths of length j ending at v_i
A "small" representative family.
We would like to store at least one path of length j
that serves the same purpose.
Then:

Given: A (BIG) family $\mathcal{F}$ of p-sized subsets of $[n]$.

$$S_1, S_2, \ldots, S_t$$

Given: A (BIG) family $\mathcal{F}$ of p-sized subsets of $[n]$.

$$S_1, S_2, \ldots, S_t$$

Want: A (small) subfamily $\widehat{\mathcal{F}}$ of $\mathcal{F}$ such that:

Given: A (BIG) family $\mathcal{F}$ of p-sized subsets of $[n]$.

$$S_1, S_2, \ldots, S_t$$

Want: A (small) subfamily $\widehat{\mathcal{F}}$ of $\mathcal{F}$ such that:

For any $X \subseteq [n]$ of size $(k - p)$,

if there is a set S in $\mathcal{F}$ such that $X \cap S = \emptyset$,
then there is a set $\widehat{S}$ in $\widehat{\mathcal{F}}$ such that $X \cap \widehat{S} = \emptyset$.

Given: A (BIG) family $\mathcal{F}$ of p-sized subsets of $[n]$.

$$S_1, S_2, \ldots, S_t$$

Want: A (small) subfamily $\widehat{\mathcal{F}}$ of $\mathcal{F}$ such that:

For any $X \subseteq [n]$ of size $(k - p)$,

if there is a set S in $\mathcal{F}$ such that $X \cap S = \emptyset$,
then there is a set $\widehat{S}$ in $\widehat{\mathcal{F}}$ such that $X \cap \widehat{S} = \emptyset$.

The "second half" of a solution – can be any subset.

Given: A (BIG) family $\mathcal{F}$ of p-sized subsets of $[n]$.

$$S_1, S_2, \ldots, S_t$$

Want: A (small) subfamily $\widehat{\mathcal{F}}$ of $\mathcal{F}$ such that:

For any $X \subseteq [n]$ of size $(k - p)$,

if there is a set S in $\mathcal{F}$ such that $X \cap S = \emptyset$,
then there is a set $\widehat{S}$ in $\widehat{\mathcal{F}}$ such that $X \cap \widehat{S} = \emptyset$.

This is a valid patch into X.

Given: A (BIG) family $\mathcal{F}$ of p-sized subsets of $[n]$.

$$S_1, S_2, \ldots, S_t$$

Want: A (small) subfamily $\widehat{\mathcal{F}}$ of $\mathcal{F}$ such that:

For any $X \subseteq [n]$ of size $(k-p)$,

if there is a set S in $\mathcal{F}$ such that $X \cap S = \emptyset$,
then there is a set $\widehat{S}$ in $\widehat{\mathcal{F}}$ such that $X \cap \widehat{S} = \emptyset$.

This is a guaranteed replacement for S.

Given: A (BIG) family $\mathcal{F}$ of p-sized subsets of $[n]$.

$$S_1, S_2, \ldots, S_t$$

Want: A (small) subfamily $\widehat{\mathcal{F}}$ of $\mathcal{F}$ such that:

For any $X \subseteq [n]$ of size $(k-p)$,

if there is a set S in $\mathcal{F}$ such that $X \cap S = \emptyset$,
then there is a set $\widehat{S}$ in $\widehat{\mathcal{F}}$ such that $X \cap \widehat{S} = \emptyset$.

Given: A $\leqslant \binom{n}{p}$ family $\mathcal{F}$ of p-sized subsets of $[n]$.

$$S_1, S_2, \ldots, S_t$$

Want: A (small) subfamily $\widehat{\mathcal{F}}$ of $\mathcal{F}$ such that:

For any $X \subseteq [n]$ of size $(k-p)$,

if there is a set S in $\mathcal{F}$ such that $X \cap S = \emptyset$,
then there is a set $\widehat{S}$ in $\widehat{\mathcal{F}}$ such that $X \cap \widehat{S} = \emptyset$.

Given: A $\leqslant \binom{n}{p}$ family $\mathcal{F}$ of p-sized subsets of $[n]$.

$$S_1, S_2, \ldots, S_t$$

Known: $\exists \binom{k}{p}$ subfamily $\widehat{\mathcal{F}}$ of $\mathcal{F}$ such that:

For any $X \subseteq [n]$ of size $(k - p)$,

if there is a set S in $\mathcal{F}$ such that $X \cap S = \emptyset$,
then there is a set $\widehat{S}$ in $\widehat{\mathcal{F}}$ such that $X \cap \widehat{S} = \emptyset$.

Bolobás, 1965.

Given: A a matroid $(M, \mathcal{I})$, and a family of p-sized subsets from $\mathcal{I}$:

$$S_1, S_2, \ldots, S_t$$

Given: A a matroid $(\mathcal{M}, \mathcal{I})$, and a family of p-sized subsets from $\mathcal{I}$:

$$S_1, S_2, \ldots, S_t$$

Want: A subfamily $\widehat{\mathcal{F}}$ of $\mathcal{F}$ such that:

Given: A a matroid $(\mathcal{M}, \mathcal{I})$, and a family of p-sized subsets from $\mathcal{I}$:

$$S_1, S_2, \ldots, S_t$$

Want: A subfamily $\widehat{\mathcal{F}}$ of $\mathcal{F}$ such that:

For any $X \subseteq [n]$ of size at most q,

if there is a set S in $\mathcal{F}$ such that $X \cap S = \emptyset$ and $X \cup S \in \mathcal{I}$,
then there is a set $\widehat{S}$ in $\widehat{\mathcal{F}}$ such that $X \cap \widehat{S} = \emptyset$ and $X \cup \widehat{S} \in \mathcal{I}$.

