Einführung und Definitionen

Caspar Nagy

13. Mai 2019

Gliederung

- Motivation
- Definitionen

Motivation – Wo wir bei TGI stehen geblieben sind

Viele Interessante Probleme $\in NP$

- Lösungen für Bar Fight Prevention (aka Vertex Cover) schon für $n = 1000$ sehr unhandlich
- Laufzeit kann drastisch reduziert werden, wenn wir den Lösungsraum einschränken

Frage:

- Welche Parameter vereinfachen unser Problem tatsächlich?
- Welche Laufzeit kann man mit Parametrisierung erreichen?

Definitionen

Definitionen 1/2

Parametrisiertes Problem

- $(X, k) \in \Sigma^* \times \mathbb{N}$, wobei X die Instanz des Problems und k die unäre Kodierung des Parameters ist. $*$

FPT (*Fixed Parameter Tractable*)

- Menge der parametrisierten Probleme, für die ein Algorithmus $\mathcal{A}$ existiert, der Instanzen in Zeit $f(k) \cdot |(x, k)|^c$ entscheidet.

XP (*slice-wise polynomial*)

- Menge der parametrisierten Probleme, für die ein Algorithmus $\mathcal{A}$ existiert, der Instanzen in Zeit $f(k) \cdot |(x, k)|^{g(k)}$ entscheidet.

Definitionen 2/2

Aus TGI kennen wir die Mengen *P* und *NP*. Für parametrisierte Probleme gibt es analog *FPT*/*XP* und *W[1]*

- W[1] ist die Menge aller parametrisierten Probleme, die mindestens so komplex sind wie das Finden einer $\textsc{Clique}$ der Größe k.
- Analog zu NP wird die W[1]-Vollständigkeit über polynomielle Transformationen gezeigt.
- Das alles ist natürlich sinnlos, sollte $P = NP$ oder $\textsc{Clique} \in FPT$ sein.

Fragen?

Normalerweise	Kernelization	Anwendung
Effizienz = Laufzeit	Effizienz = Ausgabegröße	Ausgabegröße = Laufzeit

Probleminstanz (I, k) → **Kernelization Algorithmus $\mathcal{A}$** → äquivalente Probleminstanz $\mathcal{A}(I, k) = (I', k')$ mit $|I'| + k' \leq g(k)$

Normalerweise	Kernelization	Anwendung
Effizienz = Laufzeit	Effizienz = Ausgabegröße	Ausgabegröße = Laufzeit

Probleminstanz (I, k) → **Kernelization Algorithmus $\mathcal{A}$** → äquivalente Probleminstanz $\mathcal{A}(I,k) = (I', k')$ mit $|I'| + k' \leq g(k)$

Vertex Cover → Algor. **VC** → $\mathcal{O}(k^2)$ vertices, $\mathcal{O}(k^2)$ edges

Feedback Arc Set in Tournaments → Algor. **FAST** → $k^2 + 2k$ vertices

Edge Clique Cover → Algor. **ECC** → 2^k vertices

Other Kernelization Techniques

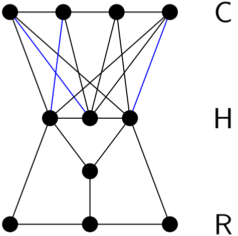
Liran Dattner

11. Mai 2019

Crown Decomposition

Zerlegung eines Graphen in C, H, R :

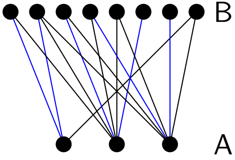
- C stabile Menge
- H separiert C und R
- $\exists$ Matching von H nach C



Expansion

Bipartiter Graph mit Partitionen A, B.
$M \subset E$ heißt q-Expansion von A nach B, falls:

- M überdeckt $q|A|$ Knoten aus B
- $\forall x \in V(A) : x$ ist inzident zu q Kanten in M



Sunflower

Mengen $(S_i)_{i \leq k}$ bilden **Sunflower** mit Zentrum V, falls:

- $\forall i \neq j : S_i \cap S_j = V$

Bounded Search Trees

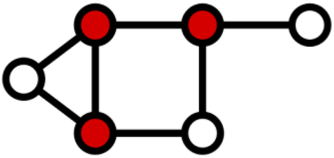


Abbildung: Wikipedia[1]

Algorithm 1 VC(Graph G, int k)

if $|E| == 0$ **then**
    return true
else if $k == 0$ **then**
    return false
else
    choose some $\{u, v\} \in E$
    return $VC(G - v, k - 1) \vee VC(G - u, k - 1)$
end if

[1] `https://en.wikipedia.org/wiki/Vertex_cover`

Bounded Search Trees

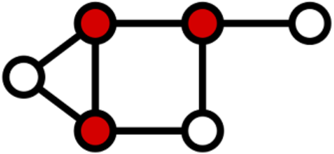


Abbildung: Wikipedia[1]

Algorithm 1 VC(Graph G, int k)

if $|E| == 0$ **then**
    return true
else if $k == 0$ **then**
    return false
else
    choose some $\{u, v\} \in E$
    return $VC(G - v, k - 1) \vee VC(G - u, k - 1)$
end if

