A Fast Branching Algorithm for Cluster Vertex Deletion

Anudhyan Boral[1] Marek Cygan[2]
Tomasz Kociumaka[2] Marcin Pilipczuk[3]

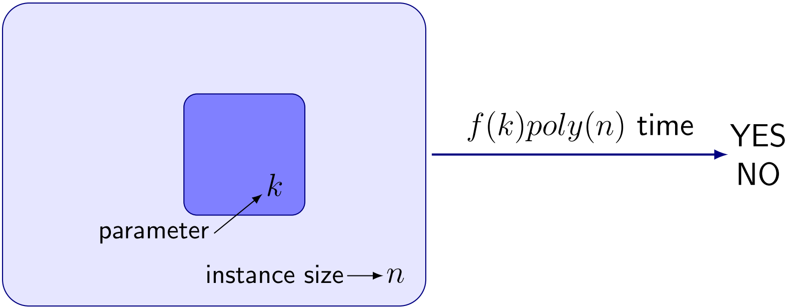
[1]Harvard University, USA

[2]University of Warsaw, Poland

[3]University of Bergen, Norway

CSR 2014
Moscow, Russia
June 10, 2014

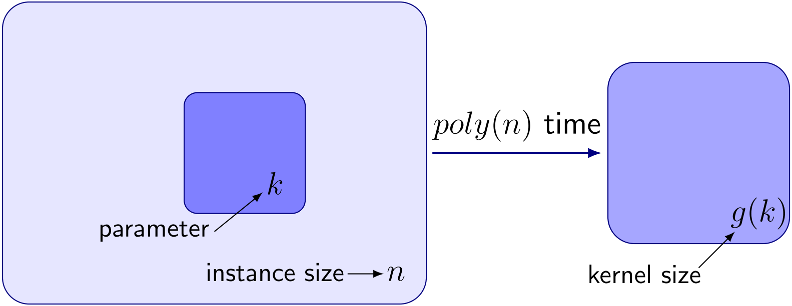
Parameterized complexity and kernelization



Definition

An FPT-algorithm for a parameterized problem runs in $\mathcal{O}(f(k)n^c)$-time, where c is a constant (independent of k).

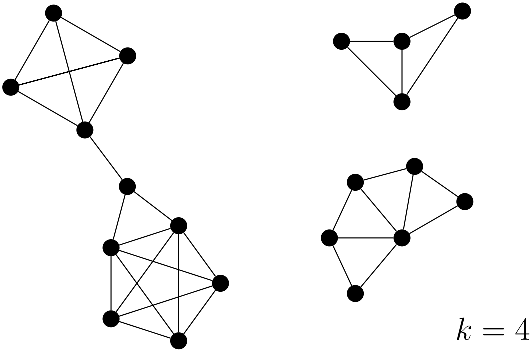
Parameterized complexity and kernelization



Definition

A kernel of size $g(k)$ is a polynomial-time algorithm, which reduces an instance of a parameterized problem to an equivalent instance of size at most $g(k)$.

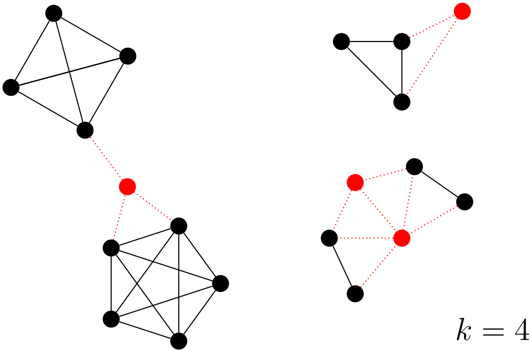
Cluster Vertex Deletion



Problem (Cluster Vertex Deletion, CVD)

Input: *an undirected graph* $G = (V, E)$*, a positive integer* k*.*
Output: *a set* $S \subseteq V$ *such that* $|S| \leq k$ *and* $G \setminus S$ *is a cluster graph (disjoint union of cliques).*

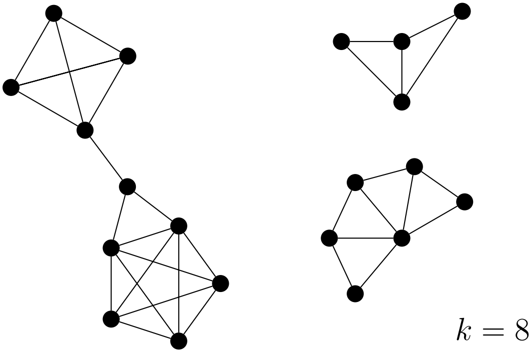
Cluster Vertex Deletion



Problem (Cluster Vertex Deletion, CVD)

Input: *an undirected graph* $G = (V, E)$, *a positive integer* k.
Output: *a set* $S \subseteq V$ *such that* $|S| \leq k$ *and* $G \setminus S$ *is a cluster graph (disjoint union of cliques).*

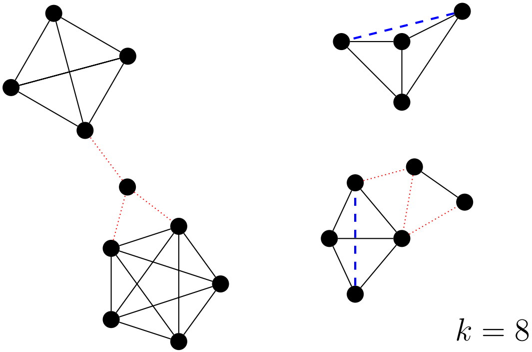
Cluster Editing



Problem (Cluster Editing, CVD)

