Sparsification of Constraint Satisfaction Problems with an application to q-Coloring

Astrid Pieterse

Based on papers at MFCS 2016 and IPEC 2017
joint work with Bart M. P. Jansen.

February 20, 2018

NET WORKS
THENETWORKCENTER.NL

TU/e Technische Universiteit Eindhoven University of Technology

Introduction

- Find exact algorithms for NP-hard problems
  - Exponential time
  - Speed-up possible?
- Preprocess the input instance
  - Aim to reduce the size
  - Polynomial time
- Many different NP-hard problems
  - Logic problems (CNF-SAT, NAE-SAT,...)
  - Graph problems (3-Coloring, Vertex Cover,...)
  - ...
- Study constraint satisfaction problems
  - All problems above can be written as a CSP instance

Introduction

- Find exact algorithms for NP-hard problems
  - Exponential time
  - Speed-up possible?
- Preprocess the input instance
  - Aim to reduce the size
  - Polynomial time
- Many different NP-hard problems
  - Logic problems (CNF-SAT, NAE-SAT,...)
  - Graph problems (3-Coloring, Vertex Cover,...)
  - ...
- Study constraint satisfaction problems
  - All problems above can be written as a CSP instance

Introduction

- Find exact algorithms for NP-hard problems
  - Exponential time
  - Speed-up possible?
- Preprocess the input instance
  - Aim to reduce the size
  - Polynomial time
- Many different NP-hard problems
  - Logic problems (CNF-SAT, NAE-SAT,...)
  - Graph problems (3-Coloring, Vertex Cover,...)
  - ...
- Study constraint satisfaction problems
  - All problems above can be written as a CSP instance

Introduction

- Find exact algorithms for NP-hard problems
  - Exponential time
  - Speed-up possible?
- Preprocess the input instance
  - Aim to reduce the size
  - Polynomial time
- Many different NP-hard problems
  - Logic problems (CNF-SAT, NAE-SAT,...)
  - Graph problems (3-Coloring, Vertex Cover,...)
  - ...
- Study constraint satisfaction problems
  - All problems above can be written as a CSP instance

Constraint Satisfaction Problems

CSP(Γ)

Input A number of constraints over a set of variables V

- Constraint $R(x_1, \ldots, x_k)$ applies relation $R \subseteq \{0,1\}^k$ to $x_1, \ldots, x_k \in V$

Question Does there exists an assignment $f : V \to \{0,1\}$, such that all constraints are satisfied?

Constraint language Γ

- Specifies the type of constraints
- Can only use constraints $R(x_1, \ldots, x_k)$ for $R \in \Gamma$

Example: 3-CNF-SAT

3-CNF-SAT:

$$F = \underbrace{(x \vee y \vee \neg z)}_{\text{clause (size 3)}} \wedge (x \vee \neg y \vee \neg w) \wedge \ldots$$

Equivalent to CSP(Γ) for $\Gamma = \{R_0, R_1, R_2, R_3\}$

- R_i represents clauses with i negations

Example: 3-CNF-SAT

3-CNF-SAT:

$$F = \underbrace{(x \vee y \vee \neg z)}_{\text{clause (size 3)}} \wedge (x \vee \neg y \vee \neg w) \wedge \ldots$$

Equivalent to CSP(Γ) for $\Gamma = \{R_0, R_1, R_2, R_3\}$

- R_i represents clauses with i negations

Example: 3-CNF-SAT

3-CNF-SAT:

$$F = \underbrace{(x \vee y \vee \neg z)}_{\text{clause (size 3)}} \wedge (x \vee \neg y \vee \neg w) \wedge \ldots$$

Equivalent to CSP(Γ) for $\Gamma = \{R_0, R_1, R_2, R_3\}$

- R_i represents clauses with i negations

R_0 is chosen such that $R_0(x, y, z)$ is equivalent to $(x \vee y \vee z)$

$$\begin{aligned} R_0 &= \{0,1\}^3 \setminus \{(0,0,0)\} \\ &= \{(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0),(1,1,1)\} \end{aligned}$$

Example: 3-CNF-SAT

3-CNF-SAT:

$$F = \underbrace{(x \vee y \vee \neg z)}_{\text{clause (size 3)}} \wedge (x \vee \neg y \vee \neg w) \wedge \ldots$$

Equivalent to CSP(Γ) for $\Gamma = \{R_0, R_1, R_2, R_3\}$

- R_i represents clauses with i negations

R_1 is chosen such that $R_1(x, y, z)$ is equivalent to $(\neg x \vee y \vee z)$

$$\begin{aligned} R_1 &= \{0,1\}^3 \setminus \{(1,0,0)\} \\ &= \{(0,0,0),(0,0,1),(0,1,0),(0,1,1),(1,0,1),(1,1,0),(1,1,1)\} \end{aligned}$$

Example: 3-CNF-SAT

3-CNF-SAT:

$$F = \underbrace{(x \vee y \vee \neg z)}_{\text{clause (size 3)}} \wedge (x \vee \neg y \vee \neg w) \wedge \ldots$$

Equivalent to CSP(Γ) for $\Gamma = \{R_0, R_1, R_2, R_3\}$

- R_i represents clauses with i negations

R_2 is chosen such that $R_2(x, y, z)$ is equivalent to $(\neg x \vee \neg y \vee z)$

$$\begin{aligned} R_2 &= \{0,1\}^3 \setminus \{(1,1,0)\} \\ &= \{(0,0,0),(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,1)\} \end{aligned}$$

Example: 3-CNF-SAT

3-CNF-SAT:

$$F = \underbrace{(x \vee y \vee \neg z)}_{\text{clause (size 3)}} \wedge (x \vee \neg y \vee \neg w) \wedge \ldots$$

Equivalent to CSP(Γ) for $\Gamma = \{R_0, R_1, R_2, R_3\}$

- R_i represents clauses with i negations

R_3 is chosen such that $R_3(x, y, z)$ is equivalent to $(\neg x \vee \neg y \vee \neg z)$

$$\begin{aligned} R_3 &= \{0,1\}^3 \setminus \{(1,1,1)\} \\ &= \{(0,0,0),(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0)\} \end{aligned}$$

Example: 3-CNF-SAT

3-CNF-SAT:

$$F = \underbrace{(x \vee y \vee \neg z)}_{\text{clause (size 3)}} \wedge (x \vee \neg y \vee \neg w) \wedge \ldots$$

Equivalent to CSP(Γ) for $\Gamma = \{R_0, R_1, R_2, R_3\}$

- R_i represents clauses with i negations

$$R_0 = \{0,1\}^3 \setminus \{(0,0,0)\}$$
$$R_1 = \{0,1\}^3 \setminus \{(1,0,0)\}$$
$$R_2 = \{0,1\}^3 \setminus \{(1,1,0)\}$$
$$R_3 = \{0,1\}^3 \setminus \{(1,1,1)\}$$

Sparsification

Constraint satisfaction problems are (often) hard

- Preprocess the input
- Reduce the number of constraints
  - Find redundant constraints
- Worst-case bound on the number of remaining constraints
  - As a function of the number of variables
- Maintaining the answer
  - Keeping all satisfying assignments?
  - Not changing satisfiability
- Efficiently!

Sparsification

Constraint satisfaction problems are (often) hard

- Preprocess the input
- Reduce the number of constraints
  - Find redundant constraints
- Worst-case bound on the number of remaining constraints
  - As a function of the number of variables
- Maintaining the answer
  - Keeping all satisfying assignments?
  - Not changing satisfiability
- Efficiently!

Sparsification

Constraint satisfaction problems are (often) hard

- Preprocess the input
- Reduce the number of constraints
  - Find redundant constraints
- Worst-case bound on the number of remaining constraints
  - As a function of the number of variables
- Maintaining the answer
  - Keeping all satisfying assignments?
  - Not changing satisfiability
- Efficiently!

Sparsification

Constraint satisfaction problems are (often) hard

- Preprocess the input
- Reduce the number of constraints
  - Find redundant constraints
- Worst-case bound on the number of remaining constraints
  - As a function of the number of variables
- Maintaining the answer
  - Keeping all satisfying assignments?
  - Not changing satisfiability
- Efficiently!

Sparsification

Constraint satisfaction problems are (often) hard

- Preprocess the input
- Reduce the number of constraints
  - Find redundant constraints
- Worst-case bound on the number of remaining constraints
  - As a function of the number of variables
- Maintaining the answer
  - Keeping all satisfying assignments?
  - Not changing satisfiability
- Efficiently!

An easy sparsification

d-CNF-SAT:

$$F = \underbrace{(x \vee y \vee \neg z)}_{\text{clause (size } d\text{)}} \wedge (x \vee \neg y \vee \neg w) \wedge \ldots$$

Sparsification procedure:
Remove duplicate clauses

Size

- n variables ($2n$ possible literals), d literals per clause
- At most $O((2n)^d)$ different possible clauses
- $O(n^d)$ for d constant

An easy sparsification

d-CNF-SAT:

$$F = \underbrace{(x \vee y \vee \neg z)}_{\text{clause (size } d\text{)}} \wedge (x \vee \neg y \vee \neg w) \wedge \ldots$$

Sparsification procedure:

Remove duplicate clauses

Size

- n variables ($2n$ possible literals), d literals per clause
- At most $O((2n)^d)$ different possible clauses
- $O(n^d)$ for d constant

An easy sparsification

d-CNF-SAT:

$$F = \underbrace{(x \vee y \vee \neg z)}_{\text{clause (size } d\text{)}} \wedge (x \vee \neg y \vee \neg w) \wedge \ldots$$

Sparsification procedure:

Remove duplicate clauses

Size

- n variables ($2n$ possible literals), d literals per clause
- At most $O((2n)^d)$ different possible clauses
- $O(n^d)$ for d constant

Known results

d-CNF-SAT:

$$F = \underbrace{(x \vee y \vee \neg z)}_{\text{clause (size } d)} \wedge (x \vee \neg y \vee \neg w) \wedge \ldots$$

d-NAE-SAT:

$$F = (x, y, z) \wedge (x, \neg y, \neg w) \wedge \ldots$$

Known results

d-CNF-SAT:

$$F = \underbrace{(x \vee y \vee \neg z)}_{\text{clause (size } d)} \wedge (x \vee \neg y \vee \neg w) \wedge \ldots$$

d-NAE-SAT:

$$F = (x, y, z) \wedge (x, \neg y, \neg w) \wedge \ldots$$

Sparsification

d-CNF-SAT	$O(n^d)$
d-NAE-SAT	$O(n^{d-1})$

no nontrivial sparsification [Dell,van Melkebeek]

- Can prove certain lower bounds

Known results

d-CNF-SAT:

$$F = \underbrace{(x \vee y \vee \neg z)}_{\text{clause (size } d)} \wedge (x \vee \neg y \vee \neg w) \wedge \ldots$$

d-NAE-SAT:

$$F = (x, y, z) \wedge (x, \neg y, \neg w) \wedge \ldots$$

Sparsification

d-CNF-SAT	$O(n^d)$	no nontrivial sparsification [Dell,van Melkebeek]
d-NAE-SAT	$O(n^{d-1})$	

- Can prove certain lower bounds

Lower bounds

Prove that no "better" sparsification exists?

- Needs assumptions, like P $\neq$ NP
  - Stronger assumption: NP $\not\subseteq$ coNP/poly

Under these assumptions, we can prove bounds of the type

There is no $O(n^{d-\varepsilon})$ size sparsification for d-CNF-SAT, for any $\varepsilon > 0$

Lower bounds

Prove that no "better" sparsification exists?

- Needs assumptions, like P $\neq$ NP
  - Stronger assumption: NP $\not\subseteq$ coNP/poly

Under these assumptions, we can prove bounds of the type

There is no $O(n^{d-\varepsilon})$ size sparsification for d-CNF-SAT, for any $\varepsilon > 0$

Lower bounds

Prove that no "better" sparsification exists?

- Needs assumptions, like P $\neq$ NP
  - Stronger assumption: NP $\not\subseteq$ coNP/poly

Under these assumptions, we can prove bounds of the type

There is no $O(n^{d-\varepsilon})$ size sparsification for d-CNF-SAT,
for any $\varepsilon > 0$

Dichotomy theorem for CSPs

Schaefer's dichotomy theorem
Depending on properties of the relations in Γ, CSP(Γ) is

Polynomial-time solvable

2-SAT	HORN-SAT
DUAL-HORN-SAT	XOR-SAT
...	...

or

NP-hard

3-CNF-SAT	3-NAE-SAT
EXACT-SAT	...

Sparsification for CSPs: Goal

Find a classification for sparsifiability

Optimal sparsification bound: $O(n)$ constraints

EXACT-SAT (upcoming), . . . ?

Optimal sparsification bound: $O(n^2)$ constraints

. . . ?

⋮

Optimal sparsification bound: $O(n^d)$ constraints

d-CNF-SAT, $(d+1)$-NAE-SAT, . . . ?

⋮

Exact satisfiability

Exact SAT

Input A formula in the following form, consisting of clauses, each consisting of a number of literals.

$$\underbrace{(\neg x, \neg y)}_{\text{clause}} \wedge (\neg y, z) \wedge (x, z)$$

Question Does there exists a boolean assignment, such that each clause contains exactly one true literal?

Sparsification for Exact SAT: Example

Example

Let 0 := false, 1 := true

$$(\neg x, \neg y) \wedge (\neg y, z) \wedge (x, z)$$

Satisfied by $x = 1, y = 0, z = 0$

Sparsification for Exact SAT: Example

Example

Let 0 := false, 1 := true

$$(\neg x, \neg y) \wedge (\neg y, z) \wedge (x, z)$$

Satisfied by $x = 1, y = 0, z = 0$

$$\begin{array}{lcll} (\neg x, \neg y) \wedge & & (1 - x) + (1 - y) = 1 & \\ (\neg y, z) \wedge & \Leftrightarrow & (1 - y) + z = 1 & \Leftrightarrow \quad z - y = 0 \\ (x, z) & & x + z = 1 & \end{array}$$

is satisfied

Sparsification for Exact SAT: Example

Example

Let 0 := false, 1 := true

$$(\neg x, \neg y) \wedge (\neg y, z) \wedge (x, z)$$

Satisfied by $x = 1, y = 0, z = 0$

$$\begin{array}{lcll} (\neg x, \neg y) \wedge & & (1 - x) + (1 - y) = 1 & \\ (\neg y, z) \wedge & \Leftrightarrow & (1 - y) + z = 1 & \Leftrightarrow \quad z - y = 0 \\ (x, z) & & x + z = 1 & \\ \text{is satisfied} & & & \end{array}$$

Sparsification for Exact SAT: Example

Example

Let 0 := false, 1 := true

$$(\neg x, \neg y) \wedge (\neg y, z) \wedge (x, z)$$

Satisfied by $x = 1, y = 0, z = 0$

$$\begin{array}{lcll} (\neg x, \neg y) \wedge & & (1 - x) + (1 - y) = 1 & \\ (\neg y, z) \wedge & \Leftrightarrow & (1 - y) + z = 1 & \Leftrightarrow \quad z - y = 0 \\ (x, z) & & x + z = 1 & \\ \text{is satisfied} & & & \end{array}$$

Sparsification for Exact SAT: Example

Example

Let 0 := false, 1 := true

$$(\neg x, \neg y) \wedge (\neg y, z) \wedge (x, z)$$

Satisfied by $x = 1, y = 0, z = 0$

$$\begin{array}{lcll} (\neg x, \neg y) \wedge & & (1 - x) + (1 - y) = 1 & \\ (\neg y, z) \wedge & \Leftrightarrow & (1 - y) + z = 1 & \Leftrightarrow \quad \dfrac{x + y = 1}{z - y = 0}+ \\ (x, z) & & x + z = 1 & \dfrac{}{x + z = 1}+ \\ \text{is satisfied} & & & \end{array}$$

Sparsification for Exact SAT: Example

Example

Let 0 := false, 1 := true

$$(\neg x, \neg y) \wedge (\neg y, z) \wedge (x, z)$$

Satisfied by $x = 1, y = 0, z = 0$

$$\begin{array}{lcll} (\neg x, \neg y) \wedge & & (1 - x) + (1 - y) = 1 & \\ (\neg y, z) \wedge & \Leftrightarrow & (1 - y) + z = 1 & \Leftrightarrow \\ (x, z) & & x + z = 1 & \end{array} \quad \begin{array}{l} x + y = 1 \\ z - y = 0 \\ \hline x + z = 1 \end{array} +$$

is satisfied

- (x, z) is always satisfied when the other clauses are
- It is redundant

Sparsification for Exact SAT

Can we do this in general?