Given: A a matroid $(M, \mathcal{I})$, and a family of p-sized subsets from $\mathcal{I}$:

$$S_1, S_2, \ldots, S_t$$

There is a subfamily $\widehat{\mathcal{F}}$ of $\mathcal{F}$ of size at most $\binom{p+q}{p}$ such that:

For any $X \subseteq [n]$ of size at most q,

if there is a set S in $\mathcal{F}$ such that $X \cap S = \emptyset$ and $X \cup S \in \mathcal{I}$,
then there is a set $\widehat{S}$ in $\widehat{\mathcal{F}}$ such that $X \cap \widehat{S} = \emptyset$ and $X \cup \widehat{S} \in \mathcal{I}$.

Lovász, 1977

Given: A a matroid $(M, \mathcal{I})$, and a family of p-sized subsets from $\mathcal{I}$:

$$S_1, S_2, \ldots, S_t$$

There is an efficiently computable subfamily $\widehat{\mathcal{F}}$ of $\mathcal{F}$ of size at most $\binom{p+q}{p}$ such that:

For any $X \subseteq [n]$ of size at most q,

if there is a set S in $\mathcal{F}$ such that $X \cap S = \emptyset$ and $X \cup S \in \mathcal{I}$,
then there is a set $\widehat{S}$ in $\widehat{\mathcal{F}}$ such that $X \cap \widehat{S} = \emptyset$ and $X \cup \widehat{S} \in \mathcal{I}$.

Márx (2009) and Fomin, Lokshtanov, Saurabh (2013)

Summary.

We have at hand a p-uniform collection of independent sets, $\mathcal{F}$ and a number q. Let X be any set of size at most q. For any set $S \in \mathcal{F}$, if:

a X is disjoint from S, and

b X and S together form an independent set,

then a q-representative family $\widehat{\mathcal{F}}$ contains a set $\widehat{S}$ that is:

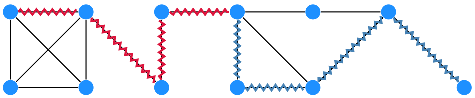
a disjoint from X, and

b forms an independent set together with X.

Such a subfamily is called a q-representative family for the given family.

REPRESENTATIVE SETS

Back to Why.



$1 \quad 2 \quad 3 \quad \cdots \quad i \quad \cdots \quad k-1 \quad k$

v_1

$\vdots$

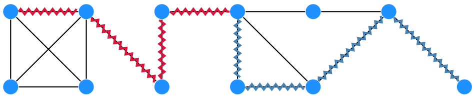
v_j

$\vdots$

v_n

[RECALL]

Worst case running time: $\mathcal{O}^\star\left(\binom{n}{k}\right)$



$1 \quad 2 \quad 3 \quad \cdots \quad i \quad \cdots \quad k-1 \quad k$

v_1

[RECALL]

