Minicourse on parameterized algorithms and complexity

Part 2: Iterative compression

Dániel Marx
(slides by Marek Cygan)

Jagiellonian University in Kraków
April 21-23, 2015

What iterative compression is?

Iterative compression — main idea

Recursive approach exploiting instance structure exposed by a bit oversized solution.

What iterative compression is?

Iterative compression — main idea

Recursive approach exploiting instance structure exposed by a bit oversized solution.

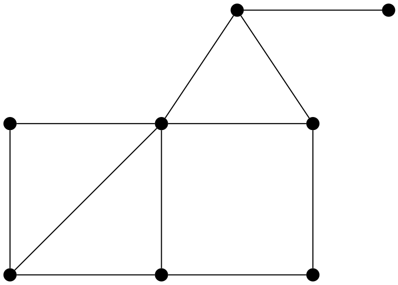
Solution compression:

1. First, apply some simple trick so that you can assume that a slightly too large solution is available.
2. Then exploit the structure it imposes on the input graph to construct an optimal solution.

Vertex Cover

Vertex Cover

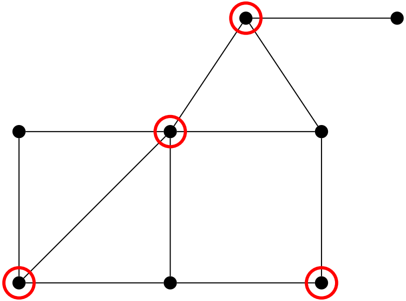
Input: undirected G, integer k
Question: is there a subset $X \subseteq V(G)$ of size at most k such that for each $uv \in E(G)$ we have $\{u, v\} \cap X \neq \emptyset$.



Vertex Cover

Vertex Cover

Input: undirected G, integer k
Question: is there a subset $X \subseteq V(G)$ of size at most k such that for each $uv \in E(G)$ we have $\{u, v\} \cap X \neq \emptyset$.



Additional input: oversized solution

We exemplify the iterative compression technique by showing $2^k n^{\mathcal{O}(1)}$ algorithm for Vertex Cover.

Additional input: oversized solution

We exemplify the iterative compression technique by showing $2^k n^{\mathcal{O}(1)}$ algorithm for Vertex Cover.

Vertex Cover Compression

Input: undirected G, integer k,
vertex cover $Z \subseteq V(G)$ of size at most $2k$

Question: is there a vertex cover of size at most k?

Additional input: oversized solution

We exemplify the iterative compression technique by showing $2^k n^{\mathcal{O}(1)}$ algorithm for Vertex Cover.

Vertex Cover Compression

Input: undirected G, integer k,
vertex cover $Z \subseteq V(G)$ of size at most $2k$

Question: is there a vertex cover of size at most k?

- Where do we get Z from?
- How do we use Z?

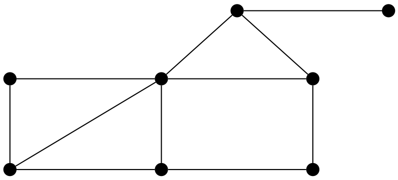
Additional input: oversized solution

Where do we get Z from?

Additional input: oversized solution

Where do we get Z from?

Use polynomial time 2-approximation:

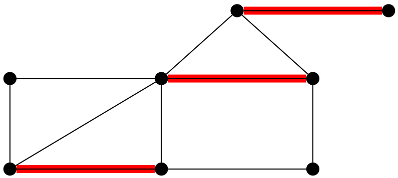


Additional input: oversized solution

Where do we get Z from?

Use polynomial time 2-approximation:

- Find any inclusionwise maximal matching M.

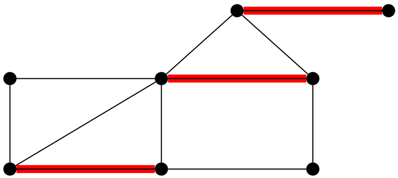


Additional input: oversized solution

Where do we get Z from?

Use polynomial time 2-approximation:

- Find any inclusionwise maximal matching M.
- If $|M| > k$, then no VC of size $\leqslant k$ exists.

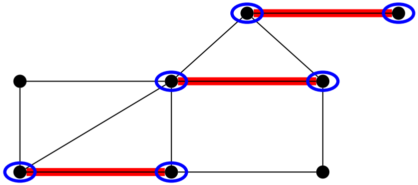


Additional input: oversized solution

Where do we get Z from?

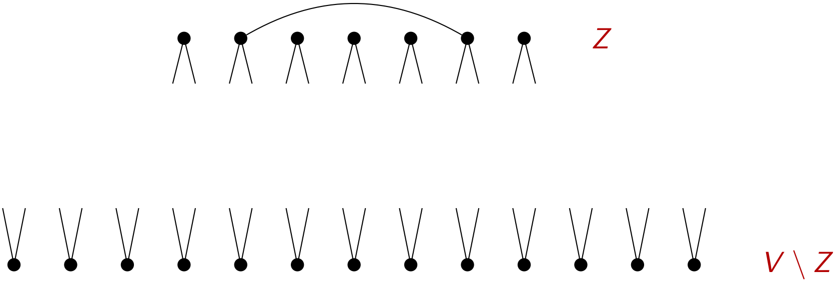
Use polynomial time 2-approximation:

- Find any inclusionwise maximal matching M.
- If $|M| > k$, then no VC of size $\leqslant k$ exists.
- Otherwise, set $Z = V(M)$, we have $|Z| \leqslant 2k$.



Additional input: oversized solution

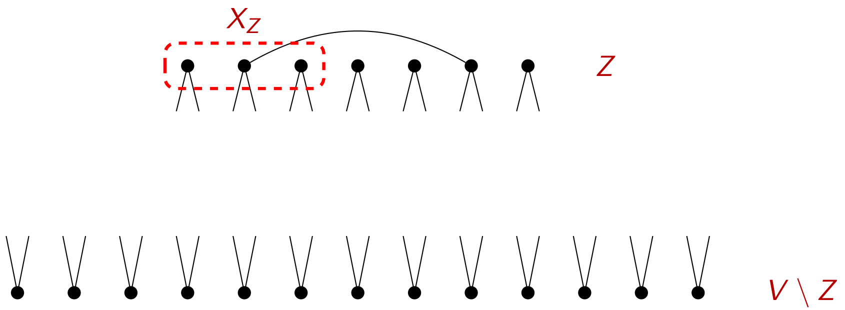
How do we use Z?