Input: *an undirected graph* $G = (V, E)$*, a positive integer* k*.*
Output: *a set* $S \subseteq \binom{V}{2}$ *such that* $|S| \leq k$ *and* $(V, E \triangle S)$ *is a cluster graph (here* $\triangle$ *is a symmetric difference).*

Cluster Editing



Problem (Cluster Editing, CVD)

Input: *an undirected graph* $G = (V, E)$*, a positive integer* k.
Output: *a set* $S \subseteq \binom{V}{2}$ *such that* $|S| \leq k$ *and* $(V, E \triangle S)$ *is a cluster graph (here* $\triangle$ *is a symmetric difference).*

Motivation

Clustering objects based on pairwise similarities:

- computational biology,
- machine learning.

Motivation

Clustering objects based on pairwise similarities:

- computational biology,
- machine learning.

Cluster Vertex Deletion vs Cluster Editing:

- more instances are tractable for CVD (more powerful operation),
- errors in the similarity relation are likely to affect few vertices (contaminated samples etc.).

Motivation

Clustering objects based on pairwise similarities:

- computational biology,
- machine learning.

Cluster Vertex Deletion vs Cluster Editing:

- more instances are tractable for CVD (more powerful operation),
- errors in the similarity relation are likely to affect few vertices (contaminated samples etc.).

Theoretical motivation:

- deletion problem for a natural graph class.

Results

Previous results: (here $n = |V|$, $m = |E|$)

- simple $\mathcal{O}(3^k(n+m))$-time branching algorithm,
- an $\mathcal{O}(2^k k^9 + nm)$-time algorithm
  iterative compression (Hüffner et al., 2008)

Results

Previous results: (here $n = |V|$, $m = |E|$)

- simple $\mathcal{O}(3^k(n+m))$-time branching algorithm,
- an $\mathcal{O}(2^k k^9 + nm)$-time algorithm
  iterative compression (Hüffner et al., 2008)

Results for a more general 3-Hitting Set problem:

- $\mathcal{O}(2.18^k + n^3)$ algorithm (Fernau, 2010)
- $\mathcal{O}(k^4)$-size kernel (Abu-Khzam, 2010; preserves CVD)

Results

Previous results: (here $n = |V|$, $m = |E|$)

- simple $\mathcal{O}(3^k(n+m))$-time branching algorithm,
- an $\mathcal{O}(2^k k^9 + nm)$-time algorithm
  iterative compression (Hüffner et al., 2008)

Results for a more general 3-Hitting Set problem:

- $\mathcal{O}(2.18^k + n^3)$ algorithm (Fernau, 2010)
- $\mathcal{O}(k^4)$-size kernel (Abu-Khzam, 2010; preserves CVD)

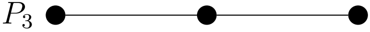
Our results:

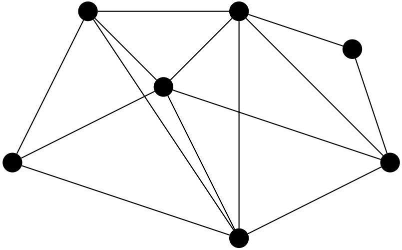
- an $\mathcal{O}(1.9102^k(n+m))$-time branching algorithm,
- $\mathcal{O}(1.9102^k k^4 + nm)$ time if combined with the kernel.

Cluster Vertex Deletion as hitting P_3's

Observation

A graph is a cluster graph if and only if it does not have P_3, the 3-vertex path, as an induced subgraph.

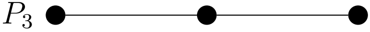


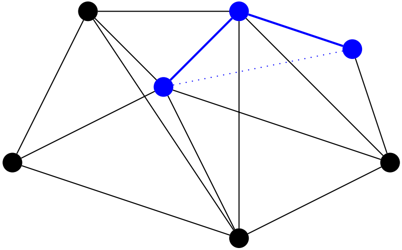


CLUSTER VERTEX DELETION as hitting P_3's

Observation

A graph is a cluster graph if and only if it does not have P_3, the 3-vertex path, as an induced subgraph.

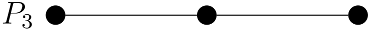


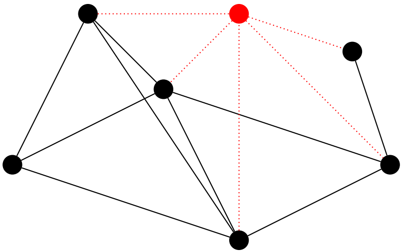


CLUSTER VERTEX DELETION as hitting P_3's

Observation

A graph is a cluster graph if and only if it does not have P_3, the 3-vertex path, as an induced subgraph.



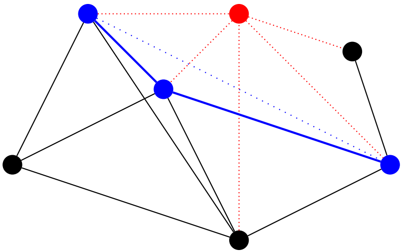


Cluster Vertex Deletion as hitting P_3's

Observation

A graph is a cluster graph if and only if it does not have P_3, the 3-vertex path, as an induced subgraph.

