Julius-Maximilians-
UNIVERSITÄT WÜRZBURG
Lehrstuhl für
INFORMATIK I
Algorithmen & Komplexität
Institut für Informatik

Algorithmische Graphentheorie

Sommersemester 2025

10. Vorlesung

Festparameter-Berechenbarkeit

Alexander Wolff

Lehrstuhl für Informatik I

Herangehensweisen an NP-schwere Probleme

Herangehensweisen an NP-schwere Probleme

- Exponentialzeit-Algorithmen, z.B. Backtracking

Herangehensweisen an NP-schwere Probleme

- Exponentialzeit-Algorithmen, z.B. Backtracking
- Approximationsalgorithmen:
  Tausche Qualität gegen Laufzeit

Herangehensweisen an NP-schwere Probleme

- Exponentialzeit-Algorithmen, z.B. Backtracking
- Approximationsalgorithmen:
  Tausche Qualität gegen Laufzeit
- Heuristiken: empirische Untersuchung auf Benchmarks

Herangehensweisen an NP-schwere Probleme

- **Exponentialzeit-Algorithmen**, z.B. Backtracking
- **Approximationsalgorithmen**:
  Tausche Qualität gegen Laufzeit
- **Heuristiken**: empirische Untersuchung auf Benchmarks
- **Randomisierung**: Suche nach der Nadel im Heuhaufen

Herangehensweisen an NP-schwere Probleme

- Exponentialzeit-Algorithmen, z.B. Backtracking
- Approximationsalgorithmen:
  Tausche Qualität gegen Laufzeit
- Heuristiken: empirische Untersuchung auf Benchmarks
- Randomisierung: Suche nach der Nadel im Heuhaufen
- Entwurf von parametrisierten Algorithmen

NEU

Ein Beispiel: Vertex Cover

Def. (Zur Erinnerung)
Sei G ein ungerichteter Graph mit Knotenmenge $V(G)$.
$C \subseteq V(G)$ heißt *Knotenüberdeckung* (eng. *vertex cover*) von G, falls

Ein Beispiel: Vertex Cover

Def. (Zur Erinnerung)
Sei G ein ungerichteter Graph mit Knotenmenge $V(G)$.
$C \subseteq V(G)$ heißt *Knotenüberdeckung* (eng. *vertex cover*) von G, falls für alle $uv \in E(G)$ gilt, dass $\{u, v\} \cap C \neq \emptyset$.

Ein Beispiel: Vertex Cover

Def. (Zur Erinnerung)
Sei G ein ungerichteter Graph mit Knotenmenge $V(G)$.
$C \subseteq V(G)$ heißt *Knotenüberdeckung* (eng. *vertex cover*) von G, falls für alle $uv \in E(G)$ gilt, dass $\{u, v\} \cap C \neq \emptyset$.

Prob. *Kleinste Knotenüberdeckung*

Gegeben: Graph G

Gesucht: eine kleinste Knotenüberdeckung von G

Ein Beispiel: Vertex Cover

Def. (Zur Erinnerung)
Sei G ein ungerichteter Graph mit Knotenmenge $V(G)$.
$C \subseteq V(G)$ heißt *Knotenüberdeckung* (eng. *vertex cover*) von G, falls für alle $uv \in E(G)$ gilt, dass $\{u, v\} \cap C \neq \emptyset$.

Prob. *Kleinste Knotenüberdeckung* – Optimierungsproblem
Gegeben: Graph G
Gesucht: eine kleinste Knotenüberdeckung von G

Ein Beispiel: Vertex Cover

Def. (Zur Erinnerung)
Sei G ein ungerichteter Graph mit Knotenmenge $V(G)$.
$C \subseteq V(G)$ heißt *Knotenüberdeckung* (eng. *vertex cover*) von G, falls für alle $uv \in E(G)$ gilt, dass $\{u, v\} \cap C \neq \emptyset$.

Prob. *Kleinste Knotenüberdeckung* – Optimierungsproblem
Gegeben: Graph G
Gesucht: eine kleinste Knotenüberdeckung von G

Prob. – Entscheidungsproblem

Ein Beispiel: Vertex Cover

Def. (Zur Erinnerung)
Sei G ein ungerichteter Graph mit Knotenmenge $V(G)$.
$C \subseteq V(G)$ heißt *Knotenüberdeckung* (eng. *vertex cover*) von G, falls für alle $uv \in E(G)$ gilt, dass $\{u, v\} \cap C \neq \emptyset$.

Prob. *Kleinste Knotenüberdeckung* – Optimierungsproblem
Gegeben: Graph G
Gesucht: eine kleinste Knotenüberdeckung von G

Prob. *k-Knotenüberdeckung (k-VC)* – Entscheidungsproblem
Gegeben: Graph G, natürliche Zahl k
Gesucht: Knotenüberdeckung der Größe $\leq k$ von G – falls eine solche existiert (sonst gib „nein“ zurück)

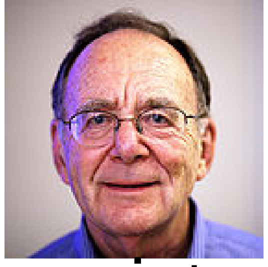
Previous Work

- eines der ersten Probleme, dessen NP-Schwere gezeigt wurde

Previous Work

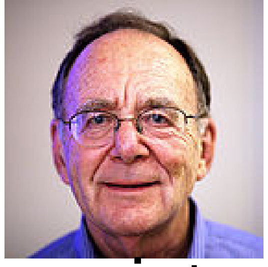
- eines der ersten Probleme, dessen NP-Schwere gezeigt wurde (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p \ldots$)

Previous Work



- eines der ersten Probleme, dessen NP-Schwere gezeigt wurde (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p \ldots$) [Karp, 1972]

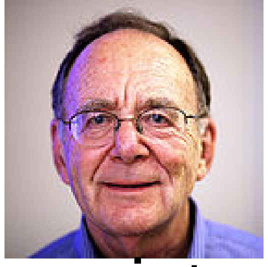
Previous Work



- eines der ersten Probleme, dessen NP-Schwere gezeigt wurde (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p \ldots$) [Karp, 1972]

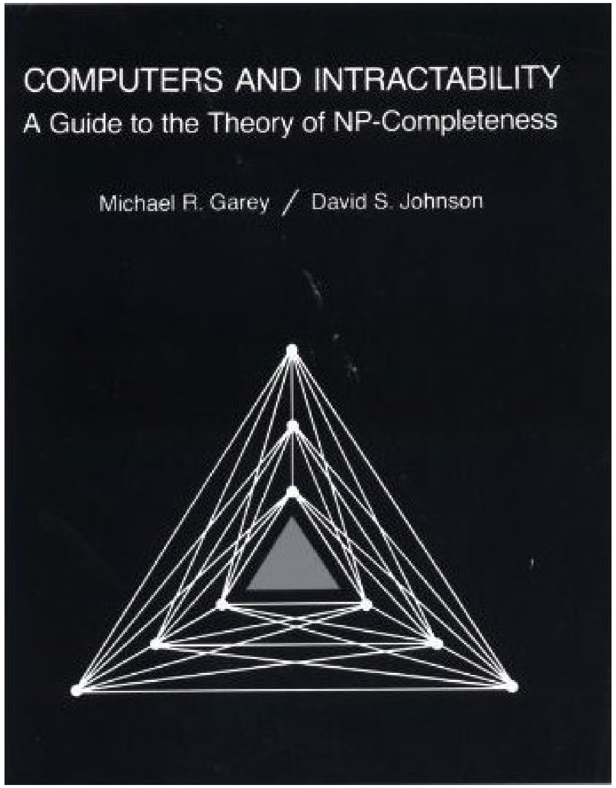
- eines der sechs grundlegenden NP-vollständigen Probleme in dem Klassiker: [Garey & Johnson, 1979]

Previous Work

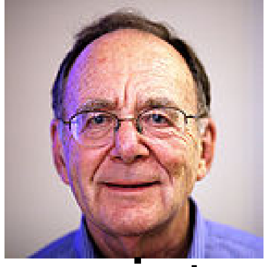


- eines der ersten Probleme, dessen NP-Schwere gezeigt wurde (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p \ldots$) [Karp, 1972]

- eines der sechs grundlegenden NP-vollständigen Probleme in dem Klassiker: [Garey & Johnson, 1979]

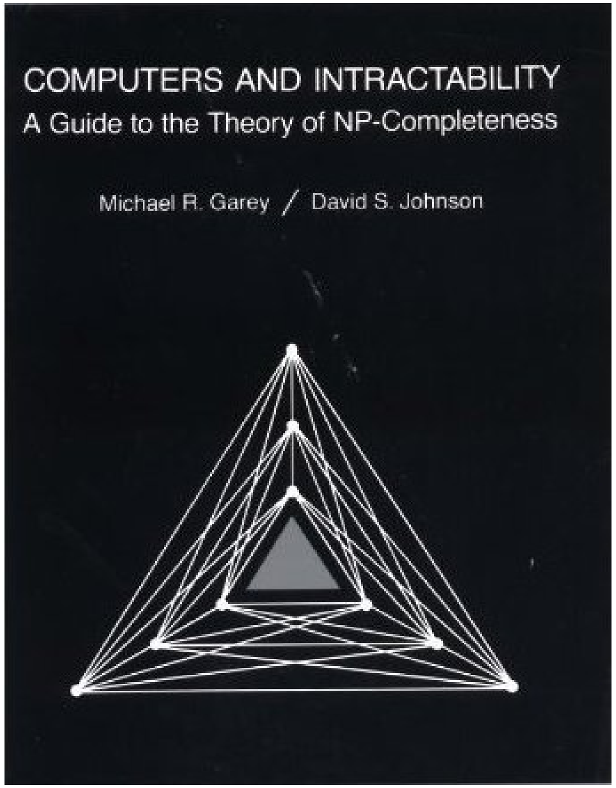


Previous Work



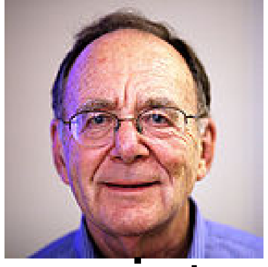
- eines der ersten Probleme, dessen NP-Schwere gezeigt wurde (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p$ ...) [Karp, 1972]