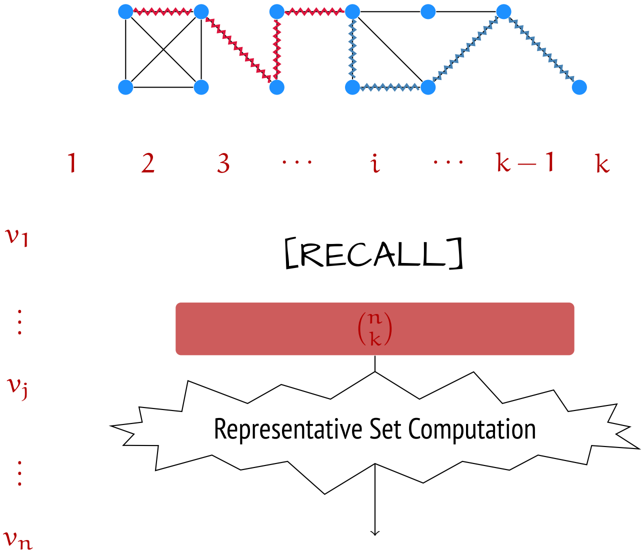
$\vdots$

$\binom{n}{k}$

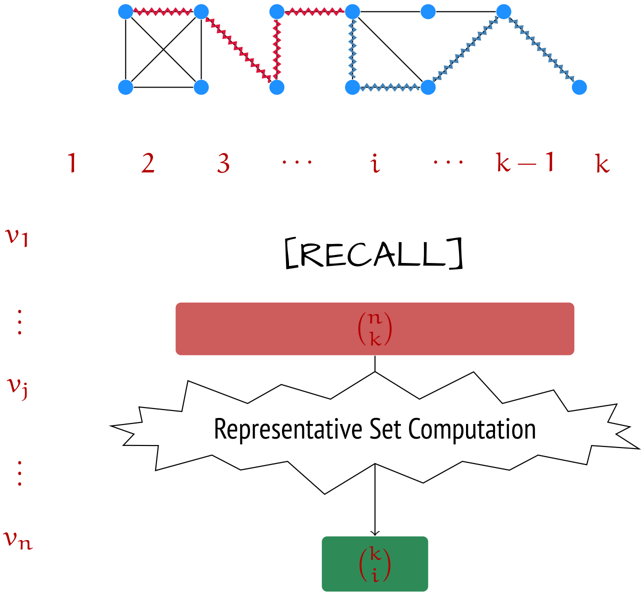
v_j

$\vdots$

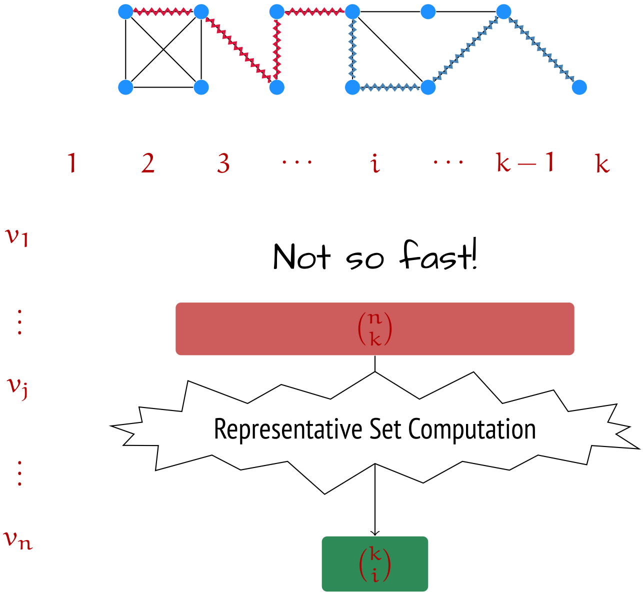
v_n



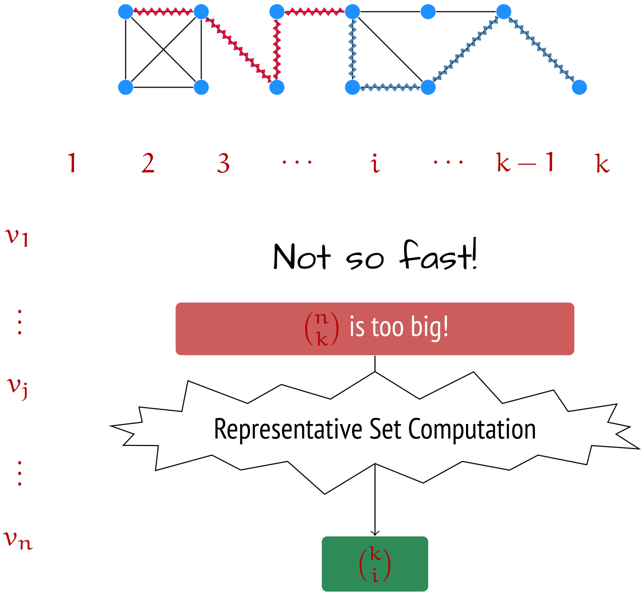
1 2 3 ⋯ i ⋯ k − 1 k
v_1
⋮
v_j
⋮
v_n
[RECALL]
$\binom{n}{k}$
Representative Set Computation



1
2
3
···
i
···
k − 1
k
v_1
[RECALL]
$\binom{n}{k}$
v_j
Representative Set Computation
v_n
$\binom{k}{i}$



1
2
3
⋯
i
⋯
k − 1
k
v_1
⋮
v_j
⋮
v_n
Not so fast!
$\binom{n}{k}$
Representative Set Computation
$\binom{k}{i}$



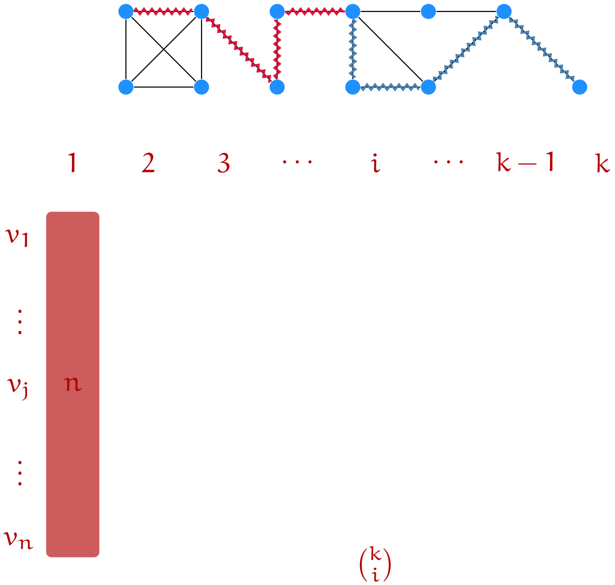
1 2 3 ⋯ i ⋯ k − 1 k
v_1
⋮
v_j
⋮
v_n
Not so fast!
$\binom{n}{k}$ is too big!
Representative Set Computation
$\binom{k}{i}$

We are going to compute representative families at every intermediate stage of the computation.

