Über Parameterisierte Aufzählkomplexität

Alexander Temper

Seminar der Komplexitätstheorie SS2024

Outline

Outline

NP-Schwere ist für generelle Instanzen

- NP-Schwere betrachtet die schwierigsten generellen Instanzen.
- In der Praxis sind sehr viele Probleme NP-schwer.
- Soll man diese Probleme aufgeben?

Praktische Probleme weisen Struktur auf I

Liste Nr.	Für die gewählte Partei im Kreis ein X einsetzen!	Kurz-bezeichnung	Parteibezeichnung	Bezeichnung einer Bewerberin oder eines Bewerbers (Name und/ oder Reihungsnummer) durch die Wählerin oder durch den Wähler
1	◯	ÖVP	Österreichische Volkspartei	
2	◯	SPÖ	Sozialdemokratische Partei Österreichs	
3	◯	FPÖ	Freiheitliche Partei Österreichs (FPÖ) – Die Freiheitlichen	
4	◯	GRÜNE	Die Grünen – Die Grüne Alternative	
5	◯	NEOS	NEOS – Das Neue Europa	
6	◯	DNA	DNA – Demokratisch – Neutral – Authentisch	
7	◯	KPÖ	Kommunistische Partei Österreichs – KPÖ Plus	

- Präferenzprofile:
  - Wählerinnen $w_1, \ldots w_n$,
  - Alternativen $a_1, \ldots, a_m$ und
  - Präferenzen

$$w_1 : a_1 \succ a_2 \succ \ldots$$

$$w_2 : a_4 \succ a_n \succ \ldots.$$

- Unzählige NP-schwere Probleme über Präferenzprofile, aber
- Anzahl der Alternativen m oft klein!

Praktische Probleme weisen Struktur auf I

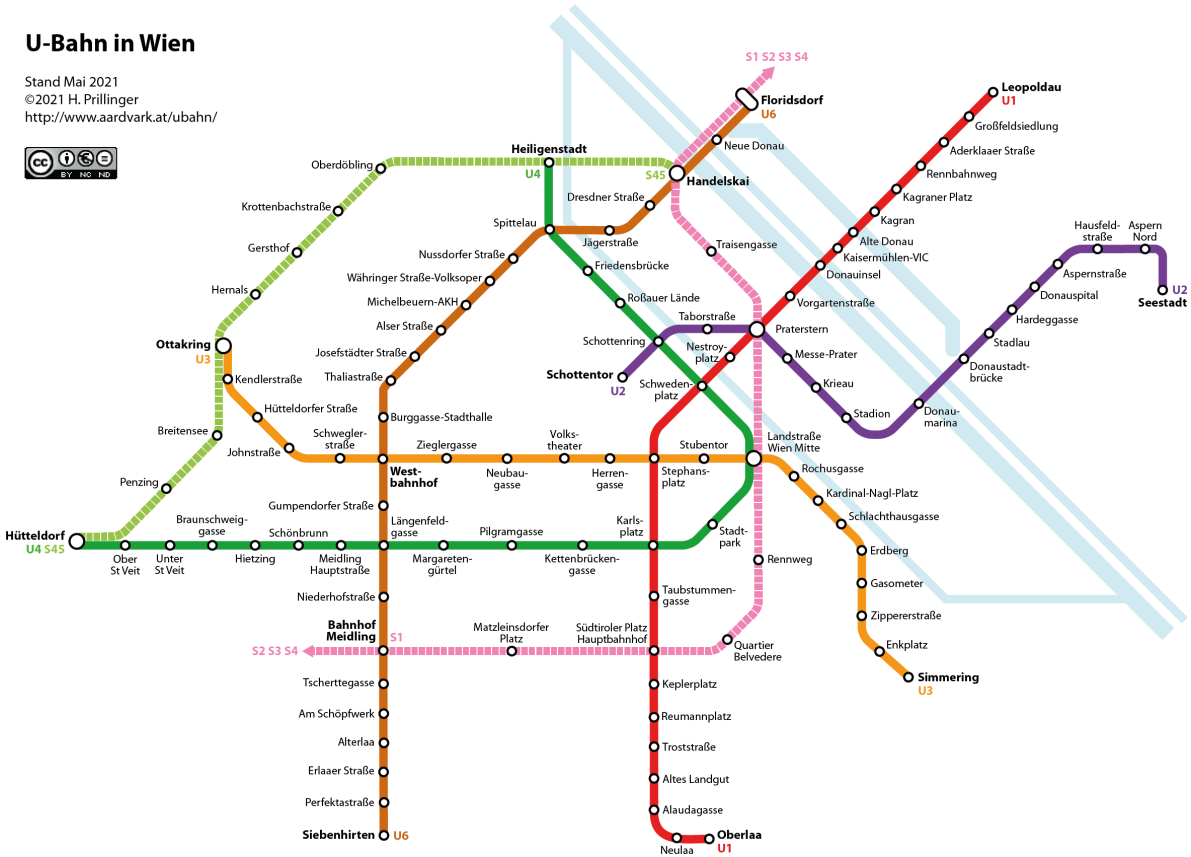
Liste Nr.	Für die gewählte Partei im Kreis ein X einsetzen!	Kurz-bezeichnung	Parteibezeichnung	Bezeichnung einer Bewerberin oder eines Bewerbers (Name und/ oder Reihungsnummer) durch die Wählerin oder durch den Wähler
1	◯	ÖVP	Österreichische Volkspartei	
2	◯	SPÖ	Sozialdemokratische Partei Österreichs	
3	◯	FPÖ	Freiheitliche Partei Österreichs (FPÖ) – Die Freiheitlichen	
4	◯	GRÜNE	Die Grünen – Die Grüne Alternative	
5	◯	NEOS	NEOS – Das Neue Europa	
6	◯	DNA	DNA – Demokratisch – Neutral – Authentisch	
7	◯	KPÖ	Kommunistische Partei Österreichs – KPÖ Plus	

- Präferenzprofile:
  - Wählerinnen $w_1, \ldots w_n$,
  - Alternativen $a_1, \ldots, a_m$ und
  - Präferenzen

$$w_1 : a_1 \succ a_2 \succ \ldots$$
$$w_2 : a_4 \succ a_n \succ \ldots.$$

- Unzählige NP-schwere Probleme über Präferenzprofile, aber
- Anzahl der Alternativen m oft klein!

Praktische Probleme weisen Struktur auf II



U-Bahn Netzwerke haben sehr viel Struktur: niedrige Grade, wenige wichtige Knoten, oft "fast" planar.

Outline

Was ist ein parameterisiertes Problem?

Definition (Parameterisiertes Problem)

Sei X eine Problemklasse.

- Funktion $p : X \to \mathbb{N}$ ist eine Parameterisierung.
- Das Paar (X, p) ist ein parameterisiertes Problem.
- Das Paar $(x, p(x))$ ist eine Instanz von (X, p). Oft $k := p(x)$.

Beispiel

- Präferenzprofile parameterisiert durch Anzahl Alternativen
- Graphen parameterisiert durch deren maximalen Grad
- Probleme parameterisiert durch die maximale Lösungsgröße

Was ist Fixed Parameter Tractability (FPT)?

Definition (Fixed Parameter Tractable, [1])

Ein parameterisiertes Problem (X, p) ist fixed parameter tractable wenn ein Algorithmus $x \in X$ in

$$f(p(x)) \cdot \text{poly}(|x|)$$

löst. (f beliebig)

Triviales Beispiel

3-SAT parameterisiert durch die Anzahl Variablen n ist FPT: alle 2^n Zuweisungen via brute-force prüfen.

Outline

Kernelization

Intuition

Kann eine Probleminstanz x so klein gemacht werden, sodass Brute-forcing ein FPT-Algorithmus ist?

Definition (Kernelization)

Eine Kernelization $K : X \to X$ für (X, p) ist eine Reduktion von $(x, k) \in (X, p)$ zu einem Kernel $K(x) \in (X, p')$ sodass

- die Berechnung des Kernels polynomielle Zeit dauert,
- der Kernel äquivalent zu x ist und
- Größen beschränkt sind: $|K(x)| + p'(x) \leq h(p(x))$, h beliebig.

Kernelization

Intuition

Kann eine Probleminstanz x so klein gemacht werden, sodass Brute-forcing ein FPT-Algorithmus ist?

Definition (Kernelization)

Eine Kernelization $K : X \to X$ für (X, p) ist eine Reduktion von $(x, k) \in (X, p)$ zu einem Kernel $K(x) \in (X, p')$ sodass

- die Berechnung des Kernels polynomielle Zeit dauert,
- der Kernel äquivalent zu x ist und
- Größen beschränkt sind: $|K(x)| + p'(x) \leq h(p(x))$, h beliebig.

Kernelization

Intuition

Kann eine Probleminstanz x so klein gemacht werden, sodass Brute-forcing ein FPT-Algorithmus ist?

Definition (Kernelization)

Eine Kernelization $K : X \to X$ für (X, p) ist eine Reduktion von $(x, k) \in (X, p)$ zu einem Kernel $K(x) \in (X, p')$ sodass

- die Berechnung des Kernels polynomielle Zeit dauert,
- der Kernel äquivalent zu x ist und
- Größen beschränkt sind: $|K(x)| + p'(x) \leq h(p(x))$, h beliebig.

Beispiel: k-Vertex Cover

Definition (Vertex Cover)

- Sei $G = (V, E)$ ein Graph.
- Teilmenge der Knoten $V' \subset V$ ist Vertex Cover für G wenn jede Kante zumindest einen Endpunkt in V' hat.
- k-Vertex Cover wenn zusätzlich $|V'| \leq k$.