- eines der sechs grundlegenden NP-vollständigen Probleme in dem Klassiker: [Garey & Johnson, 1979]

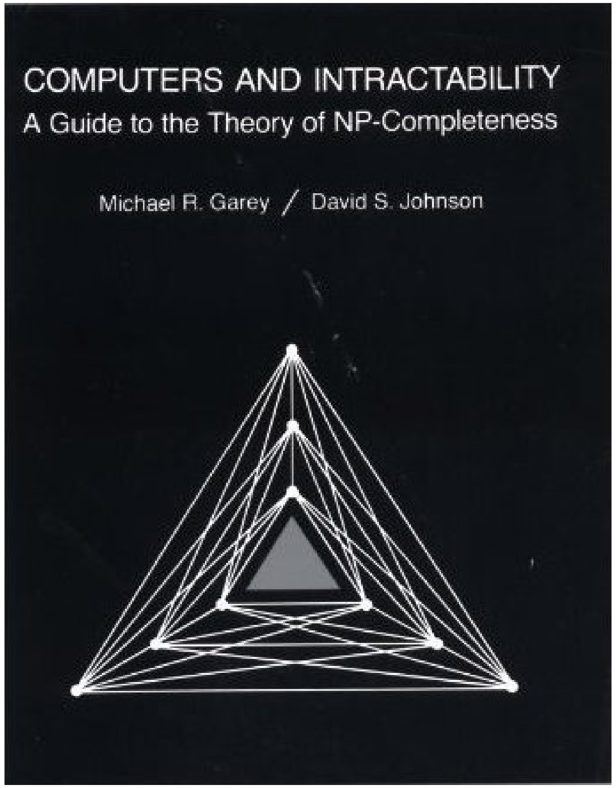


- approximierbar:

Previous Work



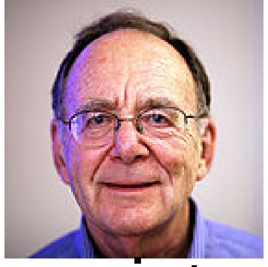
- eines der ersten Probleme, dessen NP-Schwere gezeigt wurde (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p$ ...) [Karp, 1972]
- eines der sechs grundlegenden NP-vollständigen Probleme in dem Klassiker: [Garey & Johnson, 1979]



- approximierbar:

  Nicht-erweiterbares Matching „liefert" Faktor-2-Approximation für VC.

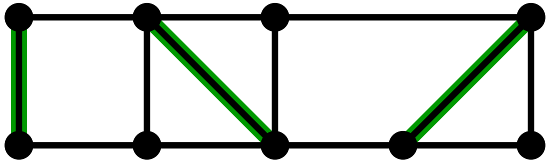
Previous Work



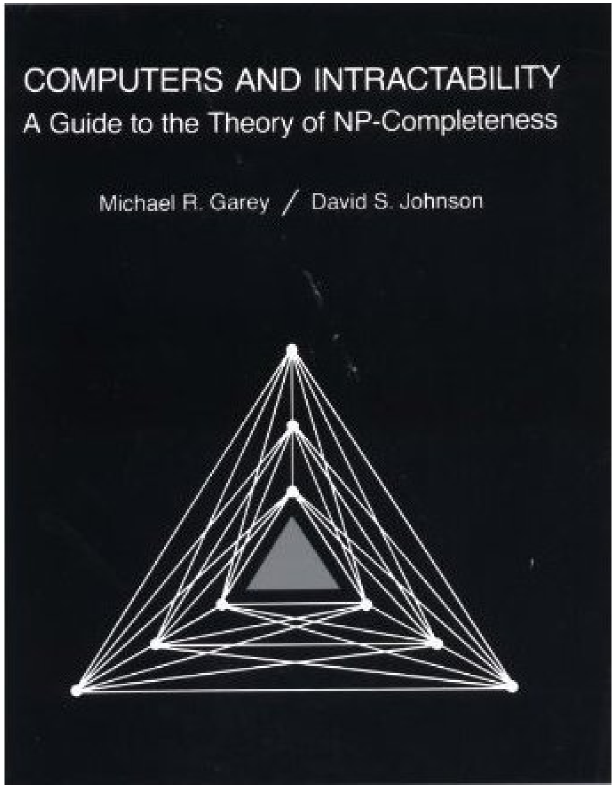
- eines der ersten Probleme, dessen NP-Schwere gezeigt wurde (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p \ldots$) [Karp, 1972]

- eines der sechs grundlegenden NP-vollständigen Probleme in dem Klassiker: [Garey & Johnson, 1979]

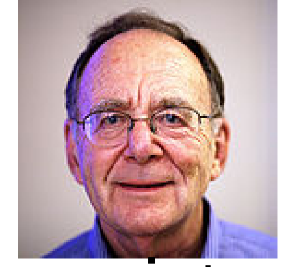
- approximierbar:



Nicht-erweiterbares Matching „liefert“ Faktor-2-Approximation für VC.



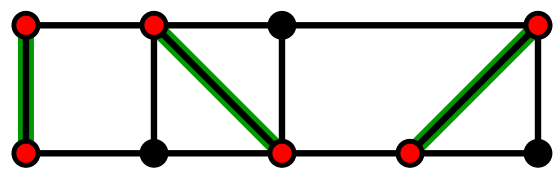
Previous Work



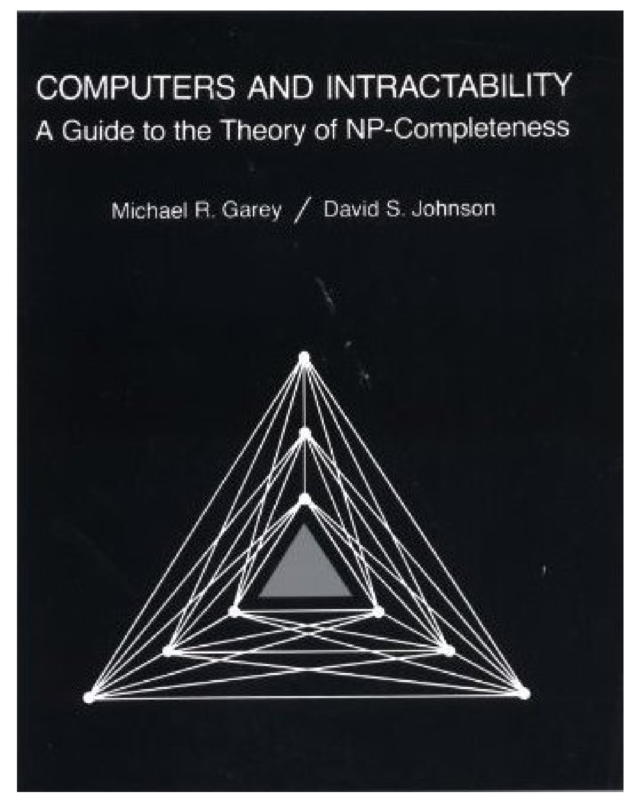
- eines der ersten Probleme, dessen NP-Schwere gezeigt wurde (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p$ . . .) [Karp, 1972]

- eines der sechs grundlegenden NP-vollständigen Probleme in dem Klassiker: [Garey & Johnson, 1979]

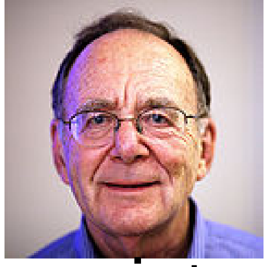
- approximierbar:



Nicht-erweiterbares Matching „liefert“ Faktor-2-Approximation für VC.

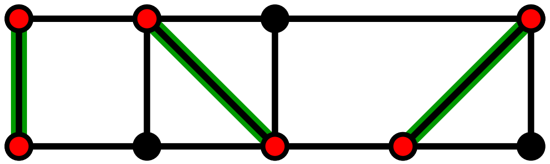


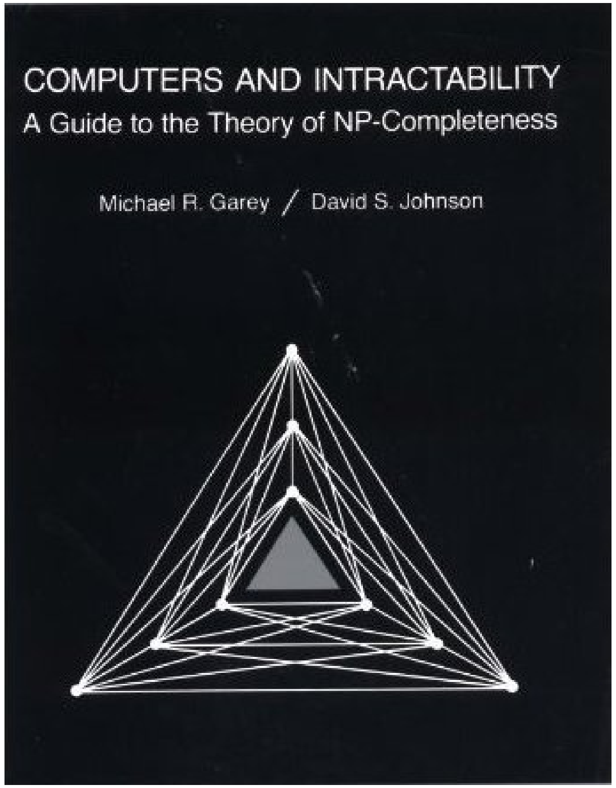
Previous Work



- eines der ersten Probleme, dessen NP-Schwere gezeigt wurde (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p$ …) [Karp, 1972]

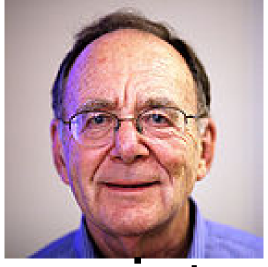
- eines der sechs grundlegenden NP-vollständigen Probleme in dem Klassiker: [Garey & Johnson, 1979]

- approximierbar: 
  Nicht-erweiterbares Matching „liefert“ Faktor-2-Approximation für VC.