Additional input: oversized solution

How do we use Z?

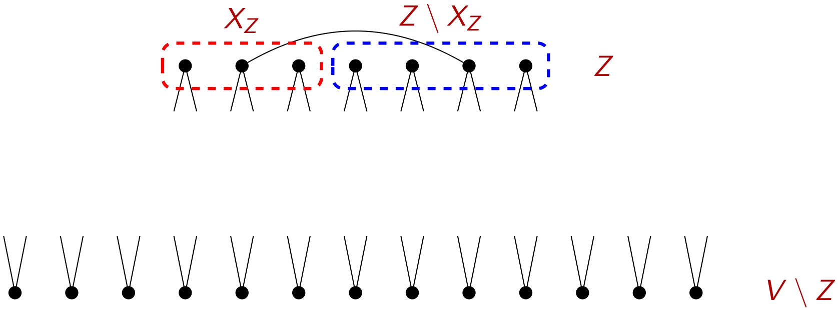
- Guess $X \cap Z = X_Z$ (by branching into $2^{|Z|} \leqslant 4^k$ cases).



Additional input: oversized solution

How do we use Z?

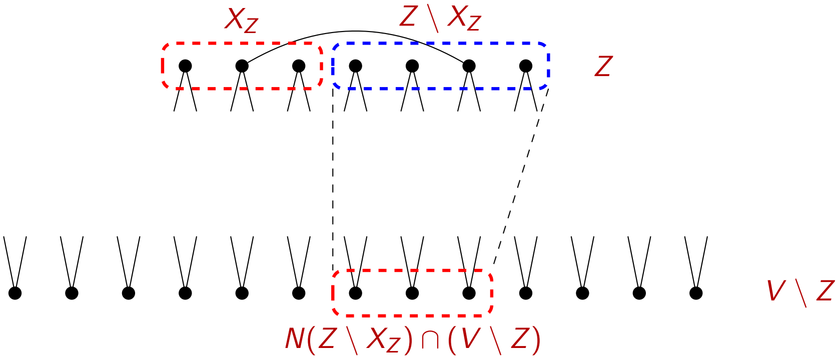
- Guess $X \cap Z = X_Z$ (by branching into $2^{|Z|} \leqslant 4^k$ cases).



Additional input: oversized solution

How do we use Z?

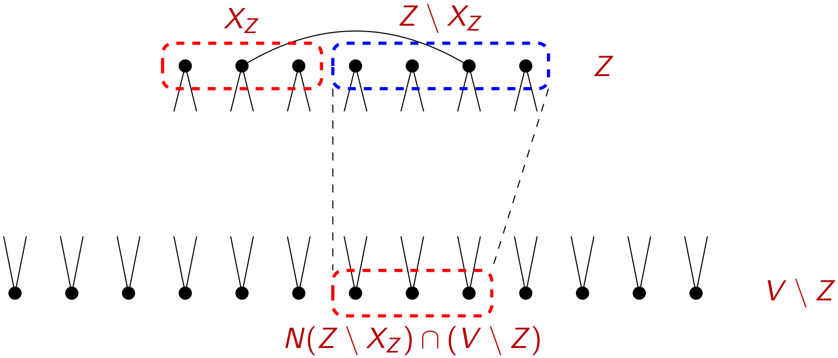
- Guess $X \cap Z = X_Z$ (by branching into $2^{|Z|} \leqslant 4^k$ cases).



Additional input: oversized solution

How do we use Z?

- Guess $X \cap Z = X_Z$ (by branching into $2^{|Z|} \leqslant 4^k$ cases).
- Check if $Z \setminus X_Z$ is independent and $|X_Z \cup N(Z \setminus X_Z)| \leqslant k$.



Additional input: oversized solution

How do we use Z?

- We have obtained $2^{|Z|} n^{\mathcal{O}(1)} \leqslant 4^k n^{\mathcal{O}(1)}$ time algorithm.

Additional input: oversized solution

How do we use Z?

- We have obtained $2^{|Z|}n^{\mathcal{O}(1)} \leqslant 4^k n^{\mathcal{O}(1)}$ time algorithm.
- Can we improve the dependency on k to 2^k?

Additional input: oversized solution

How do we use Z?

- We have obtained $2^{|Z|} n^{\mathcal{O}(1)} \leqslant 4^k n^{\mathcal{O}(1)}$ time algorithm.
- Can we improve the dependency on k to 2^k?
- Notice that it would be enough to have $|Z| \leqslant k+1$, but so far we only have $|Z| \leqslant 2k$.

Vertex Cover

Vertex Cover Compression

Input: undirected G, integer k,
vertex cover $Z \subseteq V(G)$ of size at most $k+1$
Question: is there a vertex cover of size at most k?

Vertex Cover

Vertex Cover Compression

Input: undirected G, integer k,
vertex cover $Z \subseteq V(G)$ of size at most $k+1$

Question: is there a vertex cover of size at most k?

Idea: get Z from recursion!

Bootstrapping

How to get Z of size at most $k+1$?

- Assume that an instance $I = (G, k)$ without Z is given.

Bootstrapping

How to get Z of size at most $k+1$?

- Assume that an instance $I = (G, k)$ without Z is given.
- Pick any $v \in V(G)$ and solve $I' = (G \setminus \{v\}, k)$ recursively.

Bootstrapping

How to get Z of size at most $k + 1$?

- Assume that an instance $I = (G, k)$ without Z is given.
- Pick any $v \in V(G)$ and solve $I' = (G \setminus \{v\}, k)$ recursively.
- If I' is a NO-instance then I is a NO-instance.

Bootstrapping

How to get Z of size at most $k+1$?