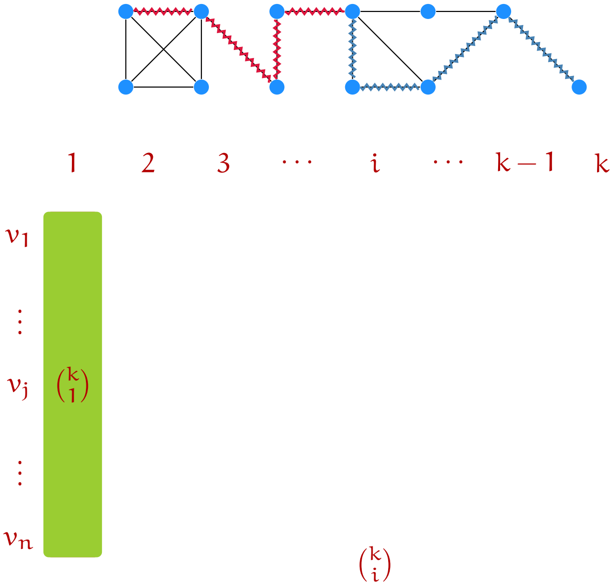
We are going to compute representative families at every intermediate stage of the computation.

For instance, in the i^{th} column, we are storing i-uniform families.
Before moving on to column $(i+1)$, we compute $(k-i)$-representative families.

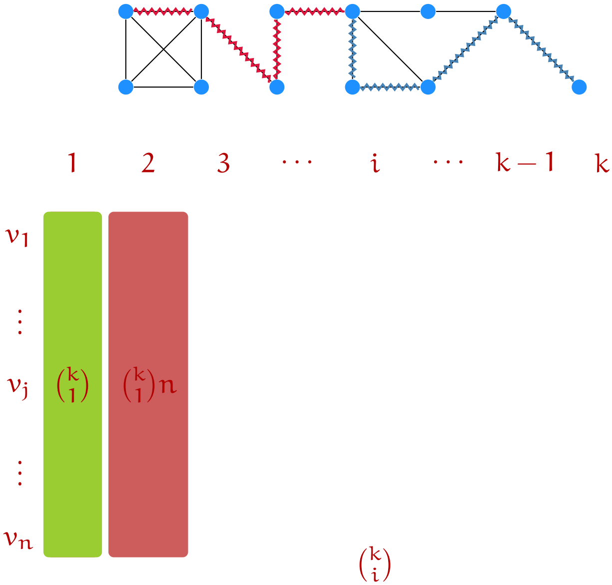
This keeps the sizes small as we go along.



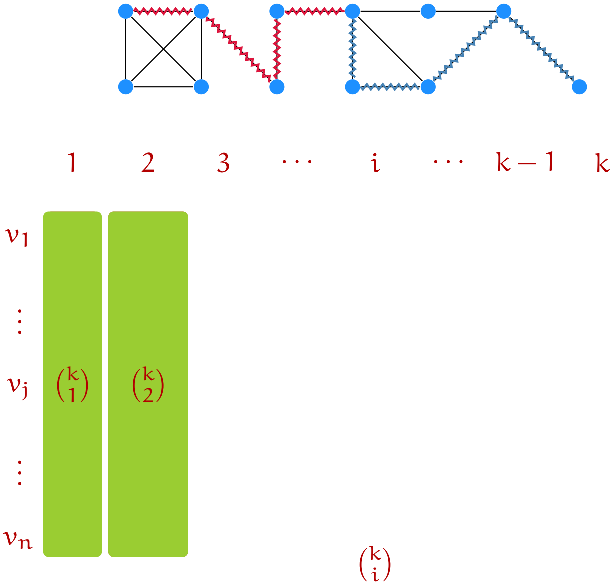
1
2
3
⋯
i
⋯
k − 1
k
v1
⋮
vj
⋮
vn
n
(k i)



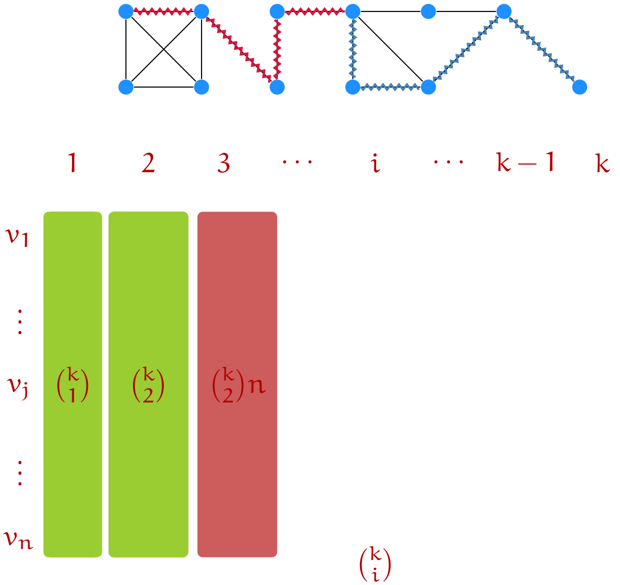
1 2 3 ⋯ i ⋯ k − 1 k
v_1
⋮
v_j
$\binom{k}{1}$
⋮
v_n
$\binom{k}{i}$