Beispiel

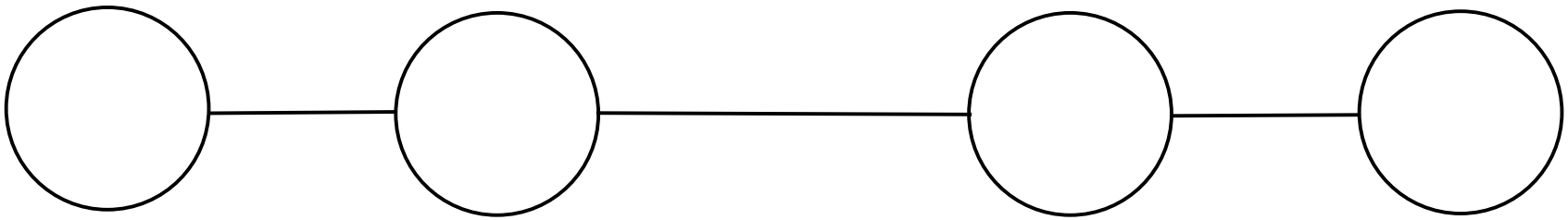
Beispiel: k-Vertex Cover

Definition (Vertex Cover)

- Sei $G = (V, E)$ ein Graph.
- Teilmenge der Knoten $V' \subset V$ ist Vertex Cover für G wenn jede Kante zumindest einen Endpunkt in V' hat.
- k-Vertex Cover wenn zusätzlich $|V'| \leq k$.

Beispiel

Vertex Cover mit $k \leq 2$?



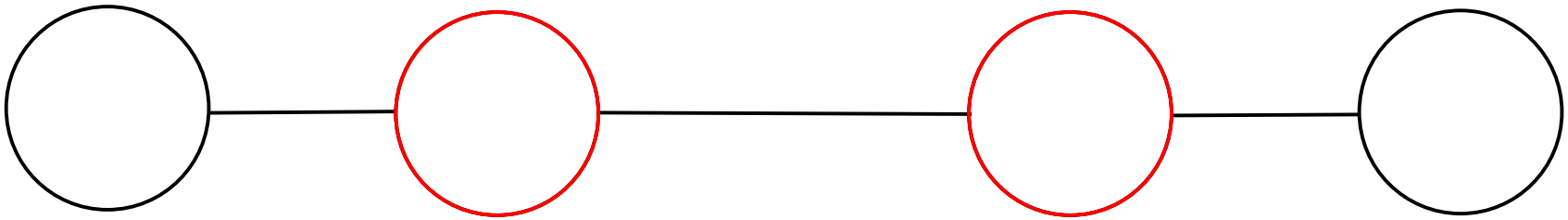
Beispiel: k-Vertex Cover

Definition (Vertex Cover)

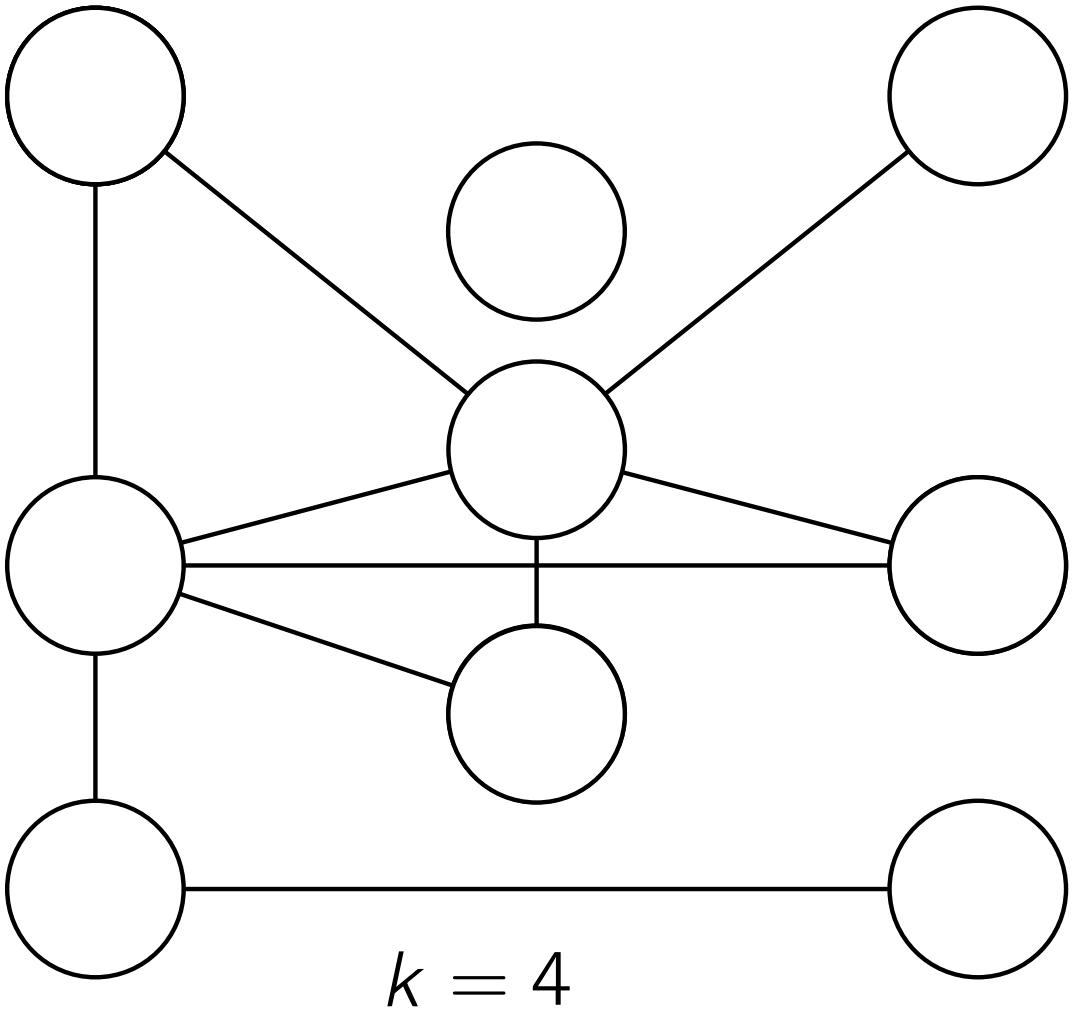
- Sei $G = (V, E)$ ein Graph.
- Teilmenge der Knoten $V' \subset V$ ist Vertex Cover für G wenn jede Kante zumindest einen Endpunkt in V' hat.
- k-Vertex Cover wenn zusätzlich $|V'| \leq k$.

Beispiel

Vertex Cover mit $k \leq 2$?



Beispiel: k-Vertex Cover Kernelization I



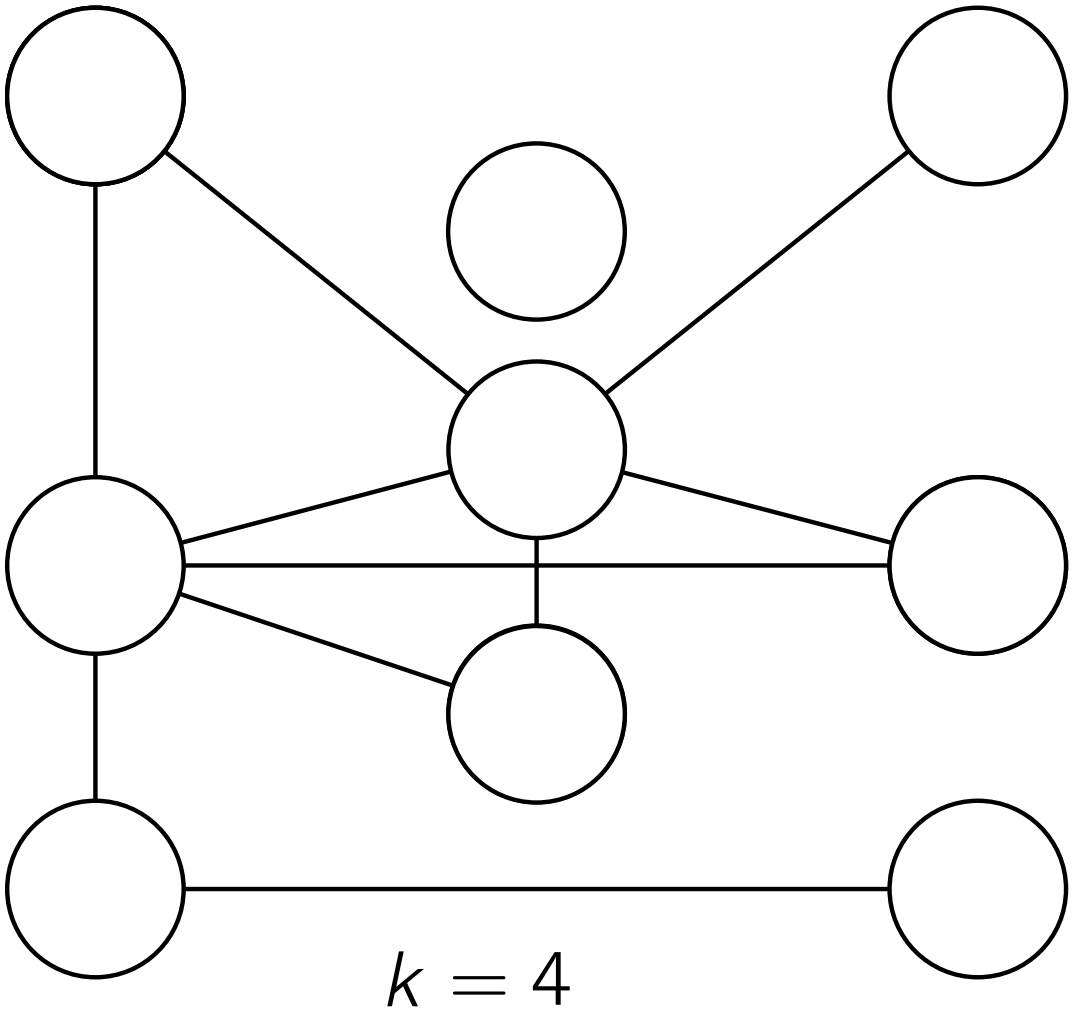
Beispiel

Finde Vertex Cover V' mit Größe $k \leq 4$.

Beobachtungen

- Isolierte Knoten können entfernt werden.
- Knoten mit Grad $> k$ müssen in V' sein.