- Assume that an instance $I = (G, k)$ without Z is given.
- Pick any $v \in V(G)$ and solve $I' = (G \setminus \{v\}, k)$ recursively.
- If I' is a NO-instance then I is a NO-instance.
- Otherwise set $Z = X \cup \{v\}$, where X is a solution for I'.

Bootstrapping

How to get Z of size at most $k+1$?

- Assume that an instance $I = (G, k)$ without Z is given.
- Pick any $v \in V(G)$ and solve $I' = (G \setminus \{v\}, k)$ recursively.
- If I' is a NO-instance then I is a NO-instance.
- Otherwise set $Z = X \cup \{v\}$, where X is a solution for I'.
- (G, k, Z) is VC Compression instance to solve.

Bootstrapping

How to get Z of size at most $k+1$?

- Assume that an instance $I = (G, k)$ without Z is given.
- Pick any $v \in V(G)$ and solve $I' = (G \setminus \{v\}, k)$ recursively.
- If I' is a NO-instance then I is a NO-instance.
- Otherwise set $Z = X \cup \{v\}$, where X is a solution for I'.
- (G, k, Z) is VC Compression instance to solve.

Lemma

$f(k)n^c$ time algorithm for VC Compression
implies $f(k)n^{c+1}$ time algorithm for VC.

Vertex Cover - summary

Lemma

$f(k)n^c$ time algorithm for VC Compression
implies $f(k)n^{c+1}$ time algorithm for VC.

Reduction: Vertex Cover $\rightarrow$ Vertex Cover Compression.

Vertex Cover - summary

Lemma

$f(k)n^c$ time algorithm for VC Compression
implies $f(k)n^{c+1}$ time algorithm for VC.

Reduction: Vertex Cover $\longrightarrow$ Vertex Cover Compression.

Vertex Cover Compression can be solved in time $2^{|Z|}n^{\mathcal{O}(1)}$,
which leads to $2^k n^{\mathcal{O}(1)}$ algorithm for VC.

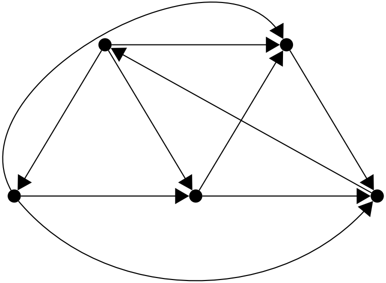
Outline

1. Iterative compression - introduction.
2. Learning by example - vertex cover.
3. Learning by example - FVS in tournament.
4. Generic steps of the method.
5. $5^k n^{\mathcal{O}(1)}$ algorithm for FVS.
6. $3^k n^{\mathcal{O}(1)}$ algorithm for OCT - sketch.

FVS in tournaments

Feedback Vertex Set (FVS) in Tournaments

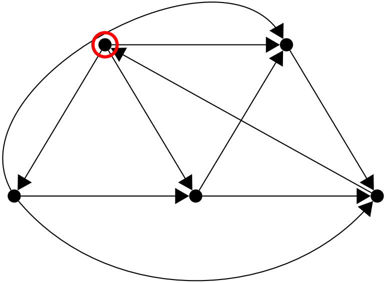
Input: a tournament (oriented clique) T, integer k
Question: is there a subset $X \subseteq V(T)$ of size at most k,
such that $T \setminus X$ is acyclic



FVS in tournaments

Feedback Vertex Set (FVS) in Tournaments

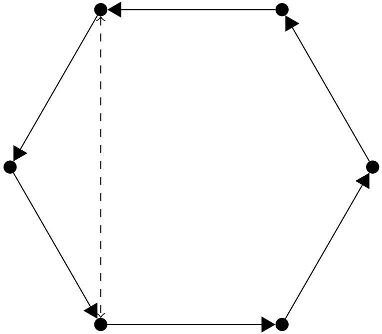
Input: a tournament (oriented clique) T, integer k
Question: is there a subset $X \subseteq V(T)$ of size at most k,
such that $T \setminus X$ is acyclic



FVS in tournaments

Lemma

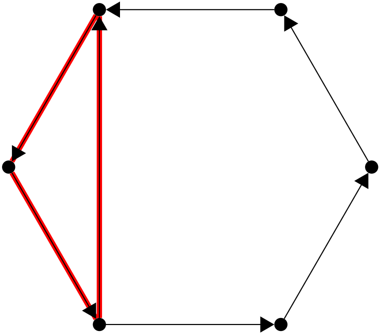
If a tournament contains a cycle, then it contains a 3-cycle.



FVS in tournaments

Lemma

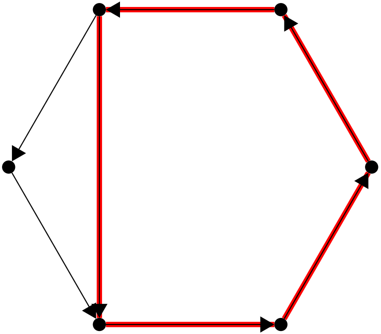
If a tournament contains a cycle, then it contains a 3-cycle.



FVS in tournaments

Lemma

If a tournament contains a cycle, then it contains a 3-cycle.



FVS in tournaments

Lemma

If a tournament contains a cycle, then it contains a 3-cycle.

- This lemma implies a simple $3^k n^{\mathcal{O}(1)}$ branching algorithm.

FVS in tournaments

Lemma

If a tournament contains a cycle, then it contains a 3-cycle.

- This lemma implies a simple $3^k n^{\mathcal{O}(1)}$ branching algorithm.
- By using iterative compression we will see how to improve the running time to $2^k n^{\mathcal{O}(1)}$.

FVS in tournaments

Start with the recursive trick, reducing the problem to its compression version.

FVS in tournaments

Start with the recursive trick, reducing the problem to its compression version.

Feedback Vertex Set (FVS) in Tournaments Compression

Input: a tournament (oriented clique) T, integer k
a FVS $Z \subseteq V(T)$ of size at most $k+1$
Question: is there a subset $X \subseteq V(T)$ of size at most k,
such that $T \setminus X$ is acyclic

FVS in tournaments

Start with the recursive trick, reducing the problem to its compression version.