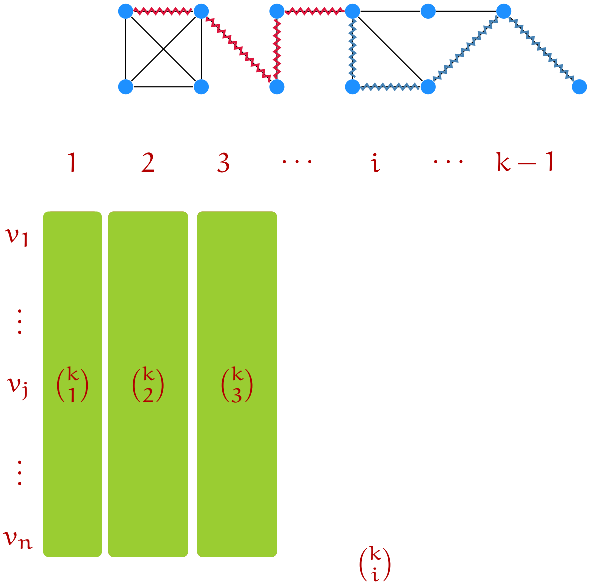
1
2
3
···
i
···
$k-1$
k
v_1
⋮
v_j
⋮
v_n
$\binom{k}{1}$
$\binom{k}{1}n$
$\binom{k}{i}$

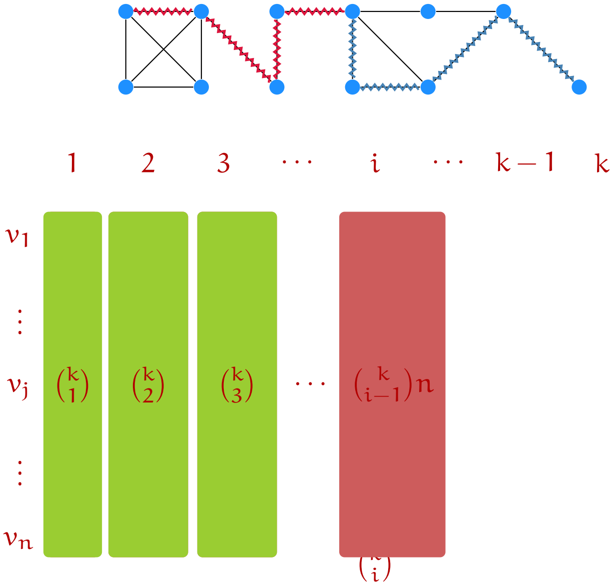


1
2
3
···
i
···
k − 1
k
v₁
⋮
v_j
(k 1)
(k 2)
⋮
v_n
(k i)

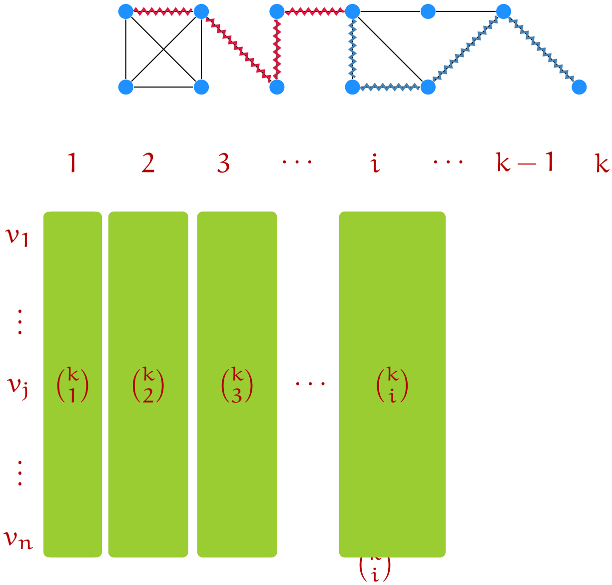


1
2
3
⋯
i
⋯
k − 1
k
v_1
⋮
v_j
⋮
v_n
$\binom{k}{1}$
$\binom{k}{2}$
$\binom{k}{2}n$
$\binom{k}{i}$

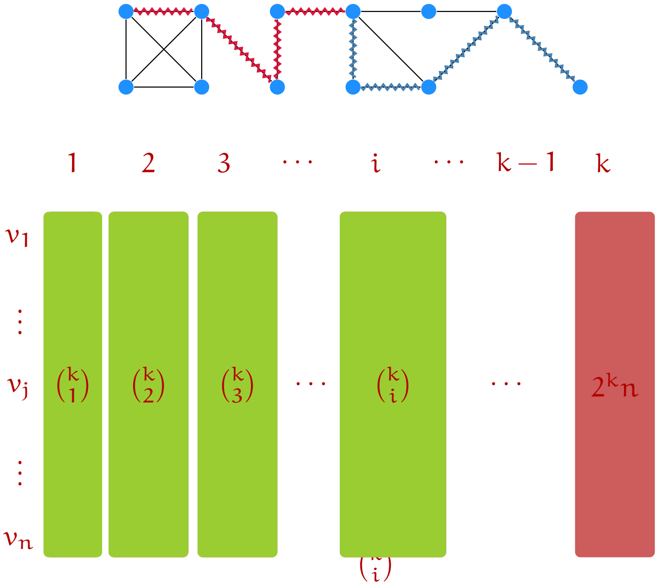

1 2 3 ⋯ i ⋯ k − 1 k
v_1
v_j
v_n
$\binom{k}{1}$
$\binom{k}{2}$
$\binom{k}{3}$
$\binom{k}{i}$