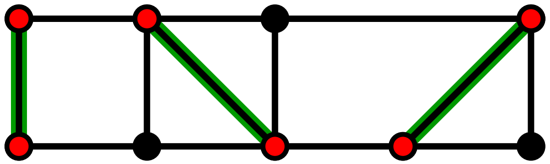
- …, aber nicht beliebig gut:

Previous Work



- eines der ersten Probleme, dessen NP-Schwere gezeigt wurde (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p$ ...) [Karp, 1972]

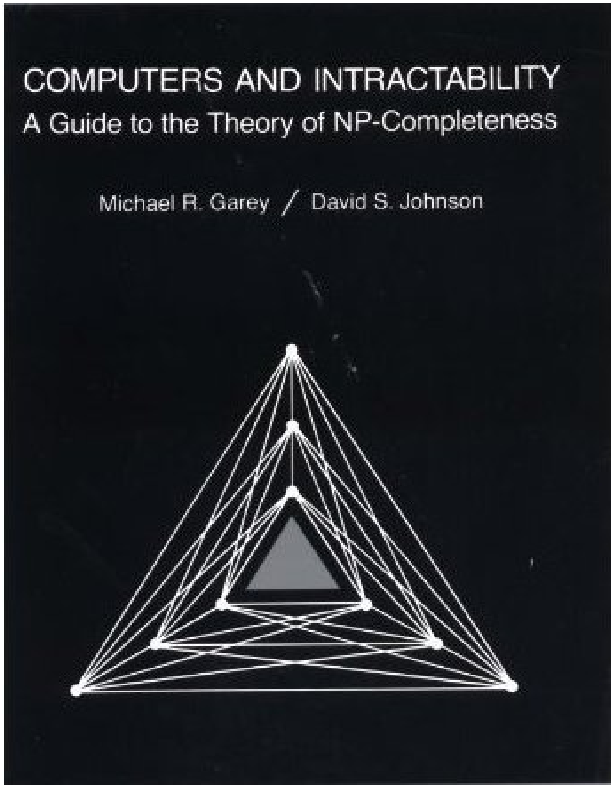
- eines der sechs grundlegenden NP-vollständigen Probleme in dem Klassiker: [Garey & Johnson, 1979]

- approximierbar: 

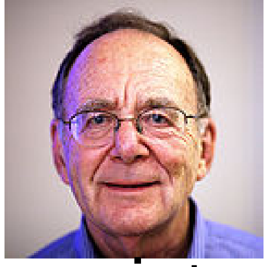
  Nicht-erweiterbares Matching „liefert" Faktor-2-Approximation für VC.

- ..., aber nicht beliebig gut:

  Es gibt keine Faktor-1,3606-Approximation für VC



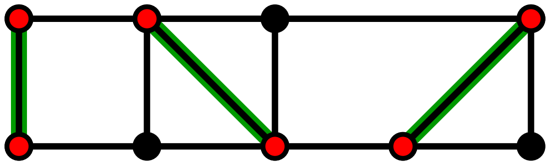
Previous Work



- eines der ersten Probleme, dessen NP-Schwere gezeigt wurde (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p$ ...) [Karp, 1972]

- eines der sechs grundlegenden NP-vollständigen Probleme in dem Klassiker: [Garey & Johnson, 1979]

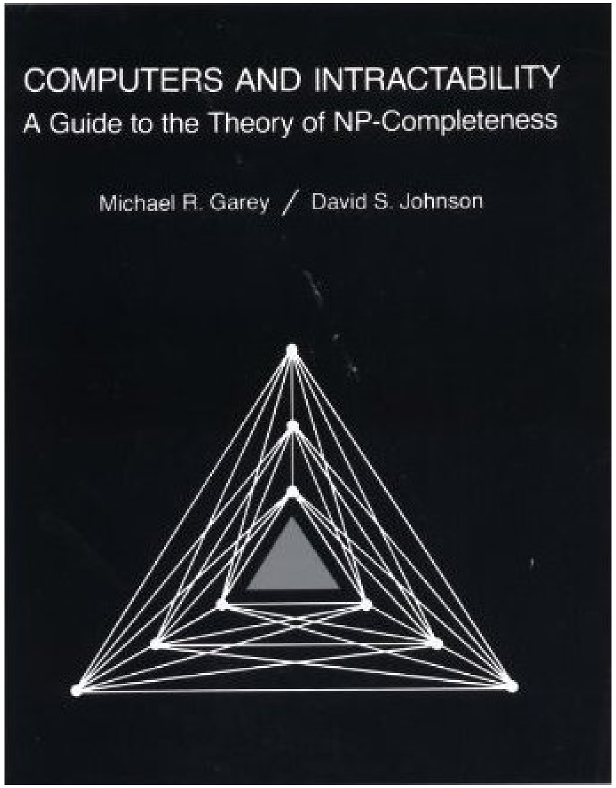
- approximierbar:

  Nicht-erweiterbares Matching „liefert“ Faktor-2-Approximation für VC.

- ..., aber nicht beliebig gut:

  Es gibt keine Faktor-1,3606-Approximation für VC, falls $\mathcal{P} \neq \mathcal{NP}$.
  [Dinur & Safra, 2004]



Ein exakter Algorithmus für k-VC

BruteForceVC(Graph G, Integer k)

Ein exakter Algorithmus für k-VC

BruteForceVC(Graph G, Integer k)
  foreach $C \in \binom{V(G)}{k}$ **do**
    // teste, ob C VC
    vc $= true$

    if vc **then**
      return (“yes”, C)
  return (“no”, $\emptyset$)

Ein exakter Algorithmus für k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V(G) choose k) do
      // teste, ob C VC
      vc = true
      foreach uv ∈ E(G) do
          if {u, v} ∩ C = ∅ then
              vc = false
      if vc then
          return ("yes", C)
  return ("no", ∅)
```

$\text{foreach } C \in \binom{V(G)}{k} \text{ do}$

Ein exakter Algorithmus für k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V(G) choose k) do
      // teste, ob C VC
      vc = true
      foreach uv ∈ E(G) do
          if {u, v} ∩ C = ∅ then
              vc = false
      if vc then
          return ("yes", C)
  return ("no", ∅)
```

Laufzeit.

Ein exakter Algorithmus für k-VC

BruteForceVC(Graph G, Integer k)
foreach $C \in \binom{V(G)}{k}$ **do**
// teste, ob C VC
vc $= true$
foreach $uv \in E(G)$ **do**
if $\{u, v\} \cap C = \emptyset$ **then**
vc $= false$
if vc **then**
return (“yes”, C)
return (“no”, $\emptyset$)

Laufzeit.

Ein exakter Algorithmus für k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V(G) choose k) do
    // teste, ob C VC
    vc = true
    foreach uv ∈ E(G) do
      if {u, v} ∩ C = ∅ then
        vc = false
    if vc then
      return ("yes", C)
  return ("no", ∅)
```

$$\left|\binom{V(G)}{k}\right| = \binom{|V(G)|}{k} = \binom{n}{k}$$

Laufzeit.

Ein exakter Algorithmus für k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V(G) choose k) do
    // teste, ob C VC
    vc = true
    foreach uv ∈ E(G) do
      if {u, v} ∩ C = ∅ then
        vc = false
    if vc then
      return ("yes", C)
  return ("no", ∅)
```

$$\left|\binom{V(G)}{k}\right| = \binom{|V(G)|}{k} = \binom{n}{k}$$
$$= O(n^k)$$

Warum?

Laufzeit.

Ein exakter Algorithmus für k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V(G) choose k) do
      // teste, ob C VC
      vc = true
      foreach uv ∈ E(G) do
          if {u, v} ∩ C = ∅ then
              vc = false
      if vc then
          return ("yes", C)
  return ("no", ∅)
```

$$\left|\binom{V(G)}{k}\right| = \binom{|V(G)|}{k} = \binom{n}{k}$$
$$= O(n^k)$$

Warum?

Laufzeit.

Ein exakter Algorithmus für k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V(G) choose k) do
    // teste, ob C VC
    vc = true
    foreach uv ∈ E(G) do
      if {u, v} ∩ C = ∅ then
        vc = false
    if vc then
      return ("yes", C)
  return ("no", ∅)
```

$$\left|\binom{V(G)}{k}\right| = \binom{|V(G)|}{k} = \binom{n}{k} = O(n^k)$$

Warum?

$$O(E(G)) = O(m)$$

Laufzeit.