[1] `https://en.wikipedia.org/wiki/Vertex_cover`

Bounded Search Trees

Problem	Suchbaumgröße	Techniken
Vertex Cover	$\mathcal{O}(2^k)$	der gerade eben vorgestellte Algorithmus
Vertex Cover	$\mathcal{O}(1.62^k)$	Kernelization; Bounding beim Grad ≤ 1; betrachte immer den Knoten mit dem höchsten Grad
Vertex Cover	$\mathcal{O}(1.47^k)$	zusätzlich: Bounding beim Grad ≤ 2
Closest String	$\mathcal{O}((d+1)^d)$	

Worum geht es?

$$
\begin{aligned}
min. & \sum_{v \in V} x_v \\
mit\ & x_u + x_v \geq 1 \quad \forall (u,v) \in E \\
& x_v \in \{0,1\} \quad \forall v \in V
\end{aligned}
$$

VERTEXCOVER als ILP

Karlsruher Institut für Technologie

Worum geht es?

Variablen

Zielfunktion

$$\begin{aligned} \mathit{min.} & \underline{\textstyle\sum_{v \in V} x_v} \\ \mathit{mit} & \; x_u + x_v \geq 1 \quad \forall (u, v) \in E \\ & \; x_v \in \{0, 1\} \quad \forall v \in V \end{aligned}$$

constraints

VertexCover als ILP

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Karlsruher Institut für Technologie

Worum geht es?

Variablen

Zielfunktion

$$\min. \sum_{v \in V} x_v$$

$$\text{mit} \quad x_u + x_v \geq 1 \quad \forall (u,v) \in E$$

$$x_v \in \{0,1\} \quad \forall v \in V$$

constraints

$$0 \leq x_v \leq 1 \quad \forall v \in V$$

Relaxation

VERTEXCOVER als ILP

- LP-RELAXATION $\equiv$ Aufhebung der Forderung nach Ganzzahligkeit

Parametrized Algorithms from LP-Relaxations

Worum geht es?

Variablen

Zielfunktion

$$\underline{\min. \sum_{v \in V} x_v}$$

$$\text{mit} \quad \begin{aligned} &x_u + x_v \geq 1 \quad \forall (u,v) \in E \\ &\underline{x_v \in \{0,1\}} \quad \forall v \in V \end{aligned} \Bigg| \text{ constraints}$$

$$\rightsquigarrow 0 \leq x_v \leq 1 \quad \forall v \in V$$

Relaxation

VERTEXCOVER als ILP

- LP-RELAXATION $\equiv$ Aufhebung der Forderung nach Ganzzahligkeit
- schnellere FPT-Algorithmen durch LP-RELAXATION

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Worum geht es?

2 Parametrisierungen von VERTEXCOVER

Parametrized Algorithms from LP-Relaxations

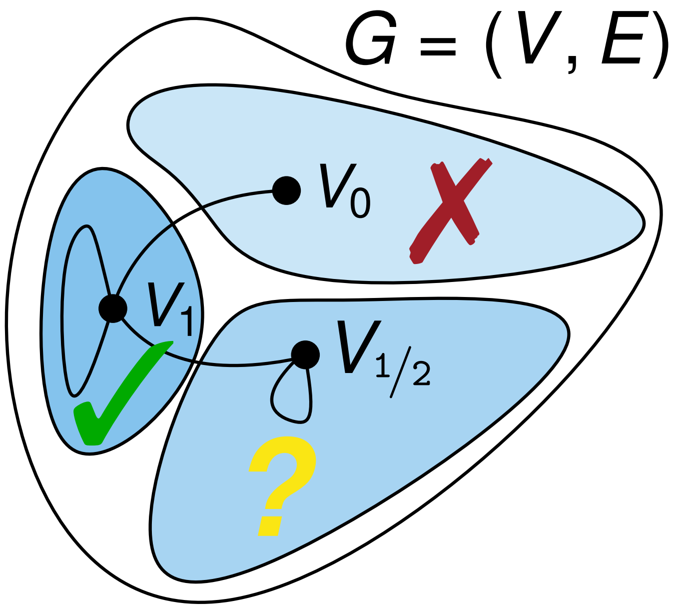
Worum geht es?

2 Parametrisierungen von VERTEXCOVER

- Parametrisierung über Größe k

  $2k$-vertex kernel für VERTEXCOVER

  Nemhauser-Trotter Theorem



$$V_1 \subseteq S \subseteq V_1 \cup V_{1/2}$$

Abb. 1: Nemhauser-Trotter Theorem

Parametrized Algorithms from LP-Relaxations

Worum geht es?