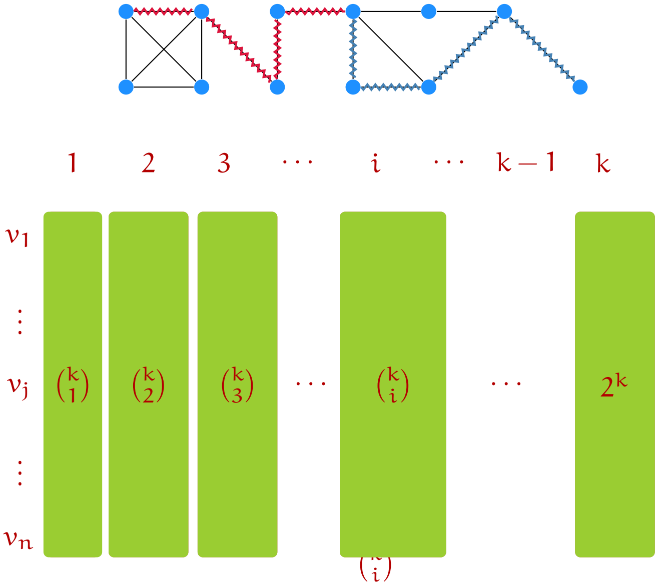
1
2
3
⋯
i
⋯
k − 1
k
v_1
⋮
v_j
⋮
v_n
$\binom{k}{1}$
$\binom{k}{2}$
$\binom{k}{3}$
⋯
$\binom{k}{i-1}n$
$\binom{k}{i}$



1
2
3
· · ·
i
· · ·
k − 1
k
v_1
⋮
v_j
⋮
v_n
$\binom{k}{1}$
$\binom{k}{2}$
$\binom{k}{3}$
· · ·
$\binom{k}{i}$
$\binom{k}{i}$



1
2
3
…
i
…
$k-1$
k
v_1
⋮
v_j
⋮
v_n
$\binom{k}{1}$
$\binom{k}{2}$
$\binom{k}{3}$
…
$\binom{k}{i}$
…
$2^k n$
$\binom{k}{i}$



1
2
3
...
i
...
k − 1
k
v_1
⋮
v_j
⋮
v_n
$\binom{k}{1}$
$\binom{k}{2}$
$\binom{k}{3}$
...
$\binom{k}{i}$
...
2^k
$\binom{k}{i}$

Let $\mathcal{P}_i^j$ be the set of all paths of length i ending at v_j.

It can be shown that the families thus computed at the i^{th} column, j^{th} row are indeed $(k-i)$-representative families for P_i^j.

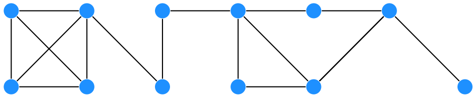
The correctness is implicit in the notion of a representative family.

REPRESENTATIVE SETS

A Different Why.

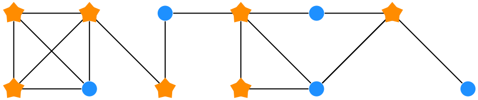
Vertex Cover

Can you delete k vertices to kill all edges?



Vertex Cover

Can you delete k vertices to kill all edges?



Let $(G = (V, E), k)$ be an instance of Vertex Cover.

Note that E can be thought of as a 2-uniform family over the ground set V.

Let $(G = (V, E), k)$ be an instance of Vertex Cover.

Note that E can be thought of as a 2-uniform family over the ground set V.

Goal: Kernelization.

In this context, we are asking if there is a small subset X of the edges such that

$G[X]$ is a YES-instance $\leftrightarrow$ G is a YES-instance.

Note: If G is a YES-instance, then $G[X]$ is a YES-instance for any subset $X \subseteq E$.

Note: If G is a YES-instance, then $G[X]$ is a YES-instance for any subset $X \subseteq E$.

We get one direction for free!

Note: If G is a YES-instance, then $G[X]$ is a YES-instance for any subset $X \subseteq E$.

We get one direction for free!

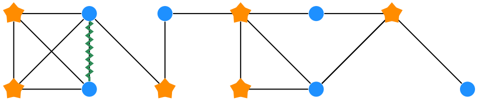
It is the NO-instances that we have to worry about preserving.

Note: If G is a YES-instance, then $G[X]$ is a YES-instance for any subset $X \subseteq E$.

We get one direction for free!

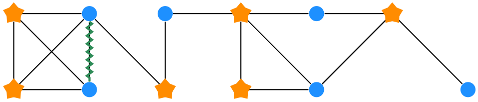
It is the NO-instances that we have to worry about preserving.

What is a NO-instance?



If G is a NO-instance:

For any subset S of size at most k,
there is an edge that is disjoint from S.



If G is a NO-instance:

For any subset S of size at most k,
there is an edge that is disjoint from S.

Ring a bell?

Recall.

We have at hand a p-uniform collection of independent sets, $\mathcal{F}$ and a number q. Let X be any set of size at most q. For any set $S \in \mathcal{F}$, if:

a X is disjoint from S, and

b X and S together form an independent set,

then a q-representative family contains a set $\widehat{S}$ that is:

a disjoint from X, and

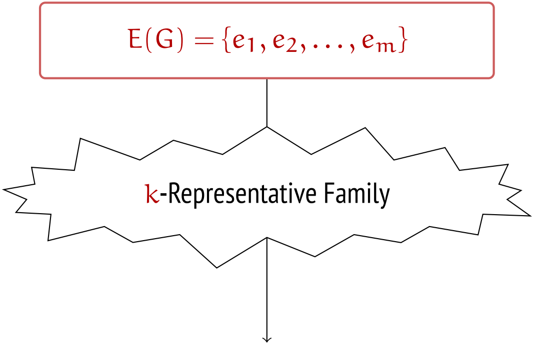
b forms an independent set together with X.