Beispiel: k-Vertex Cover Kernelization I



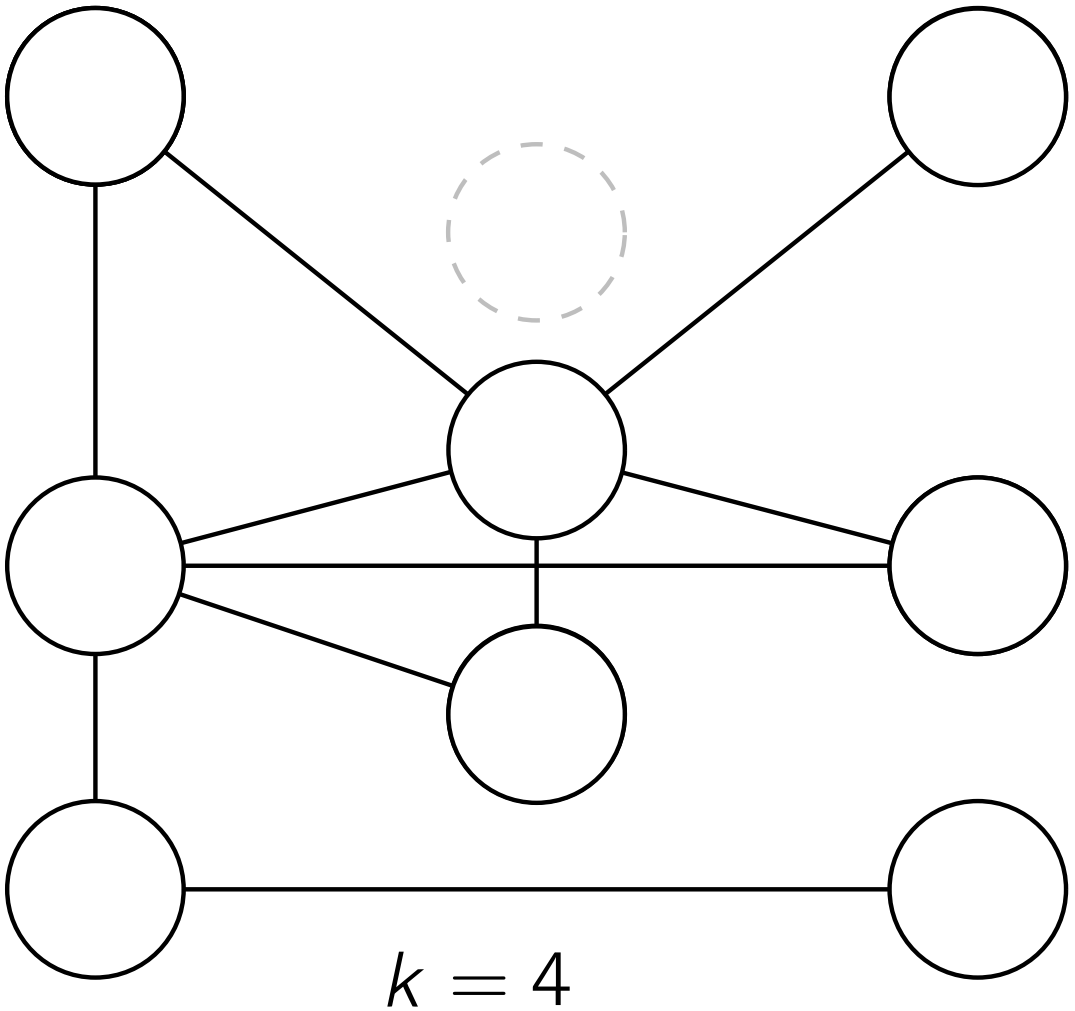
Beispiel

Finde Vertex Cover V' mit Größe $k \leq 4$.

Beobachtungen

- Isolierte Knoten können entfernt werden.
- Knoten mit Grad $> k$ müssen in V' sein.

Beispiel: k-Vertex Cover Kernelization I



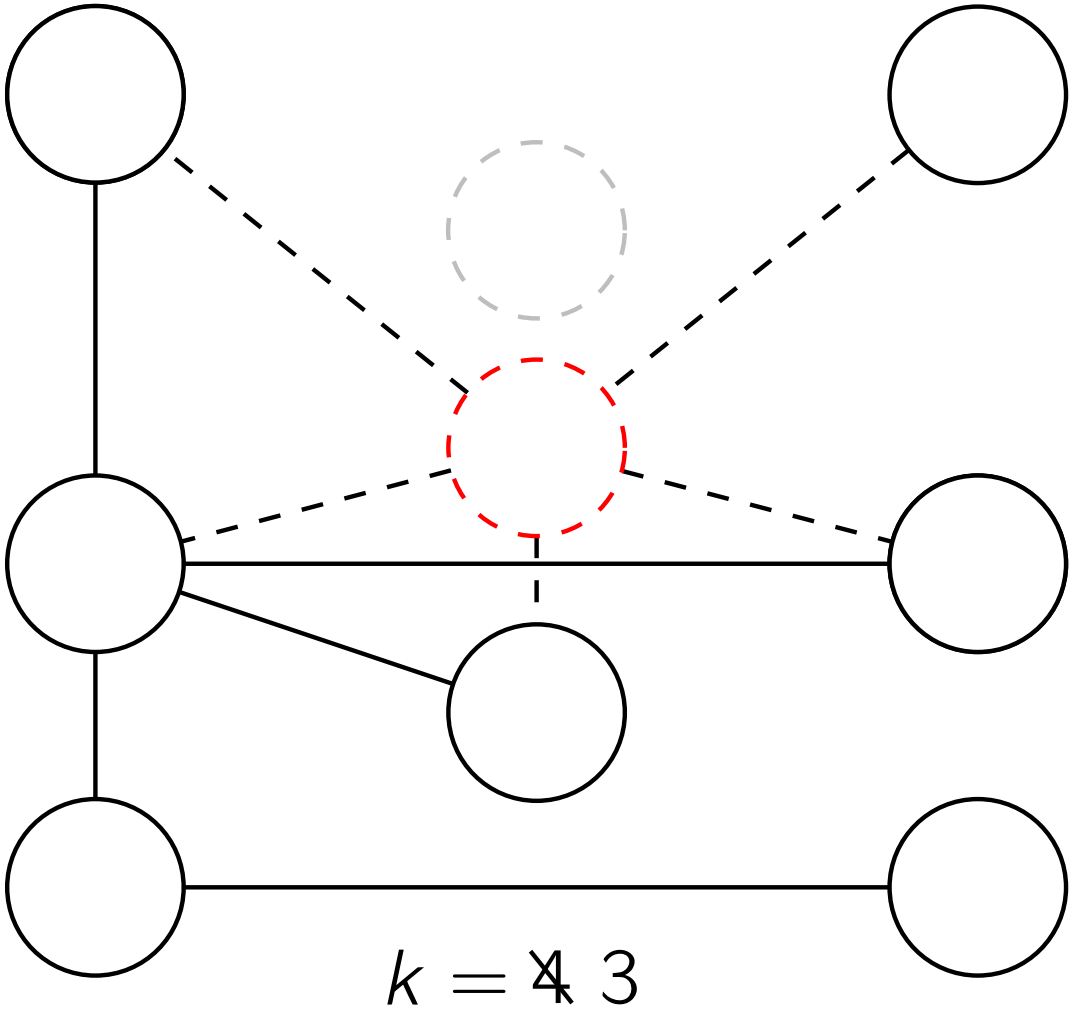
Beispiel

Finde Vertex Cover V' mit Größe $k \leq 4$.

Beobachtungen

- Isolierte Knoten können entfernt werden.
- Knoten mit Grad $> k$ müssen in V' sein.

Beispiel: k-Vertex Cover Kernelization I



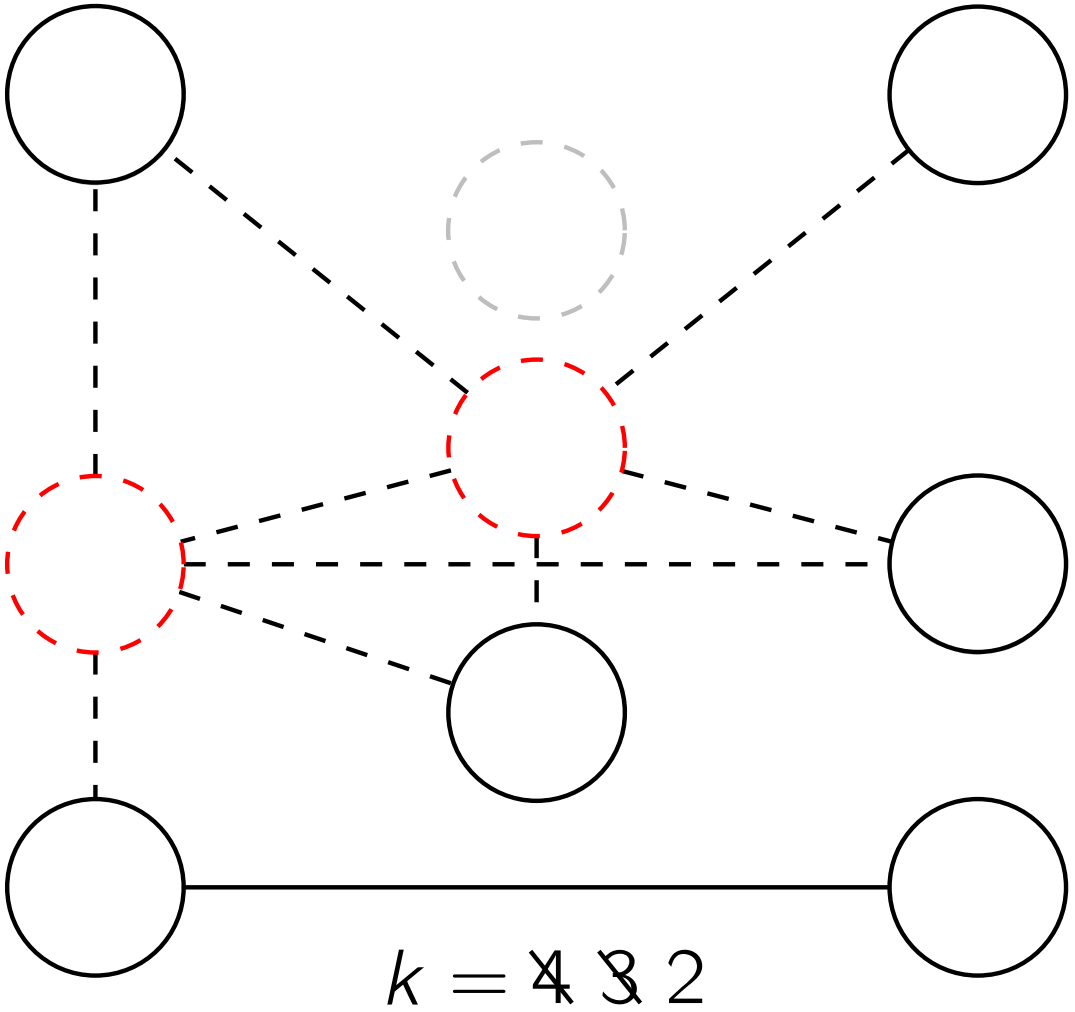
Beispiel

Finde Vertex Cover V' mit Größe $k \leq 4$.