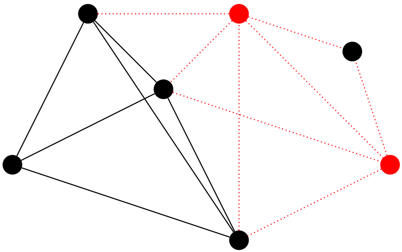
P_3 



Cluster Vertex Deletion as hitting P_3's

Observation

A graph is a cluster graph if and only if it does not have P_3, the 3-vertex path, as an induced subgraph.



Simple $\mathcal{O}(3^k(n+m))$-time branching algorithm

Corollary

X is a solution iff $X \cap P \neq \emptyset$ for any P such that $G[P]$ is isomorphic to P_3. (X must hit all P_3's).

Simple $\mathcal{O}(3^k(n+m))$-time branching algorithm

Corollary

X is a solution iff $X \cap P \neq \emptyset$ for any P such that $G[P]$ is isomorphic to P_3. (X must hit all P_3's).

Algorithm:

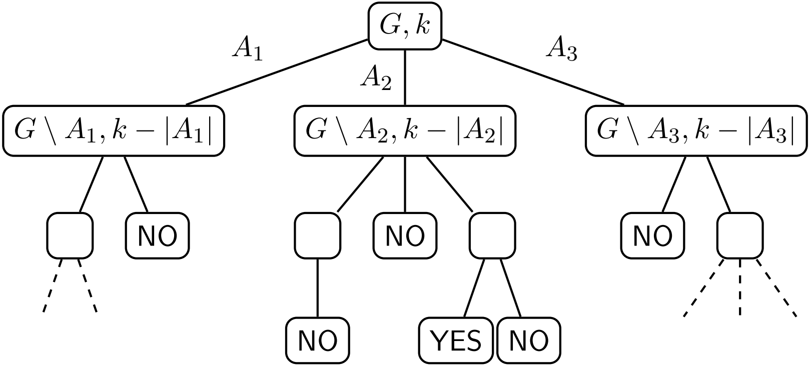
1. if G is a cluster graph, return $X = \emptyset$.
2. if $k = 0$, return NO.
3. find (v_1, v_2, v_3) inducing P_3.
4. for $i = 1, 2, 3$ recurse on $(G - v_i, k - 1)$ (adding v_i to X).

- $\mathcal{O}(3^k)$ calls in total, a single call can be implemented in $\mathcal{O}(n+m)$ time.

Branching algorithms

General framework for deletion problems:

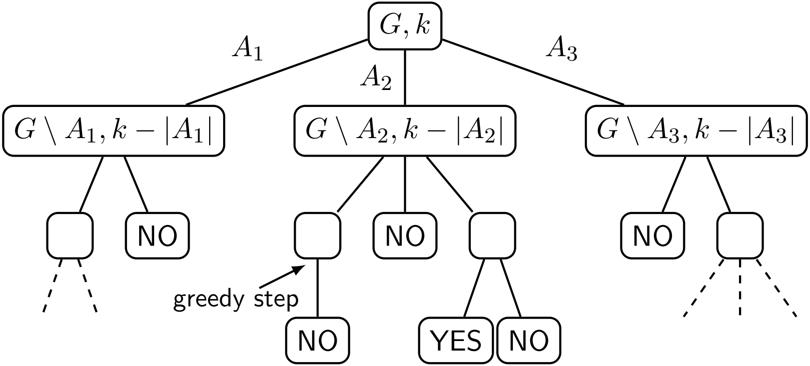
- in each step find a constant number of sets $(A_1, \ldots, A_\ell)$ such that there is a solution containing A_i for some i,
- recurse on $(G \setminus A_i, k - |A_i|)$ for each i.



Branching algorithms

General framework for deletion problems:

- in each step find a constant number of sets $(A_1, \ldots, A_\ell)$ such that there is a solution containing A_i for some i,
- recurse on $(G \setminus A_i, k - |A_i|)$ for each i.



Branching algorithms

General framework for deletion problems:

- in each step find a constant number of sets $(A_1, \ldots, A_\ell)$ such that there is a solution containing A_i for some i,
- recurse on $(G \setminus A_i, k - |A_i|)$ for each i.

Complexity analysis:

- any possible $(|A_1|, \ldots, |A_\ell|)$ is called a branching vector,
- number of recursive calls: $\mathcal{O}(c^k)$ for c such that $c^k \geq \sum_i c^{k-a_i}$ for any branching vector,
- the optimal choice of c: the largest positive root of $1 = \sum_i x^{-a_i}$ equations over all branching vectors,
- total time: $\mathcal{O}(c^k T(n))$, where $T(n)$ is the time needed for a single recursive call.

Improving the simple algorithm

Simple branching algorithm for (v, u, w) inducing P_3:

- remove one of the three vertices and recurse,
- possibly more than one of these vertices is ultimately deleted
  - single solution might be explored multiple times.

Different approach:

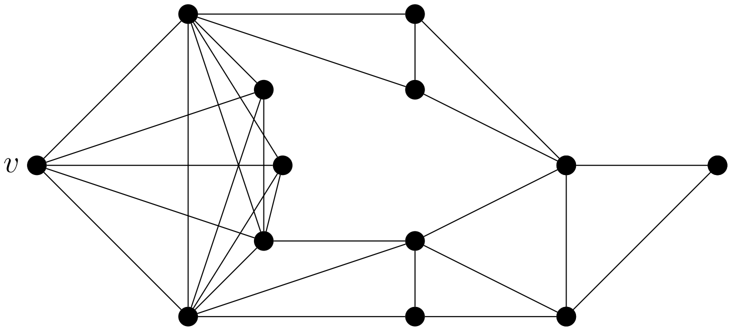
- choose a vertex v lying on some P_3
- consider two branches:
  - remove v (and recurse),
  - decide to leave v, and while v lies on P_3, branch on removing one of the other two vertices of the P_3.