Sparsification for Exact SAT

Can we do this in general?

$$\begin{array}{l} x+y=1 \\ z-y=0 \\ x+z=1 \end{array} \quad \Leftrightarrow \quad \begin{array}{r} x+y-1=0 \\ z-y=0 \\ x+z-1=0 \end{array} \quad \Leftrightarrow \quad \begin{pmatrix} 1 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \\ 1 & 0 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Sparsification for Exact SAT

Can we do this in general?

$$\begin{array}{l} x+y=1 \\ z-y=0 \\ x+z=1 \end{array} \quad \Leftrightarrow \quad \begin{array}{r} x+y-1=0 \\ z-y=0 \\ x+z-1=0 \end{array} \quad \Leftrightarrow \quad \begin{pmatrix} 1 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \\ 1 & 0 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Find redundant constraints by looking at the matrix

$$\begin{pmatrix} 1 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \\ 1 & 0 & 1 & -1 \end{pmatrix}$$

Sparsification for Exact SAT

Can we do this in general?

$$\begin{array}{l} x+y=1 \\ z-y=0 \\ x+z=1 \end{array} \quad \Leftrightarrow \quad \begin{array}{r} x+y-1=0 \\ z-y=0 \\ x+z-1=0 \end{array} \quad \Leftrightarrow \quad \begin{pmatrix} 1 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \\ 1 & 0 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Find redundant constraints by looking at the matrix

$$\begin{pmatrix} 1 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \\ 1 & 0 & 1 & -1 \end{pmatrix}$$

Find a basis of the row space

$$\begin{pmatrix} \boxed{\begin{matrix} 1 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \end{matrix}} \\ 1 & 0 & 1 & -1 \end{pmatrix}$$

Sparsification for Exact SAT

- Write down linear equation for each constraint
- Problem asks to find 0/1-solution to linear system
- Compute basis of the row space of the matrix
- Keep constraints corresponding to rows in the basis
  - Remove others

$$
\begin{aligned}
x + y - 1 &= 0 \\
z - y &= 0 \\
x + z - 1 &= 0 \\
&\dots\ \dots
\end{aligned}
$$

Sparsification for Exact SAT

- Write down linear equation for each constraint
- Problem asks to find 0/1-solution to linear system
- Compute basis of the row space of the matrix
- Keep constraints corresponding to rows in the basis
  - Remove others

$$\begin{pmatrix} 1 & 1 & 0 & \dots & -1 \\ 0 & -1 & 1 & \dots & 0 \\ 1 & 0 & 1 & \dots & -1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Sparsification for Exact SAT

- Write down linear equation for each constraint
- Problem asks to find 0/1-solution to linear system
- Compute basis of the row space of the matrix
- Keep constraints corresponding to rows in the basis
  - Remove others

$$\begin{pmatrix} 1 & 1 & 0 & \dots & -1 \\ 0 & -1 & 1 & \dots & 0 \\ 1 & 0 & 1 & \dots & -1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \end{pmatrix}$$

Sparsification for Exact SAT

- Write down linear equation for each constraint
- Problem asks to find 0/1-solution to linear system
- Compute basis of the row space of the matrix
- Keep constraints corresponding to rows in the basis
  - Remove others

$$\begin{pmatrix} 1 & 1 & 0 & \dots & -1 \\ 0 & -1 & 1 & \dots & 0 \\ 1 & 0 & 1 & \dots & -1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \end{pmatrix}$$

Sparsification for Exact SAT

- Write down linear equation for each constraint
- Problem asks to find 0/1-solution to linear system
- Compute basis of the row space of the matrix
- Keep constraints corresponding to rows in the basis
  - Remove others

Remaining constraints: (x, y) and $(\neg y, z)$

Sparsification for Exact SAT

Correctness

Removed clauses are implied by remaining ones

Size

- The matrix has $n + 1$ columns
  - One for each variable plus one for the constant
- Dimension of row space equals dimension of column space
  - Bounded by #columns
- At most $n + 1$ remaining constraints

Sparsification for Exact SAT

Theorem

There is a polynomial-time algorithm, that given an instance F of Exact-sat, produces instance F' of Exact-sat such that

- Any boolean assignment satisfies F' if and only if it satisfies F
- The number of clauses in F' is $O(n)$

The set of clauses of F' is a subset of those of F

Generalize the Exact-SAT sparsification

c-Polynomial Root CSP

Definition

Input	A list of polynomial equalities of the form $f_i(x_{i_1}, \ldots, x_{i_k}) = 0$, where f_i has degree at most c.
Question	Does there exist a 0/1-assignment to the variables, such that all equalities are satisfied?

- Exact-sat is 1-Polynomial Root CSP

c-Polynomial Root CSP

Definition

Input	A list of polynomial equalities of the form $f_i(x_{i_1}, \ldots, x_{i_k}) = 0$, where f_i has degree at most c.
Question	Does there exist a 0/1-assignment to the variables, such that all equalities are satisfied?

- Exact-sat is 1-Polynomial Root CSP

Sparsification for c-Polynomial Root CSP

Find redundant constraints in a similar way as for Exact-SAT

- Each constraint is given by a degree-c polynomial

$$f(x, y, z) = 0$$

- Find a subset S of relevant equalities
  - Equalities not in S are combinations of those in S
  - Adapt previous method to higher-degree polynomials!
- Remove redundant constraints

Sparsification for c-Polynomial Root CSP

Find redundant constraints in a similar way as for Exact-sat

- Each constraint is given by a degree-c polynomial

$$f(x, y, z) = 0$$

- Find a subset S of relevant equalities
  - Equalities not in S are combinations of those in S
  - Adapt previous method to higher-degree polynomials!
- Remove redundant constraints

Sparsification for c-Polynomial Root CSP

Find redundant constraints in a similar way as for Exact-SAT

- Each constraint is given by a degree-c polynomial

$$f(x, y, z) = 0$$

- Find a subset S of relevant equalities
  - Equalities not in S are combinations of those in S
  - Adapt previous method to higher-degree polynomials!
- Remove redundant constraints

Finding redundant polynomial equalities

Create a matrix A

- Column for each multilinear monomial of degree at most c
  - Example: degree-2 polynomials over variables x, y, z
- Row for each constraint. Consider the constraints
  - $f(x, y) = xy + 2y + 3$
  - $g(z, y) = zy + y$

Find a basis of the row space of A

- Keep corresponding constraints

Finding redundant polynomial equalities

Create a matrix A

- Column for each multilinear monomial of degree at most c
  - Example: degree-2 polynomials over variables x, y, z
- Row for each constraint. Consider the constraints
  - $f(x, y) = xy + 2y + 3$
  - $g(z, y) = zy + y$

Find a basis of the row space of A

- Keep corresponding constraints

Finding redundant polynomial equalities

Create a matrix A

- Column for each multilinear monomial of degree at most c
  - Example: degree-2 polynomials over variables x, y, z
- Row for each constraint. Consider the constraints
  - $f(x, y) = xy + 2y + 3$
  - $g(z, y) = zy + y$

Find a basis of the row space of A

- Keep corresponding constraints

$$A = \begin{array}{c} \begin{matrix} xy & xz & yz & x & y & z & 1 \end{matrix} \\ \begin{pmatrix} \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix} \end{array}$$

Finding redundant polynomial equalities

Create a matrix A

- Column for each multilinear monomial of degree at most c
  - Example: degree-2 polynomials over variables x, y, z
- Row for each constraint. Consider the constraints
  - $f(x, y) = xy + 2y + 3$
  - $g(z, y) = zy + y$

Find a basis of the row space of A

- Keep corresponding constraints

$$A = \begin{array}{c} \begin{matrix} xy & xz & yz & x & y & z & 1 \end{matrix} \\ \begin{pmatrix} \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix} \end{array}$$

Finding redundant polynomial equalities

Create a matrix A

- Column for each multilinear monomial of degree at most c
  - Example: degree-2 polynomials over variables x, y, z
- Row for each constraint. Consider the constraints
  - $f(x, y) = xy + 2y + 3$
  - $g(z, y) = zy + y$

Find a basis of the row space of A

- Keep corresponding constraints

$$A = \begin{array}{c} \begin{matrix} xy & xz & yz & x & y & z & 1 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 2 & 0 & 3 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix} \end{array} \begin{matrix} \\ f \end{matrix}$$

Finding redundant polynomial equalities

Create a matrix A

- Column for each multilinear monomial of degree at most c
  - Example: degree-2 polynomials over variables x, y, z
- Row for each constraint. Consider the constraints
  - $f(x, y) = xy + 2y + 3$
  - $g(z, y) = zy + y$

Find a basis of the row space of A

- Keep corresponding constraints

$$A = \begin{array}{c} \begin{matrix} xy & xz & yz & x & y & z & 1 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 2 & 0 & 3 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix} \end{array} \begin{matrix} \\ f \end{matrix}$$

Finding redundant polynomial equalities

Create a matrix A

- Column for each multilinear monomial of degree at most c
  - Example: degree-2 polynomials over variables x, y, z
- Row for each constraint. Consider the constraints
  - $f(x, y) = xy + 2y + 3$
  - $g(z, y) = zy + y$

Find a basis of the row space of A

- Keep corresponding constraints

$$A = \begin{array}{c} \begin{matrix} xy & xz & yz & x & y & z & 1 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 2 & 0 & 3 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix} \end{array} \begin{matrix} f \\ g \end{matrix}$$

Finding redundant polynomial equalities

Create a matrix A

- Column for each multilinear monomial of degree at most c
  - Example: degree-2 polynomials over variables x, y, z
- Row for each constraint. Consider the constraints
  - $f(x, y) = xy + 2y + 3$
  - $g(z, y) = zy + y$

Find a basis of the row space of A

- Keep corresponding constraints

$$A = \begin{array}{c} \begin{matrix} xy & xz & yz & x & y & z & 1 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 2 & 0 & 3 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix} \end{array} \begin{matrix} f \\ g \\ \end{matrix}$$

Sparsification for c-Polynomial Root CSP

Correctness

Sparsified instance satisfiable $\Leftrightarrow$ Original instance satisfiable

- Let τ be a satisfying assignment for the sparsified instance
- Suppose $f_i(x_{i_1}, \ldots, x_{i_k}) = 0$ was removed
- Then there exist $\alpha_j \in \mathbb{R}$ such that

$$f_i(\tau(x_{i_1}), \ldots, \tau(x_{i_k})) = \sum_{j \in B} \alpha_j \cdot f_j(\tau(x_{j_1}), \ldots, \tau(x_{j_k}))$$

Since $f_j(x_{j_1}, \ldots, x_{j_k}) = 0$ was in the sparsified instance

$$f_j(\tau(x_{j_1}), \ldots, \tau(x_{j_k})) = 0 \ \forall j \in B$$

Thereby

$$f_i(\tau(x_{i_1}), \ldots, \tau(x_{i_k})) = 0$$

Sparsification for c-Polynomial Root CSP

Correctness

Sparsified instance satisfiable $\Leftrightarrow$ Original instance satisfiable

- Let τ be a satisfying assignment for the sparsified instance
- Suppose $f_i(x_{i_1}, \ldots, x_{i_k}) = 0$ was removed
- Then there exist $\alpha_j \in \mathbb{R}$ such that

$$f_i(\tau(x_{i_1}), \ldots, \tau(x_{i_k})) = \sum_{j \in B} \alpha_j \cdot f_j(\tau(x_{j_1}), \ldots, \tau(x_{j_k}))$$

Since $f_j(x_{j_1}, \ldots, x_{j_k}) = 0$ was in the sparsified instance

$$f_j(\tau(x_{j_1}), \ldots, \tau(x_{j_k})) = 0 \ \forall j \in B$$

Thereby

$$f_i(\tau(x_{i_1}), \ldots, \tau(x_{i_k})) = 0$$

Sparsification for c-Polynomial Root CSP

Correctness

Sparsified instance satisfiable $\Leftrightarrow$ Original instance satisfiable

- Let τ be a satisfying assignment for the sparsified instance
- Suppose $f_i(x_{i_1}, \ldots, x_{i_k}) = 0$ was removed
- Then there exist $\alpha_j \in \mathbb{R}$ such that

$$f_i(\tau(x_{i_1}), \ldots, \tau(x_{i_k})) = \sum_{j \in B} \alpha_j \cdot f_j(\tau(x_{j_1}), \ldots, \tau(x_{j_k}))$$

Since $f_j(x_{j_1}, \ldots, x_{j_k}) = 0$ was in the sparsified instance

$$f_j(\tau(x_{j_1}), \ldots, \tau(x_{j_k})) = 0 \ \forall j \in B$$

Thereby

$$f_i(\tau(x_{i_1}), \ldots, \tau(x_{i_k})) = 0$$

Sparsification for c-Polynomial Root CSP

Correctness

Sparsified instance satisfiable $\Leftrightarrow$ Original instance satisfiable ✓

Size

Number of remaining constraints $\leq$ #columns

- Column for each multilinear monomial of degree $\leq c$
- $\binom{n}{i}$ multilinear monomials of degree exactly i
- $\sum_{i=0}^{c} \binom{n}{i} = O(n^c)$ columns

Sparsification for c-Polynomial Root CSP

Theorem

There is a polynomial-time algorithm, that given an instance F of c-Polynomial-root CSP, produces instance F' of c-Polynomial-root CSP such that

- Any boolean assignment satisfies F' if and only if it satisfies F
- The number of constraints in F' is $O(n^c)$

c-Polynomial-root CSP has a $O(n^c)$ size sparsification

Relation to earlier results

Problem	**Special case of**	**Bound**
d-CNF-SAT	d-Polynomial-root CSP	$O(n^d)$
d-NAE-SAT	$(d-1)$-Polynomial-root CSP	$O(n^{d-1})$
d-Exact-SAT	1-Polynomial-root CSP	$O(n)$

Clause can be replaced by degree-c polynomial equality

- We have seen: d-Exact-SAT
- Next: d-NAE-SAT

(Monotone) d-NAE-SAT

Find polynomial f of degree $d-1$ such that

$$f(x_1, \ldots, x_d) = 0 \Leftrightarrow \text{clause } (x_1, \ldots, x_d) \text{ is } \textsc{nae}\text{-satisfied}$$

Clause $(x_1, \ldots, x_d)$ is $\textsc{nae}$-satisfied iff

$$x_1 + x_2 + \ldots + x_d \in \{1, 2, \ldots, d-1\}$$

Let $g(x) := (x-1) \cdot (x-2) \cdots (x-(d-1))$

- $g(x) = 0$ for $x \in \{1, 2, \ldots, d-1\}$
- $g(0) \neq 0$ and $g(d) \neq 0$

$g(x_1 + x_2 + \ldots + x_d)$ is the required polynomial

- Can handle negations as well
  - Use $(1 - x_i)$ for $\neg x_i$

(Monotone) d-NAE-SAT

Find polynomial f of degree $d-1$ such that

$$f(x_1, \ldots, x_d) = 0 \Leftrightarrow \text{clause } (x_1, \ldots, x_d) \text{ is } \textsc{nae}\text{-satisfied}$$

Clause $(x_1, \ldots, x_d)$ is $\textsc{nae}$-satisfied iff

$$x_1 + x_2 + \ldots + x_d \in \{1, 2, \ldots, d-1\}$$

Let $g(x) := (x-1) \cdot (x-2) \cdots (x-(d-1))$

- $g(x) = 0$ for $x \in \{1, 2, \ldots, d-1\}$
- $g(0) \neq 0$ and $g(d) \neq 0$

$g(x_1 + x_2 + \ldots + x_d)$ is the required polynomial

- Can handle negations as well
  - Use $(1 - x_i)$ for $\neg x_i$

(Monotone) d-NAE-SAT

Find polynomial f of degree $d-1$ such that

$$f(x_1, \ldots, x_d) = 0 \Leftrightarrow \text{clause } (x_1, \ldots, x_d) \text{ is } \textsc{nae}\text{-satisfied}$$

Clause $(x_1, \ldots, x_d)$ is $\textsc{nae}$-satisfied iff

$$x_1 + x_2 + \ldots + x_d \in \{1, 2, \ldots, d-1\}$$

Let $g(x) := (x-1) \cdot (x-2) \cdots (x-(d-1))$

- $g(x) = 0$ for $x \in \{1, 2, \ldots, d-1\}$
- $g(0) \neq 0$ and $g(d) \neq 0$

$g(x_1 + x_2 + \ldots + x_d)$ is the required polynomial

- Can handle negations as well
  - Use $(1 - x_i)$ for $\neg x_i$

(Monotone) d-NAE-SAT

Find polynomial f of degree $d-1$ such that

$$f(x_1, \ldots, x_d) = 0 \Leftrightarrow \text{clause } (x_1, \ldots, x_d) \text{ is } \textsc{nae}\text{-satisfied}$$

Clause $(x_1, \ldots, x_d)$ is $\textsc{nae}$-satisfied iff

$$x_1 + x_2 + \ldots + x_d \in \{1, 2, \ldots, d-1\}$$

Let $g(x) := (x-1) \cdot (x-2) \cdots (x-(d-1))$

- $g(x) = 0$ for $x \in \{1, 2, \ldots, d-1\}$
- $g(0) \neq 0$ and $g(d) \neq 0$

$g(x_1 + x_2 + \ldots + x_d)$ is the required polynomial

- Can handle negations as well
  - Use $(1 - x_i)$ for $\neg x_i$

Generalizations

We can generalize the result for c-Polynomial-root CSP

- Finite fields
- Integers mod m
- ...

Can we also use polynomial inequalities?

Generalizations

We can generalize the result for c-Polynomial-root CSP

- Finite fields
- Integers mod m
- ...

Can we also use polynomial inequalities?

Polynomial inequalities

Each constraint given by $f(x_1, \ldots, x_k) \neq 0$

- Can represent CNF-SAT, let $x, y, z \in \{0, 1\}$

$$(\neg x \vee y \vee z) \Leftrightarrow (1 - x) + y + z \neq 0$$

- Degree-1 polynomials can represent d-CNF-SAT for any d
- Lower bound for d-CNF-SAT known
  - No $O(n^{d-\varepsilon})$ sparsification for $\varepsilon > 0$
- Polynomial-size sparsification in this case not possible

Polynomial inequalities

Each constraint given by $f(x_1, \ldots, x_k) \neq 0$

- Can represent CNF-SAT, let $x, y, z \in \{0, 1\}$

$$(\neg x \vee y \vee z) \Leftrightarrow (1 - x) + y + z \neq 0$$

- Degree-1 polynomials can represent d-CNF-SAT for any d
- Lower bound for d-CNF-SAT known
  - No $O(n^{d-\varepsilon})$ sparsification for $\varepsilon > 0$
- Polynomial-size sparsification in this case not possible

Polynomial inequalities

Each constraint given by $f(x_1, \ldots, x_k) \neq 0$

- Can represent CNF-SAT, let $x, y, z \in \{0, 1\}$

$$(\neg x \vee y \vee z) \Leftrightarrow (1 - x) + y + z \neq 0$$

- Degree-1 polynomials can represent d-CNF-SAT for any d
- Lower bound for d-CNF-SAT known
  - No $O(n^{d-\varepsilon})$ sparsification for $\varepsilon > 0$
- Polynomial-size sparsification in this case not possible

Polynomial inequalities

Each constraint given by $f(x_1, \ldots, x_k) \neq 0$

- Can represent CNF-SAT, let $x, y, z \in \{0, 1\}$

$$(\neg x \vee y \vee z) \Leftrightarrow (1 - x) + y + z \neq 0$$

- Degree-1 polynomials can represent d-CNF-SAT for any d
- Lower bound for d-CNF-SAT known
  - No $O(n^{d-\varepsilon})$ sparsification for $\varepsilon > 0$
- Polynomial-size sparsification in this case not possible

Polynomial inequalities

Each constraint given by $f(x_1, \ldots, x_k) \neq 0$

- Can represent CNF-SAT, let $x, y, z \in \{0, 1\}$

$$(\neg x \vee y \vee z) \Leftrightarrow (1 - x) + y + z \neq 0$$

- Degree-1 polynomials can represent d-CNF-SAT for any d
- Lower bound for d-CNF-SAT known
  - No $O(n^{d-\varepsilon})$ sparsification for $\varepsilon > 0$
- Polynomial-size sparsification in this case not possible

Application to q-Coloring

The q-Coloring problem

Can the vertices of a graph be colored with at most q colors?