Such a subfamily is called a q-representative family for the given family.

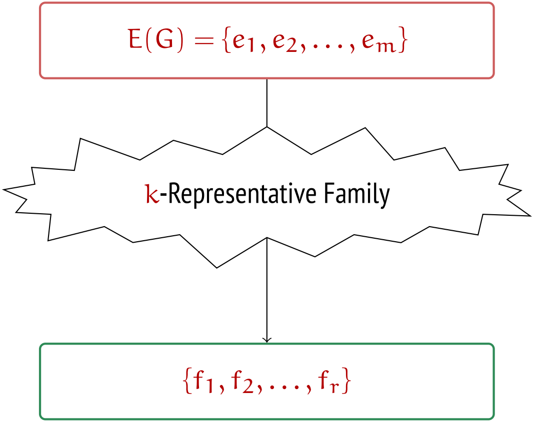
Claim: A k-representative family for E is in fact an $O(k^2)$ kernel for vertex cover.

$$E(G) = \{e_1, e_2, \ldots, e_m\}$$

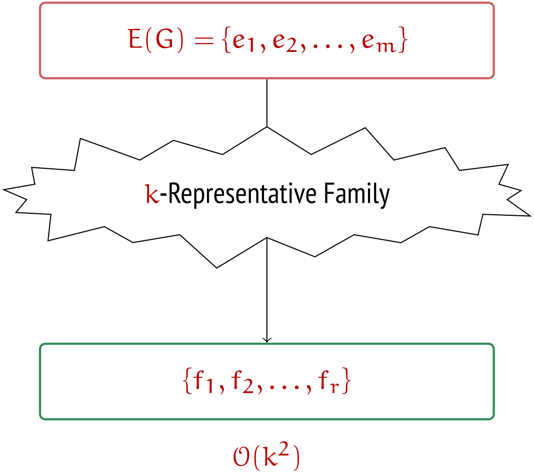
Is there a Vertex Cover of size at most k?



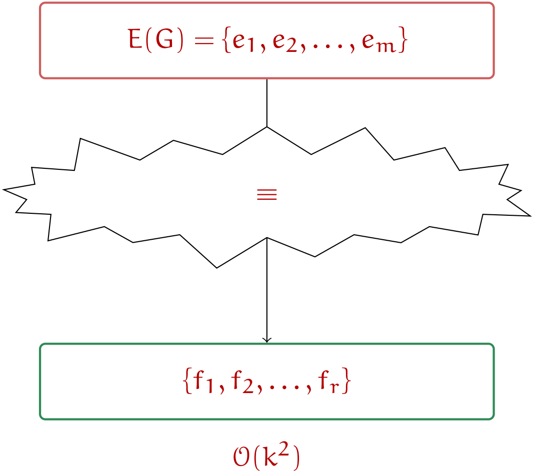
Is there a Vertex Cover of size at most k?



Is there a Vertex Cover of size at most k?



Is there a Vertex Cover of size at most k?

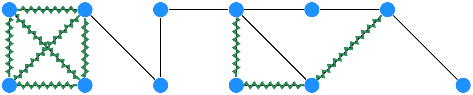


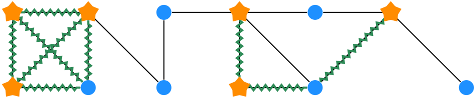
Is there a Vertex Cover of size at most k?

Let us show that if $G[X]$ is a YES-instance, then so is G.

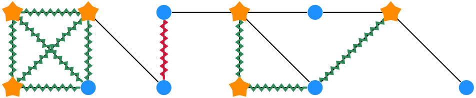
Let us show that if $G[X]$ is a YES-instance, then so is G.

This time, by contradiction.

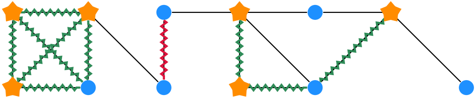




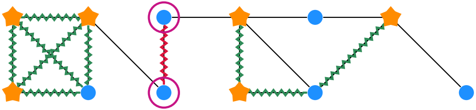
Try the solution for $G[X]$ on G.



Suppose there is an uncovered edge.

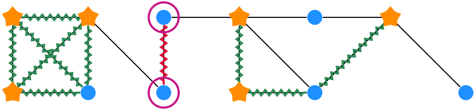


Since X is a k-representative family, for ANY $S \subseteq V$, where $|S| \leqslant k$:



Since X is a k-representative family, for ANY $S \subseteq V$, where $|S| \leqslant k$:

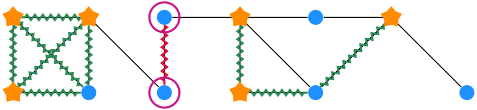
if there is a set e in E such that $e \cap S = \emptyset$,



Since X is a k-representative family, for ANY $S \subseteq V$, where $|S| \leqslant k$:

if there is a set e in E such that $e \cap S = \emptyset$,

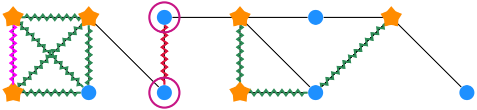
then there is a set $\widehat{e}$ in X such that $\widehat{e} \cap S = \emptyset$.