2 Parametrisierungen von VERTEXCOVER

- Parametrisierung über Größe k

  $2k$-vertex kernel für VERTEXCOVER

  Nemhauser-Trotter Theorem

- above guarantee Param. $k - LPVC(G)$

  branch Algorithmus für VERTEXCOVER

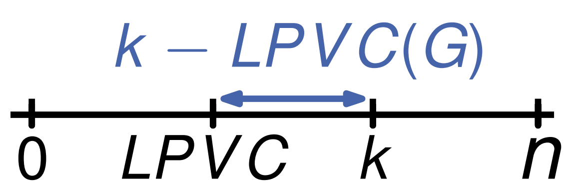
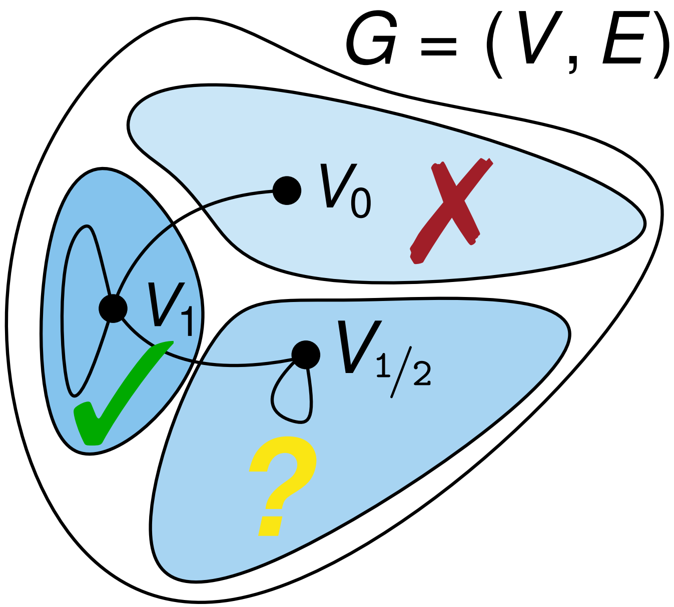


Abb. 2: VERTEXCOVER ABOVE LP



$$V_1 \subseteq S \subseteq V_1 \cup V_{1/2}$$

Abb. 1: Nemhauser-Trotter Theorem

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Definition

Iterative Compression

Eine Algorithmentechnik, die **iterative compression routine** auf eine Instanz von Problem anwendet. Das Ziel ist, einige Knoten zu entfernen, damit der resultierende Graph **einige globale Eigenschaft** erfüllt

Definition

Iterative Compression

Eine Algorithmentechnik, die **iterative compression routine** auf eine Instanz von Problem anwendet. Das Ziel ist, einige Knoten zu entfernen, damit der resultierende Graph **einige globale Eigenschaft** erfüllt

- Compression routine ist ein Algorithmus, der mit gegebener **Probleminstanz** und einer **Lösung** entweder kleinere Lösung findet oder überprüfen, dass die gegebene Lösung minimal ist

Definition

Karlsruher Institut für Technologie

Iterative Compression

Eine Algorithmentechnik, die **iterative compression routine** auf eine Instanz von Problem anwendet. Das Ziel ist, einige Knoten zu entfernen, damit der resultierende Graph **einige globale Eigenschaft** erfüllt

- Compression routine ist ein Algorithmus, der mit gegebener **Probleminstanz** und einer **Lösung** entweder kleinere Lösung findet oder überprüfen, dass die gegebene Lösung minimal ist
- Wir führen diese routine iterative, während wir den Graph bauen

Warum iterative Compression?

Karlsruher Institut für Technologie

- Wenn die Compression läuft in FPT Zeit, so ist auch der ganze Algorithmus
- Wir beobachten nicht nur das Problem sonder auch die Struktur der Lösung, um die Eigenschaften zu finden
- Deswegen ist es vielleicht leichter eine compression routine als ad hoc FPT Algorithmus zu implementieren

Randomisierte Methoden

Algorithmen für NP-schwere Probleme

Luc Mercatoris

13. Mai 2019

Karlsruher Institut für Technologie

Thema: Wie kann uns Randomisierung dabei helfen, effiziente Algorithmen für FPT-Entscheidungsprobleme zu gestalten? Welche Techniken werden dabei verwendet?

Monte-Carlo Algorithmus

Ein randomisierter Algorithmus, der mit einer nach oben beschränkten Wahrscheinlichkeit ein falsches Ergebnis zurückliefern darf.

Hier: Algorithmen haben **einseitigen Fehler** ($\longrightarrow$ false negative)

Wahrscheinlichkeiten

Beliebige Eingabe bei Ja-Instanz:

- in der Regel kleine Erfolgswahrscheinlichkeit p
- hohe Wahrscheinlichkeit für false negative: $(1-p)$

$\Rightarrow$ **wiederhole t mal**

- Wahrscheinlichkeit für false negative: $(1-p)^t$

$$(1-p)^t \leq \; (e^{-p})^t = \; 1/e^{pt}$$

Setze $t = \lceil \frac{1}{p} \rceil \quad \Rightarrow \quad$ konstante Abschätzung

Auf FPT Ebene

Ziel der Monte-Carlo Algorithmen mit einseitigem Fehler:

- Algorithmus läuft in FPT Zeit
- Wahrscheinlichkeit p liegt in $1/(f(k)n^{O(1)})$

$\Rightarrow$ wiederhole den Algorithmus $f(k)n^{O(1)}$ mal

$\Rightarrow$ neuer Algorithmus hat FPT-Laufzeit mit **konstanter Fehlerwahrscheinlichkeit**

Beispiel: Feedback Vertex Set

Sei Feedback Vertex Set-Instanz (G, k):

$\rightarrow$ Es existiert ein randomisierter Algorithmus mit polynomieller Laufzeit, der bei gegebener Ja-Instanz mit einer Wahrscheinlichkeit $p = 4^{-k}$ eine gültige Lösung zurückgibt

$\Longrightarrow$ wiederhole 4^k mal

$\rightarrow$ Neuer Algorithmus hat Laufzeit $4^k n^{O(1)}$ und hat konstante Fehlerwahrscheinlichkeit