Ein exakter Algorithmus für k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V(G) choose k) do
    // teste, ob C VC
    vc = true
    foreach uv ∈ E(G) do
      if {u, v} ∩ C = ∅ then
        vc = false
    if vc then
      return ("yes", C)
  return ("no", ∅)
```

$$\left|\binom{V(G)}{k}\right| = \binom{|V(G)|}{k} = \binom{n}{k}$$
$$= O(n^k)$$

Warum?

$$O(E(G)) = O(m)$$

Laufzeit. $O(n^k m)$

Ein exakter Algorithmus für k-VC

BruteForceVC(Graph G, Integer k)
  foreach $C \in \binom{V(G)}{k}$ **do**
    // teste, ob C VC
    vc $= \mathit{true}$
    foreach $uv \in E(G)$ **do**
      if $\{u, v\} \cap C = \emptyset$ **then**
        vc $= \mathit{false}$
    if vc **then**
      return (“yes”, C)
  return (“no”, $\emptyset$)

$$\left|\binom{V(G)}{k}\right| = \binom{|V(G)|}{k} = \binom{n}{k}$$
$$= O(n^k)$$

Warum?

$$O(E(G)) = O(m)$$

Laufzeit. $O(n^k m)$ – Dies ist *nicht* polynomiell in der Größe der Eingabe ($= n + m$), da k keine Konstante, sondern Teil der Eingabe ist.

Ein neues Ziel

Finde einen Algorithmus für k-VC mit Laufzeit

$$O(f(k) + |I|^c)$$

Ein neues Ziel

Finde einen Algorithmus für k-VC mit Laufzeit

$$O(f(k) + |I|^c)$$

wobei $f\colon \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

Ein neues Ziel

Finde einen Algorithmus für k-VC mit Laufzeit

$$O(f(k) + |I|^c)$$

wobei $f\colon \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k,
– polynomiell von der Größe $|I|$ der Instanz I.

Ein neues Ziel

Finde einen Algorithmus für k-VC mit Laufzeit

$$O(f(k) + |I|^c)$$

wobei $f : \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k, *Schwierigkeit der Instanz*
– polynomiell von der Größe $|I|$ der Instanz I.

Ein neues Ziel

Finde einen Algorithmus für k-VC mit Laufzeit

$$O(f(k) + |I|^c)$$

wobei $f\colon \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k, *Schwierigkeit der Instanz*
– polynomiell von der Größe $|I|$ der Instanz I.

Ein Problem, das in dieser Zeit gelöst werden kann, heißt *festparameterberechenbar* (*fixed-parameter tractable*) bzgl. k.

Ein neues Ziel

Finde einen Algorithmus für k-VC mit Laufzeit

$$O(f(k) + |I|^c)$$

wobei $f \colon \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k, *Schwierigkeit der Instanz*
– polynomiell von der Größe $|I|$ der Instanz I.

Ein Problem, das in dieser Zeit gelöst werden kann, heißt *festparameterberechenbar* (*fixed-parameter tractable*) bzgl. k.

$\mathcal{FPT}$ = Klasse der festparameterberechenbaren Probleme.

Ein neues Ziel

Bemerkung.
Die Klasse $\mathcal{FPT}$ ändert sich nicht, wenn statt $+$ hier $\cdot$ steht.

Finde einen Algorithmus für k-VC mit Laufzeit

$$O(f(k) + |I|^c)$$

wobei $f\colon \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k, *Schwierigkeit der Instanz*
– polynomiell von der Größe $|I|$ der Instanz I.

Ein Problem, das in dieser Zeit gelöst werden kann, heißt *festparameterberechenbar* (*fixed-parameter tractable*) bzgl. k.

$\mathcal{FPT}$ = Klasse der festparameterberechenbaren Probleme.

Ein neues Ziel

Bemerkung.
Die Klasse $\mathcal{FPT}$ ändert sich nicht, wenn statt $+$ hier $\cdot$ steht.

Finde einen Algorithmus für k-VC mit Laufzeit

$$O(f(k) + |I|^c) =: O^\star(f(k))$$

wobei $f\colon \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k, *Schwierigkeit der Instanz*
– polynomiell von der Größe $|I|$ der Instanz I.

Ein Problem, das in dieser Zeit gelöst werden kann, heißt *festparameterberechenbar* (*fixed-parameter tractable*) bzgl. k.

$\mathcal{FPT}$ = Klasse der festparameterberechenbaren Probleme.

Ein neues Ziel

Bemerkung.
Die Klasse $\mathcal{FPT}$ ändert sich nicht, wenn statt $+$ hier $\cdot$ steht.

Finde einen Algorithmus für k-VC mit Laufzeit

$$O(f(k) + |I|^c) =: O^\star(f(k))$$

ignoriert polynomielle Faktoren!

wobei $f \colon \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k, *Schwierigkeit der Instanz*
– polynomiell von der Größe $|I|$ der Instanz I.

Ein Problem, das in dieser Zeit gelöst werden kann, heißt *festparameterberechenbar* (*fixed-parameter tractable*) bzgl. k.