Feedback Vertex Set (FVS) in Tournaments Compression

Input: a tournament (oriented clique) T, integer k
a FVS $Z \subseteq V(T)$ of size at most $k+1$
Question: is there a subset $X \subseteq V(T)$ of size at most k,
such that $T \setminus X$ is acyclic

Lemma

$f(k)n^c$ time algorithm for FVST Compression implies $f(k)n^{c+1}$ time algorithm for FVST.

FVS in tournaments

Pf: this time we use induction (loop) - alternative to recursion.

- Let $V(T) = \{v_1, \ldots, v_n\}$.

FVS in tournaments

Pf: this time we use induction (loop) - alternative to recursion.

- Let $V(T) = \{v_1, \ldots, v_n\}$.
- We want to solve $FVST(T[V_i], k)$ for $i = 1, \ldots, n$, where $V_i = \{v_1, \ldots, v_i\}$.

FVS in tournaments

Pf: this time we use induction (loop) - alternative to recursion.

- Let $V(T) = \{v_1, \ldots, v_n\}$.
- We want to solve $FVST(T[V_i], k)$ for $i = 1, \ldots, n$, where $V_i = \{v_1, \ldots, v_i\}$.
- Set $X_1 = \emptyset$, which is a solution for $FVST(T[v_1], k)$.

FVS in tournaments

Pf: this time we use induction (loop) - alternative to recursion.

- Let $V(T) = \{v_1, \dots, v_n\}$.
- We want to solve $FVST(T[V_i], k)$ for $i = 1, \dots, n$, where $V_i = \{v_1, \dots, v_i\}$.
- Set $X_1 = \emptyset$, which is a solution for $FVST(T[v_1], k)$.
- For $2 \leqslant i \leqslant n$ do
  - $Z_i = X_{i-1} \cup \{v_i\}$,
  - let X_i be a solution to $FVST\ Compression(T[V_i], k, Z_i)$.
  - if no solution found for $T[V_i]$, then return NO.

FVS in tournaments

Feedback Vertex Set (FVS) in Tournaments Compression

Input: a tournament (oriented clique) T, integer k
a FVS $Z \subseteq V(T)$ of size at most $k+1$
Question: is there a subset $X \subseteq V(T)$ of size at most k,
such that $T \setminus X$ is acyclic

By guessing a partition $Z = X_Z \uplus W$, we get to the disjoint version.

FVS in tournaments

Feedback Vertex Set (FVS) in Tournaments Compression

Input: a tournament (oriented clique) T, integer k
a FVS $Z \subseteq V(T)$ of size at most $k+1$
Question: is there a subset $X \subseteq V(T)$ of size at most k,
such that $T \setminus X$ is acyclic

By guessing a partition $Z = X_Z \uplus W$, we get to the disjoint version.

Disjoint FVS in Tournaments Compression

Input: a tournament (oriented clique) T, integer k
a FVS $W \subseteq V(T)$ of size at most $k+1$
Question: is there a subset $X \subseteq V(T)$ of size at most k,
disjoint with W, such that $T \setminus X$ is acyclic

FVS in tournaments

Disjoint FVS in Tournaments Compression

Input: a tournament (oriented clique) T, integer k
a FVS $W \subseteq V(T)$ of size at most $k+1$

Question: is there a subset $X \subseteq V(T)$ of size at most k,
disjoint with W, such that $T \setminus X$ is acyclic

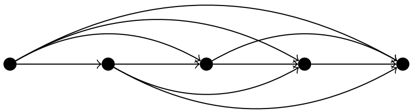
Lemma

Poly time algorithm for Disjoint FVST Compression implies $2^k n^{\mathcal{O}(1)}$ time algorithm for FVST Compression.

Disjoint FVS in tournaments

Observation

For an acyclic tournament, there is a single topological ordering.



Disjoint FVS in tournaments

Simple reduction rules:

Disjoint FVS in tournaments

Simple reduction rules:

Reduction 1

If $T[W]$ is not acyclic, then answer NO.

Disjoint FVS in tournaments

Simple reduction rules:

Reduction 1

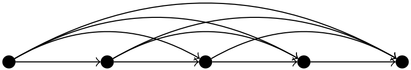
If $T[W]$ is not acyclic, then answer NO.

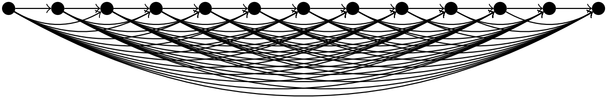
Let $A = V(T) \setminus W$ (removable set).

Reduction 2

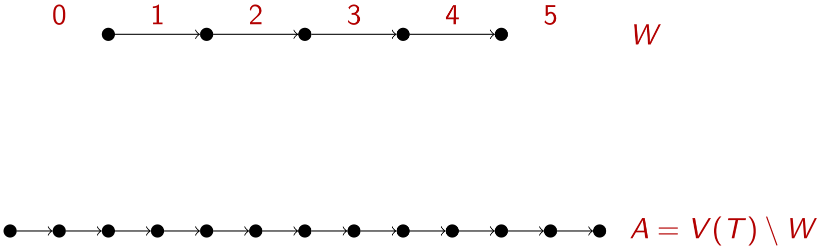
If for $v \in A$ the graph $T[W \cup \{v\}]$ contains a cycle,
then remove v and reduce k by one.