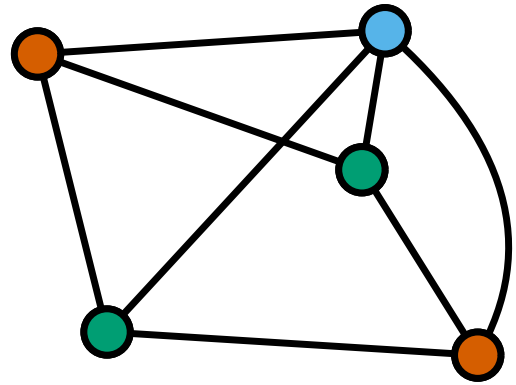
- Focus on $q = 3$
- red, green, blue

Can be seen as a CSP (domain $\{0, 1, 2\}$)

- Variable for each vertex
- Constraint for each edge

Trivial sparsification $O(n^2)$

- Matching lower bound



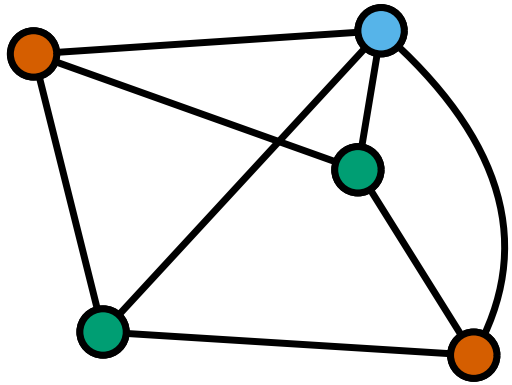
The q-Coloring problem

Can the vertices of a graph be colored with at most q colors?

- Focus on $q = 3$
- red, green, blue

Can be seen as a CSP (domain $\{0, 1, 2\}$)

- Variable for each vertex
- Constraint for each edge



Trivial sparsification $O(n^2)$

- Matching lower bound

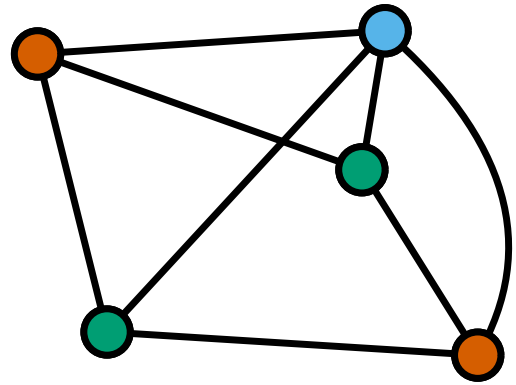
The q-Coloring problem

Can the vertices of a graph be colored with at most q colors?

- Focus on $q = 3$
- red, green, blue

Can be seen as a CSP (domain $\{0, 1, 2\}$)

- Variable for each vertex
- Constraint for each edge



Trivial sparsification $O(n^2)$

- Matching lower bound

Another complexity measure

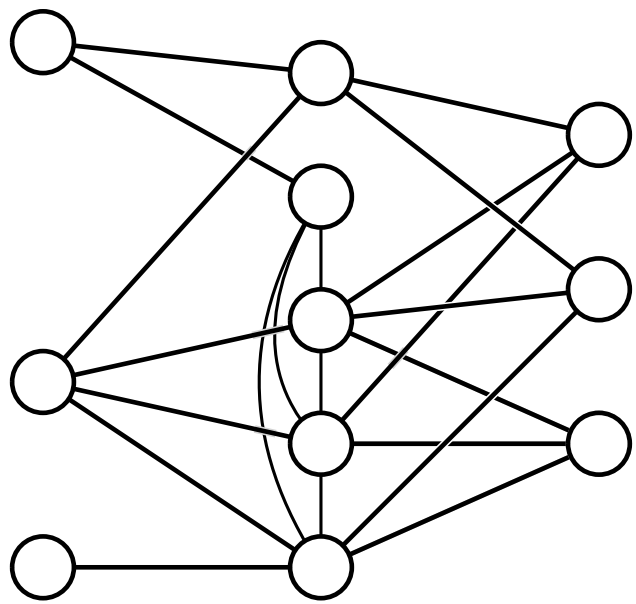
Bound sparsification size by more interesting parameter

- Measures complexity of input graph
- Size of a Vertex Cover

Another complexity measure

Bound sparsification size by more interesting parameter

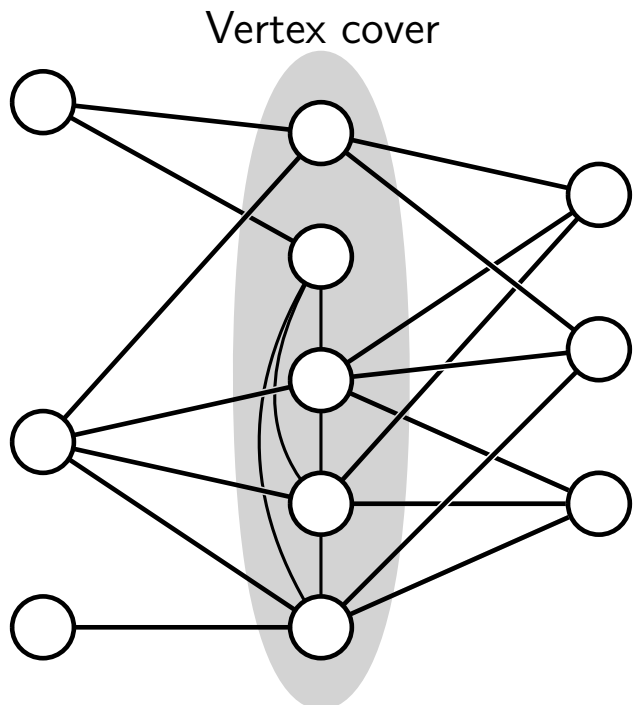
- Measures complexity of input graph
- Size of a Vertex Cover



Another complexity measure

Bound sparsification size by more interesting parameter

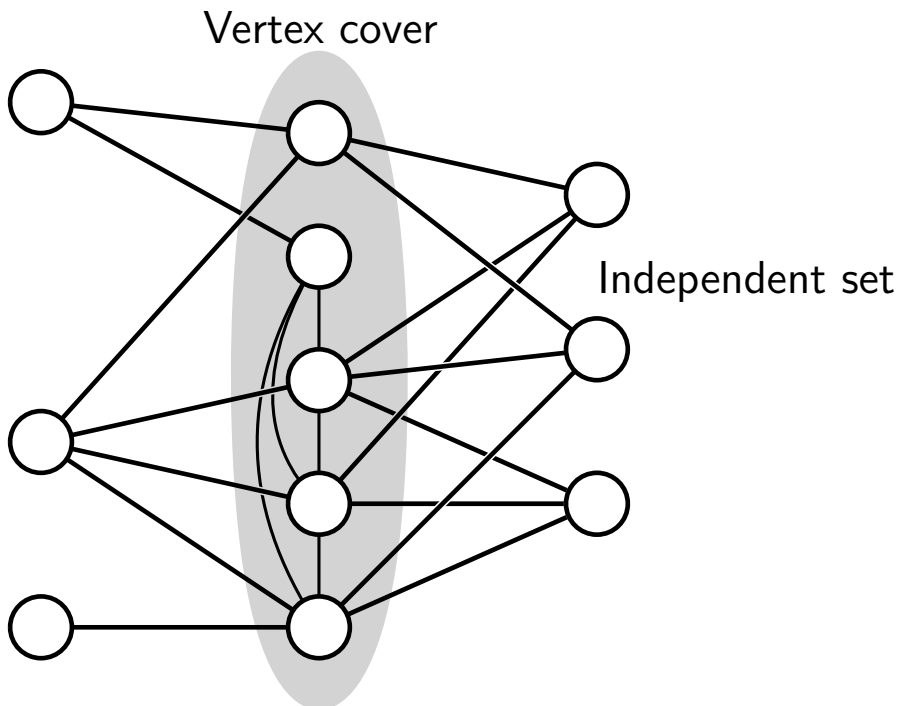
- Measures complexity of input graph
- Size of a Vertex Cover



Another complexity measure

Bound sparsification size by more interesting parameter

- Measures complexity of input graph
- Size of a Vertex Cover



Another complexity measure

Bound sparsification size by more interesting parameter

- Measures complexity of input graph
- Size of a Vertex Cover

Why Vertex Cover? Alternatives:

- Treewidth
  - No polynomial bound
- Deletion distance to (disjoint union of) paths
  - No polynomial bound [Jansen,Kratsch]

Kernelization

Efficiently reduce the size of an input instance

- Resulting size depends on parameter value
- Provably small
- Provably correct

Kernel for 3-Coloring

Polynomial-time algorithm that, given graph G with vertex cover of size k, outputs G' such that

- $|G'|$ is bounded by $f(k)$
- G is 3-Colorable if and only if G' is 3-Colorable

Kernelization

Efficiently reduce the size of an input instance

- Resulting size depends on parameter value
- Provably small
- Provably correct

Kernel for 3-Coloring

Polynomial-time algorithm that, given graph G with vertex cover of size k, outputs G' such that

- $|G'|$ is bounded by $f(k)$
- G is 3-Colorable if and only if G' is 3-Colorable

Kernel for 3-Coloring

Previous work

Jansen and Kratsch [Inf Comput. 2013] showed that

- 3-Coloring parameterized by VC has a kernel with $O(k^3)$ edges and vertices

Using the tools for CSPs, this can be improved!

Theorem

3-Coloring parameterized by Vertex Cover has a kernel with $O(k^2)$ edges and vertices

Kernel for 3-Coloring

Previous work

Jansen and Kratsch [Inf Comput. 2013] showed that

- 3-Coloring parameterized by VC has a kernel with $O(k^3)$ edges and vertices

Using the tools for CSPs, this can be improved!

Theorem

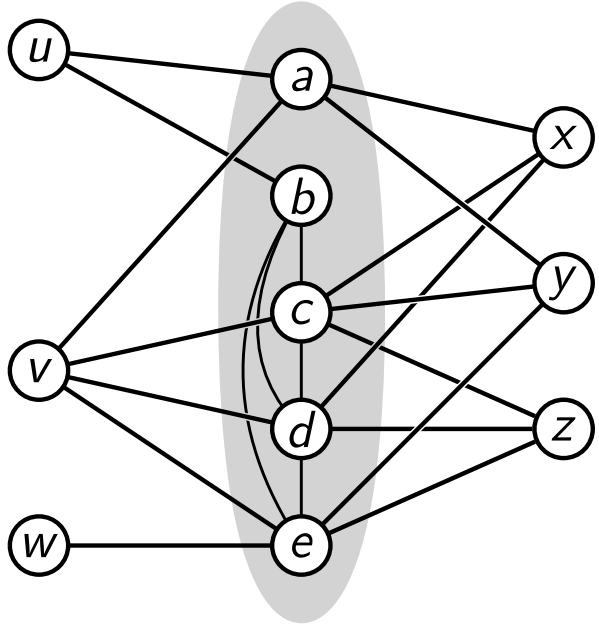
3-Coloring parameterized by Vertex Cover has a kernel with $O(k^2)$ edges and vertices

Simple kernel for 3-Coloring

Kernel for 3-Coloring: first idea

We have graph G, with vertex cover VC
The remaining vertices form independent set IS

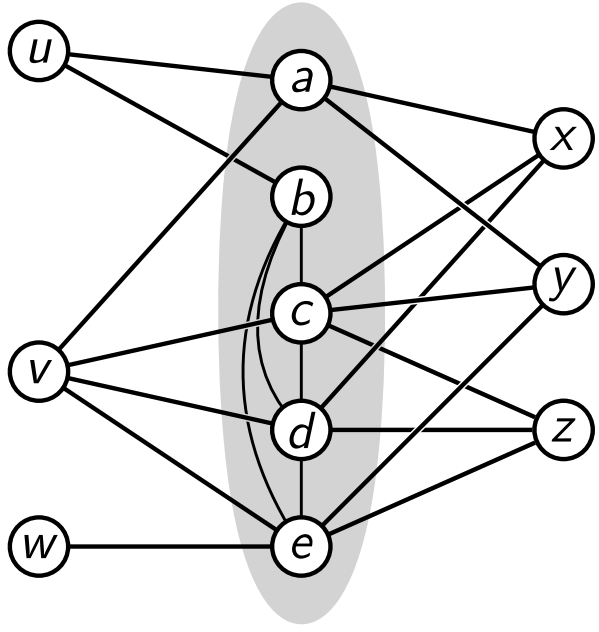
- IS may be large compared to VC
  - Find redundant vertices in IS
- Low-degree vertices can always be colored
  - They are redundant
- Coloring can be extended to vertex in IS iff ≤ 2 colors used in neighborhood
  - Consider v and z, note $N(z) \subseteq N(v)$
  - z is redundant



Kernel for 3-Coloring: first idea

We have graph G, with vertex cover VC
The remaining vertices form independent set IS

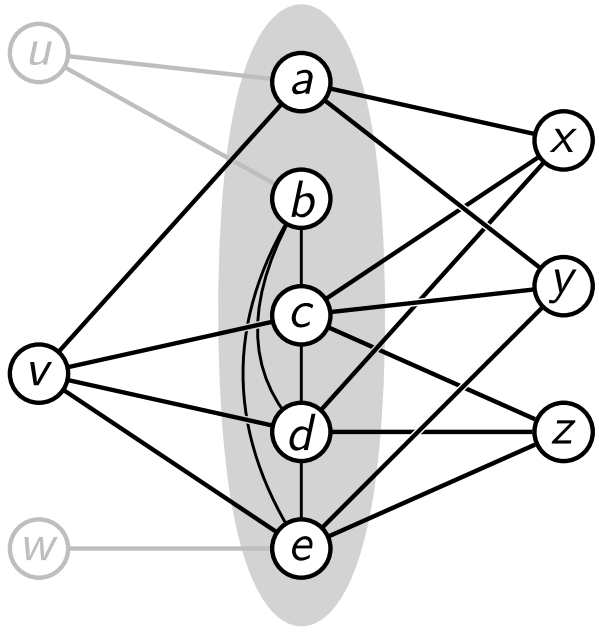
- IS may be large compared to VC
  - Find redundant vertices in IS
- Low-degree vertices can always be colored
  - They are redundant
- Coloring can be extended to vertex in IS iff ≤ 2 colors used in neighborhood
  - Consider v and z, note $N(z) \subseteq N(v)$
  - z is redundant



Kernel for 3-Coloring: first idea

We have graph G, with vertex cover VC
The remaining vertices form independent set IS

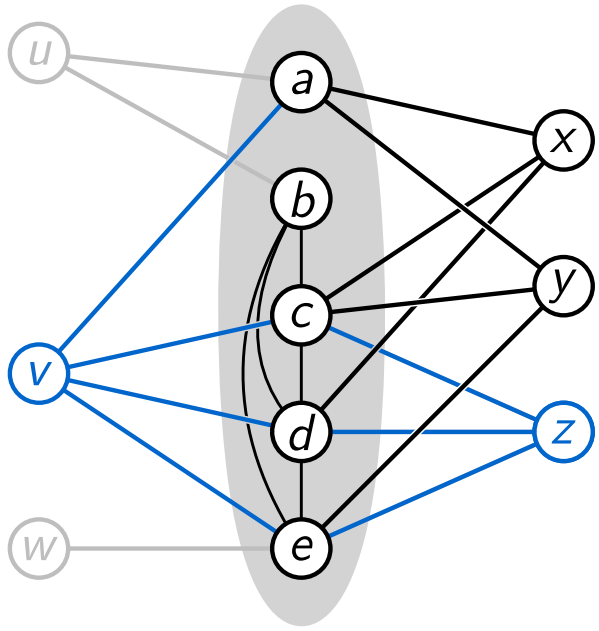
- IS may be large compared to VC
  - Find redundant vertices in IS
- Low-degree vertices can always be colored
  - They are redundant
- Coloring can be extended to vertex in IS iff ≤ 2 colors used in neighborhood
  - Consider v and z, note $N(z) \subseteq N(v)$
  - z is redundant



Kernel for 3-Coloring: first idea

We have graph G, with vertex cover VC
The remaining vertices form independent set IS

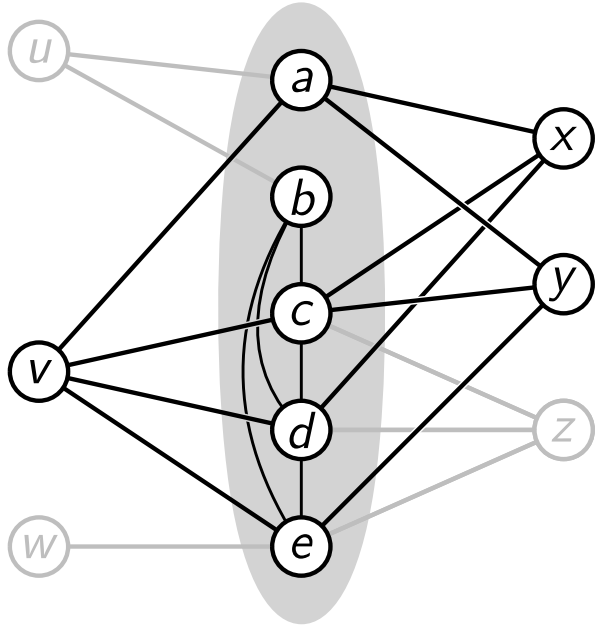
- IS may be large compared to VC
  - Find redundant vertices in IS
- Low-degree vertices can always be colored
  - They are redundant
- Coloring can be extended to vertex in IS iff ≤ 2 colors used in neighborhood
  - Consider v and z, note $N(z) \subseteq N(v)$
  - z is redundant



Kernel for 3-Coloring: first idea

We have graph G, with vertex cover VC
The remaining vertices form independent set IS

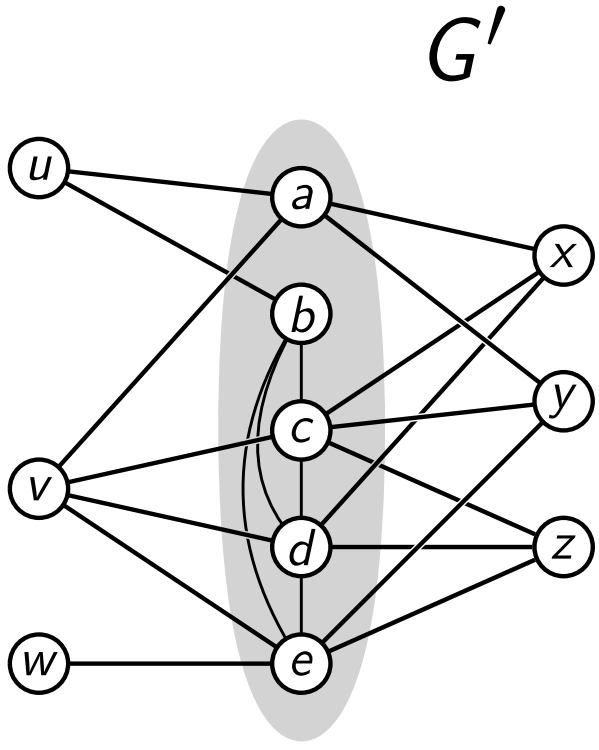
- IS may be large compared to VC
  - Find redundant vertices in IS
- Low-degree vertices can always be colored
  - They are redundant
- Coloring can be extended to vertex in IS iff ≤ 2 colors used in neighborhood
  - Consider v and z, note $N(z) \subseteq N(v)$
  - z is redundant