Since X is a k-representative family, for ANY $S \subseteq V$, where $|S| \leqslant k$:

if there is a set e in E such that $e \cap S = \emptyset$,
then there is a set $\hat{e}$ in X such that $\hat{e} \cap S = \emptyset$.

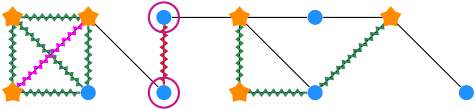
Note that the green edges denote $G[X]$.



Since X is a k-representative family, for ANY $S \subseteq V$, where $|S| \leqslant k$:

if there is a set e in E such that $e \cap S = \emptyset$,
then there is a set $\widehat{e}$ in X such that $\widehat{e} \cap S = \emptyset$.

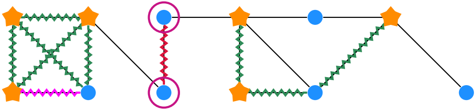
Note that the green edges denote $G[X]$.



Since X is a k-representative family, for ANY $S \subseteq V$, where $|S| \leqslant k$:

if there is a set e in E such that $e \cap S = \emptyset$,
then there is a set $\widehat{e}$ in X such that $\widehat{e} \cap S = \emptyset$.

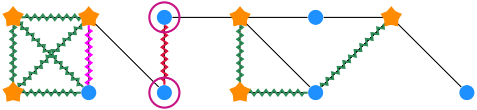
Note that the green edges denote $G[X]$.



Since X is a k-representative family, for ANY $S \subseteq V$, where $|S| \leqslant k$:

if there is a set e in E such that $e \cap S = \emptyset$,
then there is a set $\widehat{e}$ in X such that $\widehat{e} \cap S = \emptyset$.

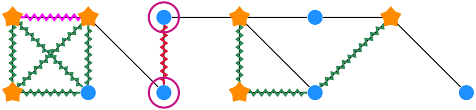
Note that the green edges denote $G[X]$.



Since X is a k-representative family, for ANY $S \subseteq V$, where $|S| \leqslant k$:

if there is a set e in E such that $e \cap S = \emptyset$,
then there is a set $\widehat{e}$ in X such that $\widehat{e} \cap S = \emptyset$.

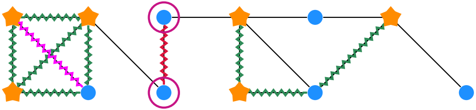
Note that the green edges denote $G[X]$.



Since X is a k-representative family, for ANY $S \subseteq V$, where $|S| \leqslant k$:

if there is a set e in E such that $e \cap S = \emptyset$,
then there is a set $\hat{e}$ in X such that $\hat{e} \cap S = \emptyset$.

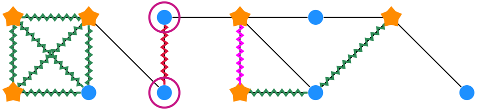
Note that the green edges denote $G[X]$.



Since X is a k-representative family, for ANY $S \subseteq V$, where $|S| \leqslant k$:

if there is a set e in E such that $e \cap S = \emptyset$,
then there is a set $\widehat{e}$ in X such that $\widehat{e} \cap S = \emptyset$.

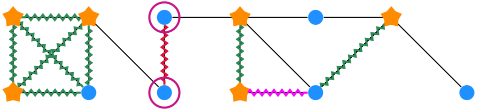
Note that the green edges denote $G[X]$.



Since X is a k-representative family, for ANY $S \subseteq V$, where $|S| \leqslant k$:

if there is a set e in E such that $e \cap S = \emptyset$,
then there is a set $\widehat{e}$ in X such that $\widehat{e} \cap S = \emptyset$.

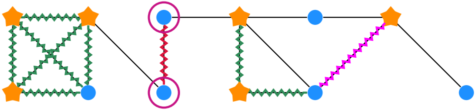
Note that the green edges denote $G[X]$.



Since X is a k-representative family, for ANY $S \subseteq V$, where $|S| \leqslant k$:

if there is a set e in E such that $e \cap S = \emptyset$,
then there is a set $\hat{e}$ in X such that $\hat{e} \cap S = \emptyset$.

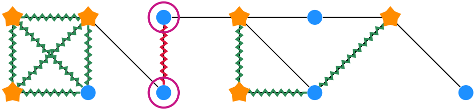
Note that the green edges denote $G[X]$.



Since X is a k-representative family, for ANY $S \subseteq V$, where $|S| \leqslant k$:

if there is a set e in E such that $e \cap S = \emptyset$,
then there is a set $\hat{e}$ in X such that $\hat{e} \cap S = \emptyset$.

Note that the green edges denote $G[X]$.



Since X is a k-representative family, for ANY $S \subseteq V$, where $|S| \leqslant k$:

if there is a set e in E such that $e \cap S = \emptyset$,
then there is a set $\widehat{e}$ in X such that $\widehat{e} \cap S = \emptyset$.

Note that the green edges denote $G[X]$.

Contradiction!

A k-representative family for $E(G)$ is in fact an $O(k^2)$ instance kernel for Vertex Cover!

REPRESENTATIVE SETS

Why, What and How.

REPRESENTATIVE SETS

And that will be all!