Disjoint FVS in tournaments

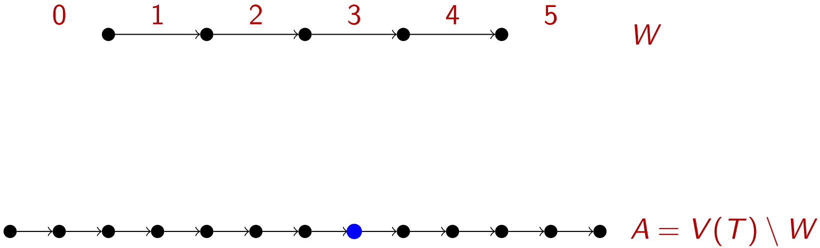




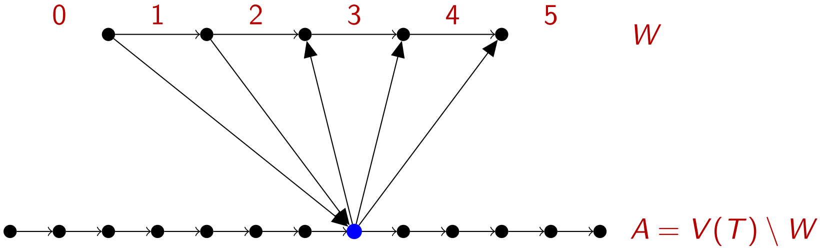
Disjoint FVS in tournaments



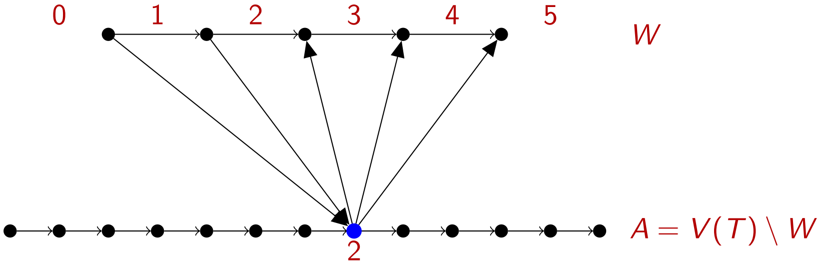
Disjoint FVS in tournaments



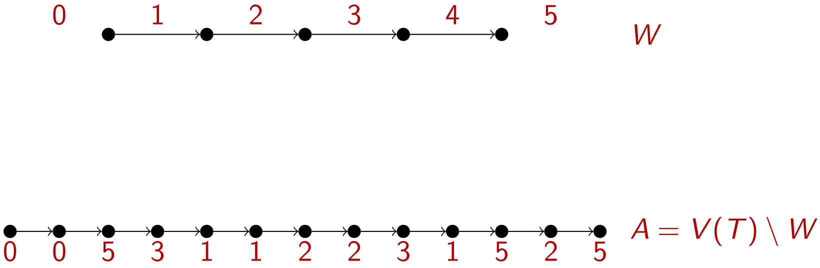
Disjoint FVS in tournaments



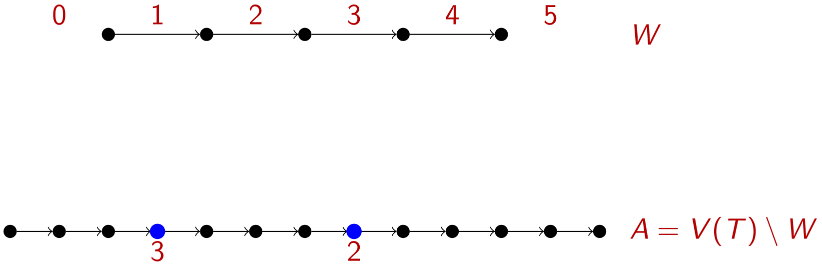
Disjoint FVS in tournaments



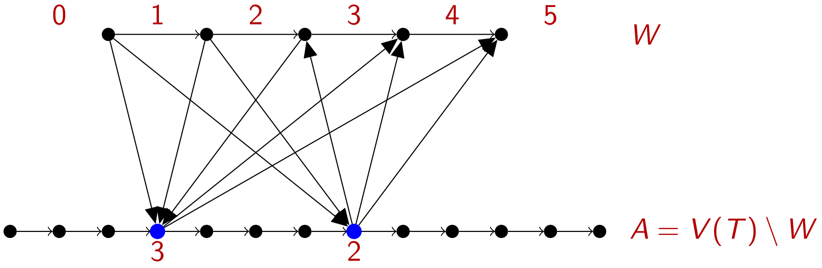
Disjoint FVS in tournaments



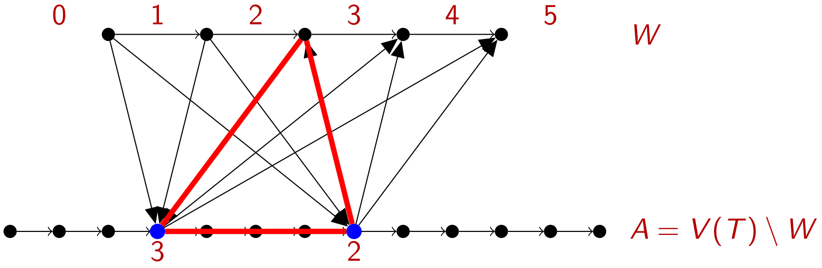
Disjoint FVS in tournaments



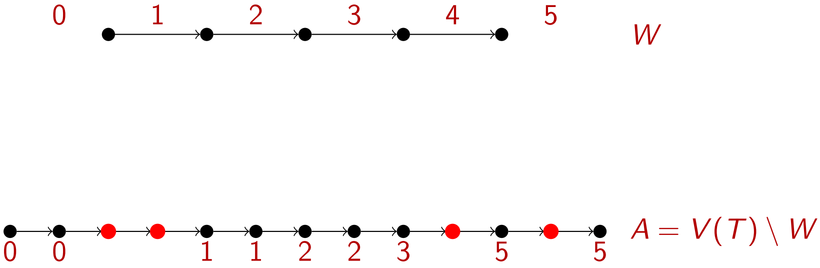
Disjoint FVS in tournaments



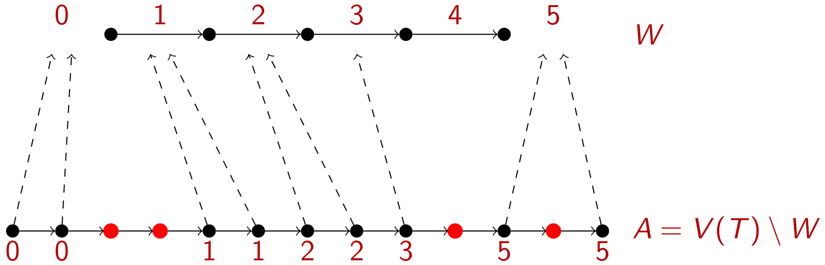
Disjoint FVS in tournaments



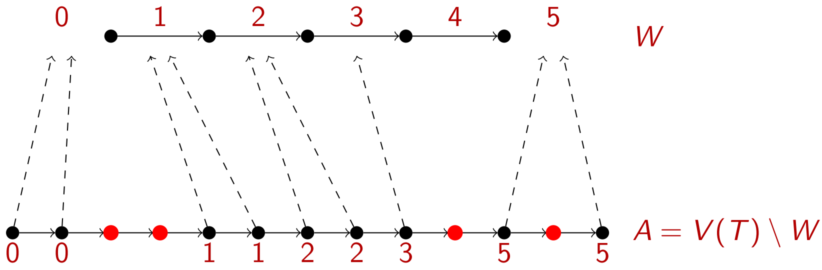
Disjoint FVS in tournaments



Disjoint FVS in tournaments



Disjoint FVS in tournaments



Consequently Disjoint FVST Compression may be reduced to finding longest nondecreasing subsequence.

General framework

Iterative compression schema:

- By using induction we can assume that a solution $Z \subseteq V(G)$, $|Z| \leqslant k+1$ is given as part of input.

General framework

Iterative compression schema:

- By using induction we can assume that a solution $Z \subseteq V(G)$, $|Z| \leqslant k+1$ is given as part of input.
- Branch into $2^{|Z|}$ cases, guessing what part of Z should be in a solution.

General framework

Iterative compression schema:

- By using induction we can assume that a solution $Z \subseteq V(G)$, $|Z| \leqslant k+1$ is given as part of input.
- Branch into $2^{|Z|}$ cases, guessing what part of Z should be in a solution.
- Solve a disjoint version of the problem, where given a solution $W \subseteq V(G)$ we look for $X \subseteq V(G) \setminus W$ of size at most k.

General framework

Iterative compression schema:

- By using induction we can assume that a solution $Z \subseteq V(G)$, $|Z| \leqslant k+1$ is given as part of input.
- Branch into $2^{|Z|}$ cases, guessing what part of Z should be in a solution.
- Solve a disjoint version of the problem, where given a solution $W \subseteq V(G)$ we look for $X \subseteq V(G) \setminus W$ of size at most k.
- $c^k n^{\mathcal{O}(1)}$ time algorithm for the disjoint version implies $(2c)^k n^{\mathcal{O}(1)}$ time algorithm for the general problem.

General framework

Lemma

$c^k n^{\mathcal{O}(1)}$ time algorithm for the disjoint version implies $(c+1)^k n^{\mathcal{O}(1)}$ time algorithm for the general problem.

General framework

Lemma

$c^k n^{\mathcal{O}(1)}$ time algorithm for the disjoint version implies $(c+1)^k n^{\mathcal{O}(1)}$ time algorithm for the general problem.