$\mathcal{FPT}$ = Klasse der festparameterberechenbaren Probleme.

Ein neues Ziel

Bemerkung.
Die Klasse $\mathcal{FPT}$ ändert sich nicht, wenn statt $+$ hier $\cdot$ steht.

Finde einen Algorithmus für k-VC mit Laufzeit

$$O(f(k) + |I|^c) =: O^\star(f(k))$$

ignoriert polynomielle Faktoren!

wobei $f\colon \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k, *Schwierigkeit der Instanz*
– polynomiell von der Größe $|I|$ der Instanz I.

Ein Problem, das in dieser Zeit gelöst werden kann, heißt *festparameterberechenbar* (*fixed-parameter tractable*) bzgl. k.

$\mathcal{FPT}$ = Klasse der festparameterberechenbaren Probleme.

Bem. BruteForceVC hat die gewünschte Laufzeit.

Ein neues Ziel

Bemerkung.
Die Klasse $\mathcal{FPT}$ ändert sich nicht, wenn statt $+$ hier $\cdot$ steht.

Finde einen Algorithmus für k-VC mit Laufzeit

$$O(f(k) + |I|^c) =: O^\star(f(k))$$

ignoriert polynomielle Faktoren!

wobei $f\colon \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k, *Schwierigkeit der Instanz*
– polynomiell von der Größe $|I|$ der Instanz I.

Ein Problem, das in dieser Zeit gelöst werden kann, heißt *festparameterberechenbar* (*fixed-parameter tractable*) bzgl. k.

$\mathcal{FPT}$ = Klasse der festparameterberechenbaren Probleme.

Bem. BruteForceVC hat *nicht* die gewünschte Laufzeit.

Ein paar einfache Beobachtungen. . .

Sei G Graph, C ein VC für G, v Knoten, der nicht in C liegt.
Welche Knoten liegen dann sicher in C?

Ein paar einfache Beobachtungen...

Sei G Graph, C ein VC für G, v Knoten, der nicht in C liegt. Welche Knoten liegen dann sicher in C?

Beob. 1. Sei G ein Graph, C ein VC für G, v ein Knoten. Dann gilt: $v \in C$ oder $N_G(v) \subseteq C$.

Ein paar einfache Beobachtungen. . .

Sei G Graph, C ein VC für G, v Knoten, der nicht in C liegt.
Welche Knoten liegen dann sicher in C?

Beob. 1. Sei G ein Graph, C ein VC für G, v ein Knoten.
Dann gilt: $v \in C$ oder $N_G(v) \subseteq C$.

Betrachte Entscheidungsproblem k-VC.
Was gilt für Knoten mit Grad $> k$?

Ein paar einfache Beobachtungen. . .

Sei G Graph, C ein VC für G, v Knoten, der nicht in C liegt.
Welche Knoten liegen dann sicher in C?

Beob. 1. Sei G ein Graph, C ein VC für G, v ein Knoten.
Dann gilt: $v \in C$ oder $N_G(v) \subseteq C$.

Betrachte Entscheidungsproblem k-VC.
Was gilt für Knoten mit Grad $> k$?

Beob. 2. Jeder Knoten mit Grad $> k$ ist in jedem k-VC von G enthalten.

Ein paar einfache Beobachtungen...

Sei G Graph, C ein VC für G, v Knoten, der nicht in C liegt.
Welche Knoten liegen dann sicher in C?

Beob. 1. Sei G ein Graph, C ein VC für G, v ein Knoten.
Dann gilt: $v \in C$ oder $N_G(v) \subseteq C$.

Betrachte Entscheidungsproblem k-VC.
Was gilt für Knoten mit Grad $> k$?

Beob. 2. Jeder Knoten mit Grad $> k$ ist in jedem k-VC von G enthalten.

Was gilt, falls $|E(G)| > k^2$ und alle Knoten Grad $\leq k$ haben?

Ein paar einfache Beobachtungen...

Sei G Graph, C ein VC für G, v Knoten, der nicht in C liegt.
Welche Knoten liegen dann sicher in C?

Beob. 1. Sei G ein Graph, C ein VC für G, v ein Knoten. Dann gilt: $v \in C$ oder $N_G(v) \subseteq C$.

Betrachte Entscheidungsproblem k-VC.
Was gilt für Knoten mit Grad $> k$?

Beob. 2. Jeder Knoten mit Grad $> k$ ist in jedem k-VC von G enthalten.

Was gilt, falls $|E(G)| > k^2$ und alle Knoten Grad $\leq k$ haben?

Beob. 3. Falls $|E(G)| > k^2$ und $\Delta(G) \leq k$ (wobei $\Delta(G) = \max_{v \in V(G)} \deg(v)$), so hat G kein k-VC.

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$

(yesorno, C') $=$ BruteForceVC(G', k')

return (yesorno, $C \cup C'$)

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

Benützen Sie Beobachtungen 2 & 3, um eine Instanz (G, k) in eine Instanz (G', k') zu verwandeln, so dass $|G'| \in O(k^2)$!

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$

(yesorno, C') $=$ BruteForceVC(G', k')

return (yesorno, $C \cup C'$)