Kernel for 3-Coloring

Build the kernel G' as follows

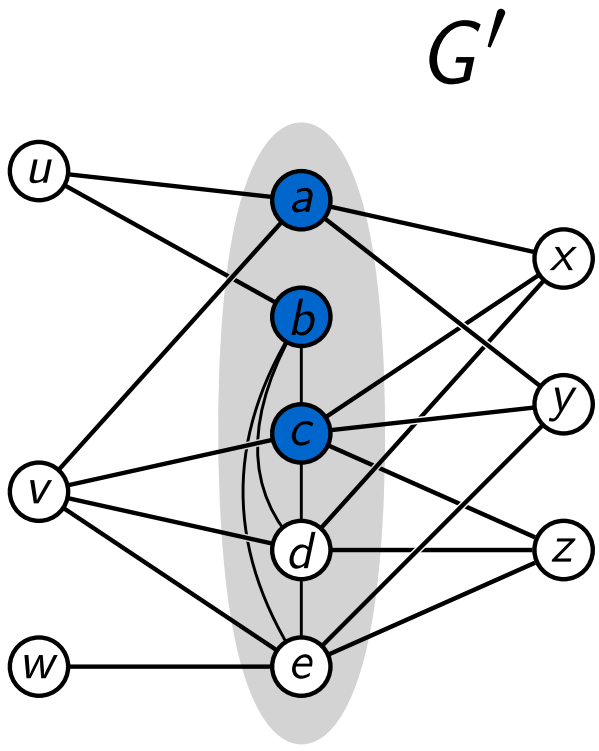
- Start with $G' = G$
- For each triple $v_1, v_2, v_3 \in \text{VC}$
  - Check if they have a common neighbor in IS in G
  - If so, mark one such neighbor
- Remove all unmarked nodes from IS



Kernel for 3-Coloring

Build the kernel G' as follows

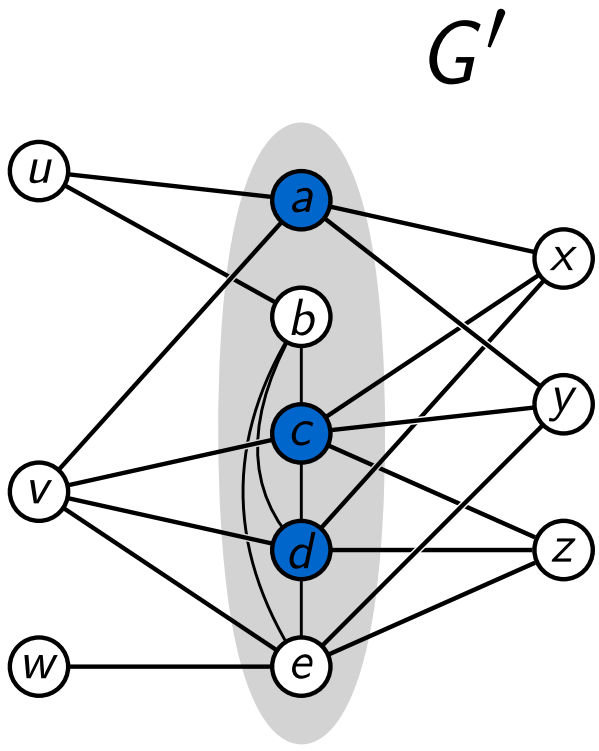
- Start with $G' = G$
- For each triple $v_1, v_2, v_3 \in \text{VC}$
  - Check if they have a common neighbor in IS in G
  - If so, mark one such neighbor
- Remove all unmarked nodes from IS



Kernel for 3-Coloring

Build the kernel G' as follows

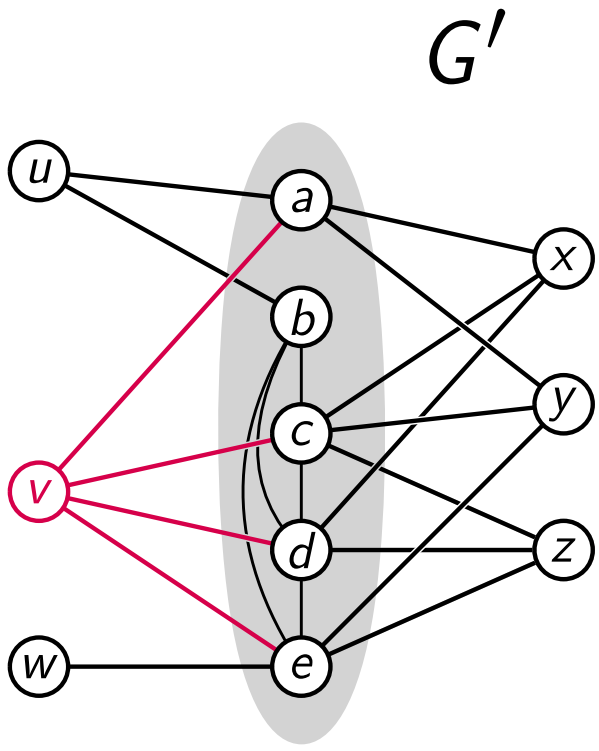
- Start with $G' = G$
- For each triple $v_1, v_2, v_3 \in \text{VC}$
  - Check if they have a common neighbor in IS in G
  - If so, mark one such neighbor
- Remove all unmarked nodes from IS



Kernel for 3-Coloring

Build the kernel G' as follows

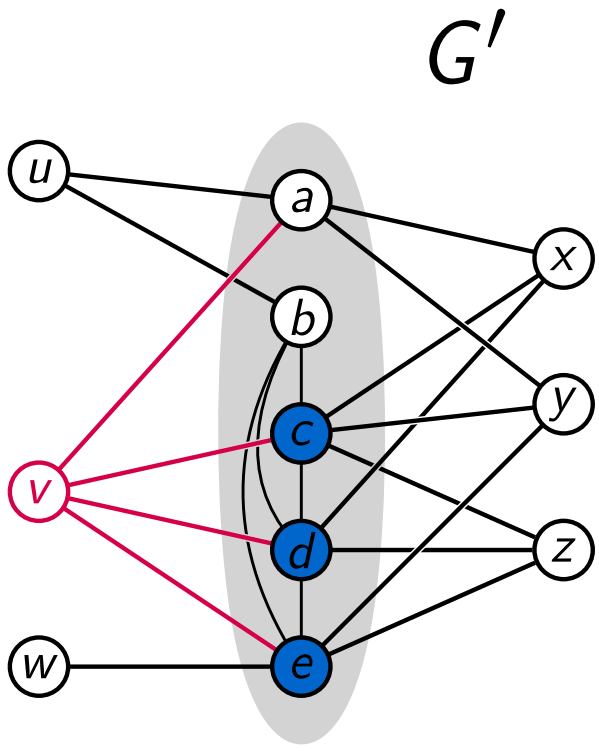
- Start with $G' = G$
- For each triple $v_1, v_2, v_3 \in \mathrm{VC}$
  - Check if they have a common neighbor in IS in G
  - If so, mark one such neighbor
- Remove all unmarked nodes from IS



Kernel for 3-Coloring

Build the kernel G' as follows

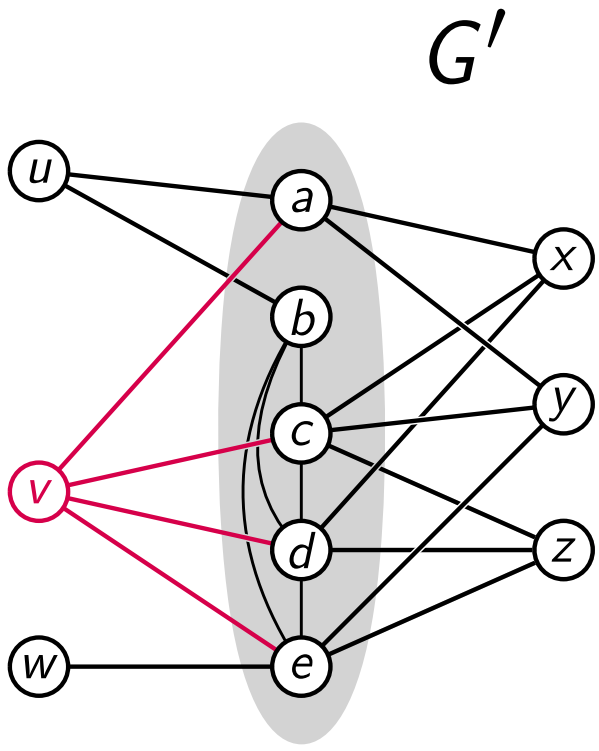
- Start with $G' = G$
- For each triple $v_1, v_2, v_3 \in \text{VC}$
  - Check if they have a common neighbor in IS in G
  - If so, mark one such neighbor
- Remove all unmarked nodes from IS



Kernel for 3-Coloring

Build the kernel G' as follows

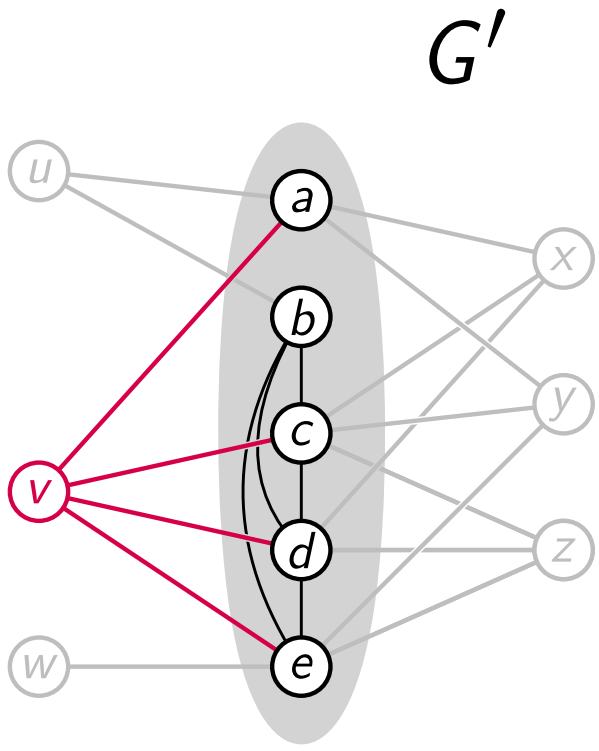
- Start with $G' = G$
- For each triple $v_1, v_2, v_3 \in \mathrm{VC}$
  - Check if they have a common neighbor in IS in G
  - If so, mark one such neighbor
- Remove all unmarked nodes from IS



Kernel for 3-Coloring

Build the kernel G' as follows

- Start with $G' = G$
- For each triple $v_1, v_2, v_3 \in \text{VC}$
  - Check if they have a common neighbor in IS in G
  - If so, mark one such neighbor
- Remove all unmarked nodes from IS

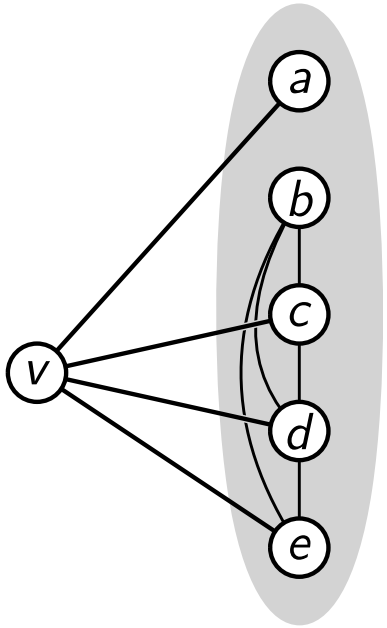


Kernel for 3-Coloring

Build the kernel G' as follows

- Start with $G' = G$
- For each triple $v_1, v_2, v_3 \in \mathrm{VC}$
  - Check if they have a common neighbor in IS in G
  - If so, mark one such neighbor
- Remove all unmarked nodes from IS

G'



Kernel for 3-Coloring

Correctness

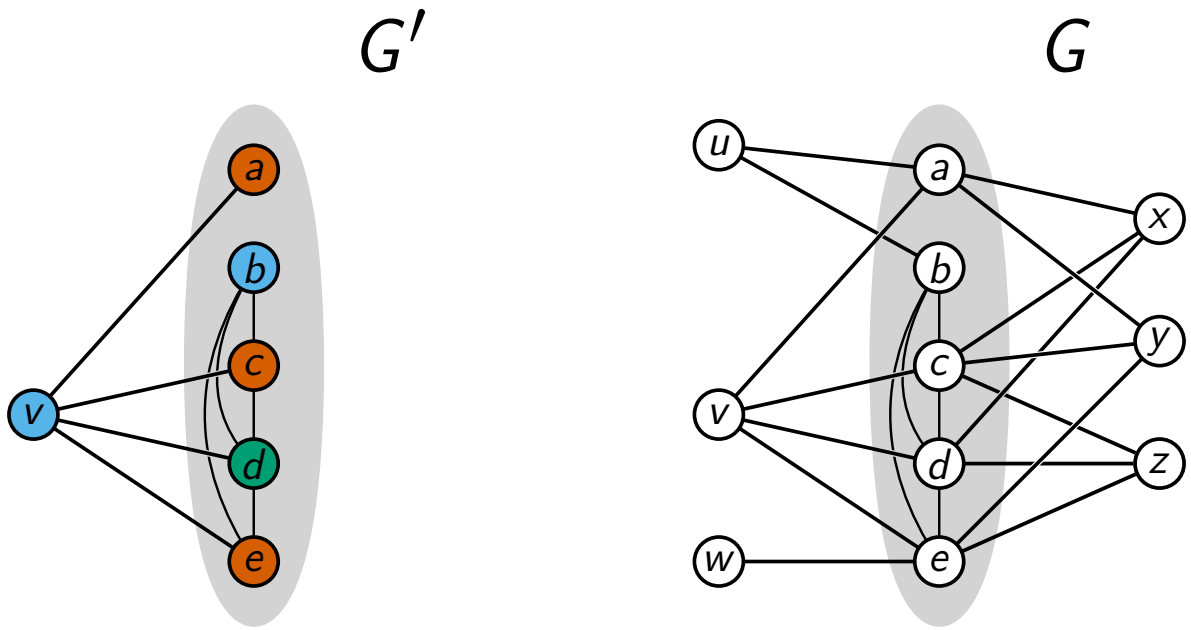
($\Rightarrow$) Clearly if G is 3-colorable, so is G'

Kernel for 3-Coloring

Correctness

($\Leftarrow$) Suppose G' is 3-colorable, we show how to color G

- Each vertex of VC receives the same color as in G'
- Let $v \in \mathrm{IS}$
  - $N(v) \subseteq \mathrm{VC}$ is already colored
  - Assign v with a color not used in $N(v)$



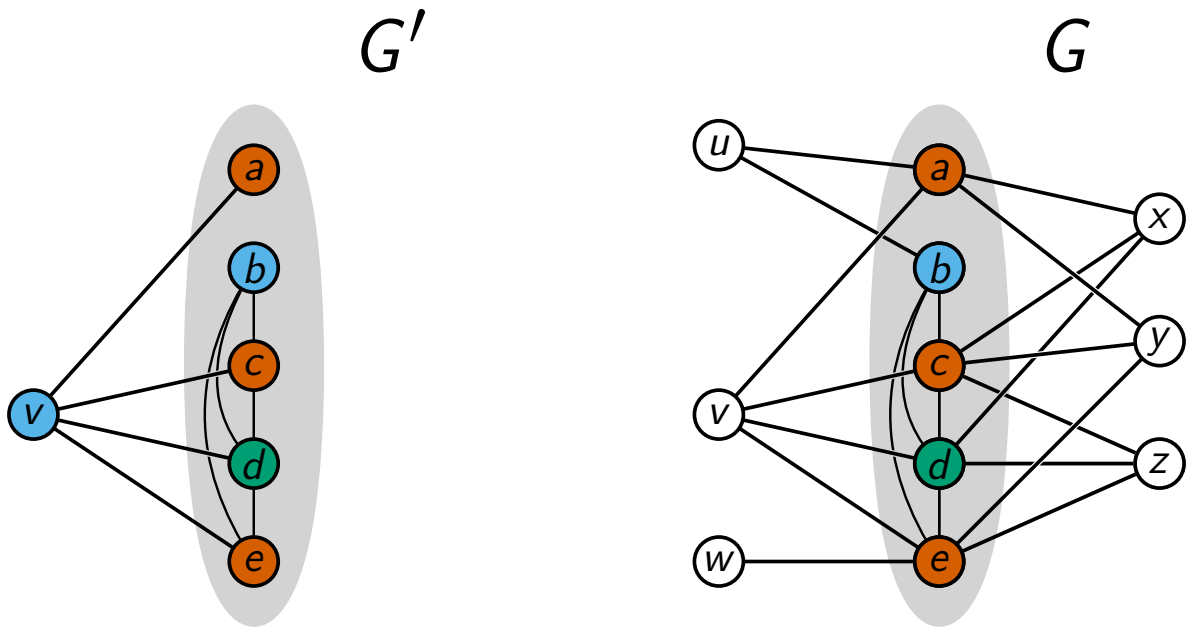
Is this always possible?

Kernel for 3-Coloring

Correctness

($\Leftarrow$) Suppose G' is 3-colorable, we show how to color G

- Each vertex of VC receives the same color as in G'
- Let $v \in \mathrm{IS}$
  - $N(v) \subseteq \mathrm{VC}$ is already colored
  - Assign v with a color not used in $N(v)$



Is this always possible?