Conflict graph H_v

If we decide to leave v, we still need to hit P_3's containing v.

Definition

Conflict graph H_v: $uw \in E(H_v)$ iff u, v and w induce P_3.

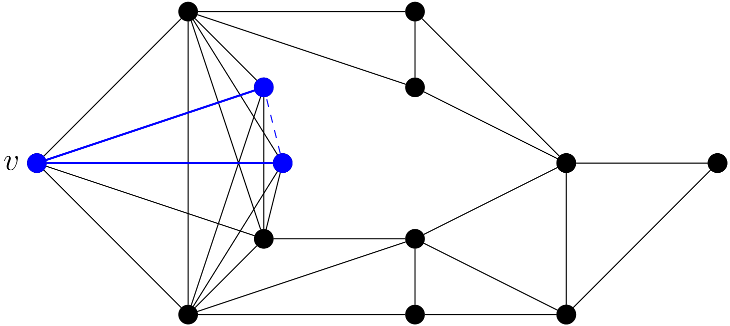


Conflict graph H_v

If we decide to leave v, we still need to hit P_3's containing v.

Definition

Conflict graph H_v: $uw \in E(H_v)$ iff u, v and w induce P_3.

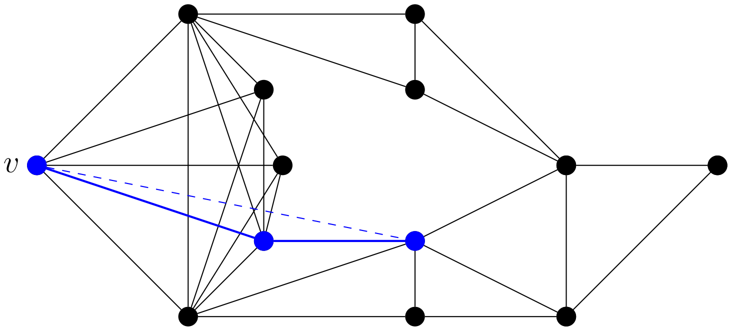


Conflict graph H_v

If we decide to leave v, we still need to hit P_3's containing v.

Definition

Conflict graph H_v: $uw \in E(H_v)$ iff u, v and w induce P_3.

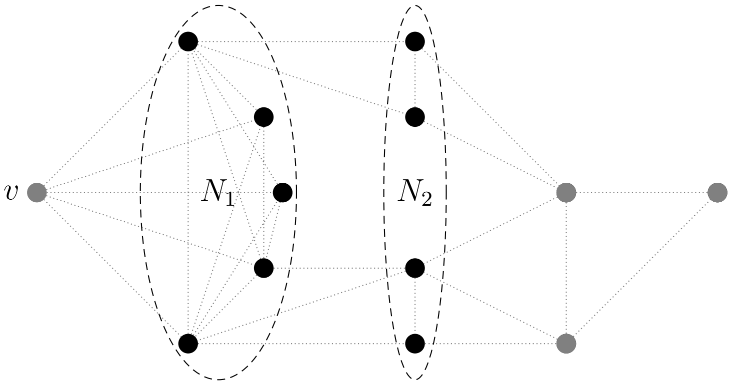


Conflict graph H_v

If we decide to leave v, we still need to hit P_3's containing v.

Definition

Conflict graph H_v: $uw \in E(H_v)$ iff u, v and w induce P_3.

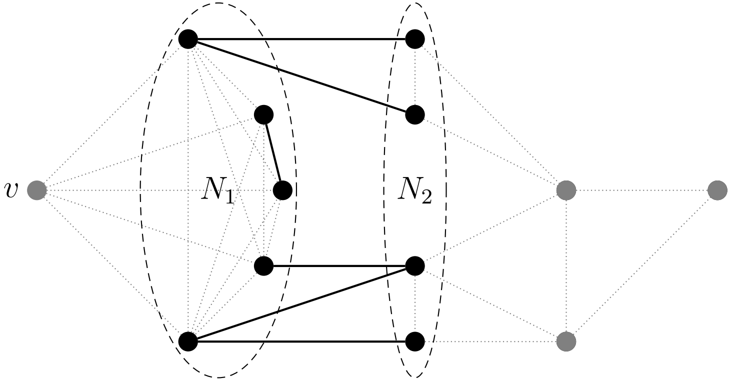


Conflict graph H_v

If we decide to leave v, we still need to hit P_3's containing v.

Definition

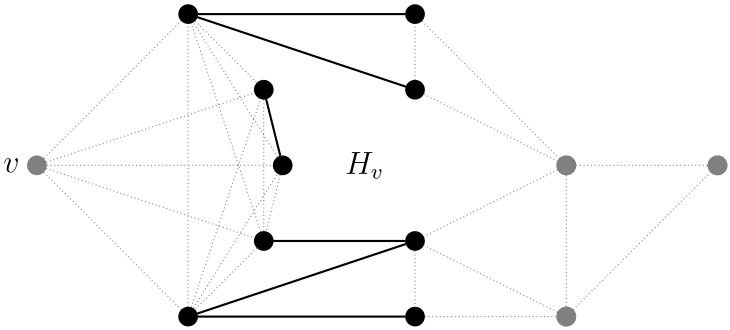
Conflict graph H_v: $uw \in E(H_v)$ iff u, v and w induce P_3.