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$C = \{v \in V(G) \mid \deg_G(v) > k\}$

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$

(yesorno, C') $=$ BruteForceVC(G', k')

return (yesorno, $C \cup C'$)

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$C = \{v \in V(G) \mid \deg_G(v) > k\}$

if $|C| > k$ **then return** (“no”, $\emptyset$)

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$

(yesorno, C') $=$ BruteForceVC(G', k')

return (yesorno, $C \cup C'$)

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$C = \{v \in V(G) \mid \deg_G(v) > k\}$

if $|C| > k$ **then return** (“no”, $\emptyset$)

$G' := G[V \setminus C]$ (entferne auch isolierte Knoten)

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$

(yesorno, C') $=$ BruteForceVC(G', k')

return (yesorno, $C \cup C'$)

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$C = \{v \in V(G) \mid \deg_G(v) > k\}$

if $|C| > k$ **then return** (“no”, $\emptyset$)

$G' := G[V \setminus C]$ (entferne auch isolierte Knoten)

if $|E(G')| > k^2$ **then return** (“no”, $\emptyset$)

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$

(yesorno, C') = BruteForceVC(G', k')

return (yesorno, $C \cup C'$)

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$C = \{v \in V(G) \mid \deg_G(v) > k\}$

if $|C| > k$ **then return** (“no”, $\emptyset$)

$G' := G[V \setminus C]$ (entferne auch isolierte Knoten)

if $|E(G')| > k^2$ **then return** (“no”, $\emptyset$)

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$

(yesorno, C') = BruteForceVC(G', k')

return (yesorno, $C \cup C'$)

Laufzeit.

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$C = \{v \in V(G) \mid \deg_G(v) > k\}$
if $|C| > k$ **then return** (“no”, $\emptyset$)
$G' := G[V \setminus C]$ (entferne auch isolierte Knoten)
if $|E(G')| > k^2$ **then return** (“no”, $\emptyset$)

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$
(yesorno, C') $=$ BruteForceVC(G', k')
return (yesorno, $C \cup C'$)

Laufzeit.

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$$\left.\begin{array}{l} C = \{v \in V(G) \mid \deg_G(v) > k\} \\ \textbf{if } |C| > k \textbf{ then return } (\text{“no”}, \emptyset) \\ G' := G[V \setminus C] \quad \text{(entferne auch isolierte Knoten)} \\ \textbf{if } |E(G')| > k^2 \textbf{ then return } (\text{“no”}, \emptyset) \end{array}\right\} O(n+m) \text{ Zeit}$$

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$

$(\text{yesorno}, C') = \text{BruteForceVC}(G', k')$

return $(\text{yesorno}, C \cup C')$

Laufzeit.

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$$\left.\begin{array}{l} C = \{v \in V(G) \mid \deg_G(v) > k\} \\ \textbf{if } |C| > k \textbf{ then return } (\text{“no”}, \emptyset) \\ G' := G[V \setminus C] \quad \text{(entferne auch isolierte Knoten)} \\ \textbf{if } |E(G')| > k^2 \textbf{ then return } (\text{“no”}, \emptyset) \end{array}\right\} \begin{array}{l} O(n+m) \\ \text{Zeit} \end{array}$$

II) Lösung der Restinstanz mit roher Gewalt

$$k' = k - |C|$$

$$\left.\begin{array}{l} (\text{yesorno}, C') = \text{BruteForceVC}(G', k') \\ \textbf{return } (\text{yesorno}, C \cup C') \end{array}\right\}$$

Laufzeit.

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$C = \{v \in V(G) \mid \deg_G(v) > k\}$
if $|C| > k$ **then return** (“no”, $\emptyset$)
$G' := G[V \setminus C]$ (entferne auch isolierte Knoten)
if $|E(G')| > k^2$ **then return** (“no”, $\emptyset$)

$O(n+m)$ Zeit

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$
(yesorno, C') $=$ BruteForceVC(G', k')
return (yesorno, $C \cup C'$)

$O(m' \cdot (n')^{k'})$ Zeit

Laufzeit.

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$C = \{v \in V(G) \mid \deg_G(v) > k\}$
if $|C| > k$ **then return** (“no”, $\emptyset$)
$G' := G[V \setminus C]$ (entferne auch isolierte Knoten)
if $|E(G')| > k^2$ **then return** (“no”, $\emptyset$)

$O(n+m)$ Zeit

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$
(yesorno, C') = BruteForceVC(G', k')
return (yesorno, $C \cup C'$)