Chromatische Codierung

- Problem: Graph G mit höchstens k Modifikationen so ändern, dass er bestimmte Eigenschaften hat
- Modifikation: Kante hinzufügen oder entfernen

Chromatische Codierung

- Problem: Graph G mit höchstens k Modifikationen so ändern, dass er bestimmte Eigenschaften hat
- Modifikation: Kante hinzufügen oder entfernen
- Chromatische Codierung: Knoten werden zufällig gefärbt
- Ob Lösung existiert, die korrekt gefärbt ist, ist effizient berechenbar
- Lösung korrekt gefärbt: modifizierte Kanten haben unterschiedlich gefärbte Endknoten

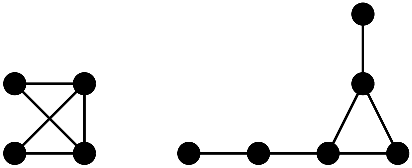
d-CLUSTERING

ℓ-Cluster-Graph: Einfacher Graph aus ℓ Zusammenhangskomponenten, die jeweils Cliquen sind

d-CLUSTERING

ℓ-Cluster-Graph: Einfacher Graph aus ℓ Zusammenhangskomponenten, die jeweils Cliquen sind

Beispiel:

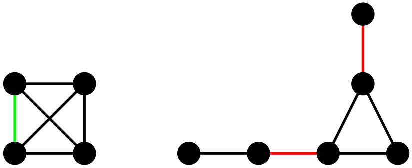


Kann der Graph mithilfe von höchstens drei Modifikationen in 4-Cluster-Graph umgewandelt werden?

d-CLUSTERING

ℓ-Cluster-Graph: Einfacher Graph aus ℓ Zusammenhangskomponenten, die jeweils Cliquen sind

Beispiel:

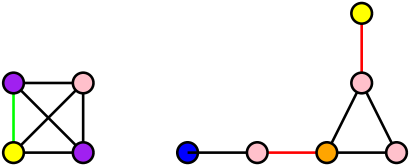


Kann der Graph mithilfe von höchstens drei Modifikationen in 4-Cluster-Graph umgewandelt werden? Ja!

d-CLUSTERING

ℓ-Cluster-Graph: Einfacher Graph aus ℓ Zusammenhangskomponenten, die jeweils Cliquen sind

Beispiel:

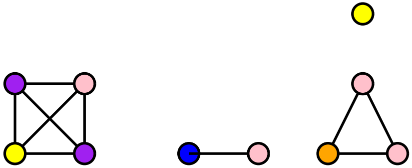


Kann der Graph mithilfe von höchstens drei Modifikationen in 4-Cluster-Graph umgewandelt werden? Ja!

d-CLUSTERING

ℓ-Cluster-Graph: Einfacher Graph aus ℓ Zusammenhangskomponenten, die jeweils Cliquen sind

Beispiel:



Kann der Graph mithilfe von höchstens drei Modifikationen in 4-Cluster-Graph umgewandelt werden? Ja!

Splitter

- Ziel: Derandomisierung der Färbung
- Idee: Familie von deterministisch erzeugten Funktionen $\mathcal{F}$
- Garantie: Es exist. ein $f \in \mathcal{F}$, sodass f den Graphen korrekt färbt

Splitter

- Ziel: Derandomisierung der Färbung
- Idee: Familie von deterministisch erzeugten Funktionen $\mathcal{F}$
- Garantie: Es exist. ein $f \in \mathcal{F}$, sodass f den Graphen korrekt färbt
- (n, k, ℓ)-Splitter $\mathcal{F}$ ist Familie von Funktionen von $[n]$ nach $[\ell]$, sodass für jedes $S \subseteq [n]$ mit Größe k eine Funktion f existiert, die S gleichmäßig aufteilt
- Behauptung: Bestimmte Splitter können effizient konstruiert werden

Treewidth

Karlsruher Institut für Technologie

- Quantifizierung der Ähnlichkeit zwischen Graph - Baum
- über Baumzerlegung, Generalisierung der Pfadzerlegung
- „gute“ Baumzerlegung $\Rightarrow$ Dynamische Programmierung
- Treewidth groß $\Rightarrow$ Aussagen über NP-schwere Probleme

Definition

Sei $\mathcal{P} = (X_1, \dots, X_r)$, $X_i \subseteq V(G) \forall i \in \{1, \dots, r\}$. $\mathcal{P}$ ist eine Pfadzerlegung eines Graphen G $:\Leftrightarrow$

P1 $\cup_{i=1}^{r} X_i = V(G)$

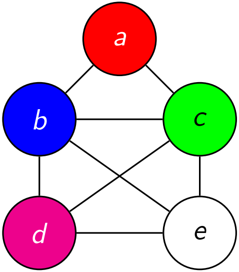
P2 $\forall (u, v) \in E(G) \exists k \in \{1, \dots, r\} : u, v \in X_k$