Vertex covers in H_v

A vertex cover of a graph G is a set $X \subseteq V(G)$ such that $G \setminus X$ has no edges.

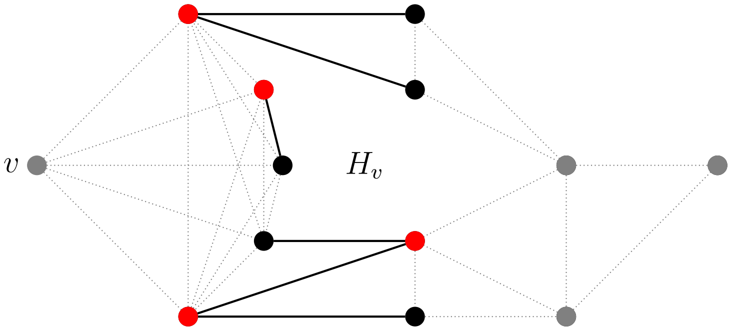
- any solution leaving v contains a vertex cover of H_v,
- after removing a vertex cover of H_v, the component of H_v is a clique.



Vertex covers in H_v

A vertex cover of a graph G is a set $X \subseteq V(G)$ such that $G \setminus X$ has no edges.

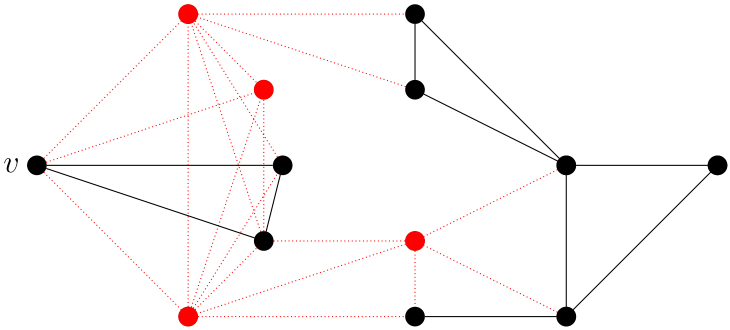
- any solution leaving v contains a vertex cover of H_v,
- after removing a vertex cover of H_v, the component of H_v is a clique.



Vertex covers in H_v

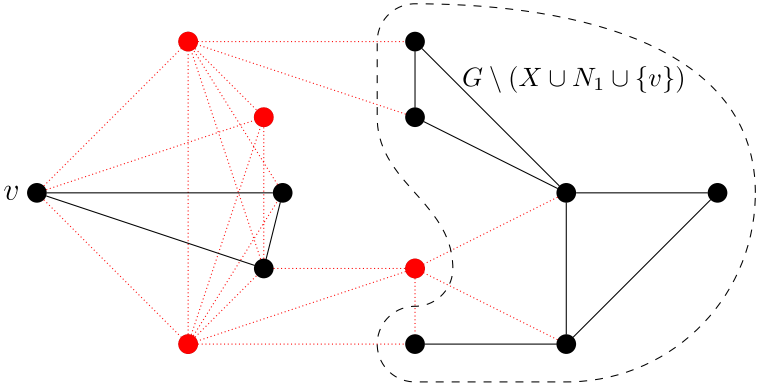
A vertex cover of a graph G is a set $X \subseteq V(G)$ such that $G \setminus X$ has no edges.

- any solution leaving v contains a vertex cover of H_v,
- after removing a vertex cover of H_v, the component of H_v is a clique.



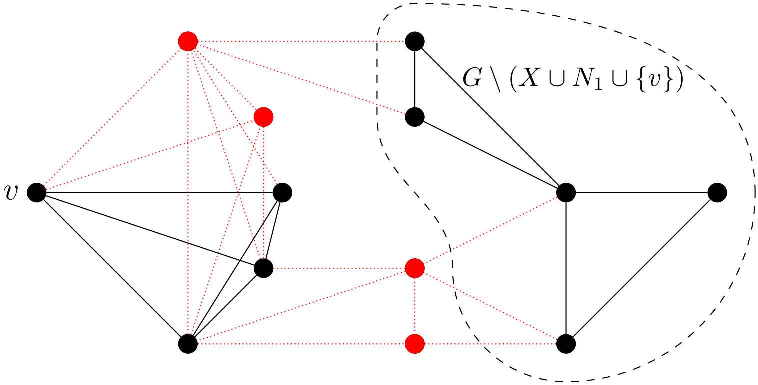
Greedy choices

- Let X, X' be vertex covers of H_v. We say that X *dominates* X' if $|X| \leq |X'|$ and $X \cap N_2 \supseteq X' \cap N_2$.
- If X dominates X', then we can replace X' with X in any solution containing X but not v.