$O(m' \cdot (n')^{k'})$ Zeit
wobei $m' := |E(G')| \leq k^2$

Laufzeit.

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$C = \{v \in V(G) \mid \deg_G(v) > k\}$
if $|C| > k$ **then return** (“no”, $\emptyset$)
$G' := G[V \setminus C]$ (entferne auch isolierte Knoten)
if $|E(G')| > k^2$ **then return** (“no”, $\emptyset$)

$O(n + m)$ Zeit

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$
(yesorno, C') = BruteForceVC(G', k')
return (yesorno, $C \cup C'$)

$O(m' \cdot (n')^{k'})$ Zeit
wobei $m' := |E(G')| \leq k^2$
$\Rightarrow n' := |V(G')| \leq 2k^2$

Laufzeit.

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$$\left.\begin{array}{l} C = \{v \in V(G) \mid \deg_G(v) > k\} \\ \textbf{if } |C| > k \textbf{ then return } (\text{“no”}, \emptyset) \\ G' := G[V \setminus C] \quad \text{(entferne auch isolierte Knoten)} \\ \textbf{if } |E(G')| > k^2 \textbf{ then return } (\text{“no”}, \emptyset) \end{array}\right\} O(n+m) \text{ Zeit}$$

II) Lösung der Restinstanz mit roher Gewalt

$$k' = k - |C|$$

$$\left.\begin{array}{l} (\text{yesorno}, C') = \text{BruteForceVC}(G', k') \\ \textbf{return } (\text{yesorno}, C \cup C') \end{array}\right\} \begin{array}{l} O(m' \cdot (n')^{k'}) \text{ Zeit} \\ \text{wobei } m' := |E(G')| \leq k^2 \\ \quad \Rightarrow \; n' := |V(G')| \leq 2k^2 \end{array}$$

Laufzeit. $O\big(n + m + k^2 \cdot (2k^2)^k\big)$

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$C = \{v \in V(G) \mid \deg_G(v) > k\}$

if $|C| > k$ **then return** (“no”, $\emptyset$)

$G' := G[V \setminus C]$ (entferne auch isolierte Knoten)

if $|E(G')| > k^2$ **then return** (“no”, $\emptyset$)

$O(n+m)$ Zeit

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$

(yesorno, C') = BruteForceVC(G', k')

return (yesorno, $C \cup C'$)

$O(m' \cdot (n')^{k'})$ Zeit

wobei $m' := |E(G')| \leq k^2$

$\Rightarrow n' := |V(G')| \leq 2k^2$

Laufzeit. $O\big(n + m + k^2 \cdot (2k^2)^k\big) = O\big(n + m + k^2 2^k k^{2k}\big)$

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$$\left.\begin{array}{l} C = \{v \in V(G) \mid \deg_G(v) > k\} \\ \textbf{if } |C| > k \textbf{ then return } (\text{“no”}, \emptyset) \\ G' := G[V \setminus C] \quad \text{(entferne auch isolierte Knoten)} \\ \textbf{if } |E(G')| > k^2 \textbf{ then return } (\text{“no”}, \emptyset) \end{array}\right\} \begin{array}{l} O(n+m) \\ \text{Zeit} \end{array}$$

II) Lösung der Restinstanz mit roher Gewalt

$$k' = k - |C|$$

$$\left.\begin{array}{l} (\text{yesorno}, C') = \text{BruteForceVC}(G', k') \\ \textbf{return } (\text{yesorno}, C \cup C') \end{array}\right\} \begin{array}{l} O(m' \cdot (n')^{k'}) \text{ Zeit} \\ \text{wobei } m' := |E(G')| \leq k^2 \\ \quad \Rightarrow \; n' := |V(G')| \leq 2k^2 \end{array}$$

Laufzeit. $O\big(n + m + k^2 \cdot (2k^2)^k\big) = O\big(\underbrace{n + m}_{|I|^1} + k^2 2^k k^{2k}\big)$

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$C = \{v \in V(G) \mid \deg_G(v) > k\}$
if $|C| > k$ **then return** ("no", $\emptyset$)
$G' := G[V \setminus C]$ (entferne auch isolierte Knoten)
if $|E(G')| > k^2$ **then return** ("no", $\emptyset$)

} $O(n+m)$ Zeit

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$
(yesorno, C') = BruteForceVC(G', k')
return (yesorno, $C \cup C'$)

} $O(m' \cdot (n')^{k'})$ Zeit
wobei $m' := |E(G')| \leq k^2$
$\Rightarrow n' := |V(G')| \leq 2k^2$

Laufzeit. $O\big(n + m + k^2 \cdot (2k^2)^k\big) = O\big(\underbrace{n+m}_{|I|^1} + \underbrace{k^2 2^k k^{2k}}_{f(k)}\big)$

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$$\left.\begin{array}{l} C = \{v \in V(G) \mid \deg_G(v) > k\} \\ \textbf{if } |C| > k \textbf{ then return } (\text{“no”}, \emptyset) \\ G' := G[V \setminus C] \quad \text{(entferne auch isolierte Knoten)} \\ \textbf{if } |E(G')| > k^2 \textbf{ then return } (\text{“no”}, \emptyset) \end{array}\right\} O(n+m) \text{ Zeit}$$

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$