Beobachtungen

- Isolierte Knoten können entfernt werden.
- Knoten mit Grad $> k$ müssen in V' sein.

Beispiel: k-Vertex Cover Kernelization I



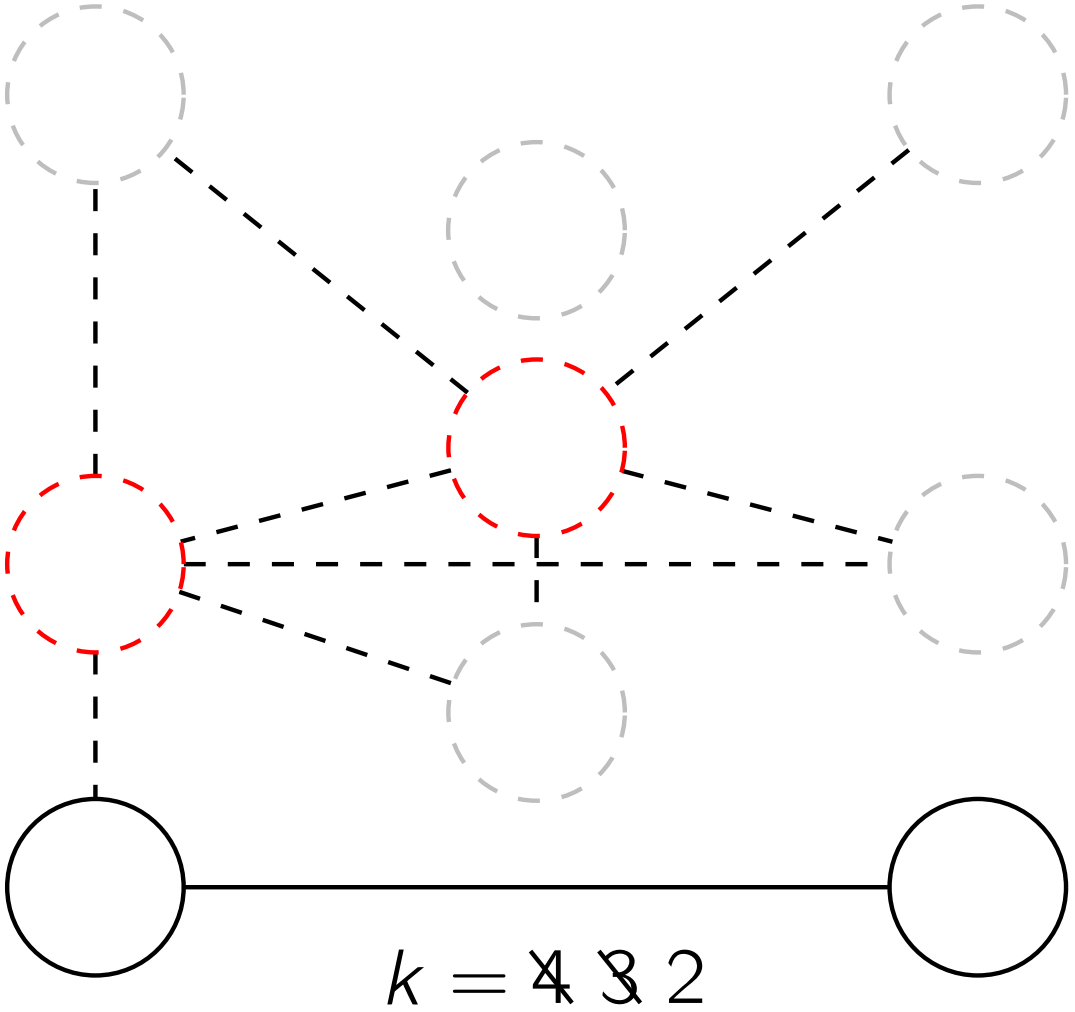
Beispiel

Finde Vertex Cover V' mit Größe $k \leq 4$.

Beobachtungen

- Isolierte Knoten können entfernt werden.
- Knoten mit Grad $> k$ müssen in V' sein.

Beispiel: k-Vertex Cover Kernelization I



Beispiel

Finde Vertex Cover V' mit Größe $k \leq 4$.

Beobachtungen

- Isolierte Knoten können entfernt werden.
- Knoten mit Grad $> k$ müssen in V' sein.

Beispiel: Vertex Cover Kernelization II

Satz

Die "exhaustive" Anwendung der Reduktionsregeln

1. *Entferne isolierte Knoten.*
2. *Falls ein Knoten Grad höher als k entferne ihn und seine Kanten und setze $k := k - 1$.*

ist eine Kernelization für Vertex Cover *parameterisiert durch die Lösungsgröße.*

Beweis.

- Läuft in polynomieller Zeit: maximal $|V|$ Reduktionschritte.
- Beide Regeln führen zu äquivalenten Instanzen.
- Danach haben Knoten Grad zwischen 1 und $k' < k$. Wenn mehr als k'^2 Kanten, gib NEIN-Instanz aus, sonst reduzierte Instanz mit $O(k'^2)$ Knoten.

Kernelization charakterisiert FPT

Satz (Folklore [1])

Die folgenden Sätze sind äquivalent:

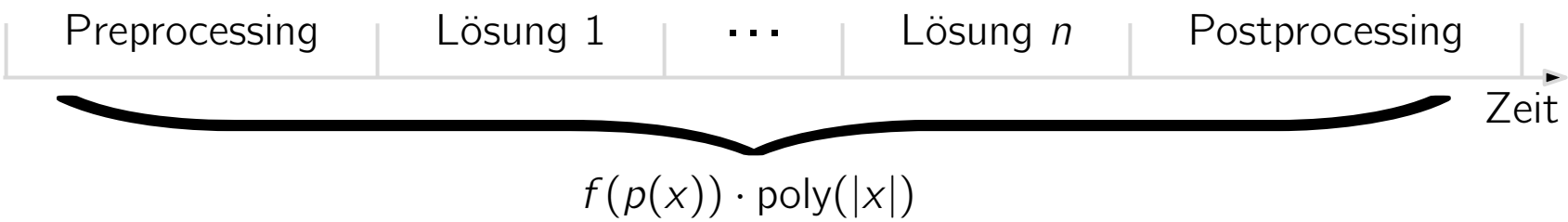
1. (X, p) *hat einen FPT-Algorithmus.*
2. *Es gibt eine Kernelization für* (X, p).

Outline

FPT Aufzählprobleme

Definition (FPT Aufzählalgorithmus)

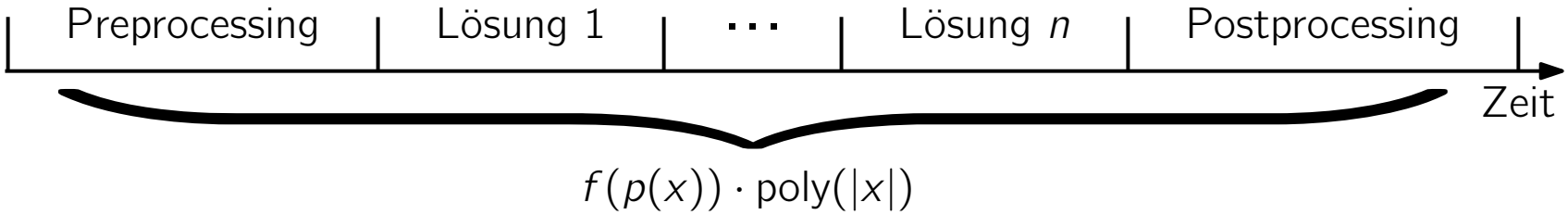
Ein FPT Aufzählalgorithmus listet alle Lösungen für (X, p) in Zeit $f(p(x)) \cdot \text{poly}(|x|)$.



FPT Aufzählprobleme

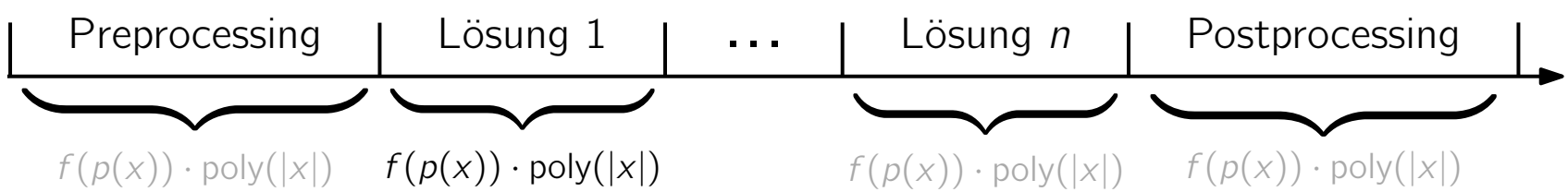
Definition (FPT Aufzählalgorithmus)

Ein FPT Aufzählalgorithmus listet alle Lösungen für (X, p) in Zeit $f(p(x)) \cdot \text{poly}(|x|)$.