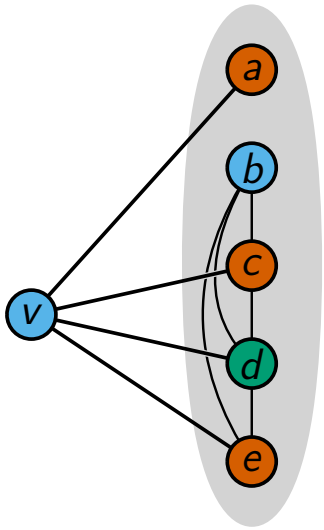
Kernel for 3-Coloring

Correctness

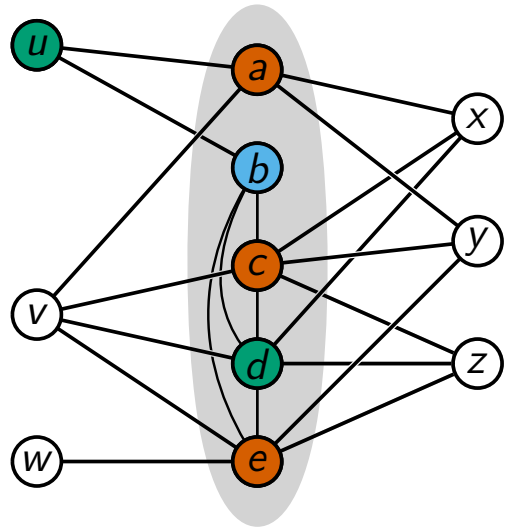
($\Leftarrow$) Suppose G' is 3-colorable, we show how to color G

- Each vertex of $\textsc{vc}$ receives the same color as in G'
- Let $v \in \textsc{is}$
  - $N(v) \subseteq \textsc{vc}$ is already colored
  - Assign v with a color not used in $N(v)$

G'



G



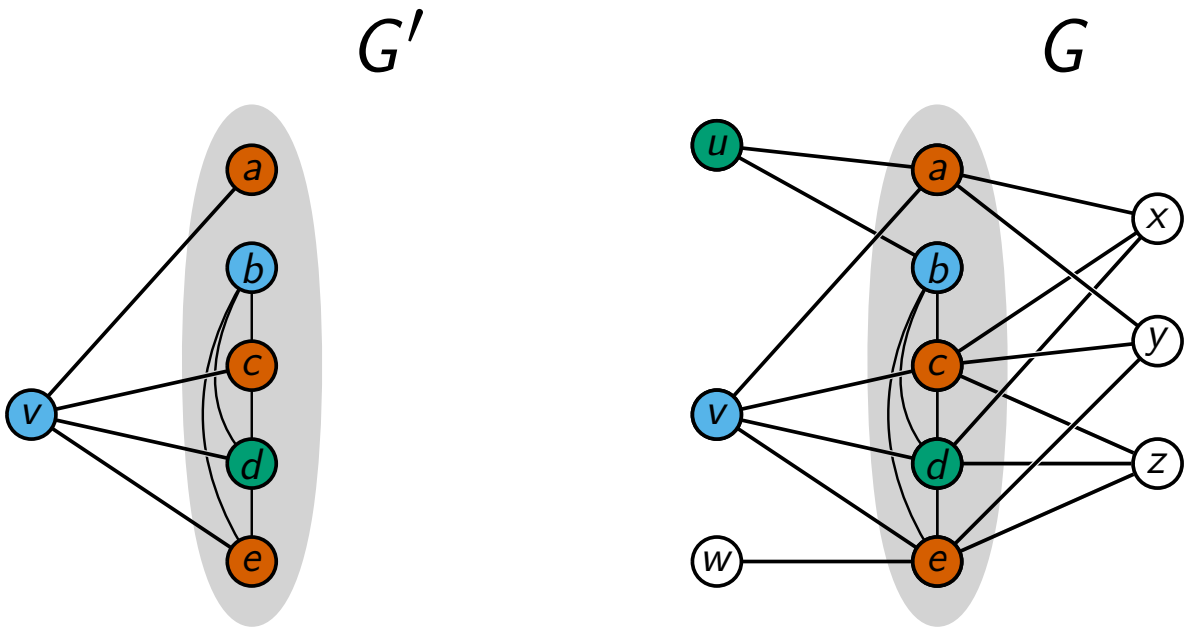
Is this always possible?

Kernel for 3-Coloring

Correctness

($\Leftarrow$) Suppose G' is 3-colorable, we show how to color G

- Each vertex of $\textsc{vc}$ receives the same color as in G'
- Let $v \in \textsc{is}$
  - $N(v) \subseteq \textsc{vc}$ is already colored
  - Assign v with a color not used in $N(v)$



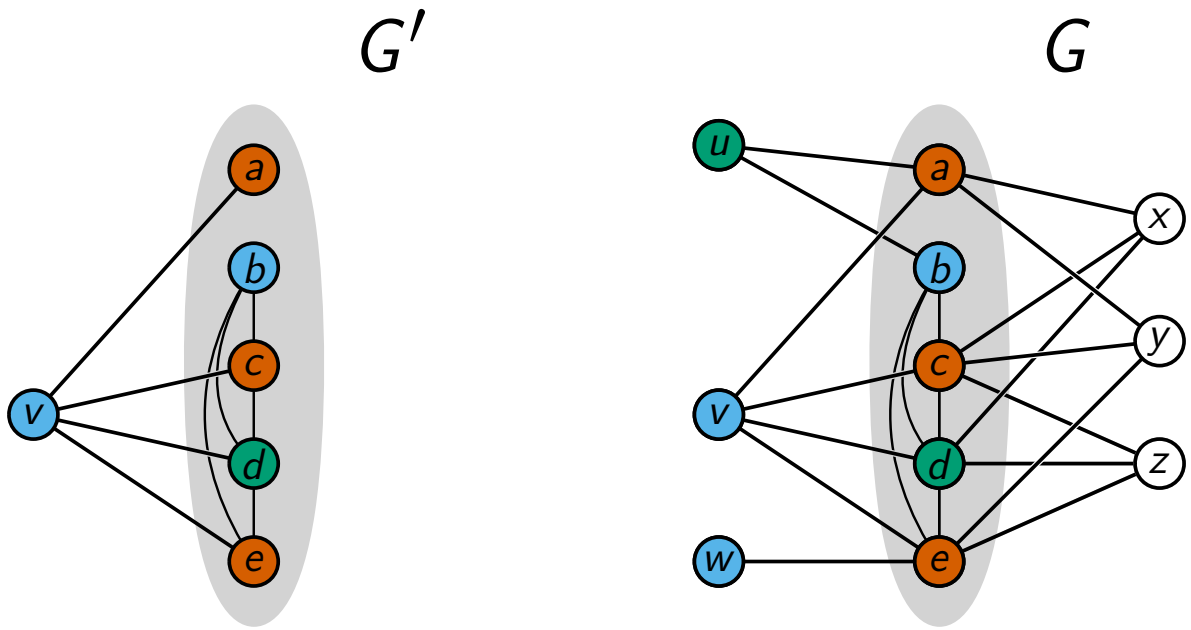
Is this always possible?

Kernel for 3-Coloring

Correctness

($\Leftarrow$) Suppose G' is 3-colorable, we show how to color G

- Each vertex of VC receives the same color as in G'
- Let $v \in \mathrm{IS}$
  - $N(v) \subseteq \mathrm{VC}$ is already colored
  - Assign v with a color not used in $N(v)$



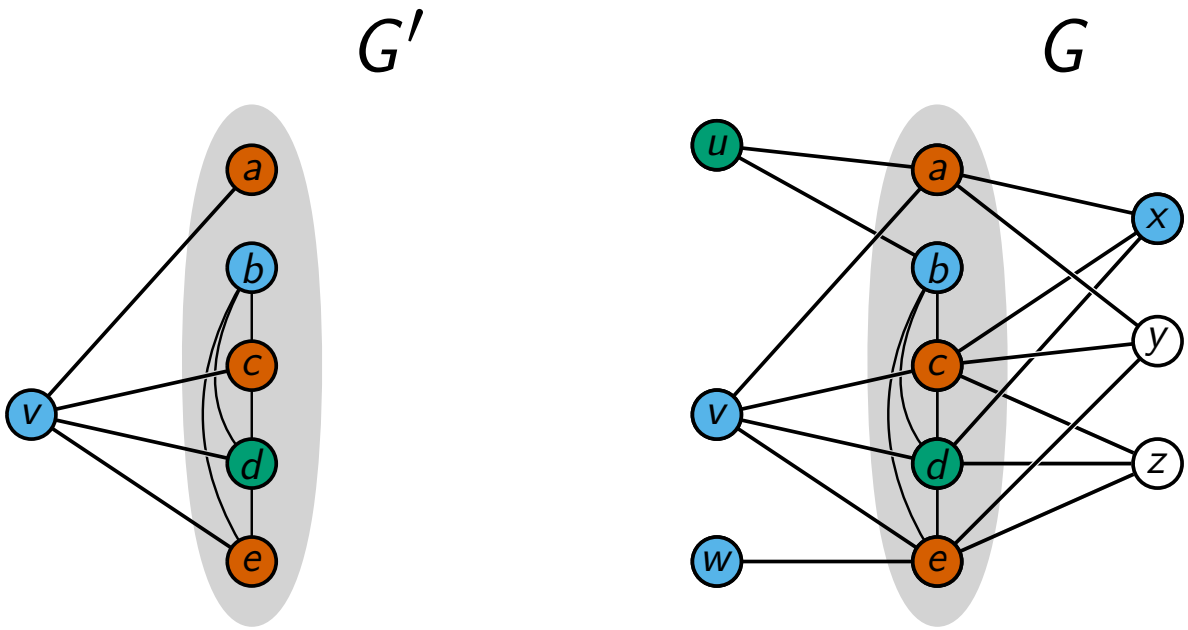
Is this always possible?

Kernel for 3-Coloring

Correctness

($\Leftarrow$) Suppose G' is 3-colorable, we show how to color G

- Each vertex of $\textsc{vc}$ receives the same color as in G'
- Let $v \in \textsc{is}$
  - $N(v) \subseteq \textsc{vc}$ is already colored
  - Assign v with a color not used in $N(v)$



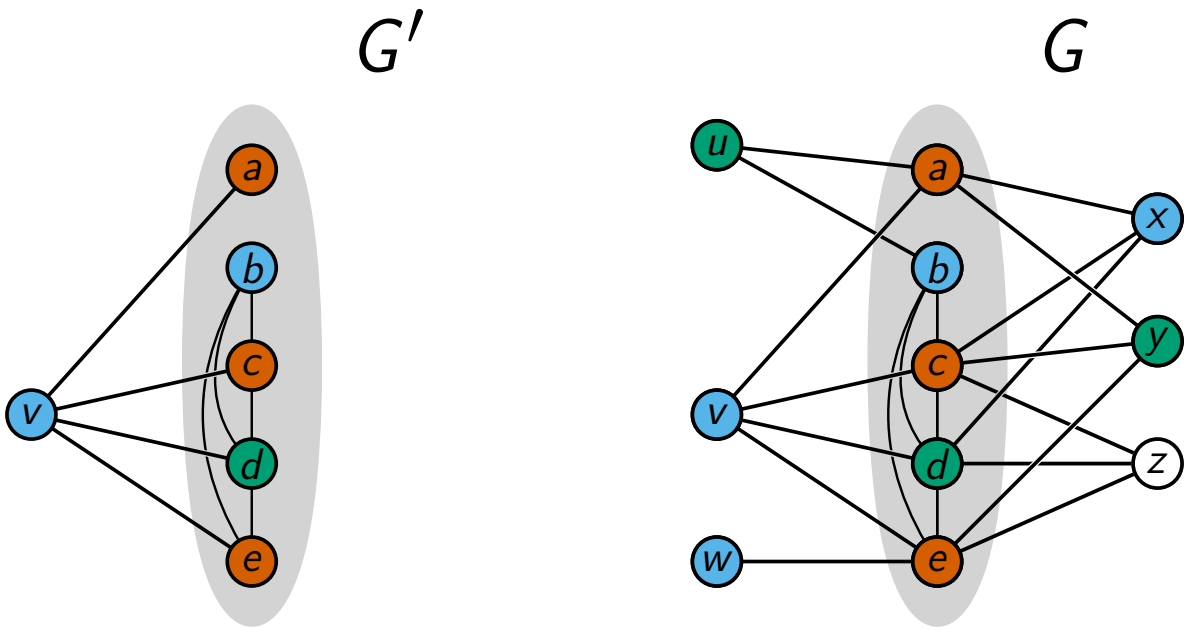
Is this always possible?

Kernel for 3-Coloring

Correctness

($\Leftarrow$) Suppose G' is 3-colorable, we show how to color G

- Each vertex of $\textsc{vc}$ receives the same color as in G'
- Let $v \in \textsc{is}$
  - $N(v) \subseteq \textsc{vc}$ is already colored
  - Assign v with a color not used in $N(v)$



Is this always possible?

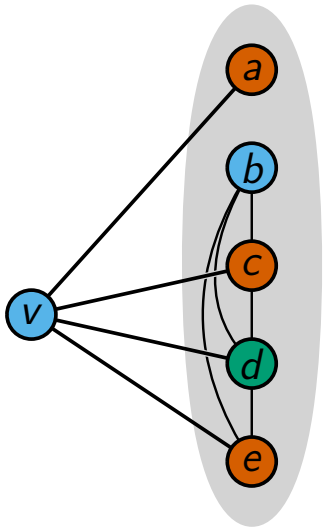
Kernel for 3-Coloring

Correctness

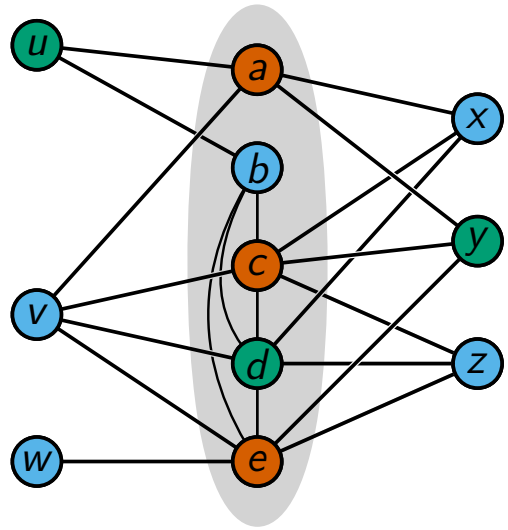
($\Leftarrow$) Suppose G' is 3-colorable, we show how to color G

- Each vertex of $\textsc{vc}$ receives the same color as in G'
- Let $v \in \textsc{is}$
  - $N(v) \subseteq \textsc{vc}$ is already colored
  - Assign v with a color not used in $N(v)$

G'



G



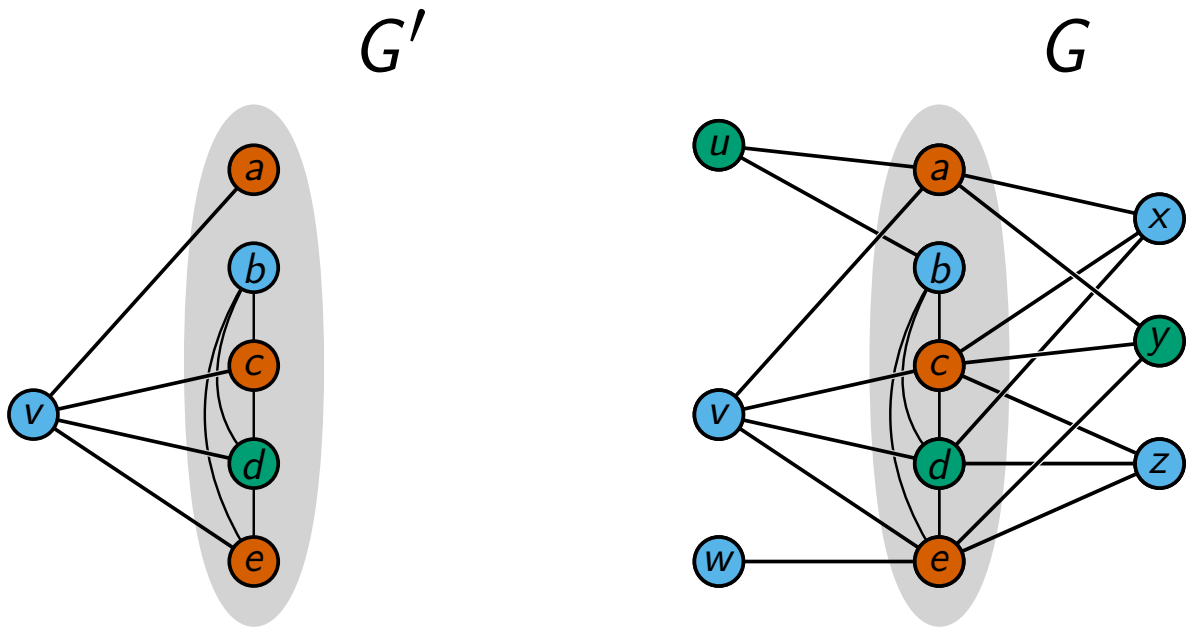
Is this always possible?

Kernel for 3-Coloring

Correctness

($\Leftarrow$) Suppose G' is 3-colorable, we show how to color G

- Each vertex of VC receives the same color as in G'
- Let $v \in \mathrm{IS}$
  - $N(v) \subseteq \mathrm{VC}$ is already colored
  - Assign v with a color not used in $N(v)$

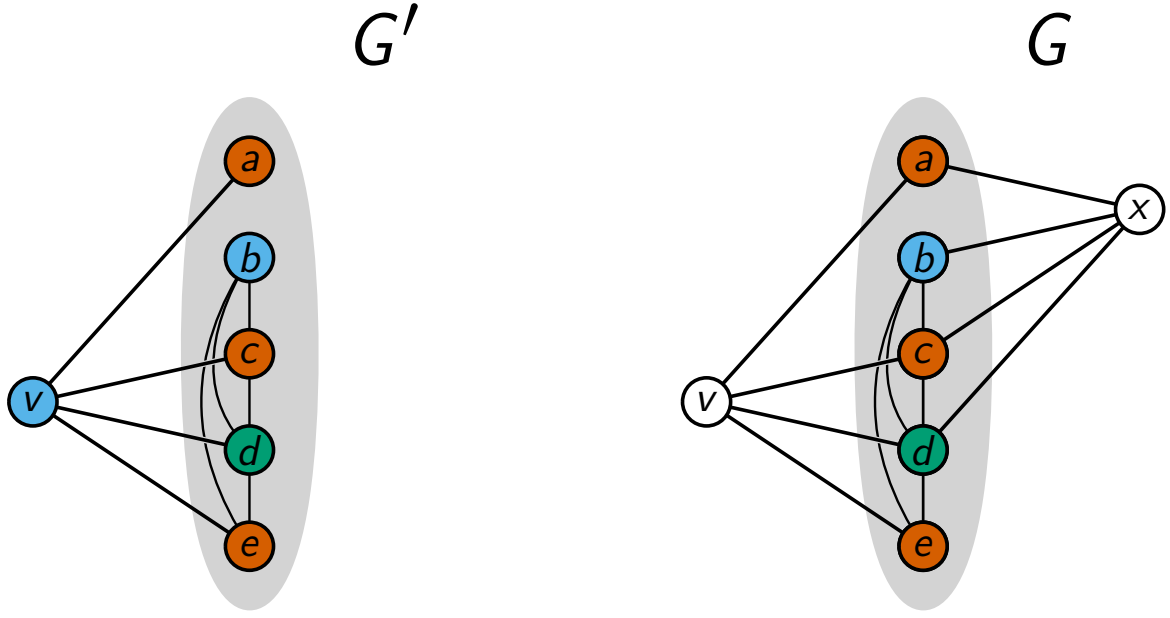


Is this always possible?

Is this always possible?