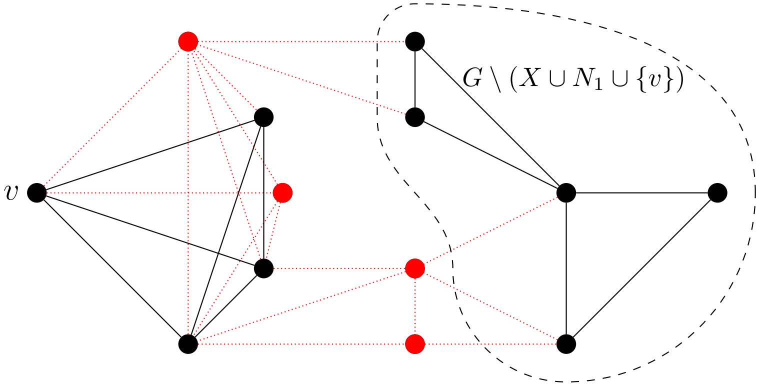
Greedy choices

- Let X, X' be vertex covers of H_v. We say that X *dominates* X' if $|X| \leq |X'|$ and $X \cap N_2 \supseteq X' \cap N_2$.
- If X dominates X', then we can replace X' with X in any solution containing X but not v.



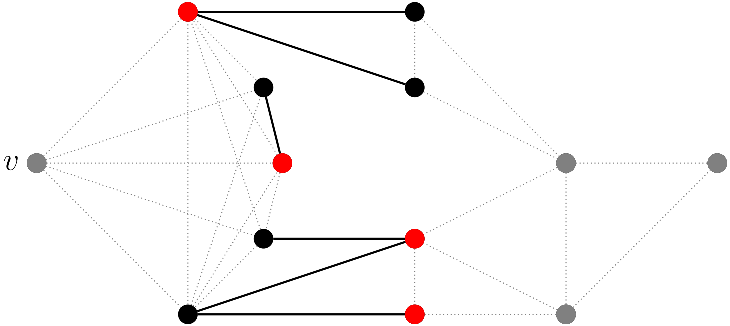
Greedy choices

- Let X, X' be vertex covers of H_v. We say that X *dominates* X' if $|X| \leq |X'|$ and $X \cap N_2 \supseteq X' \cap N_2$.
- If X dominates X', then we can replace X' with X in any solution containing X but not v.



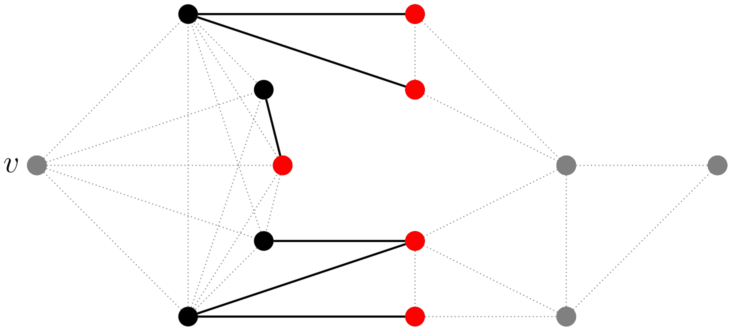
Greedy choices

- Let X, X' be vertex covers of H_v. We say that X *dominates* X' if $|X| \leq |X'|$ and $X \cap N_2 \supseteq X' \cap N_2$.
- If X dominates X', then we can replace X' with X in any solution containing X but not v.



Greedy choices

- Let X, X' be vertex covers of H_v. We say that X *dominates* X' if $|X| \leq |X'|$ and $X \cap N_2 \supseteq X' \cap N_2$.
- If X dominates X', then we can replace X' with X in any solution containing X but not v.



Branching on H_v

Summary of the "leave v" branch.

- Compute H_v.
- Generate several vertex covers of H_v, which in total dominate all vertex covers.
- Interpret steps of the (branching) algorithm generating covers as recursive calls for CVD.
- Branching vectors $(1, 2)$ ($c < 1.62$) and better.

Branching on H_v

Summary of the "leave v" branch.

- Compute H_v.
- Generate several vertex covers of H_v, which in total dominate all vertex covers.
- Interpret steps of the (branching) algorithm generating covers as recursive calls for CVD.
- Branching vectors $(1, 2)$ ($c < 1.62$) and better.

Issue:
With the "remove v" branch, the initial step may have branching vector $(1, 1, 2)$ (with $c = 1 + \sqrt{2}$).

Branching on H_v

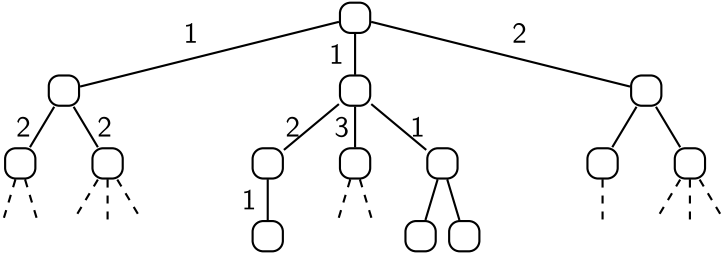
Summary of the "leave v" branch.

- Compute H_v.
- Generate several vertex covers of H_v, which in total dominate all vertex covers.
- Interpret steps of the (branching) algorithm generating covers as recursive calls for CVD.
- Branching vectors $(1, 2)$ ($c < 1.62$) and better.

Issue:
With the "remove v" branch, the initial step may have branching vector $(1, 1, 2)$ (with $c = 1 + \sqrt{2}$).