$$\sum_{X \subseteq Z} c^{k-|X|} = \sum_{i=0}^{k+1} \binom{k+1}{i} c^{k-i} 1^i = (c+1)^{k+1}/c$$

General framework

Remarks:

- To make induction work, we need to find a solution (answering YES/NO is not enough).

General framework

Remarks:

- To make induction work, we need to find a solution (answering YES/NO is not enough).
- By default iterative compression adds n factor to the running time.

General framework

Remarks:

- To make induction work, we need to find a solution (answering YES/NO is not enough).
- By default iterative compression adds n factor to the running time.
- Ex: show that for VC and FVST this factor can be reduced to $\mathcal{O}(k)$ (hint: use $\mathcal{O}(1)$-approximation).

General framework

Remarks:

- To make induction work, we need to find a solution (answering YES/NO is not enough).
- By default iterative compression adds n factor to the running time.
- Ex: show that for VC and FVST this factor can be reduced to $\mathcal{O}(k)$ (hint: use $\mathcal{O}(1)$-approximation).
- Some natural problems are not vertex deletion closed.

General framework

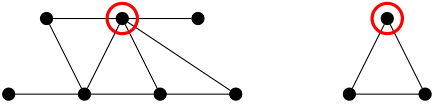
Remarks:

- To make induction work, we need to find a solution (answering YES/NO is not enough).
- By default iterative compression adds n factor to the running time.
- Ex: show that for VC and FVST this factor can be reduced to $\mathcal{O}(k)$ (hint: use $\mathcal{O}(1)$-approximation).
- Some natural problems are not vertex deletion closed.
- Ex: reduce Connected Vertex Cover (CVC) to CVC-Compression.

FVS

Feedback Vertex Set (FVS)

Input: undirected G, integer k
Question: is there a subset $X \subseteq V(G)$ of size at most k, such that $G \setminus X$ is a forest



FVS

FVS is vertex deletion closed, so we can apply iterative compression schema and solving the following problem in time $c^k n^{\mathcal{O}(1)}$ leads to $(c+1)^k n^{\mathcal{O}(1)}$ time algorithm for FVS.

FVS

FVS is vertex deletion closed, so we can apply iterative compression schema and solving the following problem in time $c^k n^{\mathcal{O}(1)}$ leads to $(c+1)^k n^{\mathcal{O}(1)}$ time algorithm for FVS.

Disjoint FVS Compression

Input: undirected G, integer k
a FVS $W \subseteq V(G)$ of size at most $k+1$
Question: is there a subset $X \subseteq V(G)$ of size at most k, disjoint with W, such that $G \setminus X$ is a forest

FVS - reduction rules

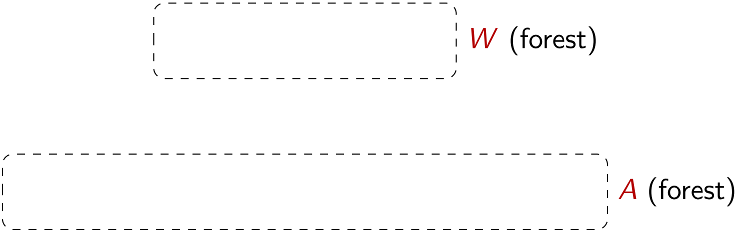
Reduction 0

If $G[W]$ contains a cycle, return NO.