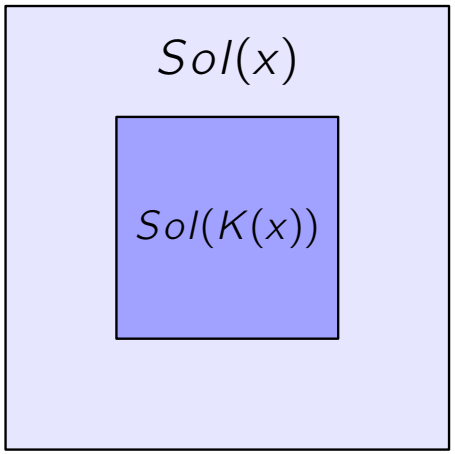
Definition (delayFPT Aufzählalgorithmus [2])

Ein delayFPT Aufzählalgorithmus listet alle Lösungen für (X, p), und zwischen jeder Lösung vergeht $f(p(x)) \cdot \text{poly}(|x|)$ Zeit.



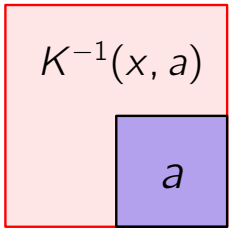
Outline

enum-Kernelization



Intuition

Können die Lösungen zu (x, k) von den Lösungen des Kernels $K(x)$ aufgebaut werden?



enum-Kernelization

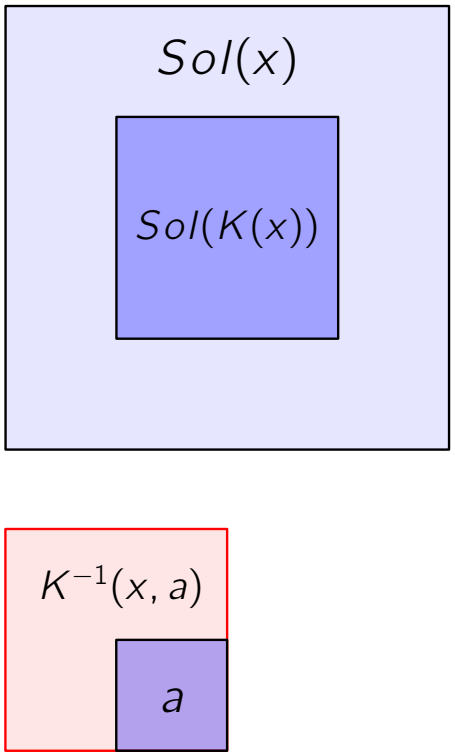
Definition (enum-Kernelization [2])

Besteht aus

- Kernelization K
- Algorithmus K^{-1}, der Probleminstanz x und eine Lösung $y \in Sol(K(x))$ zu Lösungen von x mappt

sodass

- $K^{-1}(x, y_1) \cap K^{-1}(x, y_2) = \emptyset$,
- $\cup_{y \in Sol(K(x))} K^{-1}(x, y) = Sol(x)$ und
- K^{-1} ein delayFPT Aufzählalgorithmus ist.



enum-Kernelization

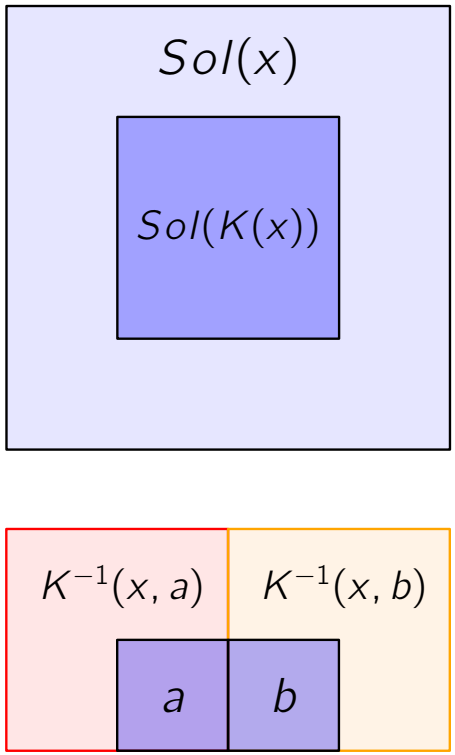
Definition (enum-Kernelization [2])

Besteht aus

- Kernelization K
- Algorithmus K^{-1}, der Probleminstanz x und eine Lösung $y \in Sol(K(x))$ zu Lösungen von x mappt

sodass

- $K^{-1}(x, y_1) \cap K^{-1}(x, y_2) = \emptyset$,
- $\cup_{y \in Sol(K(x))} K^{-1}(x, y) = Sol(x)$ und
- K^{-1} ein delayFPT Aufzählalgorithmus ist.



enum-Kernelization

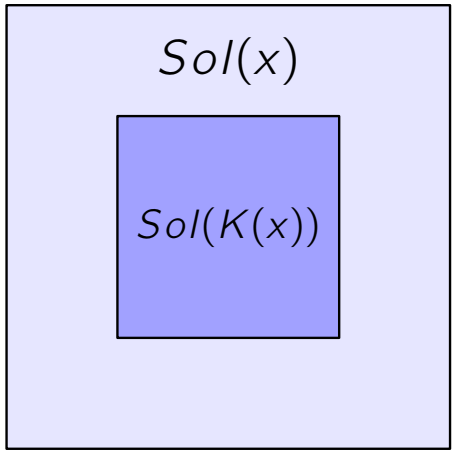
Definition (enum-Kernelization [2])

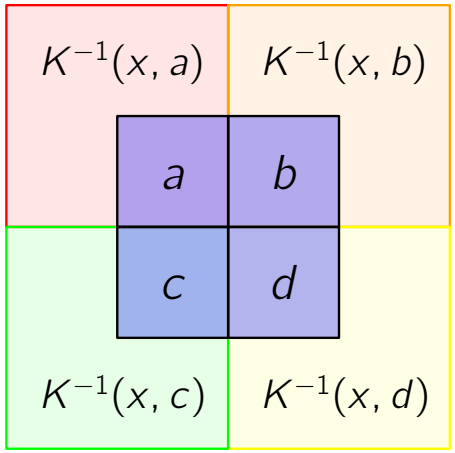
Besteht aus

- Kernelization K
- Algorithmus K^{-1}, der Probleminstanz x und eine Lösung $y \in Sol(K(x))$ zu Lösungen von x mappt

sodass

- $K^{-1}(x, y_1) \cap K^{-1}(x, y_2) = \emptyset$,
- $\cup_{y \in \text{Sol}(K(x))} K^{-1}(x, y) = Sol(x)$ und
- K^{-1} ein delayFPT Aufzählalgorithmus ist.





enum-Kernelization

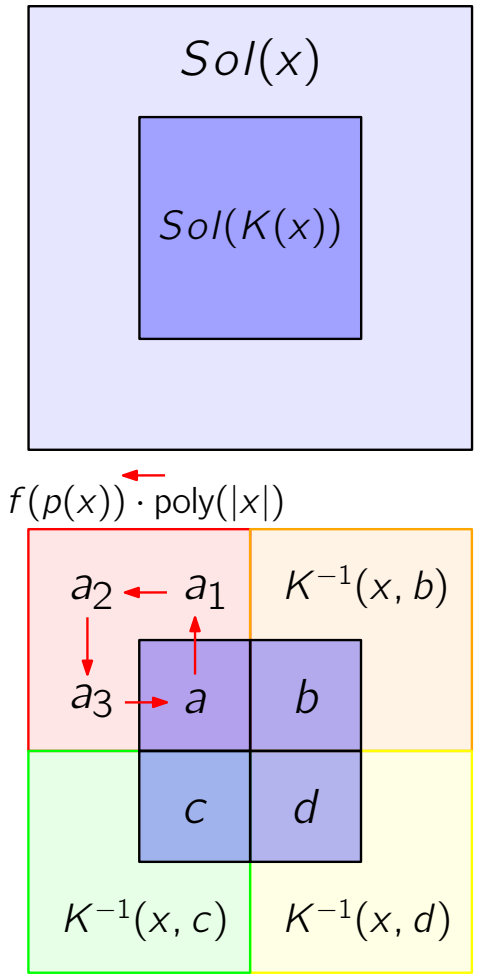
Definition (enum-Kernelization [2])