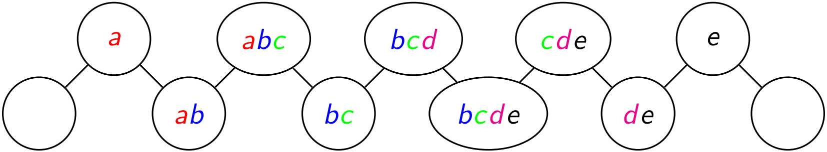
P3 $\forall u \in V(G) : u \in X_i \cap X_k \ (i \leq k) \Rightarrow u \in X_j \forall j \in \{i, \dots, k\}$

Treewidth: Beispiel

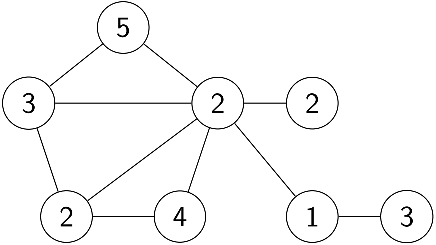
„Schöne“ Pfadzerlegung:

- erste und letzte Knotenmenge leer
- in jedem Pfadnoten wird nur ein Knoten hinzugefügt (*introduce node*)oder entfernt (*forget node*)

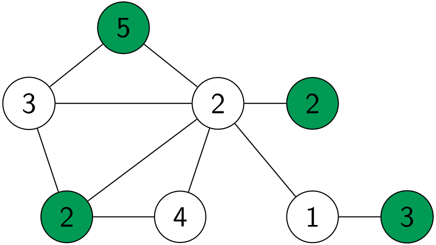




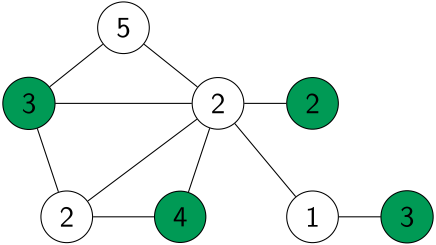
Weighted Independent Set



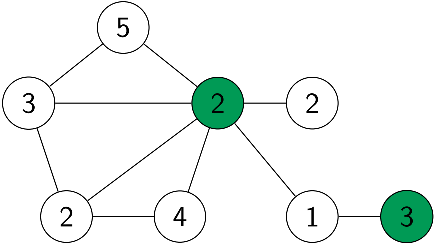
Weighted Independent Set



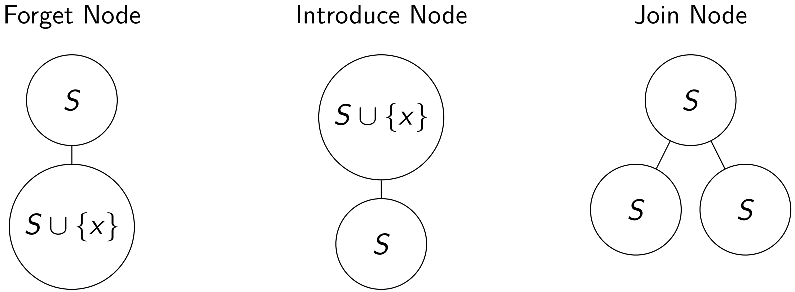
Weighted Independent Set

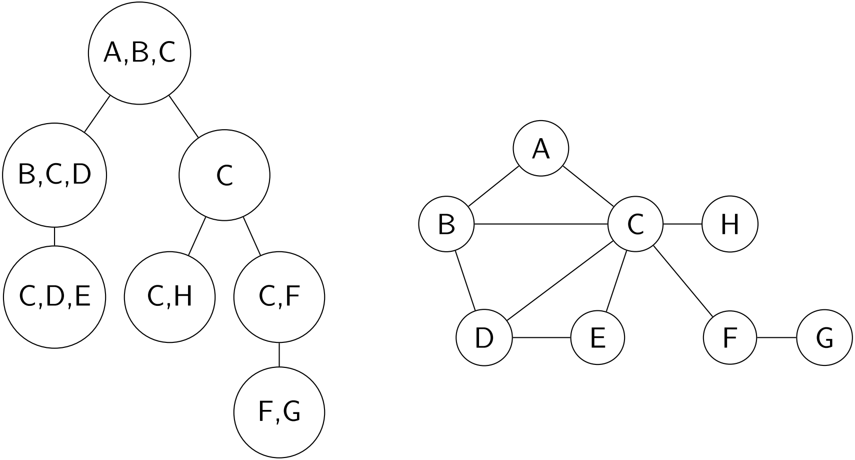


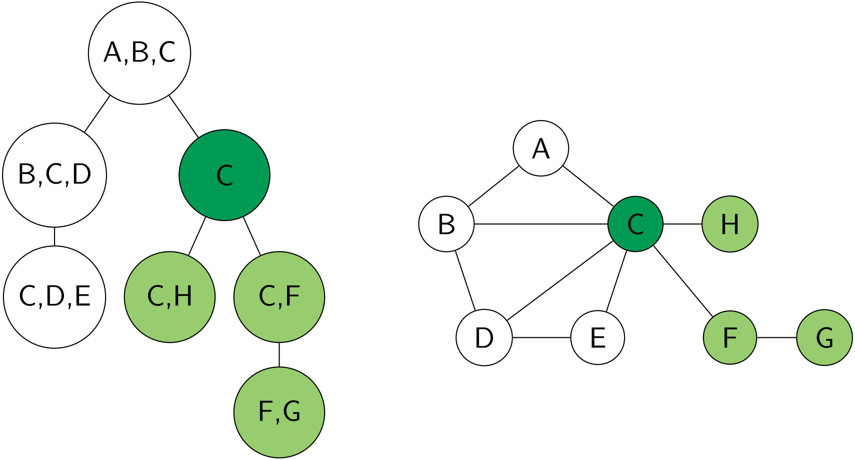
Weighted Independent Set

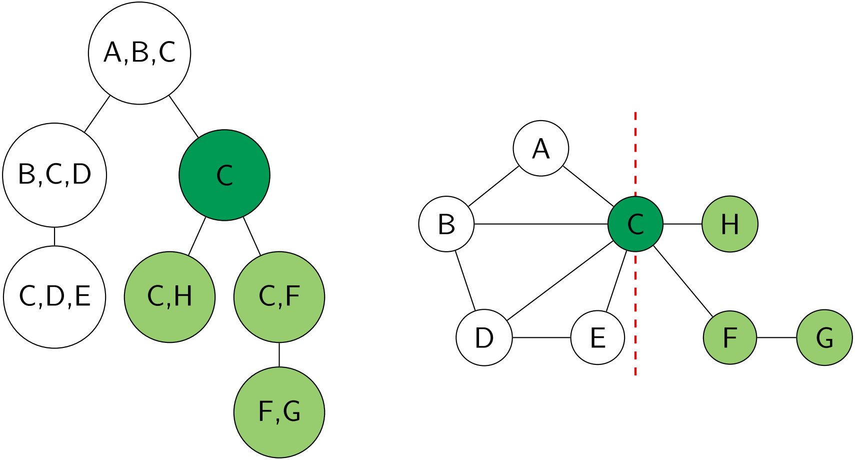


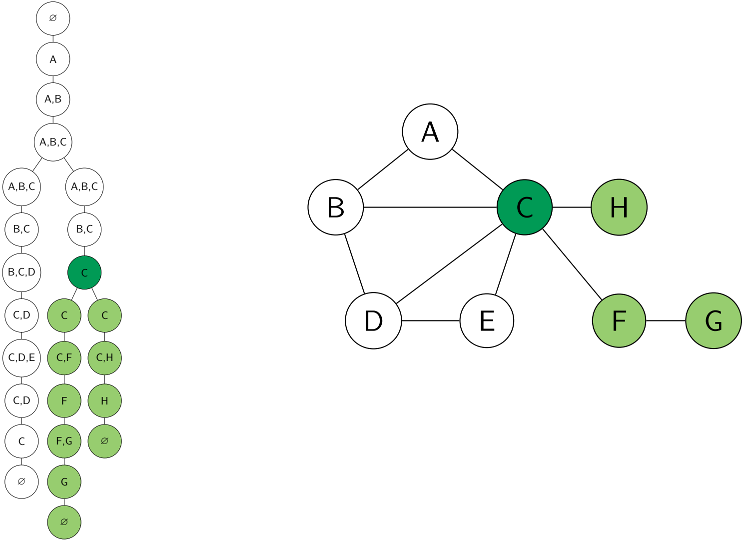
Nice Tree Decomposition

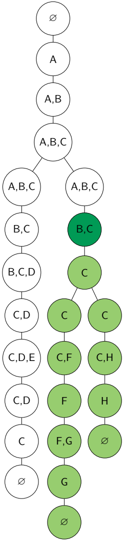