FVS - reduction rules

Reduction 0

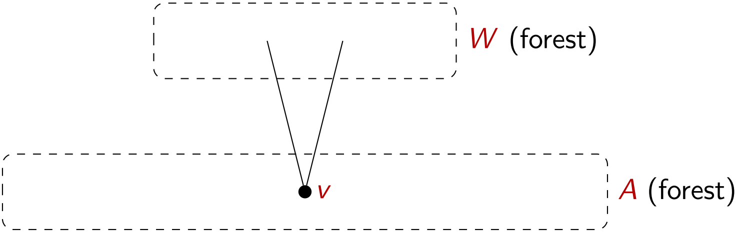
If $G[W]$ contains a cycle, return NO.



FVS - reduction rules

Reduction 0

If $G[W]$ contains a cycle, return NO.

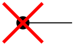


We want v to have $\geqslant 2$ incident edges going to W.

FVS - reduction rules

Reduction 1

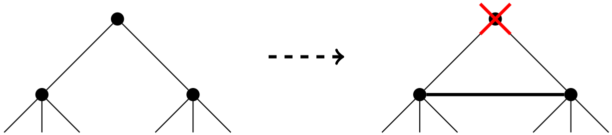
Remove all degree at most 1 vertices from G.



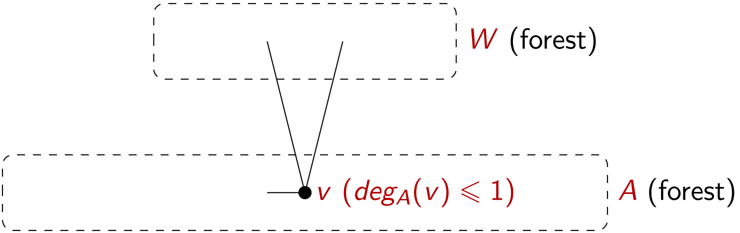
FVS - reduction rules

Reduction 2

If there is $v \in A$ with $\deg(v) = 2$ and at least one neighbor in A, then add an edge between neighbours of v (even if there was one) and remove v.

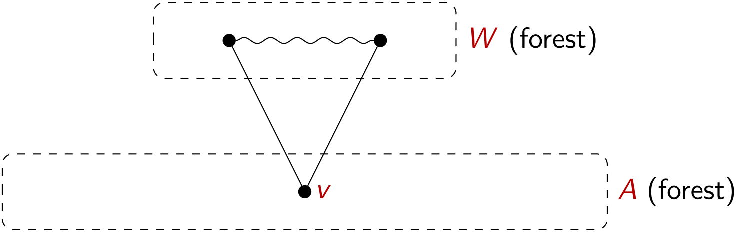


FVS - reduction rules

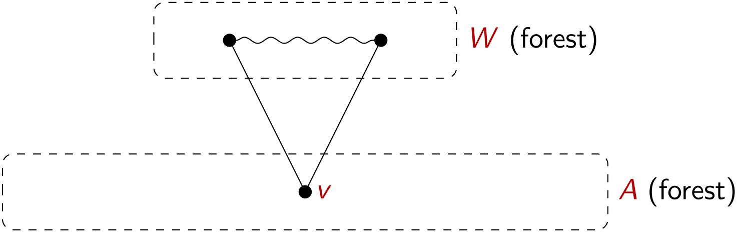


Any leaf v in A has now at least two edges to W.

FVS - one more reduction rule



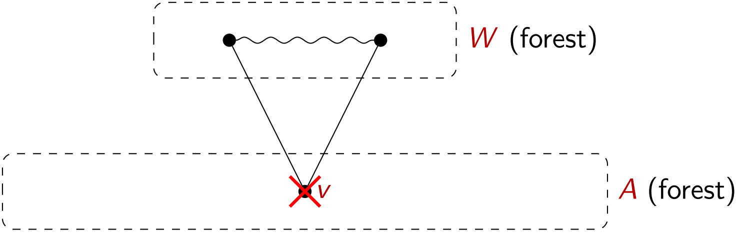
FVS - one more reduction rule



Reduction 3

If for $v \in A = V(G) \setminus W$ the graph $G[W \cup \{v\}]$ contains a cycle, then remove v and decrease k by one.

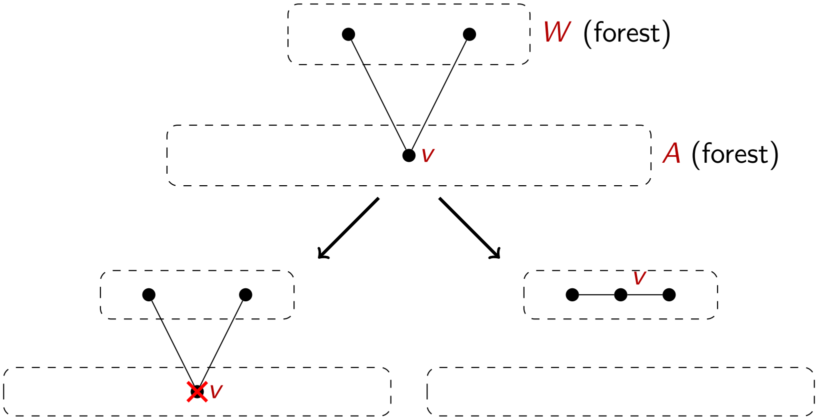
FVS - one more reduction rule



Reduction 3

If for $v \in A = V(G) \setminus W$ the graph $G[W \cup \{v\}]$ contains a cycle, then remove v and decrease k by one.

FVS branching



FVS branching

Formally, we branch into instances:

- $(G \setminus \{v\}, k-1, W)$,
- $(G, k, W \cup \{v\})$.

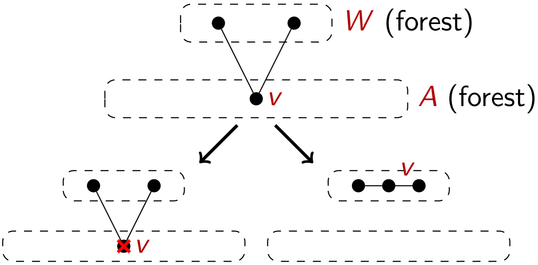
FVS branching

Formally, we branch into instances:

- $(G \setminus \{v\}, k-1, W)$,
- $(G, k, W \cup \{v\})$.