Besteht aus

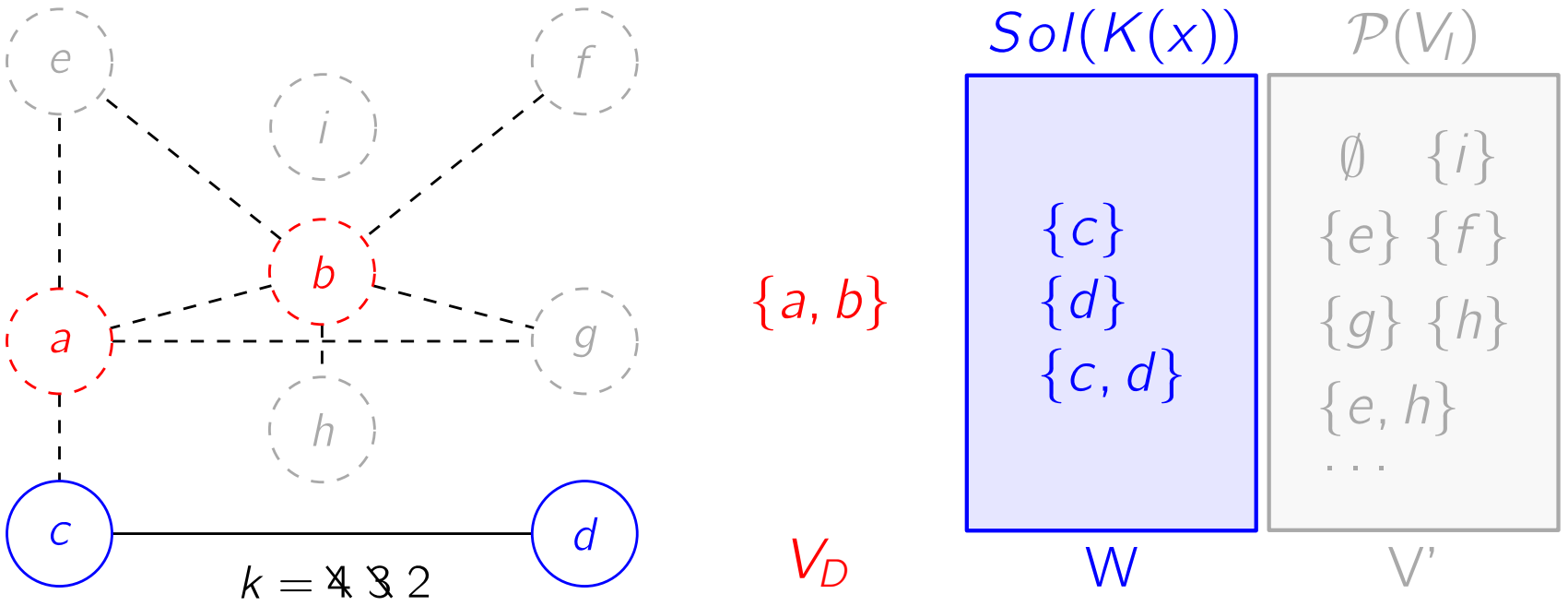
- Kernelization K
- Algorithmus K^{-1}, der Probleminstanz x und eine Lösung $y \in Sol(K(x))$ zu Lösungen von x mappt

sodass

- $K^{-1}(x, y_1) \cap K^{-1}(x, y_2) = \emptyset$,
- $\cup_{y \in \text{Sol}(K(x))} K^{-1}(x, y) = Sol(x)$ und
- K^{-1} ein delayFPT Aufzählalgorithmus ist.



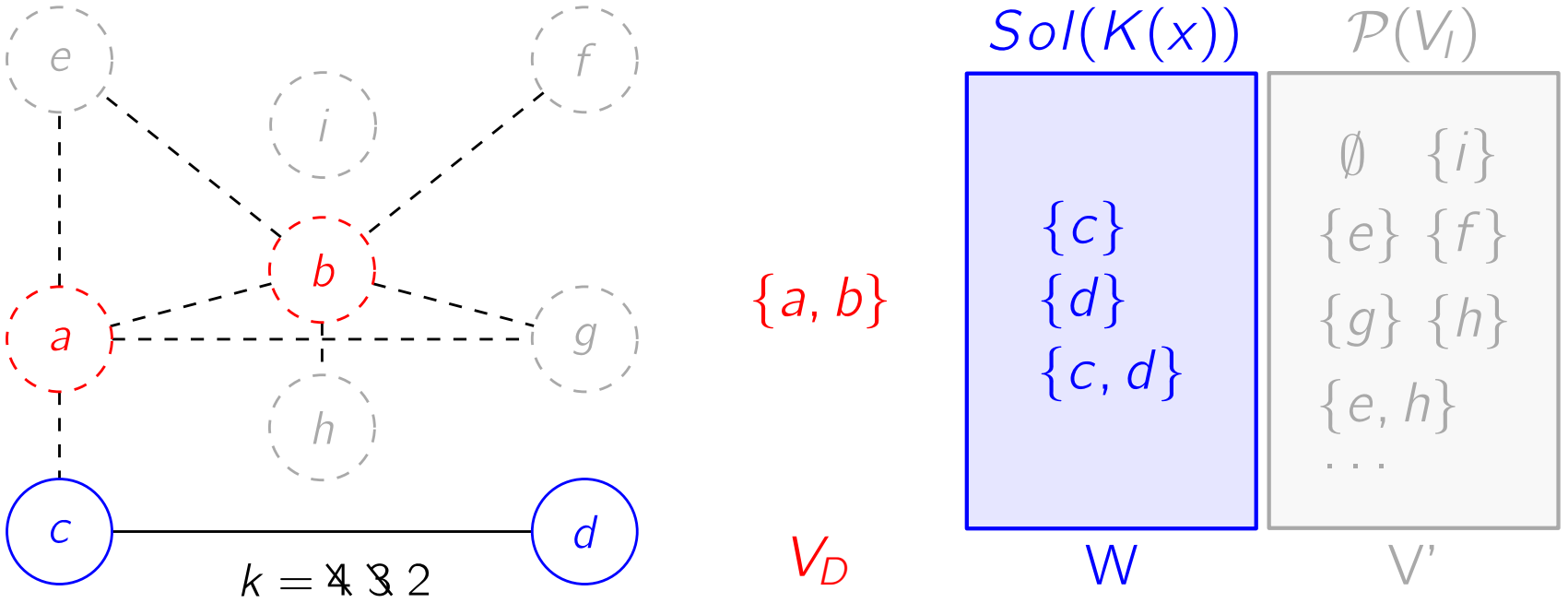
enum-Kernelization für All-Vertex Cover



- Es gibt ein K^{-1} für die vorherige Kernelization:

$$K^{-1}(G, W) := \{W \cup V_D \cup V' | V' \subset V_I, W \cup V_D \cup V' \leq k\}$$

enum-Kernelization für All-Vertex Cover



- Verschiedene W geben disjunkte Lösungsmengen.
- Die Lösungsmengen $K^{-1}(G, W)$ vereint über alle W sind alle Lösungen.
- Bei gegebenenem W $|V_D| + |W|$ berechnen und alle V' einer bestimmten Menge berechnen ist delayFPT.

Enum-Kernelization charakterisiert delayFPT I

Satz

Die folgenden Sätze sind äquivalent:

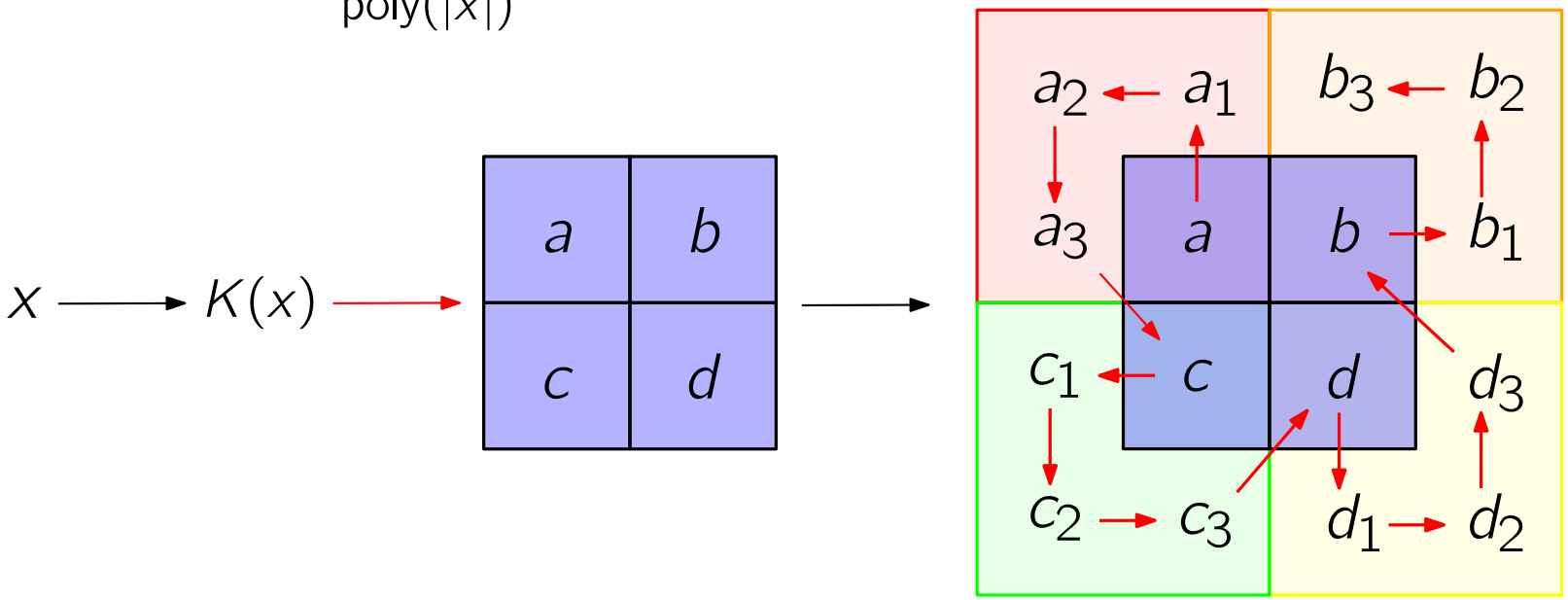
1. (X, p) *hat einen delayFPT-Algorithmus.*
2. *Es gibt eine enum-Kernelization für* (X, p).

Enum-Kernelization charakterisiert delayFPT II

enum-Kernelization $\Rightarrow$ delayFPT Algorithmus

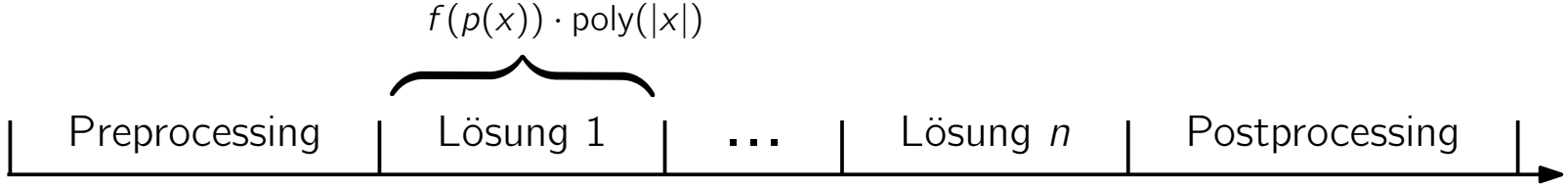
$f(p(x)) \cdot \text{poly}(|x|)$

$\text{poly}(|x|)$



Enum-Kernelization charakterisiert delayFPT III

delayFPT Algorithmus $\Rightarrow$ enum-Kernelization



Simuliere $\mathcal{A}$ für $\text{poly}(|x|)^2$ Schritte!