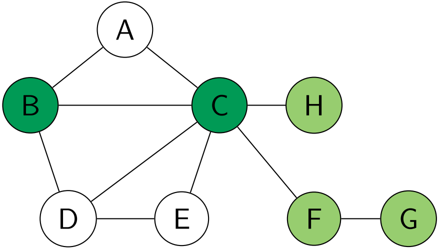

A,B,C
B,C,D
C
C,D,E
C,H
C,F
F,G
A
B
C
H
D
E
F
G

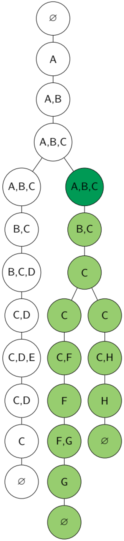

A,B,C
B,C,D
C
C,D,E
C,H
C,F
F,G
A
B
C
H
D
E
F
G

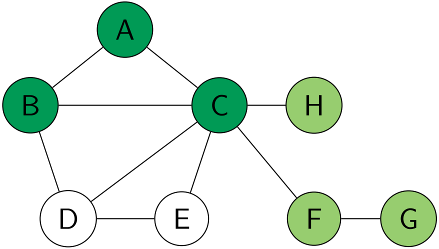

A,B,C
B,C,D
C
C,D,E
C,H
C,F
F,G
A
B
C
H
D
E
F
G

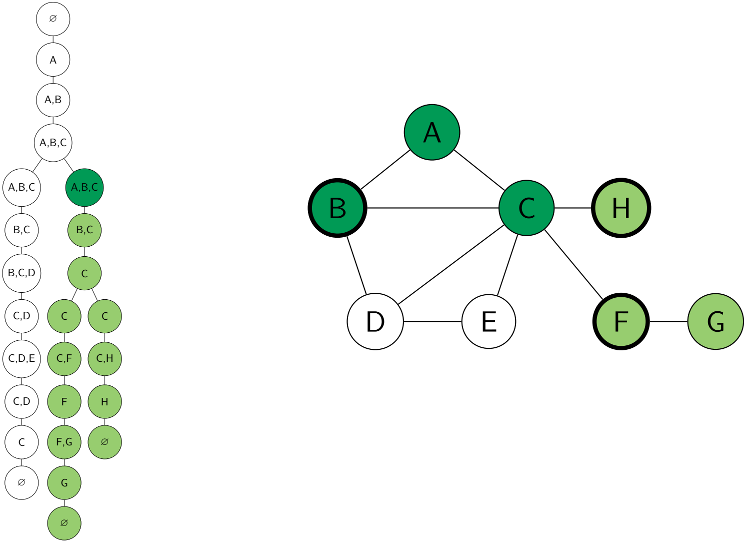

∅
A
A,B
A,B,C
A,B,C
A,B,C
B,C
B,C
B,C,D
C
C,D
C
C
C,D,E
C,F
C,H
C,D
F
H
C
F,G
∅
∅
G
∅
A
B
C
H
D
E
F
G

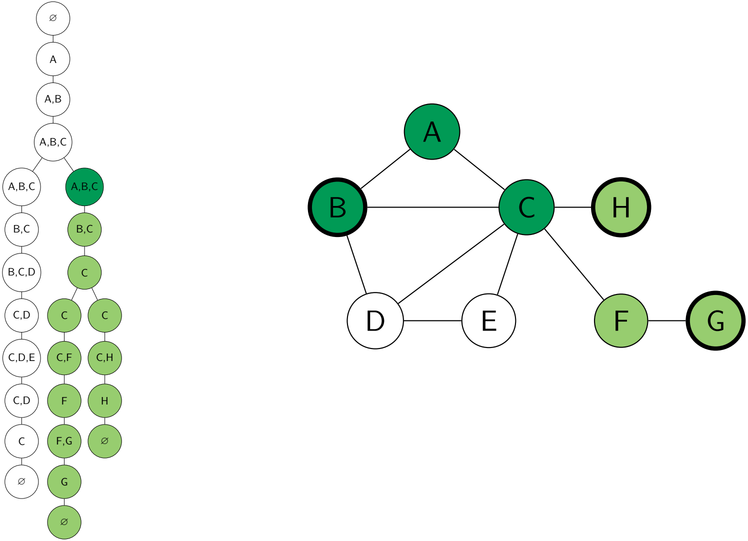

∅
A
A,B
A,B,C
A,B,C
A,B,C
B,C
B,C
B,C,D
C
C,D
C
C
C,D,E
C,F
C,H
C,D
F
H
C
F,G
∅
∅
G
∅


A
B
C
H
D
E
F
G

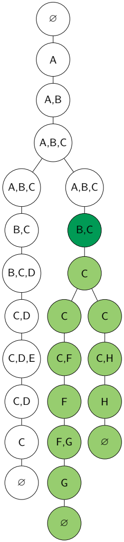

∅
A
A,B
A,B,C
A,B,C
A,B,C
B,C
B,C
B,C,D
C
C,D
C
C
C,D,E
C,F
C,H
C,D
F
H
C
F,G
∅
∅
G
∅

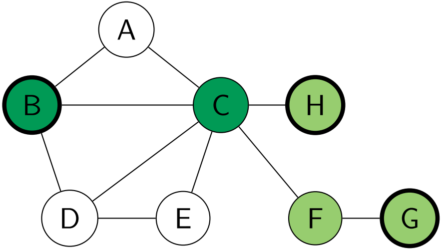

A
B
C
H
D
E
F
G


∅
A
A,B
A,B,C
A,B,C
A,B,C
B,C
B,C
B,C,D
C
C,D
C
C
C,D,E
C,F
C,H
C,D
F
H
C
F,G
∅
∅
G
∅
A
B
C
H
D
E
F
G



∅
A
A,B
A,B,C
A,B,C
A,B,C
B,C
B,C
B,C,D
C
C,D
C
C
C,D,E