Observation

A potential $\pi(I) = k + \#cc(G[W])$ decreases in each branch.



FVS branching

Formally, we branch into instances:

- $(G \setminus \{v\}, k-1, W)$,
- $(G, k, W \cup \{v\})$.

Observation

A potential $\pi(I) = k + \#cc(G[W])$ decreases in each branch.

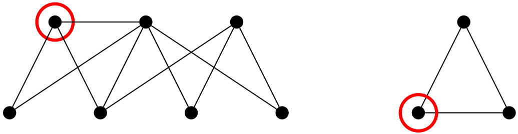
Lemma

Disjoint FVS Compression can be solved in time $4^k n^{\mathcal{O}(1)}$, consequently there is $5^k n^{\mathcal{O}(1)}$ time algorithm for FVS.

OCT

Odd Cycle Transversal (OCT)

Input: undirected G, integer k
Question: is there a subset $X \subseteq V(G)$ of size at most k, such that $G \setminus X$ is bipartite



OCT

The heart of the solution for OCT by iterative compression is the following problem, which can be solved in polynomial time!

Annotated Bipartite Coloring

Input: bipartite $G = (V_1, V_2, E)$, integer k,
a partial coloring $f_0 : V(G) \to \{1, 2, ?\}$

Question: is there a subset $X \subseteq V(G)$ of size at most k,
and a proper coloring f of $G \setminus X$ consistent with f_0.

OCT

Annotated Bipartite Coloring

Input: bipartite $G = (V_1, V_2, E)$, integer k,
a partial coloring $f_0 : V(G) \to \{1, 2, ?\}$

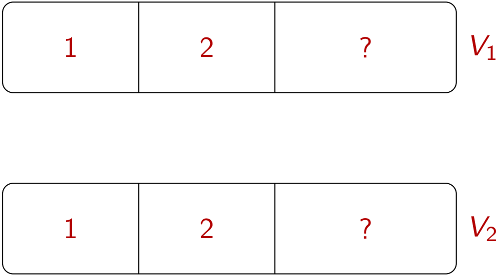
Question: is there a subset $X \subseteq V(G)$ of size at most k,
and a proper coloring f of $G \setminus X$ consistent with f_0.

OCT

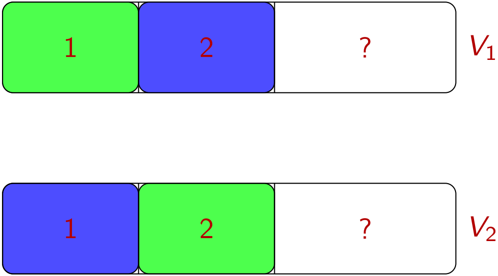
Annotated Bipartite Coloring

Input: bipartite $G = (V_1, V_2, E)$, integer k,
a partial coloring $f_0 : V(G) \to \{1, 2, ?\}$

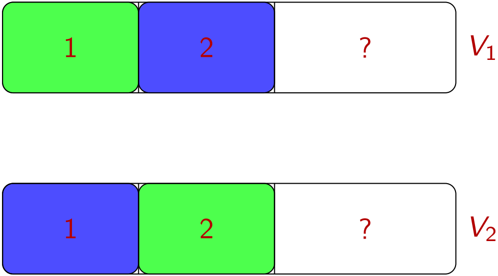
Question: is there a subset $X \subseteq V(G)$ of size at most k,
and a proper coloring f of $G \setminus X$ consistent with f_0.



OCT

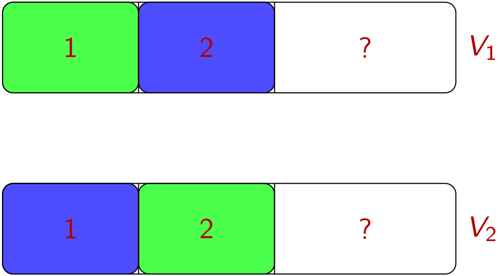


OCT



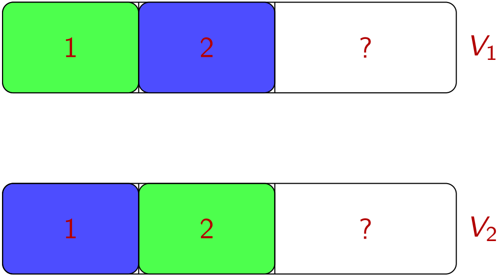
- each blue vertex is either removed or recolored wrt $V_1 \uplus V_2$,

OCT



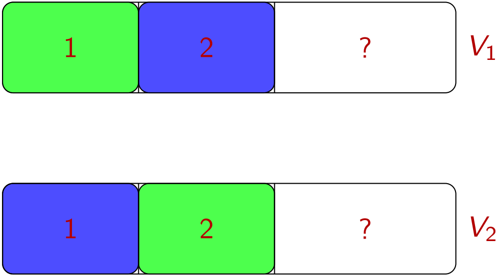
- each blue vertex is either removed or recolored wrt $V_1 \uplus V_2$,
- each green vertex is removed or maintains color wrt $V_1 \uplus V_2$,

OCT



- each blue vertex is either removed or recolored wrt $V_1 \uplus V_2$,
- each green vertex is removed or maintains color wrt $V_1 \uplus V_2$,
- for each $e \in E(G \setminus X)$ either both vertices are recolored, or none,

OCT



- each blue vertex is either removed or recolored wrt $V_1 \uplus V_2$,
- each green vertex is removed or maintains color wrt $V_1 \uplus V_2$,
- for each $e \in E(G \setminus X)$ either both vertices are recolored, or none,
- algorithm: find min cut between green and blue!

Summary

Iterative compression

Recursive approach exploiting instance structure exposed by a bit oversized solution.

We have seen it applied to:

- Vertex Cover,
- FVS in Tournaments,
- FVS,
- OCT (sketch).