Yes! Proof: Suppose not

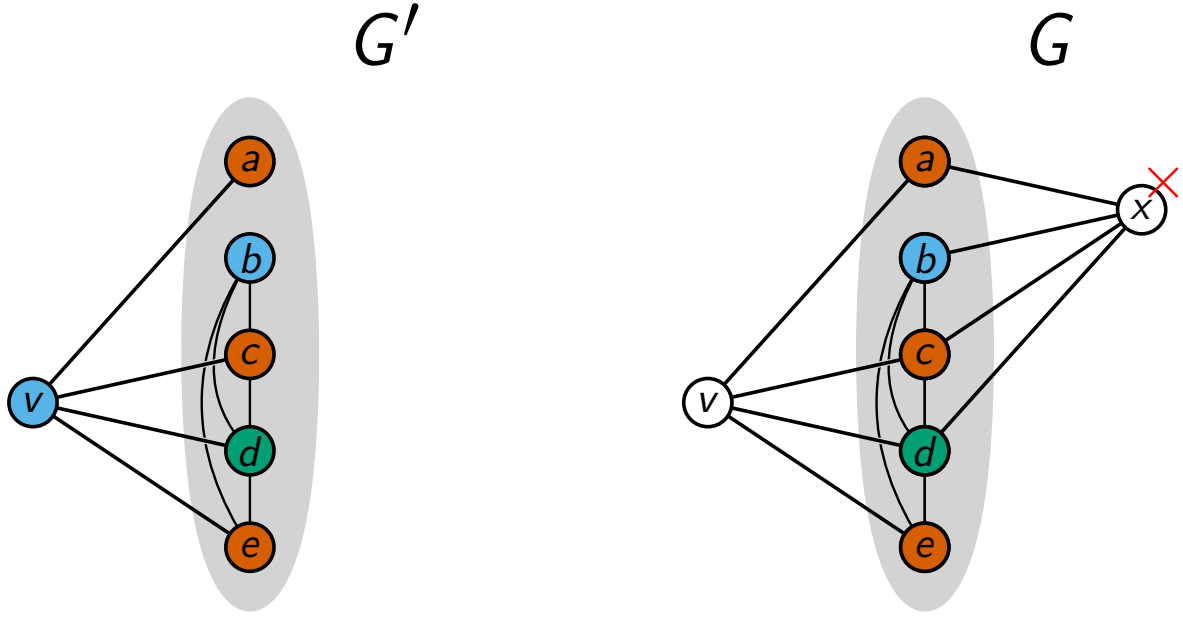
- Coloring of VC cannot be extended to some x
- Pick $r, g, b \in N(x)$ with colors red,green,blue
- r,g,b have common neighbor x
- Some common neighbor was marked
- G' contains vertex y such that $r, g, b \in N(y)$
- Contradiction



Is this always possible?

Yes! Proof: Suppose not

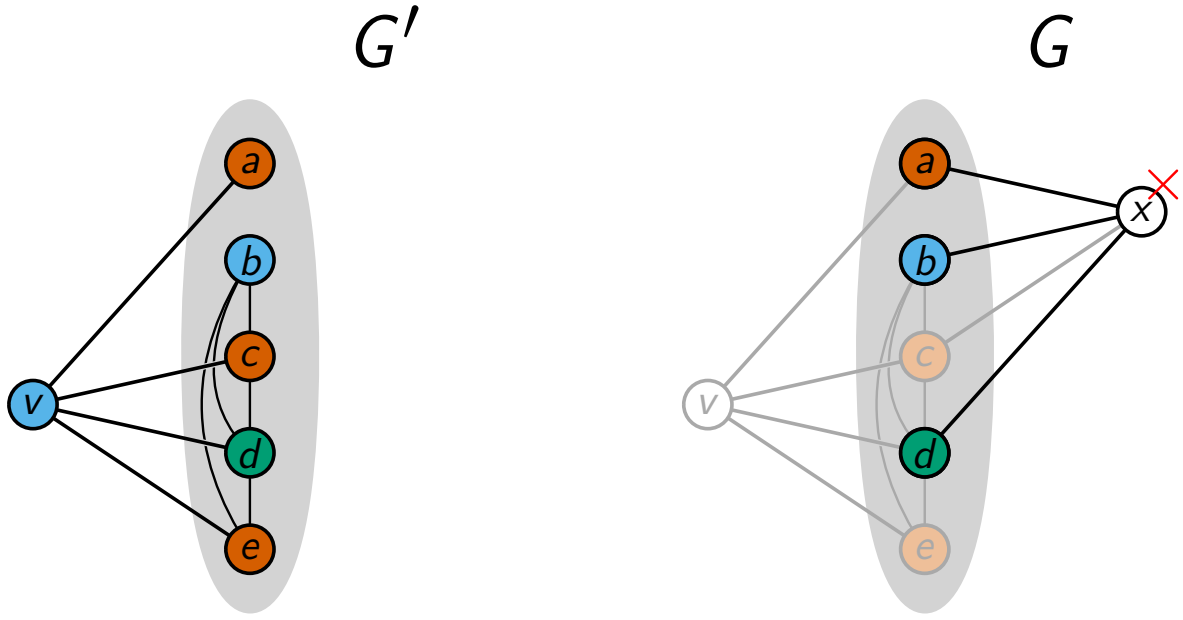
- Coloring of VC cannot be extended to some x
- Pick $r, g, b \in N(x)$ with colors red,green,blue
- r,g,b have common neighbor x
- Some common neighbor was marked
- G' contains vertex y such that $r, g, b \in N(y)$
- Contradiction



Is this always possible?

Yes! Proof: Suppose not

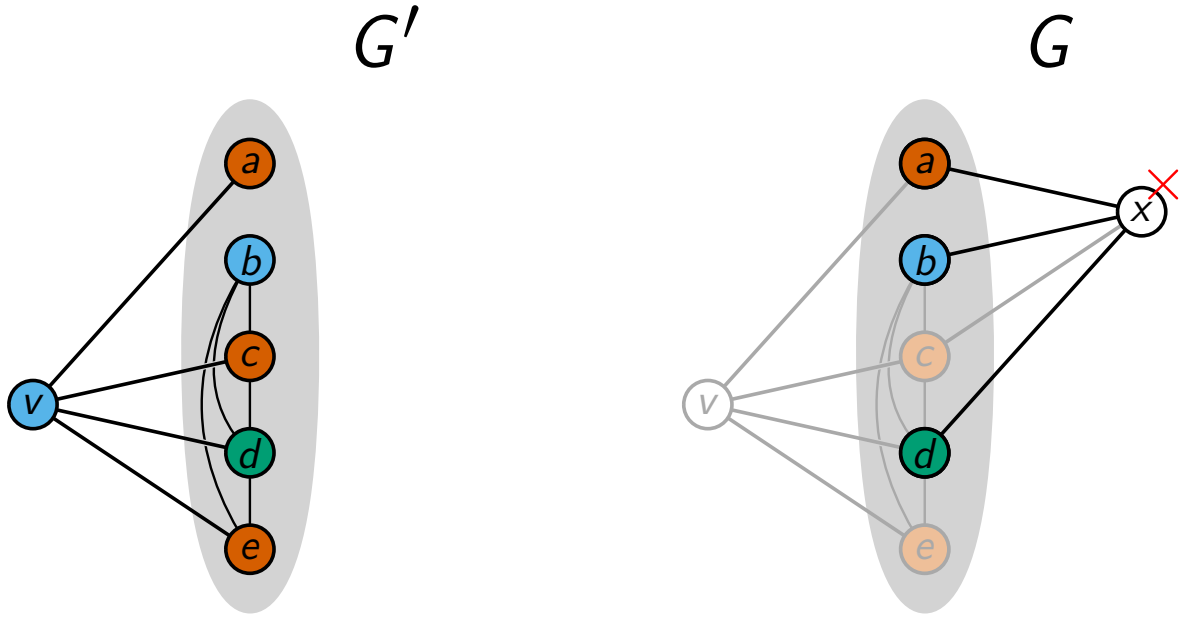
- Coloring of VC cannot be extended to some x
- Pick $r, g, b \in N(x)$ with colors red,green,blue
- r,g,b have common neighbor x
- Some common neighbor was marked
- G' contains vertex y such that $r, g, b \in N(y)$
- Contradiction



Is this always possible?

Yes! Proof: Suppose not

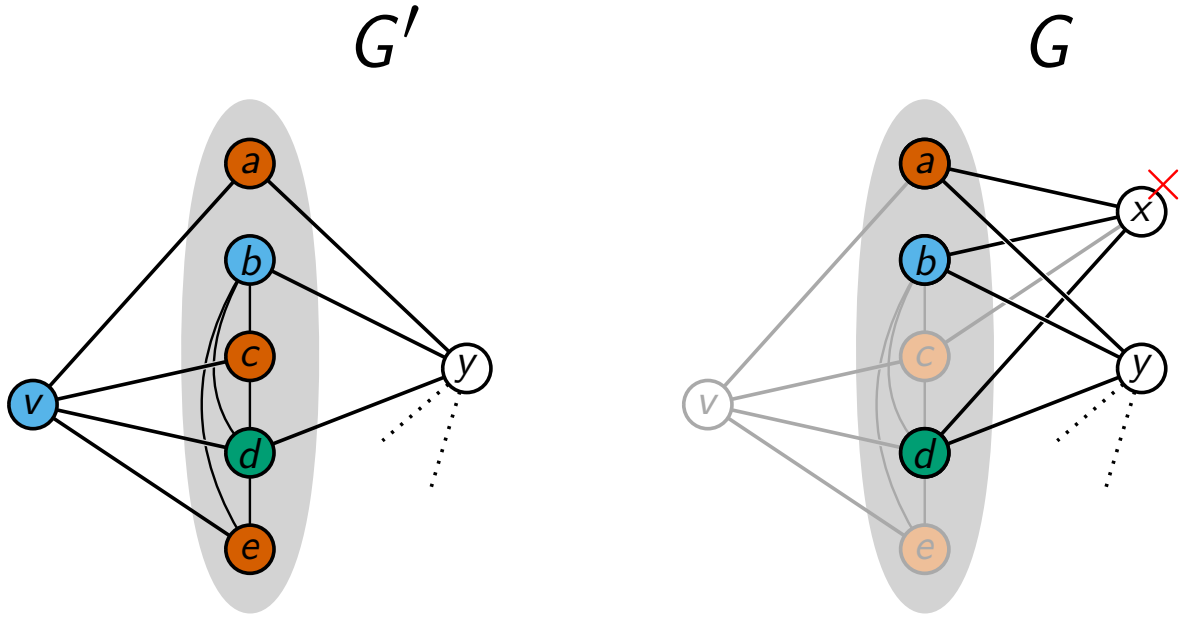
- Coloring of VC cannot be extended to some x
- Pick $r, g, b \in N(x)$ with colors red,green,blue
- r,g,b have common neighbor x
- Some common neighbor was marked
- G' contains vertex y such that $r, g, b \in N(y)$
- Contradiction



Is this always possible?

Yes! Proof: Suppose not

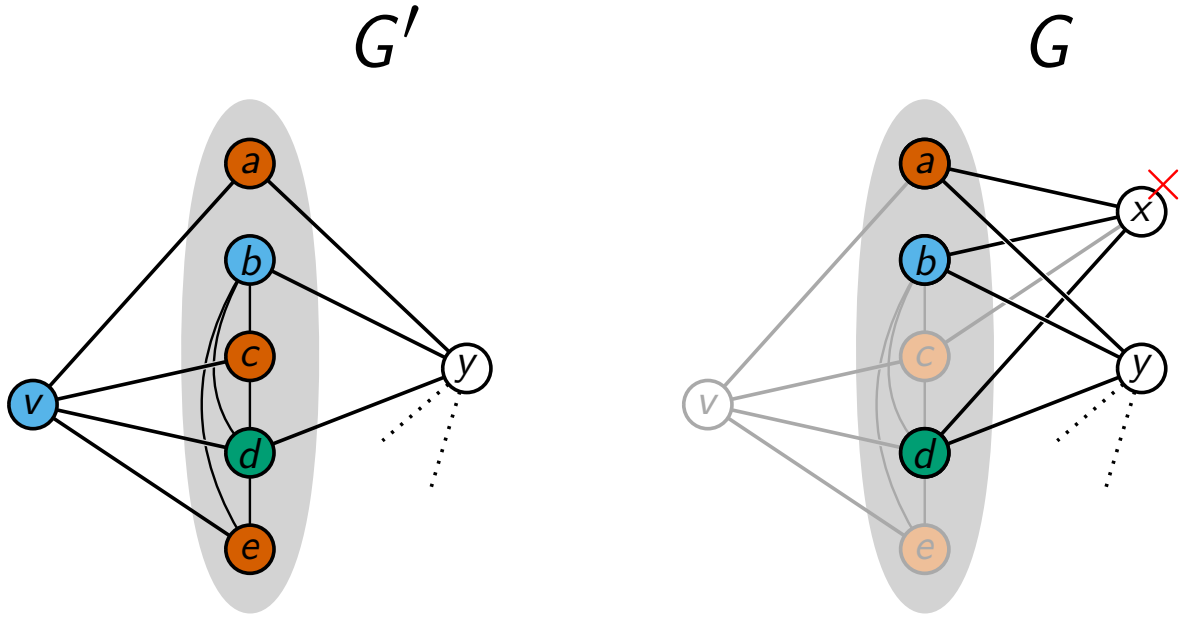
- Coloring of VC cannot be extended to some x
- Pick $r, g, b \in N(x)$ with colors red,green,blue
- r,g,b have common neighbor x
- Some common neighbor was marked
- G' contains vertex y such that $r, g, b \in N(y)$
- Contradiction



Is this always possible?

Yes! Proof: Suppose not

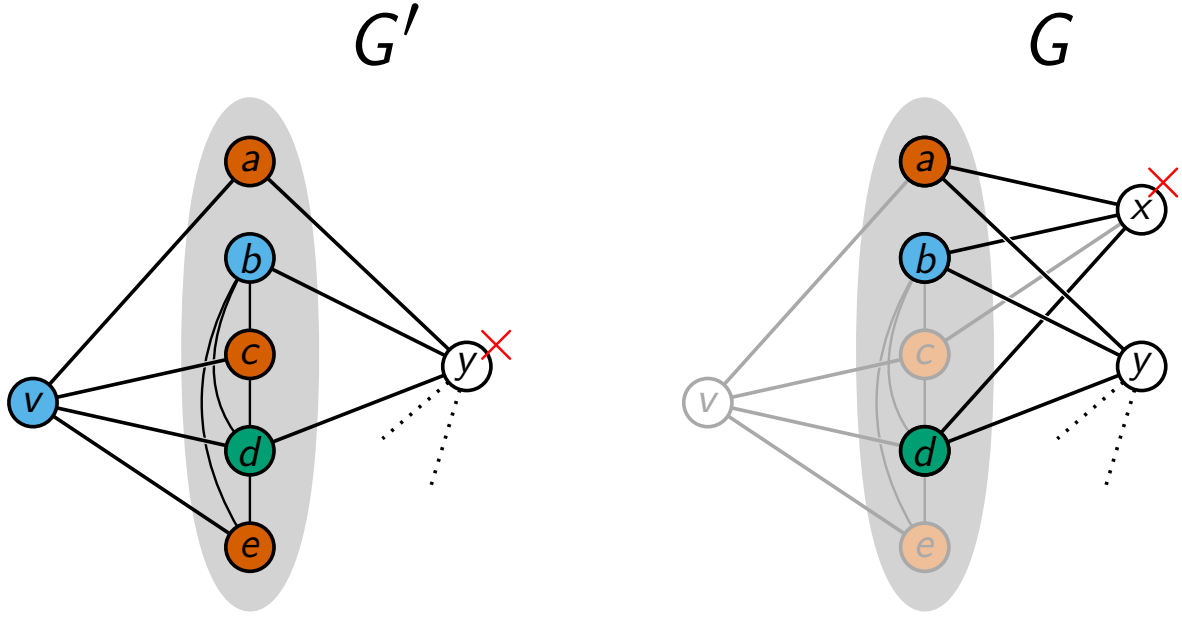
- Coloring of VC cannot be extended to some x
- Pick $r, g, b \in N(x)$ with colors red,green,blue
- r,g,b have common neighbor x
- Some common neighbor was marked
- G' contains vertex y such that $r, g, b \in N(y)$
- Contradiction



Is this always possible?