C,F
C,H
C,D
F
H
C
F,G
∅
∅
G
∅
A
B
C
H
D
E
F
G


∅
A
A,B
A,B,C
A,B,C
A,B,C
B,C
B,C
B,C,D
C
C,D
C
C
C,D,E
C,F
C,H
C,D
F
H
C
F,G
∅
∅
G
∅


A
B
C
H
D
E
F
G

Baumweite und Dynamische Programmierung

Voraussetzungen

- Existenz von Nice Tree Decompositions

Satz Weighted Independent Set

Sei $G = (V, E)$ mit $|V| \leq n$ ein Graph mit Knotengewichten zusammen mit einer Tree Decomposition mit Weite k gegeben, dann kann ein Maximum Weighted Independent Set in $\mathcal{O}(2^k \cdot k^{\mathcal{O}(1)} \cdot n)$ berechnet werden

Baumweite und Dynamische Programmierung

Voraussetzungen

- Existenz von Nice Tree Decompositions

Satz Weighted Independent Set

Sei $G = (V, E)$ mit $|V| \leq n$ ein Graph mit Knotengewichten zusammen mit einer Tree Decomposition mit Weite k gegeben, dann kann ein Maximum Weighted Independent Set in $\mathcal{O}(2^k \cdot k^{\mathcal{O}(1)} \cdot n)$ berechnet werden