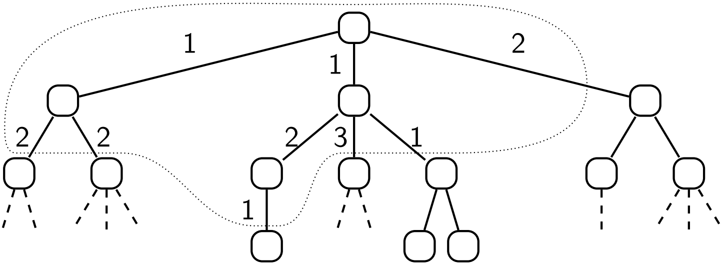
Intuitive solution:
If H_v has small vertex cover, there is structure to exploit.
Otherwise the subsequent steps "pay off" the poor initial one,

Formalizing the idea



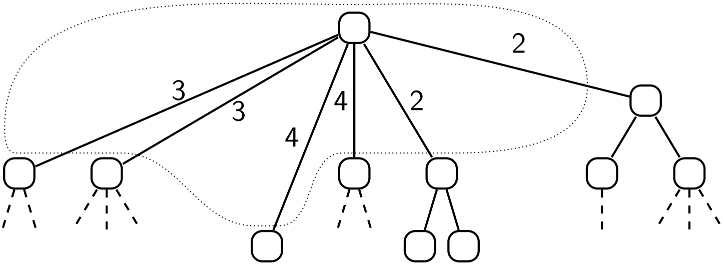
- Try to avoid the worst $(1,2)$ branching and describe the structure of the H_v when it cannot be avoided.
- Treat several initial recursive steps as a single 'virtual' one
  - removing a_i nodes can decrease vertex cover only by a_i.
- Many possible combinations of branching rules
  - automated case-analysis to check all possibilities.

Formalizing the idea



- Try to avoid the worst $(1, 2)$ branching and describe the structure of the H_v when it cannot be avoided.
- Treat several initial recursive steps as a single 'virtual' one
  - removing a_i nodes can decrease vertex cover only by a_i.
- Many possible combinations of branching rules
  - automated case-analysis to check all possibilities.

Formalizing the idea



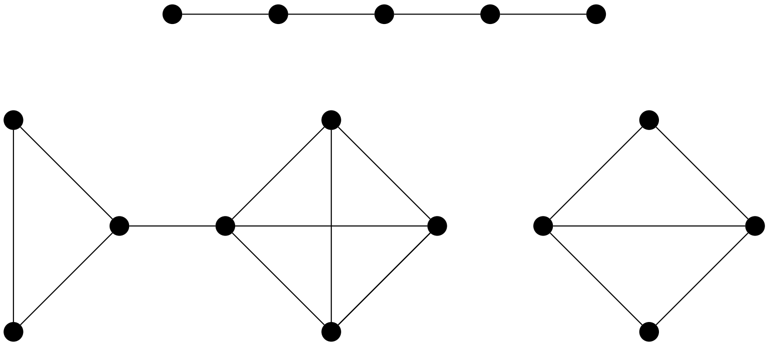
- Try to avoid the worst $(1, 2)$ branching and describe the structure of the H_v when it cannot be avoided.
- Treat several initial recursive steps as a single 'virtual' one
  - removing a_i nodes can decrease vertex cover only by a_i.
- Many possible combinations of branching rules
  - automated case-analysis to check all possibilities.

More greedy choices

Are "leave v" and "remove v" branches always necessary?

Observation

Let C be a connected component of v. If $C - v$ is a cluster graph, one can greedily remove v.

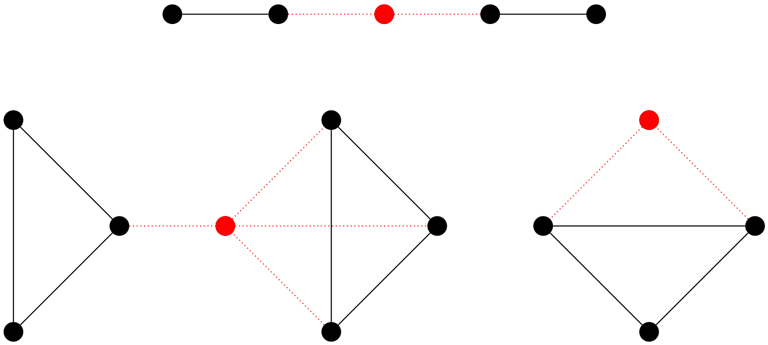


More greedy choices

Are "leave v" and "remove v" branches always necessary?

Observation

Let C be a connected component of v. If $C - v$ is a cluster graph, one can greedily remove v.

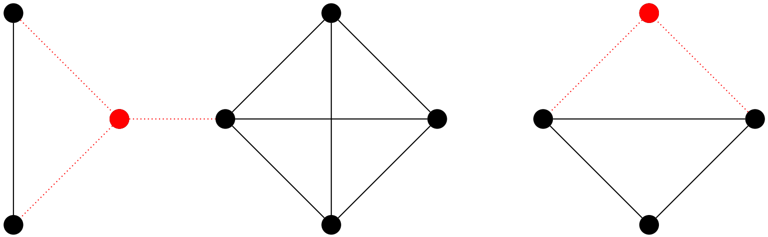


More greedy choices

Are "leave v" and "remove v" branches always necessary?

Observation

Let C be a connected component of v. If $C - v$ is a cluster graph, one can greedily remove v.