Fall 1: Keine Lösung. - Sei K(x) triviale NEIN-Instanz x_0.

Fall 2: Lösung y_1 aufgelistet. Sei x^* triviale JA-Instanz mit Lösung y^*.
Lass $K(x) := x^*$, $K^{-1}(x, y) :=$ simuliere $\mathcal{A}$ falls $y = y^*$ sonst tu nichts.

Fall 3: Noch kein Output. $\text{poly}(|x|)^2 < 2 \cdot f(p(x)) \cdot poly(|x|)$. Lass $K(x) := x, K^{-1}(x, y) = \{y\}$.

Outline

Max Ones 2-SAT

Definition

Gegeben: eine 2-CNF Formel Φ über Variablen $X := \{x_1, \ldots, x_n\}$
Parameter: k
Frage: Kann man mindestens k Variablen auf 1 setzen sodass Φ erfüllt ist?

Beispiel

Die Formel $\Phi := (a \vee b) \wedge (\neg a \vee \neg c)$ kann mit mindestens zwei 1 erfüllt werden. $I(a) := 1, I(b) := 1, I(c) := 0$.

Satz ([3], Thm. 7)

Es gibt einen FPT Algorithmus `hasMaxOnes` *(und viele andere, generellere SAT-Probleme) für das obrige Problem.*

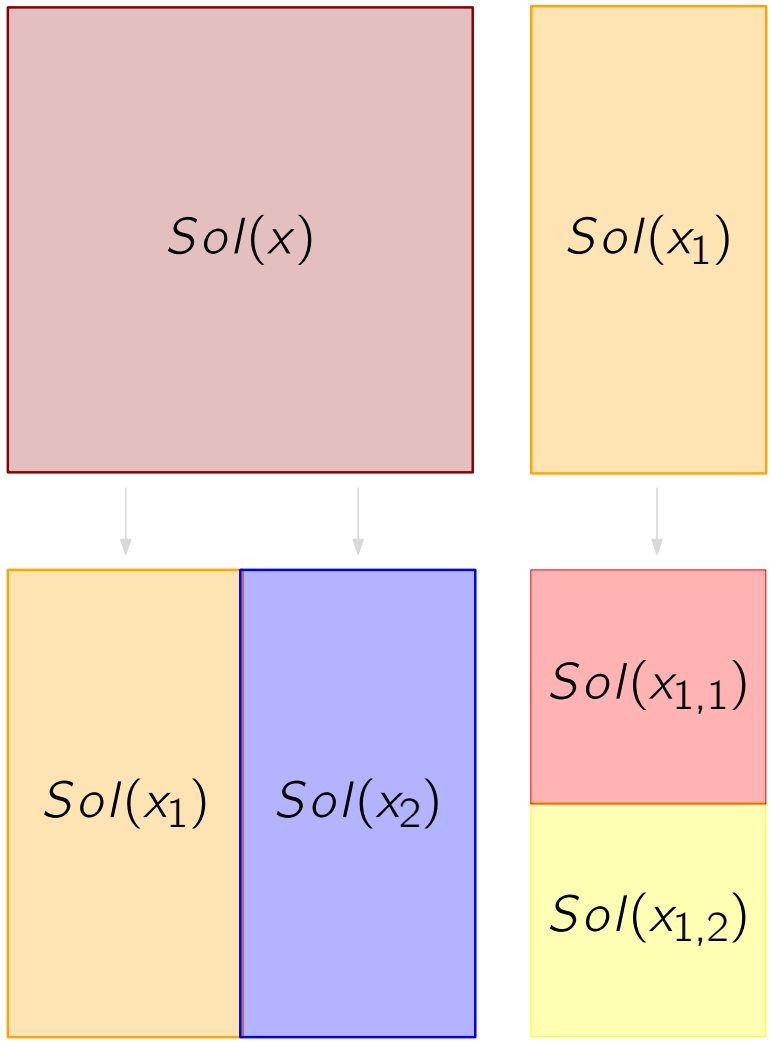
Self Reducibility

Intuition

Kann man ein Problem x in mehrere Subprobleme $x_1, x_2, \ldots$ mit gleichen Eigenschaften teilen; sodass die Lösungen der Teilprobleme Lösungen von x sind?

Beispiel

Max Ones 2-SAT ist self reducible! Zerteile (Φ, k) zu $(\Phi[v_0 := 1], k-1)$ und $(\Phi[v_0 := 0], k)$.



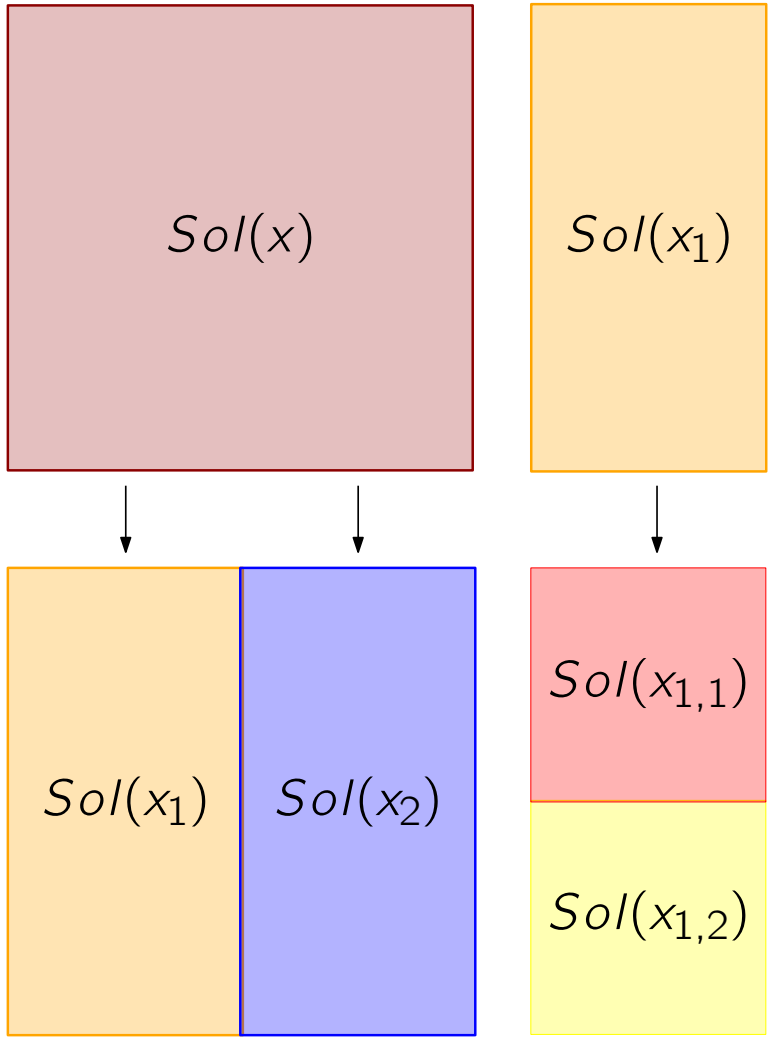
Self Reducibility

Intuition

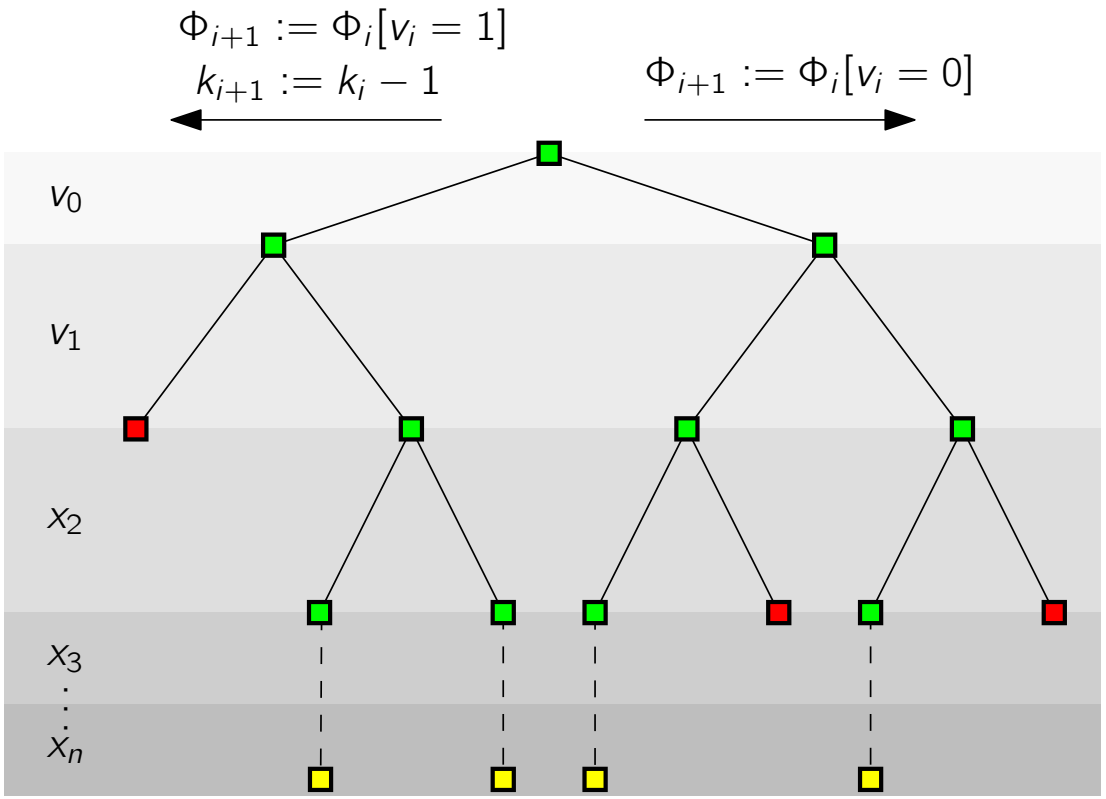
Kann man ein Problem x in mehrere Subprobleme $x_1, x_2, \ldots$ mit gleichen Eigenschaften teilen; sodass die Lösungen der Teilprobleme Lösungen von x sind?