Yes! Proof: Suppose not

- Coloring of VC cannot be extended to some x
- Pick $r, g, b \in N(x)$ with colors red,green,blue
- r,g,b have common neighbor x
- Some common neighbor was marked
- G' contains vertex y such that $r, g, b \in N(y)$
- Contradiction



Kernel for 3-Coloring

Size

Number of vertices

- VC: Exactly k by definition
- IS: At most $\binom{k}{3} = O(k^3)$

Number of edges

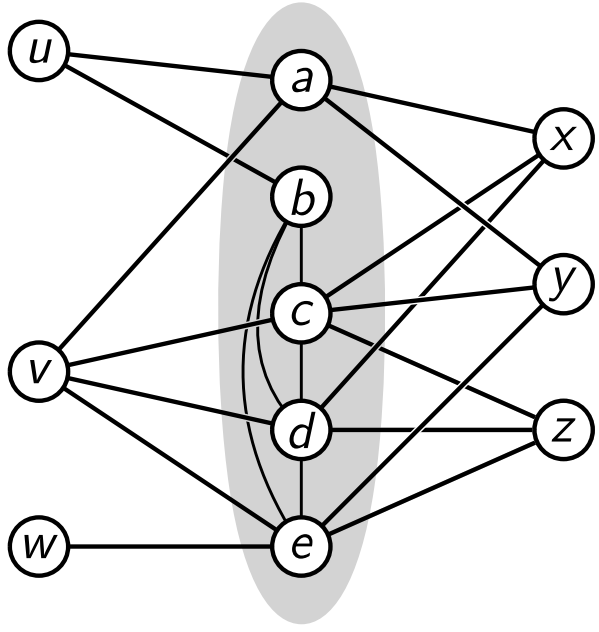
- Within VC: $O(k^2)$
- Between VC and IS: can be improved to $O(k^3)$

Improved kernel for 3-Coloring

Kernel: general idea

We have graph G, with vertex cover VC
The remaining vertices form independent set IS

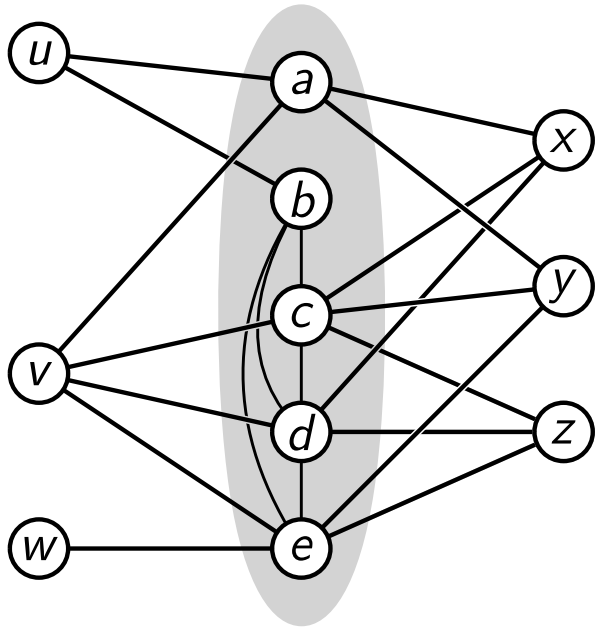
- IS may be large compared to VC
- Find more redundant vertices
  - Any coloring of $G - u$ can be extended to G
- Similarly, find redundant edges
- Improve on previous kernel



Kernel: general idea

We have graph G, with vertex cover VC
The remaining vertices form independent set IS

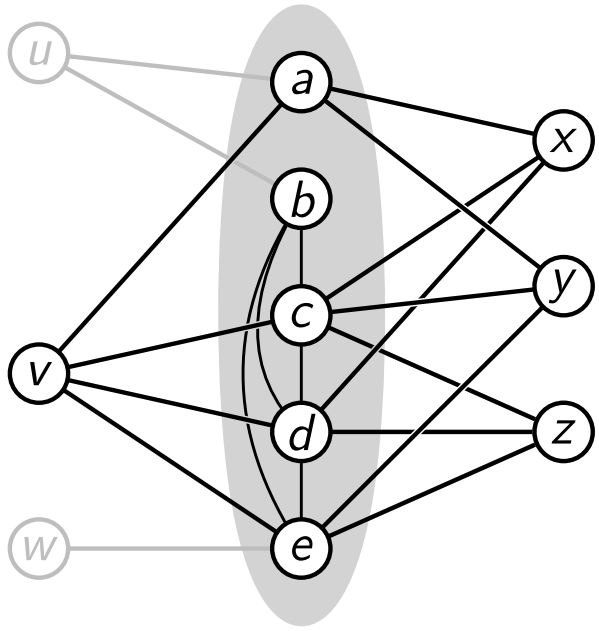
- IS may be large compared to VC
- Find more redundant vertices
  - Any coloring of $G - u$ can be extended to G
- Similarly, find redundant edges
- Improve on previous kernel



Kernel: general idea

We have graph G, with vertex cover VC
The remaining vertices form independent set IS

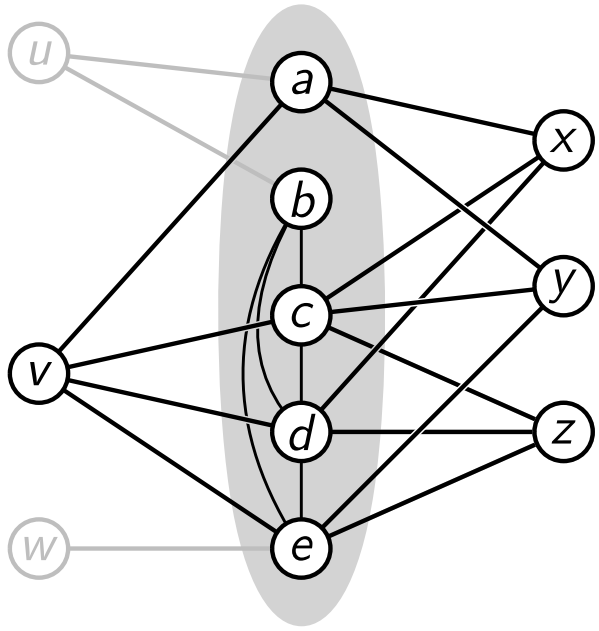
- IS may be large compared to VC
- Find more redundant vertices
  - Any coloring of $G - u$ can be extended to G
- Similarly, find redundant edges
- Improve on previous kernel



Kernel: general idea

We have graph G, with vertex cover VC
The remaining vertices form independent set IS

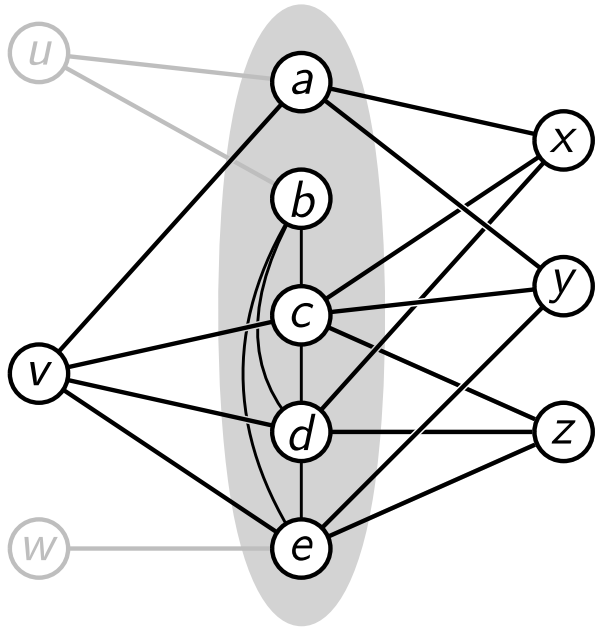
- IS may be large compared to VC
- Find more redundant vertices
  - Any coloring of $G - u$ can be extended to G
- Similarly, find redundant edges
- Improve on previous kernel



Kernel: general idea

We have graph G, with vertex cover VC
The remaining vertices form independent set IS

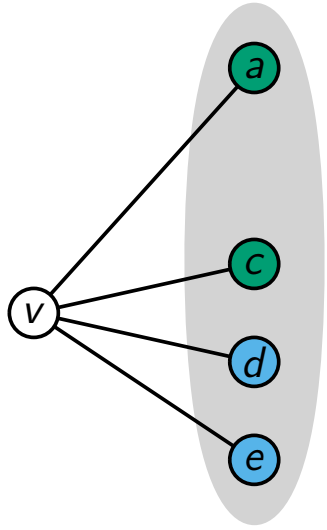
- IS may be large compared to VC
- Find more redundant vertices
  - Any coloring of $G - u$ can be extended to G
- Similarly, find redundant edges
- Improve on previous kernel



Kernel: general idea

Vertices in IS can be colored independently

- Each vertex in IS corresponds to a constraint
  - Neighborhood does not use all 3 colors
- Gives constraints on the coloring of VC
  - If some coloring of VC satisfies all constraints, it can be extended to IS
  - Previously, ensured by marking vertices



Alternatively:

For all $S \subseteq N(v)$ with $|S| = 3$, some color is used twice for S

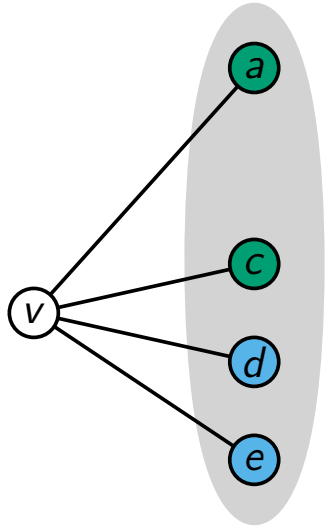
$\Updownarrow$

coloring can be extended to v

Kernel: general idea

Vertices in IS can be colored independently

- Each vertex in IS corresponds to a constraint
  - Neighborhood does not use all 3 colors
- Gives constraints on the coloring of VC
  - If some coloring of VC satisfies all constraints, it can be extended to IS
  - Previously, ensured by marking vertices



Alternatively:

For all $S \subseteq N(v)$ with $|S| = 3$, some color is used twice for S

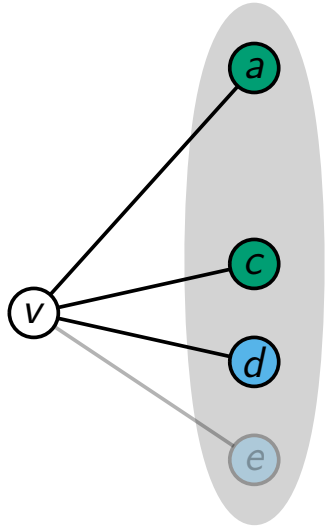
$$\Updownarrow$$

coloring can be extended to v

Kernel: general idea

Vertices in IS can be colored independently

- Each vertex in IS corresponds to a constraint
  - Neighborhood does not use all 3 colors
- Gives constraints on the coloring of VC
  - If some coloring of VC satisfies all constraints, it can be extended to IS
  - Previously, ensured by marking vertices



Alternatively:

For all $S \subseteq N(v)$ with $|S| = 3$, some color is used twice for S

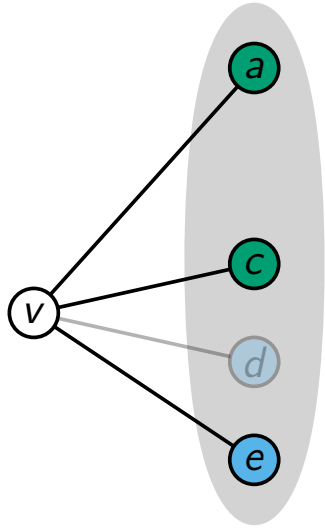
$$\Updownarrow$$

coloring can be extended to v

Kernel: general idea

Vertices in IS can be colored independently

- Each vertex in IS corresponds to a constraint
  - Neighborhood does not use all 3 colors
- Gives constraints on the coloring of VC
  - If some coloring of VC satisfies all constraints, it can be extended to IS
  - Previously, ensured by marking vertices



Alternatively:

For all $S \subseteq N(v)$ with $|S| = 3$, some color is used twice for S

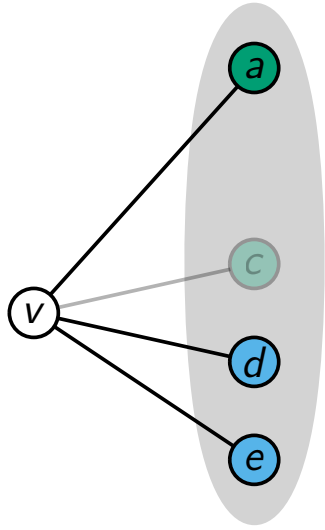
$$\Updownarrow$$

coloring can be extended to v

Kernel: general idea

Vertices in IS can be colored independently

- Each vertex in IS corresponds to a constraint
  - Neighborhood does not use all 3 colors
- Gives constraints on the coloring of VC
  - If some coloring of VC satisfies all constraints, it can be extended to IS
  - Previously, ensured by marking vertices



Alternatively:

For all $S \subseteq N(v)$ with $|S| = 3$, some color is used twice for S

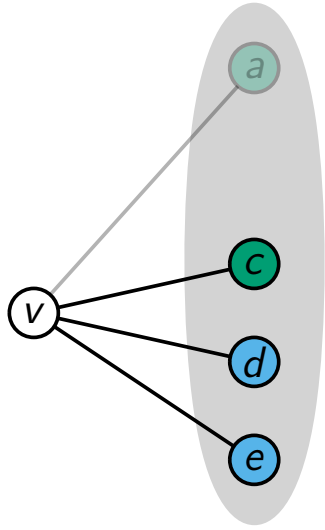
$$\Updownarrow$$

coloring can be extended to v

Kernel: general idea

Vertices in IS can be colored independently

- Each vertex in IS corresponds to a constraint
  - Neighborhood does not use all 3 colors
- Gives constraints on the coloring of VC
  - If some coloring of VC satisfies all constraints, it can be extended to IS
  - Previously, ensured by marking vertices



Alternatively:

For all $S \subseteq N(v)$ with $|S| = 3$, some color is used twice for S

$$\Updownarrow$$

coloring can be extended to v

Finding redundant vertices using constraints

Finding redundant constraints

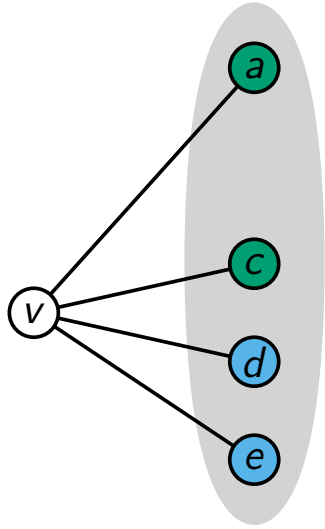
If constraints are given by degree-c polynomial equalities

- We have seen a sparsification!
  - c-Polynomial root CSP on k variables
- There are at most $O(k^c)$ relevant constraints

Modeling vertices as constraints

Polynomial equalities

- Create 3 boolean variables for each vertex in VC.
  - $a\,a\,a \in \{0,1\}$ indicate the color of a
- For each vertex v in IS, $S \subseteq N(v)$ with $|S| = 3$
  - Constraint: S does not use all 3 colors.



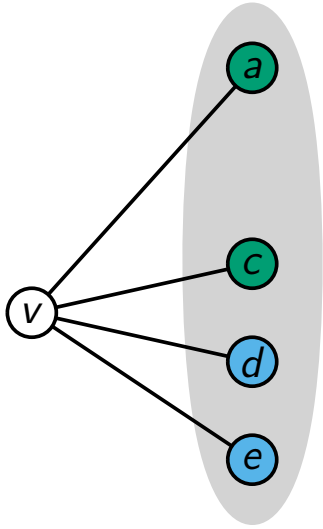
Which polynomial to use?

- Needs to have degree ≤ 2

Modeling vertices as constraints

Polynomial equalities

- Create 3 boolean variables for each vertex in VC.
  - $a\,a\,a \in \{0, 1\}$ indicate the color of a
- For each vertex v in IS, $S \subseteq N(v)$ with $|S| = 3$
  - Constraint: S does not use all 3 colors.



Which polynomial to use?

- Needs to have degree ≤ 2

Polynomial equalities for 3-Coloring

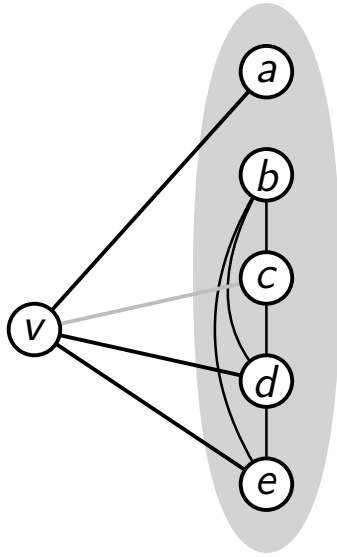
- 3 variables for each vertex in VC

Let $v \in \text{IS}$, for each $S \subseteq N(v) : |S| = 3$

- Polynomial equality of degree 2
  - For $S = \{a, d, e\}$:

$$a \wedge d + a \wedge e + d \wedge e +$$
$$a \wedge d + a \wedge e + d \wedge e +$$
$$a \wedge d + a \wedge e + d \wedge e \equiv_2 1$$

- Expresses: a,d, and e do not use all 3 colors
  - Three equal colors gives $3 \equiv_2 1$
  - Two equal colors gives 1
  - Three different colors gives 0



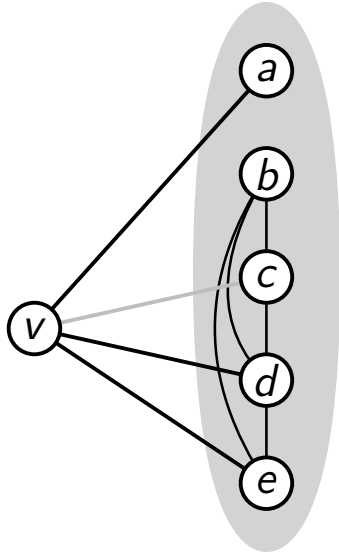
Polynomial equalities for 3-Coloring

- 3 variables for each vertex in VC

Let $v \in \text{IS}$, for each $S \subseteq N(v) : |S| = 3$

- Polynomial equality of degree 2
  - For $S = \{a, d, e\}$:

$$a \wedge d + a \wedge e + d \wedge e +$$
$$a \wedge d + a \wedge e + d \wedge e +$$
$$a \wedge d + a \wedge e + d \wedge e \equiv_2 1$$



- Expresses: a,d, and e do not use all 3 colors
  - Three equal colors gives $3 \equiv_2 1$
  - Two equal colors gives 1
  - Three different colors gives 0

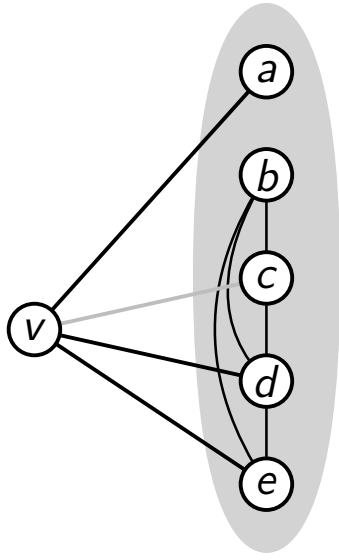
Polynomial equalities for 3-Coloring

- 3 variables for each vertex in VC

Let $v \in \text{IS}$, for each $S \subseteq N(v) : |S| = 3$

- Polynomial equality of degree 2
  - For $S = \{a, d, e\}$:

$$a \wedge d + a \wedge e + d \wedge e +$$
$$a \wedge d + a \wedge e + d \wedge e +$$
$$a \wedge d + a \wedge e + d \wedge e \equiv_2 1$$



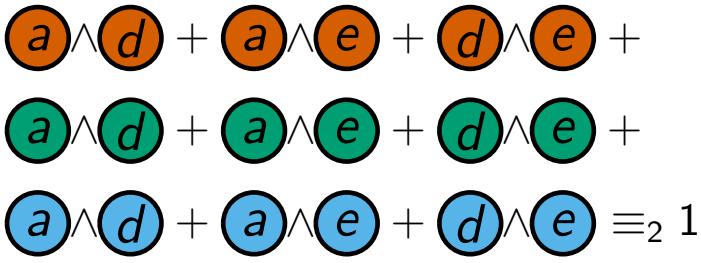
- Expresses: a,d, and e do not use all 3 colors
  - Three equal colors gives $3 \equiv_2 1$
  - Two equal colors gives 1
  - Three different colors gives 0

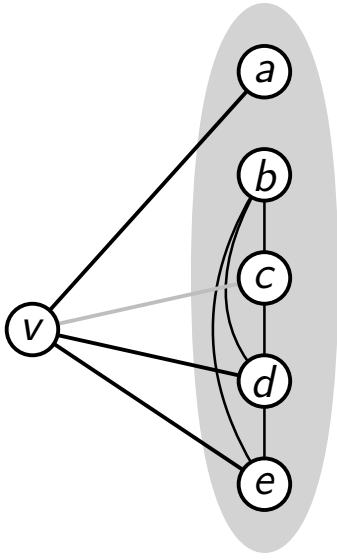
Polynomial equalities for 3-Coloring

- 3 variables for each vertex in VC 

Let $v \in \text{IS}$, for each $S \subseteq N(v) : |S| = 3$

- Polynomial equality of degree 2
  - For $S = \{a, d, e\}$:

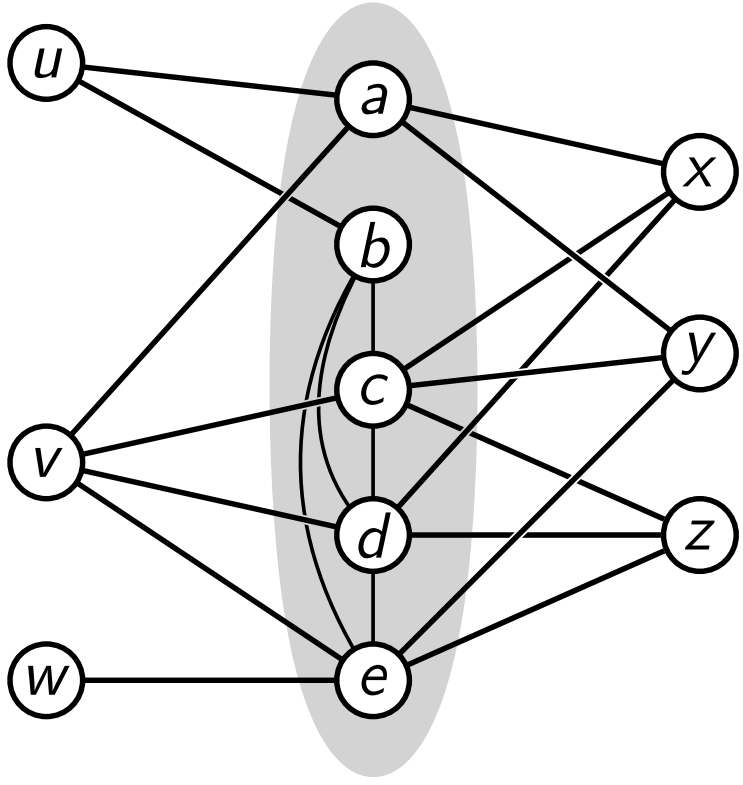




- Expresses: *a*,*d*, and *e* do not use all 3 colors
  - Three equal colors gives $3 \equiv_2 1$
  - Two equal colors gives 1
  - Three different colors gives 0

Kernel for 3-Coloring

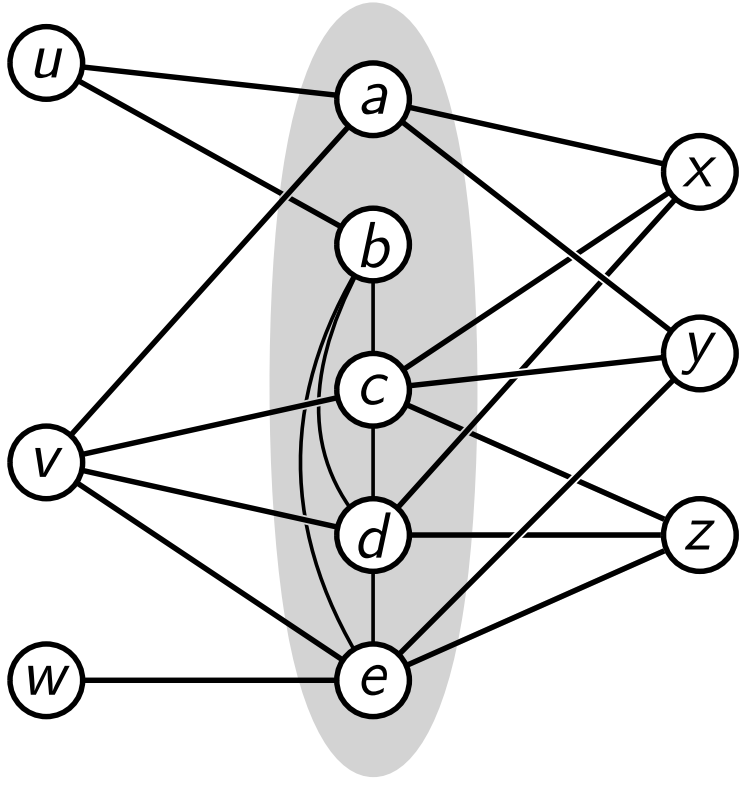
- Model vertices in IS by constraints
- Use Theorem to find subset of relevant constraints
- Keep only vertices and edges used for relevant constraints



u, w: $\emptyset$
v:
$C(a, c, d)$
$C(a, d, e)$
$C(c, d, e)$
$C(a, c, e)$
x:
$C(a, c, d)$
y:
$C(a, c, e)$
z:
$C(c, d, e)$

Kernel for 3-Coloring

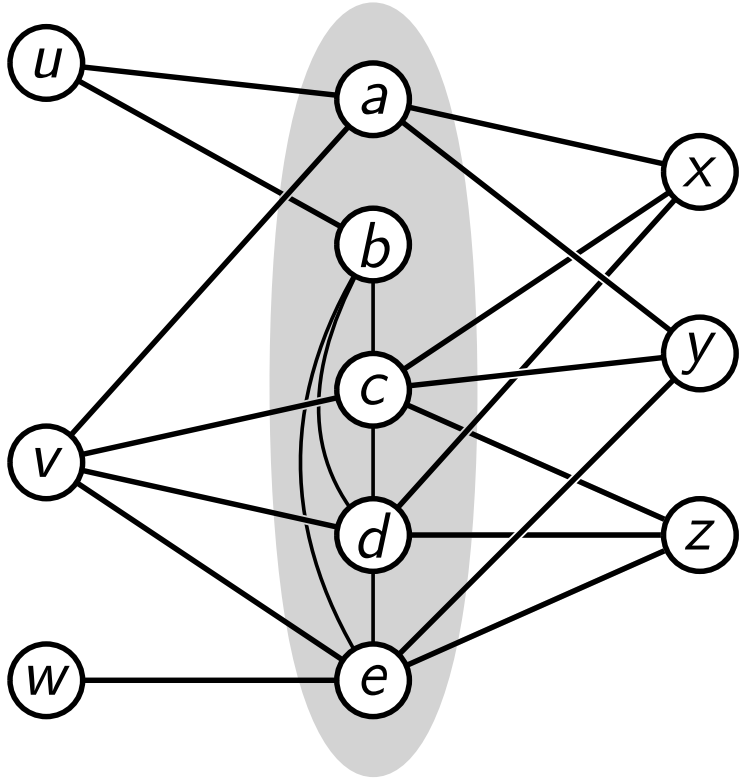
- Model vertices in IS by constraints
- Use Theorem to find subset of relevant constraints
- Keep only vertices and edges used for relevant constraints



u, w: $\emptyset$
v:
$C(a, c, d)$
$C(a, d, e)$
$C(c, d, e)$
$C(a, c, e)$
x:
$C(a, c, d)$
y:
$C(a, c, e)$
z:
$C(c, d, e)$

Kernel for 3-Coloring

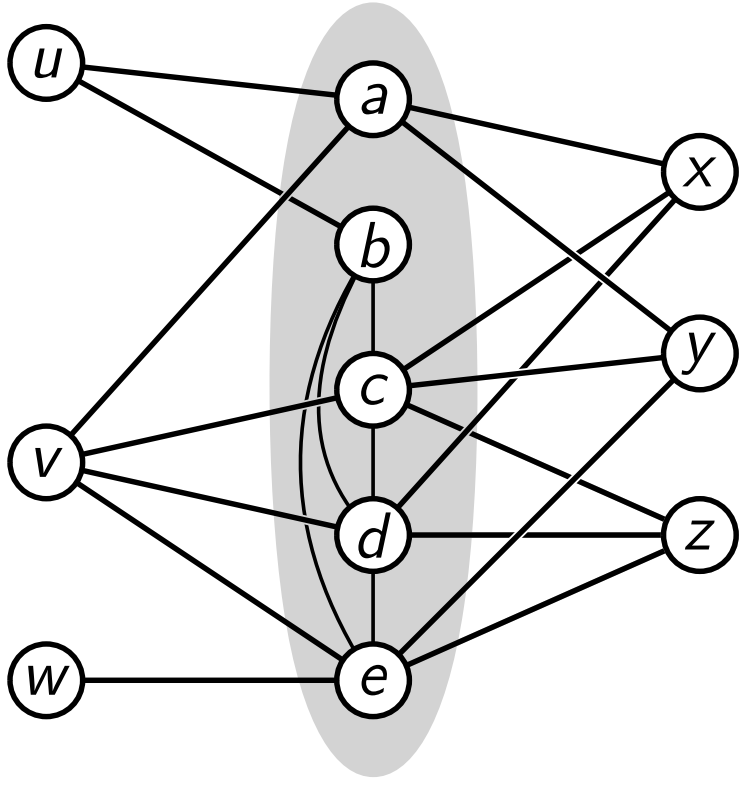
- Model vertices in IS by constraints
- Use Theorem to find subset of relevant constraints
- Keep only vertices and edges used for relevant constraints



u, w: $\emptyset$
v:
~~$C(a, c, d)$~~
$C(a, d, e)$
~~$C(c, d, e)$~~
~~$C(a, c, e)$~~
x:
$C(a, c, d)$
y:
$C(a, c, e)$
z:
~~$C(c, d, e)$~~

Kernel for 3-Coloring

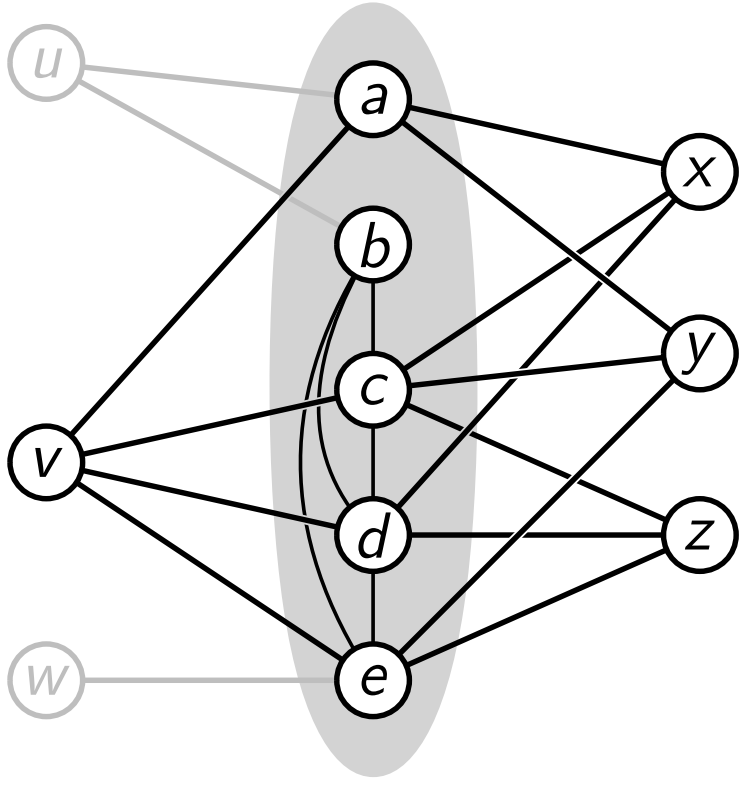
- Model vertices in IS by constraints
- Use Theorem to find subset of relevant constraints
- Keep only vertices and edges used for relevant constraints



u, w: $\emptyset$
v:
~~$C(a, c, d)$~~
$C(a, d, e)$
~~$C(c, d, e)$~~
~~$C(a, c, e)$~~
x:
$C(a, c, d)$
y:
$C(a, c, e)$
z:
~~$C(c, d, e)$~~

Kernel for 3-Coloring

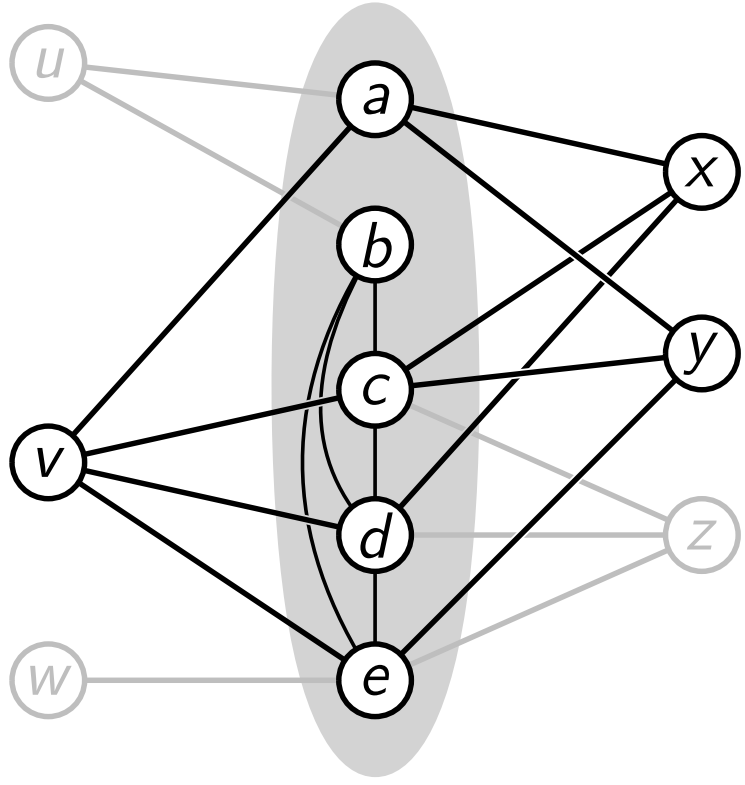
- Model vertices in IS by constraints
- Use Theorem to find subset of relevant constraints
- Keep only vertices and edges used for relevant constraints



u, w: $\emptyset$
v:
~~$C(a, c, d)$~~
$C(a, d, e)$
~~$C(c, d, e)$~~
~~$C(a, c, e)$~~
x:
$C(a, c, d)$
y:
$C(a, c, e)$
z:
~~$C(c, d, e)$~~

Kernel for 3-Coloring

- Model vertices in IS by constraints
- Use Theorem to find subset of relevant constraints
- Keep only vertices and edges used for relevant constraints



u, w: $\emptyset$

v:

~~$C(a, c, d)$~~

$C(a, d, e)$

~~$C(c, d, e)$~~

~~$C(a, c, e)$~~

x:

$C(a, c, d)$

y:

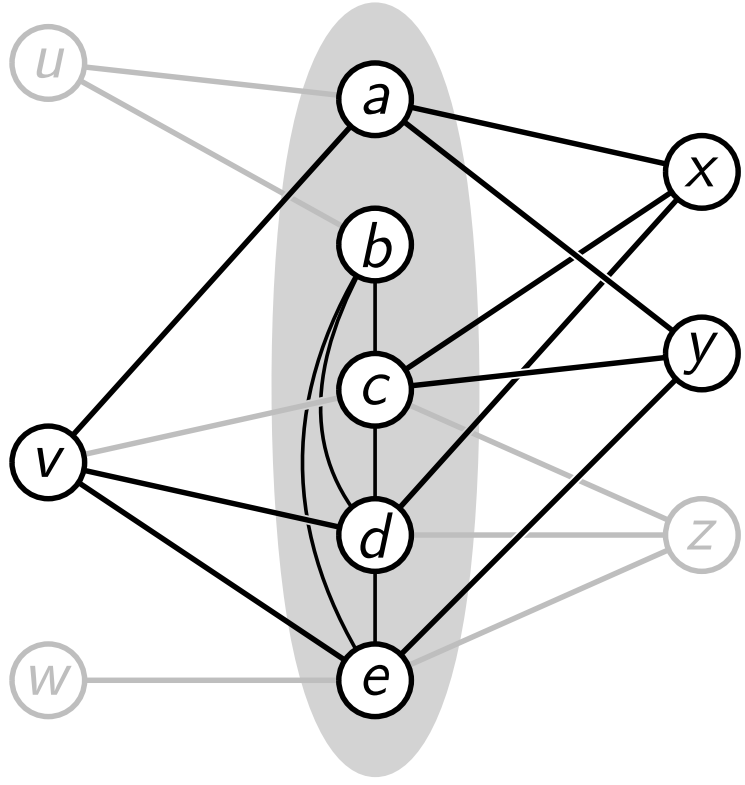
$C(a, c, e)$

z:

~~$C(c, d, e)$~~

Kernel for 3-Coloring

- Model vertices in IS by constraints
- Use Theorem to find subset of relevant constraints
- Keep only vertices and edges used for relevant constraints



u, w: $\emptyset$
v:
~~$C(a, c, d)$~~
$C(a, d, e)$
~~$C(c, d, e)$~~
~~$C(a, c, e)$~~
x:
$C(a, c, d)$
y:
$C(a, c, e)$
z:
~~$C(c, d, e)$~~

Kernel size

- Number of constraints $O(\#vars^{degree})$
  - $3 \cdot k$ variables
  - degree 2
- Number of constraints $O((3k)^2)$
- Constraint corresponds to ≤ 1 vertex and ≤ 3 edges
- $O(k^2)$ vertices and edges

Theorem

There exists a polynomial-time algorithm that when given graph G with vertex cover S of size k, outputs graph G' such that

- G' is 3-Colorable if and only if G is 3-Colorable, and
- G' has $O(k^2)$ vertices and edges.

q-Coloring

Coloring with q colors

Theorem

q-Coloring parameterized by Vertex Cover has a kernel with $O(k^{q-1})$ vertices and edges.

Can we do better?

Theorem

q-Coloring parameterized by Vertex Cover has no kernel of bitsize $O(k^{q-1-\varepsilon})$, unless NP $\subseteq$ coNP/poly.

q-Coloring

Coloring with q colors

Theorem

q-Coloring parameterized by Vertex Cover has a kernel with $O(k^{q-1})$ vertices and edges.

Can we do better?

Theorem

q-Coloring parameterized by Vertex Cover has no kernel of bitsize $O(k^{q-1-\varepsilon})$, unless NP $\subseteq$ coNP/poly.

Conclusion

Summary

- Sparsification upper and lower bounds for a number of CSPs
  - Using low-degree polynomial equalities
- Application to q-Coloring parameterized by Vertex Cover
  - Kernel with $O(k^{q-1})$ vertices and edges

Open problems

- Full classification of sparsifiability of CSPs
  - Non-boolean domain?
- Sparsifiability of H-Coloring
  - How is this influenced by the graph structure?

Conclusion

Summary

- Sparsification upper and lower bounds for a number of CSPs
  - Using low-degree polynomial equalities
- Application to q-Coloring parameterized by Vertex Cover
  - Kernel with $O(k^{q-1})$ vertices and edges

Open problems

- Full classification of sparsifiability of CSPs
  - Non-boolean domain?
- Sparsifiability of H-Coloring
  - How is this influenced by the graph structure?

Thank you!