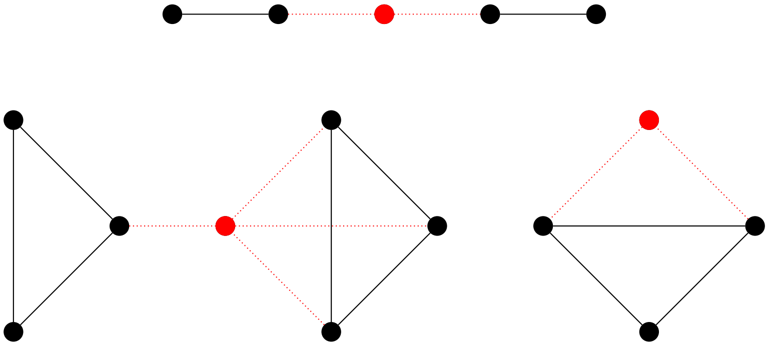


More greedy choices

Are "leave v" and "remove v" branches always necessary?

Observation

Let C be a connected component of v. If $C - v$ is a cluster graph, one can greedily remove v.

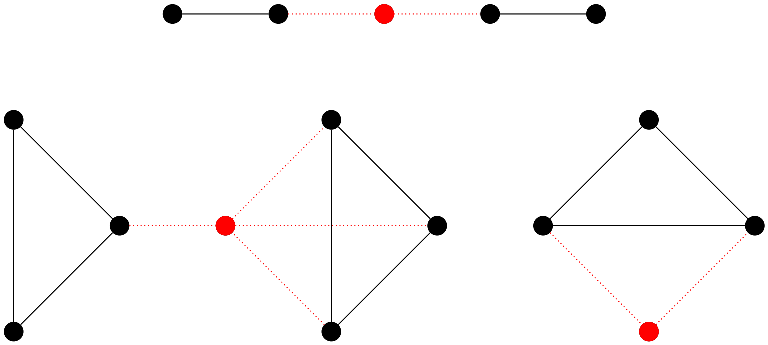


More greedy choices

Are "leave v" and "remove v" branches always necessary?

Observation

Let C be a connected component of v. If $C - v$ is a cluster graph, one can greedily remove v.

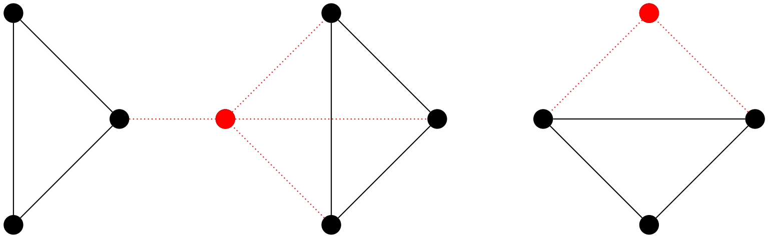


More greedy choices

Are "leave v" and "remove v" branches always necessary?

Observation

Let C be a connected component of v. If $C - v$ is a cluster graph, one can greedily remove v.



More greedy choices

Lemma

Suppose X is a vertex cover of H_v. Then there is a minimum solution S such that $v \notin S$ or $|X \setminus S| \geq 2$.

- If $|X| = 1$, greedily leave v and proceed to H_v.
- If $|X| = 2$ in the "remove v" branch proceed to H_x for some $x \in X$
  - if $C - v$ is not a cluster graph, then X intersect a P_3 disjoint with v,
  - the first branching after removing v is no worse than $(1, 2)$.

Algorithm summary

- If $VC(H_v) = 1$, we greedily leave v proceed immediately to branching H_v (branching vectors $(1, 2)$ and better)
- If $VC(H_v) = 2$, the "remove v" branch starts with a $(1, 2)$ or better branching, i.e. contributes to $(2, 3)$ in the branching vector of the 'virtual' initial step. Analysis of branching on H_v gives vectors, combined with $(2, 3)$, values $c < 1.9448$.
- If $VC(H_v) \geq 3$, analysis of branching in H_v, combined with (1) corresponding to removing v, gives vectors of values $c < 1.9338$.

Algorithm summary

- If $VC(H_v) = 1$, we greedily leave v proceed immediately to branching H_v (branching vectors $(1, 2)$ and better)
- If $VC(H_v) = 2$, the "remove v" branch starts with a $(1, 2)$ or better branching, i.e. contributes to $(2, 3)$ in the branching vector of the 'virtual' initial step. Analysis of branching on H_v gives vectors, combined with $(2, 3)$, values $c < 1.9448$.
- If $VC(H_v) \geq 3$, analysis of branching in H_v, combined with (1) corresponding to removing v, gives vectors of values $c < 1.9338$.

In the worst cases (if initally only $(1, 2)$ branching can be applied in H_v), v we can also greedily leave v.

- 'virtual' inital steps have vectors of value $c < 1.9102$.

Conclusions & open problems

Our results:

- $\mathcal{O}^*(1.9102^k)$-time branching algorithm.
- Single step implemented in linear time given G or $\bar{G}$:
  - $\mathcal{O}(1.9102^k(n+m))$ time for Cluster vertex deletion and Co-cluster vertex deletion.

Conclusions & open problems

Our results:

- $\mathcal{O}^*(1.9102^k)$-time branching algorithm.
- Single step implemented in linear time given G or $\bar{G}$:
  - $\mathcal{O}(1.9102^k(n+m))$ time for Cluster vertex deletion and Co-cluster vertex deletion.

Open problems:

- Does Cluster vertex deletion admit a small kernel (for example with $O(k)$ vertices)?
  - Cluster editing has $2k$-vertex kernel.
- Can the $\mathcal{O}^*(1.9102^k)$ time be improved?
  - more detailed analysis of the worst case could probably improve 1.9102 by a tiny amount.
- Weighted case (different prices for removing vertices).

Thank you

Thank you for your attention!