Satz Dominating Set

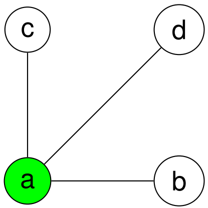
Sei $G = (V, E)$ mit $|V| \leq n$ ein Graph zusammen mit einer Tree Decomposition mit Weite k gegeben, dann kann ein Dominating Set in $\mathcal{O}(4^k \cdot k^{\mathcal{O}(1)} \cdot n)$ berechnet werden

11: Treewidth and MSO_2

Monadic Second Order Logic (MSO_2)

Grapheigenschaften mithilfe prädikatenlogischer Formeln darstellen

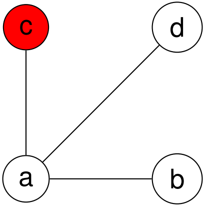
Beispiel(v) = $\forall_{Y \subseteq E}(\exists_{e \in Y} \Rightarrow \exists_{x \in V} \exists_{y \in Y}(inc(v, y) \wedge inc(x, y)))$



Monadic Second Order Logic (MSO_2)

Grapheigenschaften mithilfe prädikatenlogischer Formeln darstellen

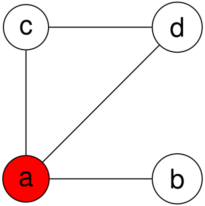
Beispiel(v) = $\forall_{Y \subseteq E}(\exists_{e \in Y} \Rightarrow \exists_{x \in V} \exists_{y \in Y}(inc(v, y) \wedge inc(x, y)))$



Monadic Second Order Logic (MSO_2)

Grapheigenschaften mithilfe prädikatenlogischer Formeln darstellen

Beispiel(v) = $\forall_{Y \subseteq E} (\exists_{e \in Y} \Rightarrow \exists_{x \in V} \exists_{y \in Y} (inc(v, y) \land inc(x, y)))$



11: Treewidth and MSO_2

Karlsruher Institut für Technologie

Satz von Courcelle

Wenn ein **Problem in MSO_2** dargestellt werden kann, kann ein **Graph mit gegebener Baumzerlegung** in **Zeit** $f(||\varphi||, t) \cdot n$ überprüft werden.

(Dabei ist f eine berechenbare Funktion, t Baumweite, $||\varphi||$ Formellänge)

Also:

- Für konstante Formellänge FPT in Baumweite
- Bei zusätzlich konstanter Baumweite in Linearzeit lösbar

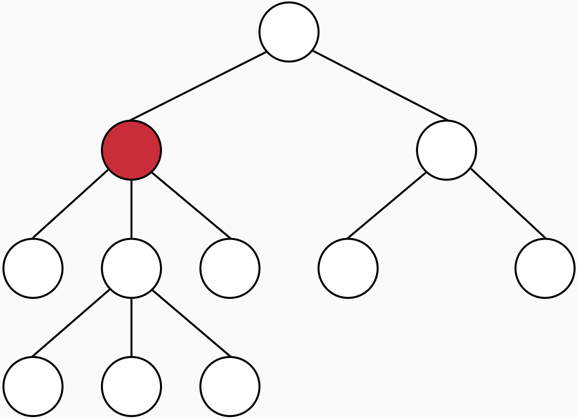
Baumweite berechnen

Es gibt einen Algorithmus, der für einen Graphen G mit n Knoten und einer ganzen Zahl k in $\mathcal{O}(8^k k^2 \cdot n^2)$ entweder eine Baumzerlegung von G mit Baumweite höchstens $4k+4$ berechnet oder erkennt, dass die Baumweite von G größer als k ist.

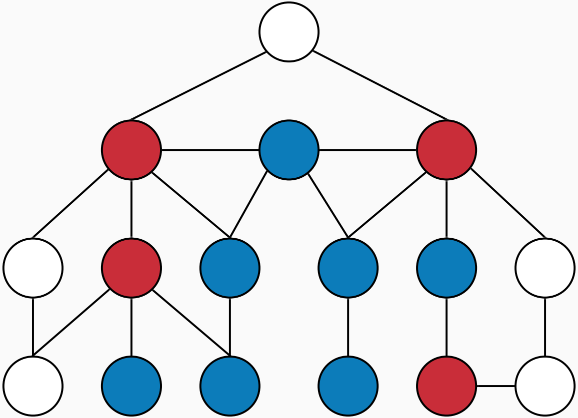
(Paul Seymour & Neil Robertson)

Baumweite berechnen - Vorgehen

- Bei Baumbreite höchstens k gibt es einen Balanced Seperator der Größe höchstens $k+1$



- Bei Baumbreite höchstens k gibt es einen Balanced Seperator der Größe höchstens $k + 1$
- Rekursive Zerlegung des Graphen, wobei in jedem Schritt ein Teilgraph mit Boundary nur etwa $3k$ betrachtet wird



- Bei Baumbreite höchstens k gibt es einen Balanced Seperator der Größe höchstens $k+1$
- Rekursive Zerlegung des Graphen, wobei in jedem Schritt ein Teilgraph mit Boundary nur etwa $3k$ betrachtet wird
- Gesucht ist kleiner Seperator, der den Teilgraphen so aufteilt, dass seine Boundary gleichmäßig auf seine Teile aufgeteilt wird