Beispiel

Max Ones 2-SAT ist self reducible! Zerteile (Φ, k) zu $(\Phi[v_0 := 1], k-1)$ und $(\Phi[v_0 := 0], k)$.



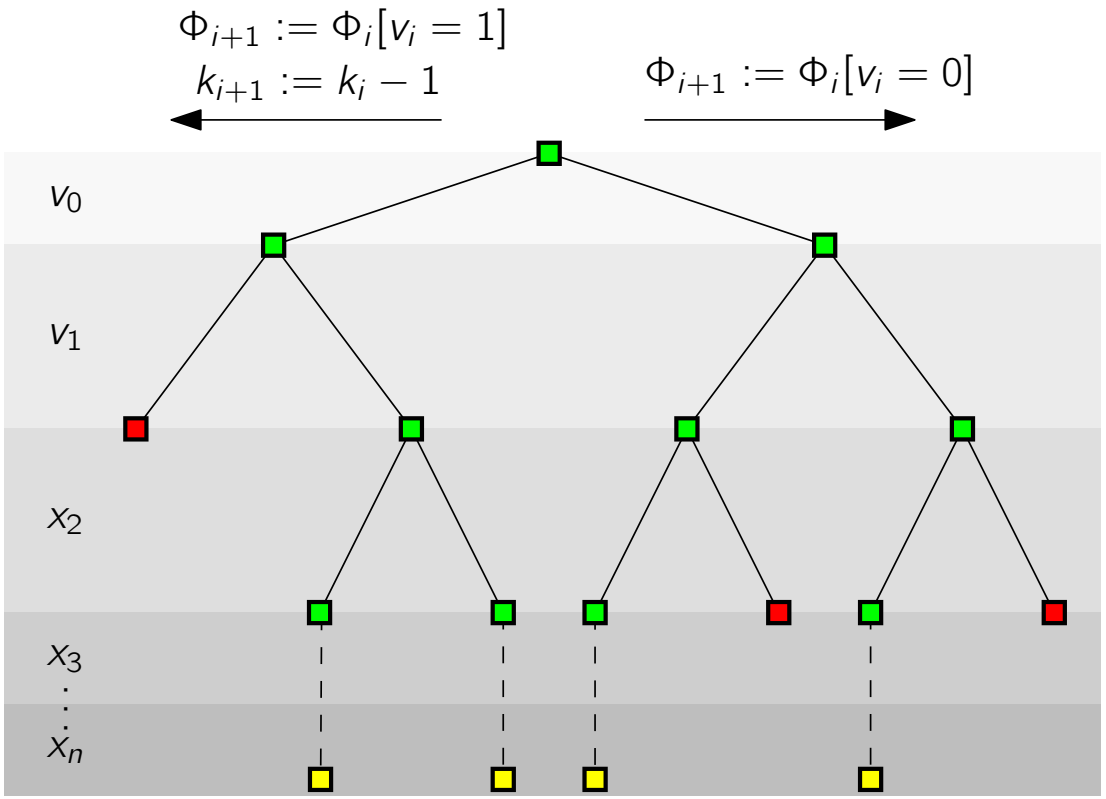
FPT Aufzählalgorithmus für Max Ones 2-SAT



- $\approx \texttt{hasMaxOnes}(\Phi_i, k_i) = \top$
  Erkunde den Zweig!
- $\approx \texttt{hasMaxOnes}(\Phi_i, k_i) = \bot$
  Prune den Zweig!
- $\approx i = n \wedge k_i \leq 0 \wedge sat.$
  Zähle Zuweisung des Zweiges auf!

- Erste Lösung: $n \cdot h(k) \cdot \text{poly}(|\Phi|)$ Zeit.
- Nächste Lösung ist erste Lösung von kleinerem Baum, daher auch $n \cdot h(k) \cdot \text{poly}(|\Phi|)$ Zeit!

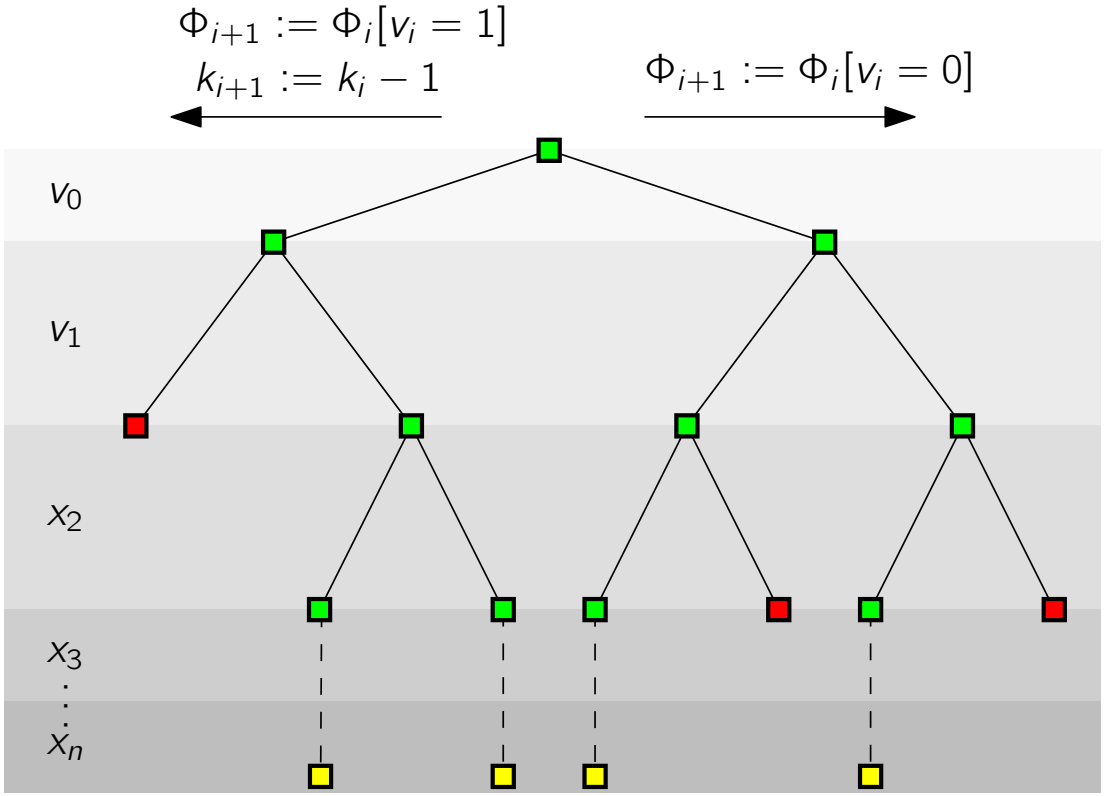
FPT Aufzählalgorithmus für Max Ones 2-SAT



- $\approx \texttt{hasMaxOnes}(\Phi_i, k_i) = \top$ Erkunde den Zweig!
- $\approx \texttt{hasMaxOnes}(\Phi_i, k_i) = \bot$ Prune den Zweig!
- $\approx i = n \wedge k_i \leq 0 \wedge sat.$ Zähle Zuweisung des Zweiges auf!

- Erste Lösung: $n \cdot h(k) \cdot \text{poly}(|\Phi|)$ Zeit.
- Nächste Lösung ist erste Lösung von kleinerem Baum, daher auch $n \cdot h(k) \cdot \text{poly}(|\Phi|)$ Zeit!

FPT Aufzählalgorithmus für Max Ones 2-SAT



- $\approx \texttt{hasMaxOnes}(\Phi_i, k_i) = \top$ Erkunde den Zweig!
- $\approx \texttt{hasMaxOnes}(\Phi_i, k_i) = \bot$ Prune den Zweig!
- $\approx i = n \wedge k_i \leq 0 \wedge sat.$ Zähle Zuweisung des Zweiges auf!

- Erste Lösung: $n \cdot h(k) \cdot \text{poly}(|\Phi|)$ Zeit.
- Nächste Lösung ist erste Lösung von kleinerem Baum, daher auch $n \cdot h(k) \cdot \text{poly}(|\Phi|)$ Zeit!

Zusammenfassung

- Fixed Parameter Tractability Algorithmen sind Algorithmen die Probleme mit einem fixierten Parameter in quasi polynomieller Zeit lösen.
- Ein FPT Aufzählalgorithmus zählt alle Lösungen in FPT Zeit auf, DelayFPT Algorithmen haben maximal FPT-Zeit langen Delay zwischen den Lösungen.
- Kernelization lässt sich gut auf delayFPT übersetzen und charakterisiert diese Klasse.
- Self-reducible Probleme lassen sich in mehrere Probleme zerteilen sodass die Teillösungen auch Teile der ganzen Lösungen sind.

Bibliographie I

[1] Marek Cygan et al. *Parameterized Algorithms*. Cham: Springer International Publishing, 2015. isbn: 978-3-319-21274-6 978-3-319-21275-3. doi: 10.1007/978-3-319-21275-3. (Visited on 04/02/2024).

[2] Nadia Creignou et al. "Paradigms for Parameterized Enumeration". In: *Mathematical Foundations of Computer Science 2013*. Ed. by Krishnendu Chatterjee and Jiri Sgall. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 290–301. isbn: 978-3-642-40313-2.

Bibliographie II

[3] Stefan Kratsch, Dániel Marx, and Magnus Wahlström. "Parameterized Complexity and Kernelizability of Max Ones and Exact Ones Problems". In: *ACM Trans. Comput. Theory* 8.1 (Feb. 2016). issn: 1942-3454. doi: 10.1145/2858787. url: https://doi.org/10.1145/2858787.