$$\left.\begin{array}{l} (\text{yesorno}, C') = \text{BruteForceVC}(G', k') \\ \textbf{return } (\text{yesorno}, C \cup C') \end{array}\right\} \begin{array}{l} O(m' \cdot (n')^{k'}) \text{ Zeit} \\ \text{wobei } m' := |E(G')| \leq k^2 \\ \quad \Rightarrow \; n' := |V(G')| \leq 2k^2 \end{array}$$

Laufzeit. $O\big(n + m + k^2 \cdot (2k^2)^k\big) = O\big(\underbrace{n + m}_{|I|^1} + \underbrace{k^2 2^k k^{2k}}_{f(k)}\big)$

Also:

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$$\left.\begin{array}{l} C = \{v \in V(G) \mid \deg_G(v) > k\} \\ \textbf{if } |C| > k \textbf{ then return } (\text{“no”}, \emptyset) \\ G' := G[V \setminus C] \quad \text{(entferne auch isolierte Knoten)} \\ \textbf{if } |E(G')| > k^2 \textbf{ then return } (\text{“no”}, \emptyset) \end{array}\right\} \begin{array}{l} O(n+m) \\ \text{Zeit} \end{array}$$

II) Lösung der Restinstanz mit roher Gewalt

$$\begin{array}{l} k' = k - |C| \\ \left.\begin{array}{l} (\text{yesorno}, C') = \text{BruteForceVC}(G', k') \\ \textbf{return } (\text{yesorno}, C \cup C') \end{array}\right\} \begin{array}{l} O(m' \cdot (n')^{k'}) \text{ Zeit} \\ \text{wobei } m' := |E(G')| \leq k^2 \\ \quad \Rightarrow n' := |V(G')| \leq 2k^2 \end{array} \end{array}$$

Laufzeit. $O\big(n + m + k^2 \cdot (2k^2)^k\big) = O\big(\underbrace{n + m}_{|I|^1} + \underbrace{k^2 2^k k^{2k}}_{f(k)}\big)$

Also: $k\text{-VC} \in \mathcal{FPT}$!

Algorithmus von Buss

BussVC(Graph G, Integer k)

I) Reduktion der Instanz auf ihren harten Kern

$C = \{v \in V(G) \mid \deg_G(v) > k\}$
if $|C| > k$ **then return** (“no”, $\emptyset$)
$G' := G[V \setminus C]$ (entferne auch isolierte Knoten)
if $|E(G')| > k^2$ **then return** (“no”, $\emptyset$)

$O(n+m)$ Zeit

II) Lösung der Restinstanz mit roher Gewalt

$k' = k - |C|$
(yesorno, C') = BruteForceVC(G', k')
return (yesorno, $C \cup C'$)

$O(m' \cdot (n')^{k'})$ Zeit
wobei $m' := |E(G')| \le k^2$
$\Rightarrow\ n' := |V(G')| \le 2k^2$

Laufzeit. $O\big(n + m + k^2 \cdot (2k^2)^k\big) = O\big(\underbrace{n + m}_{|I|^1} + \underbrace{k^2 2^k \boxed{k^{2k}}}_{f(k)}\big)$

Also: k-VC $\in \mathcal{FPT}$!

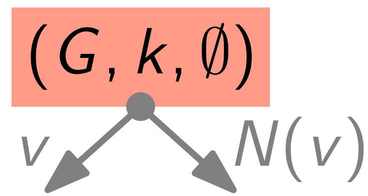
Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.

$(G, k, \emptyset)$

Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.



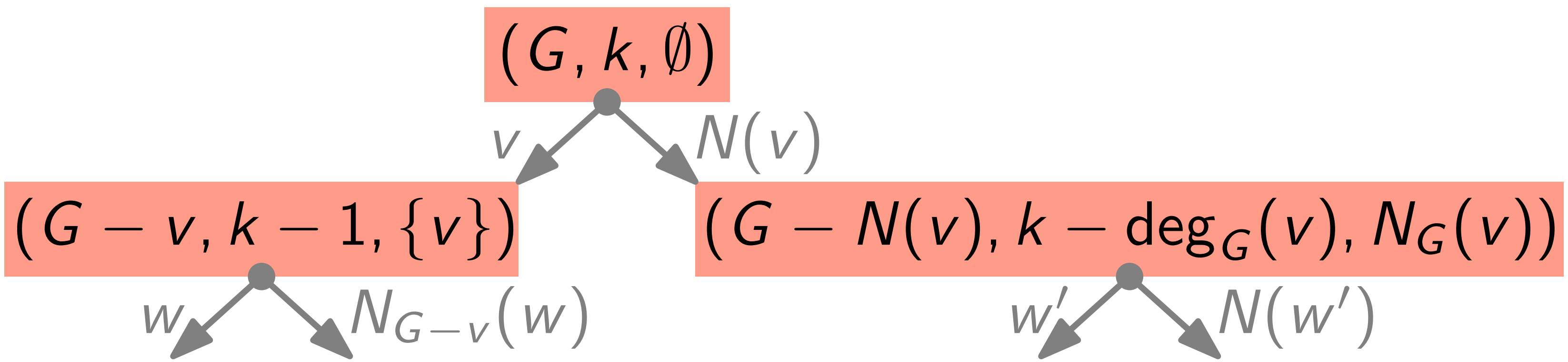
Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.

$$\begin{tikzcd}[column sep=small]
 & (G, k, \emptyset) \arrow[dl, "v"'] \arrow[dr, "N(v)"] & \\
(G - v, k - 1, \{v\}) & & (G - N(v), k - \deg_G(v), N_G(v))
\end{tikzcd}$$

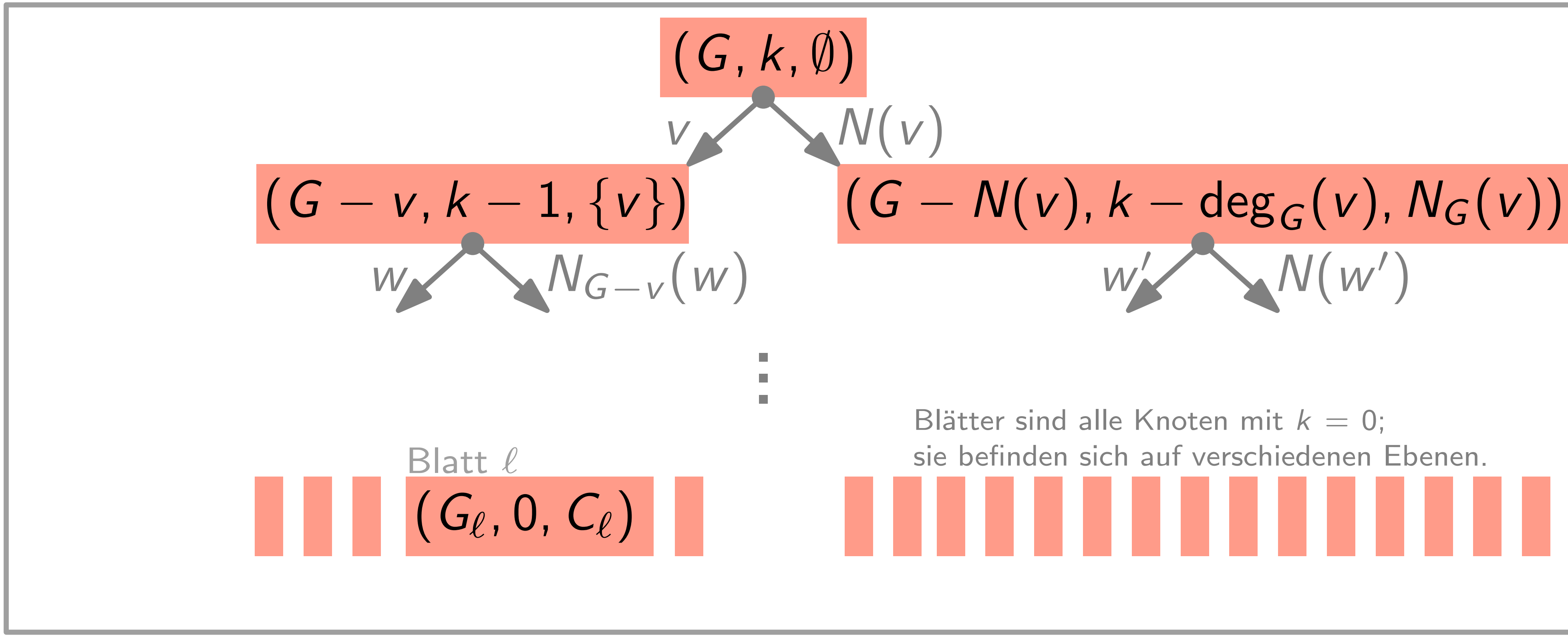
Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.



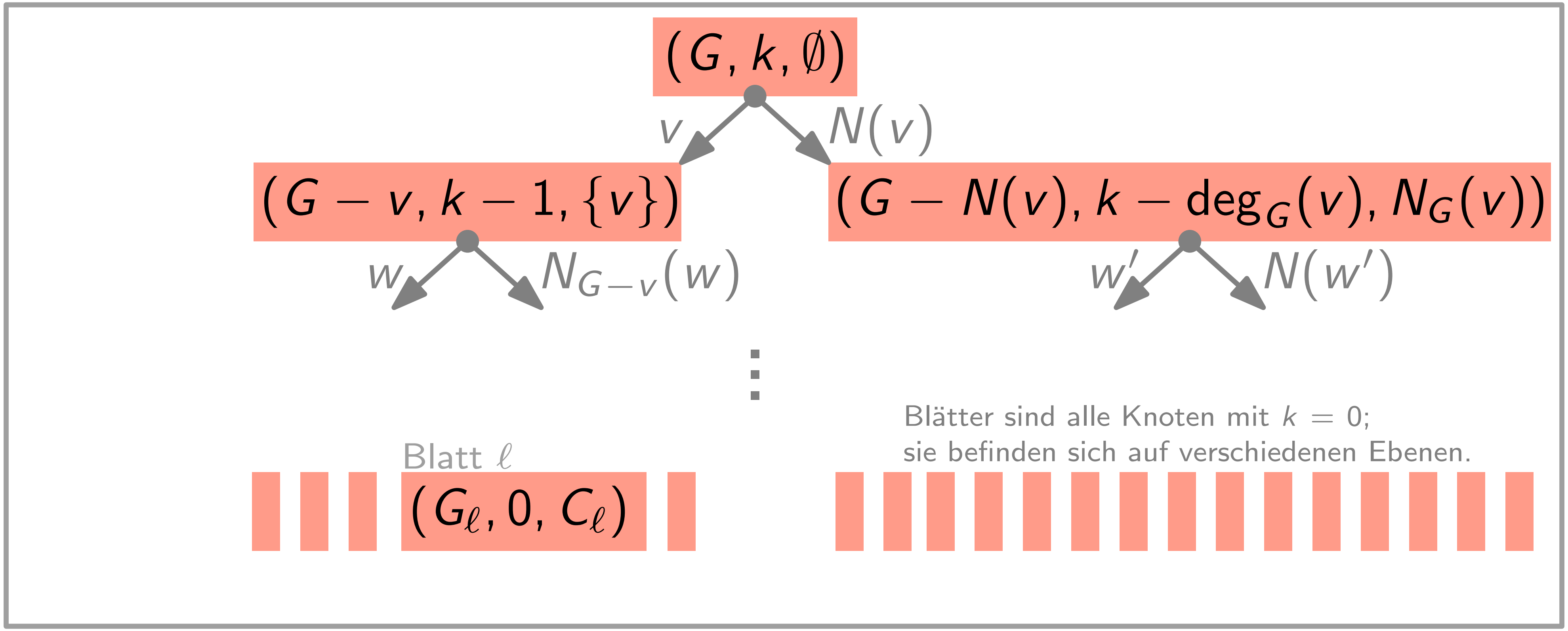
Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.



Suchbaum-Algorithmus

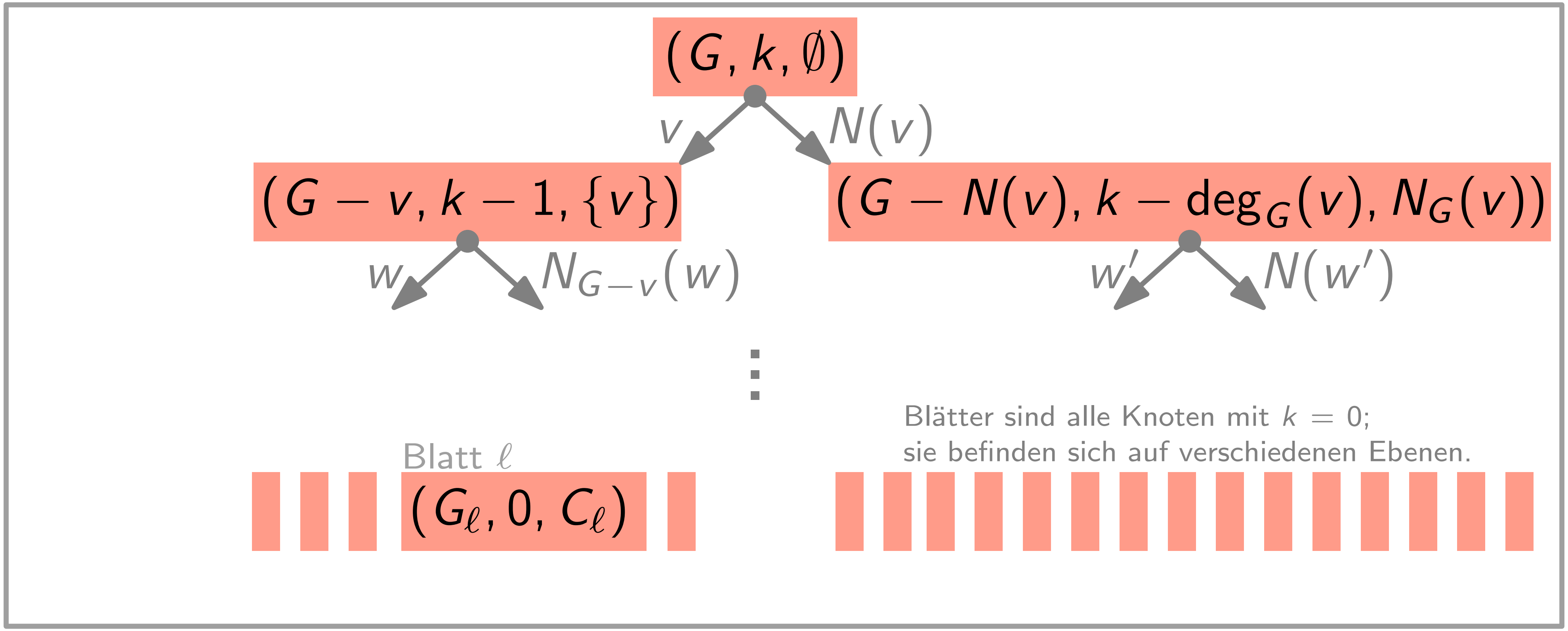
Idee. Verbessere Phase II durch Aufbau eines Suchbaums.



JA:

Suchbaum-Algorithmus

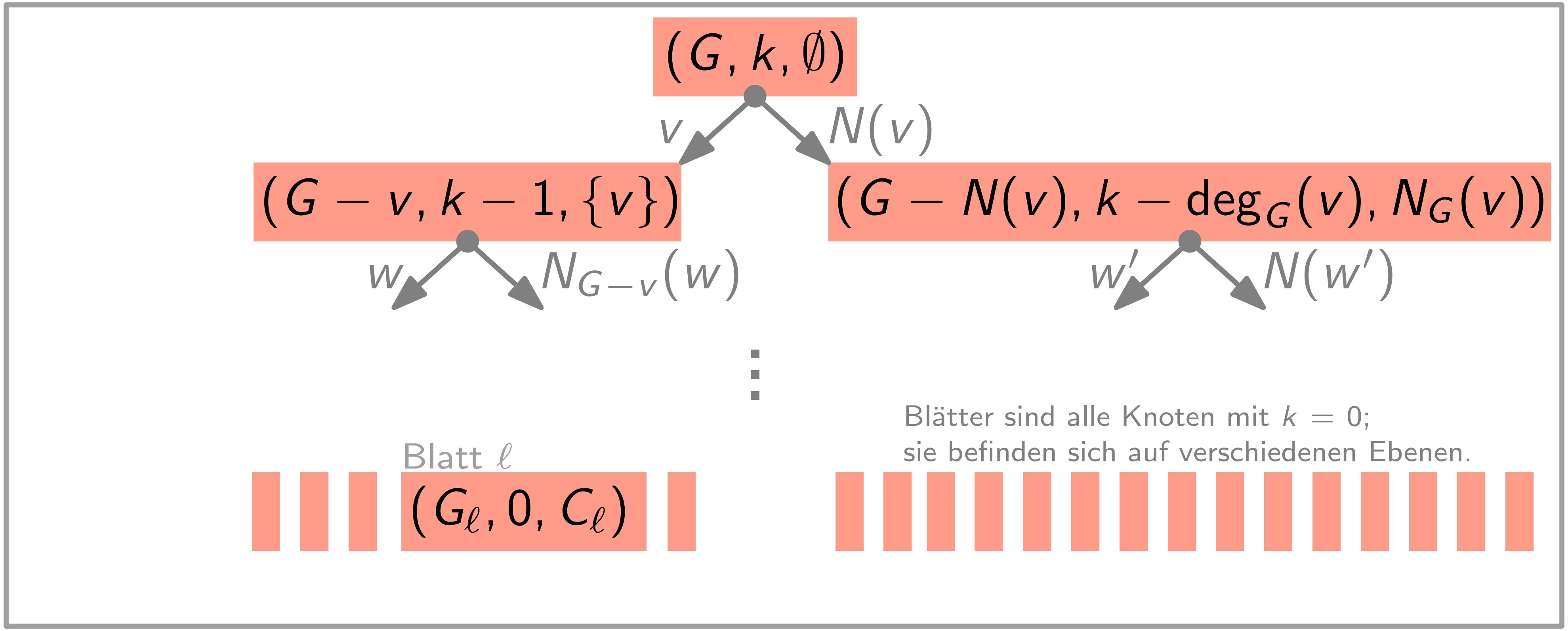
Idee. Verbessere Phase II durch Aufbau eines Suchbaums.



JA: Gibt es ein Blatt ℓ mit $E_\ell = \emptyset$, so ist C_ℓ ein k-VC von G.

Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.

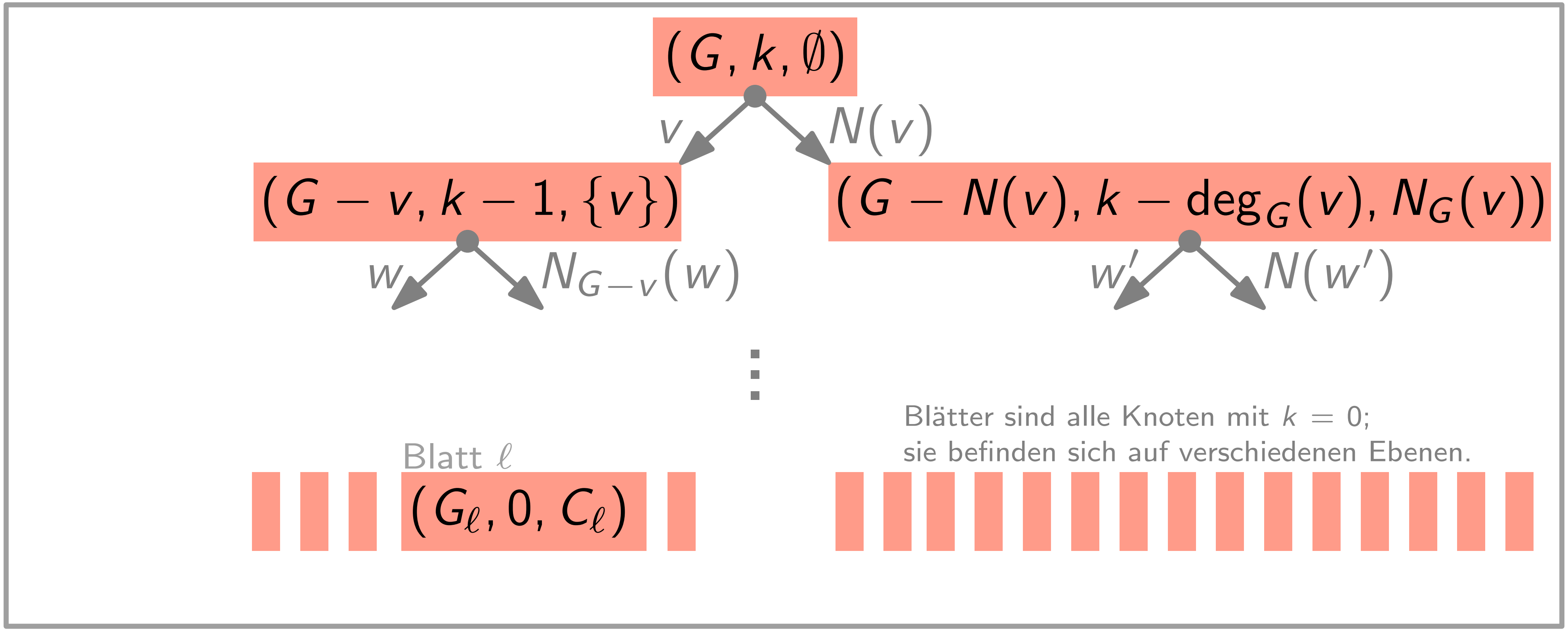


JA: Gibt es ein Blatt ℓ mit $E_\ell = \emptyset$, so ist C_ℓ ein k-VC von G.

NEIN:

Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.

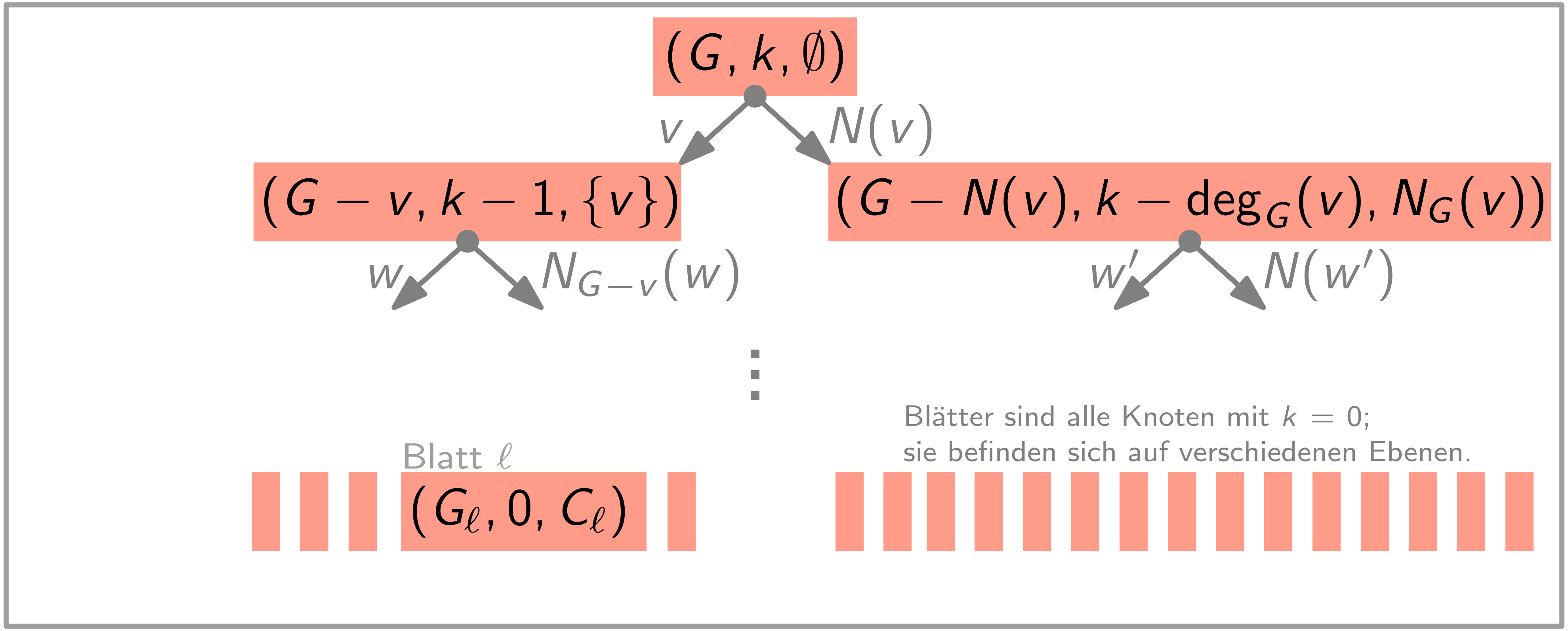


JA: Gibt es ein Blatt ℓ mit $E_\ell = \emptyset$, so ist C_ℓ ein k-VC von G.

NEIN: Gibt es kein solches Blatt, so hat G kein k-VC.

Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.

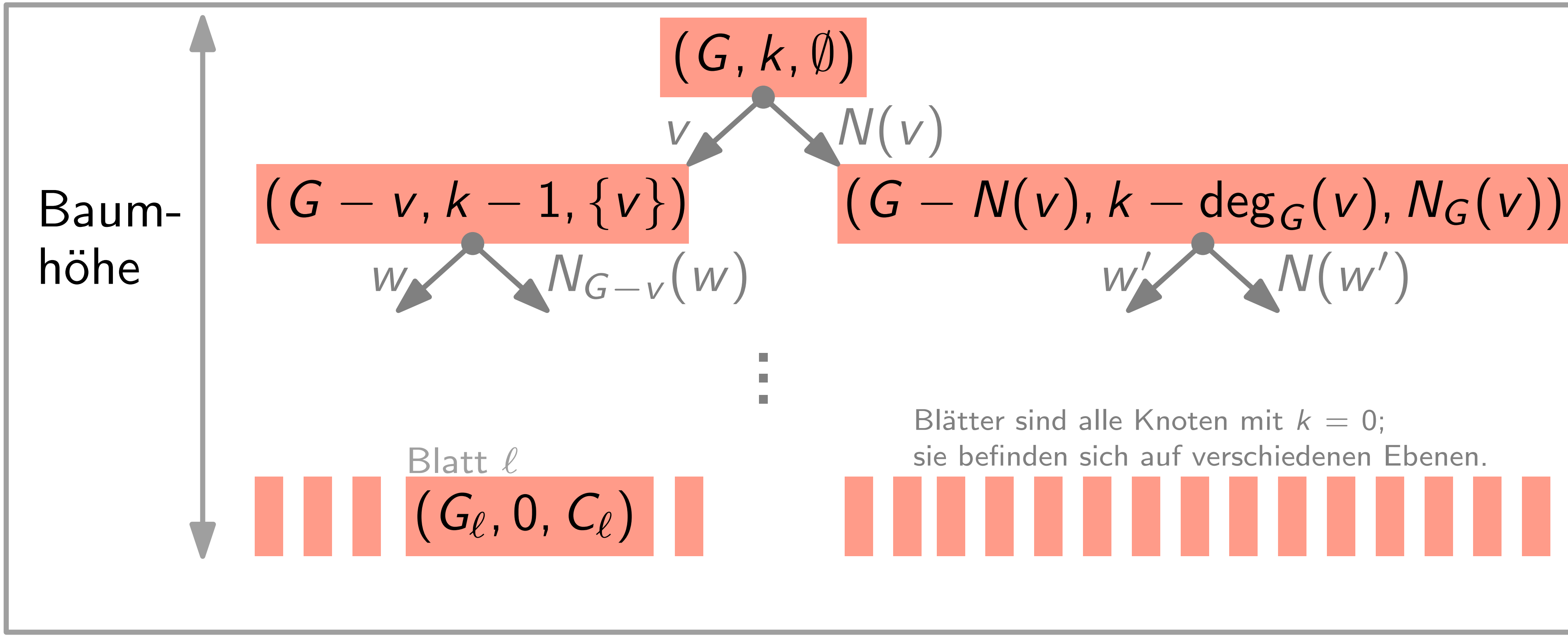


JA: Gibt es ein Blatt ℓ mit $E_\ell = \emptyset$, so ist C_ℓ ein k-VC von G.

NEIN: Gibt es kein solches Blatt, so hat G kein k-VC. (Warum?)

Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.

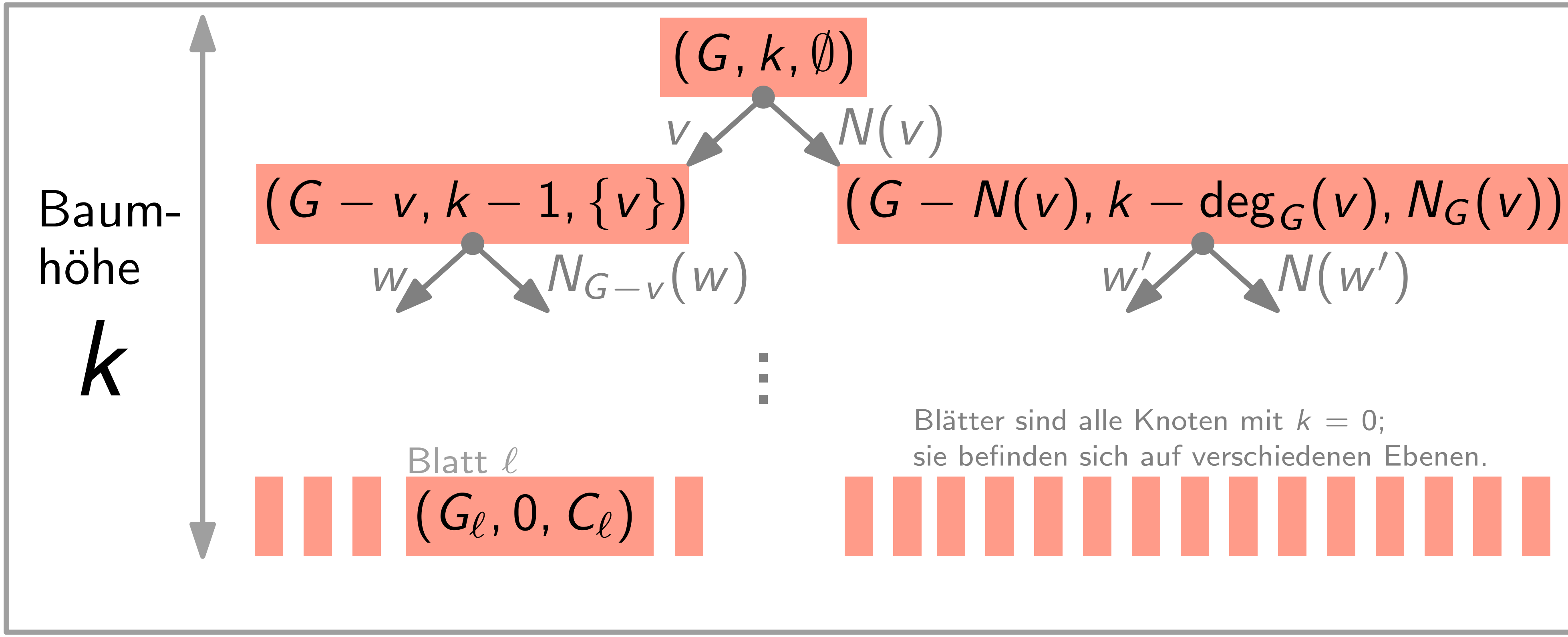


JA: Gibt es ein Blatt ℓ mit $E_\ell = \emptyset$, so ist C_ℓ ein k-VC von G.

NEIN: Gibt es kein solches Blatt, so hat G kein k-VC. (Warum?)

Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.

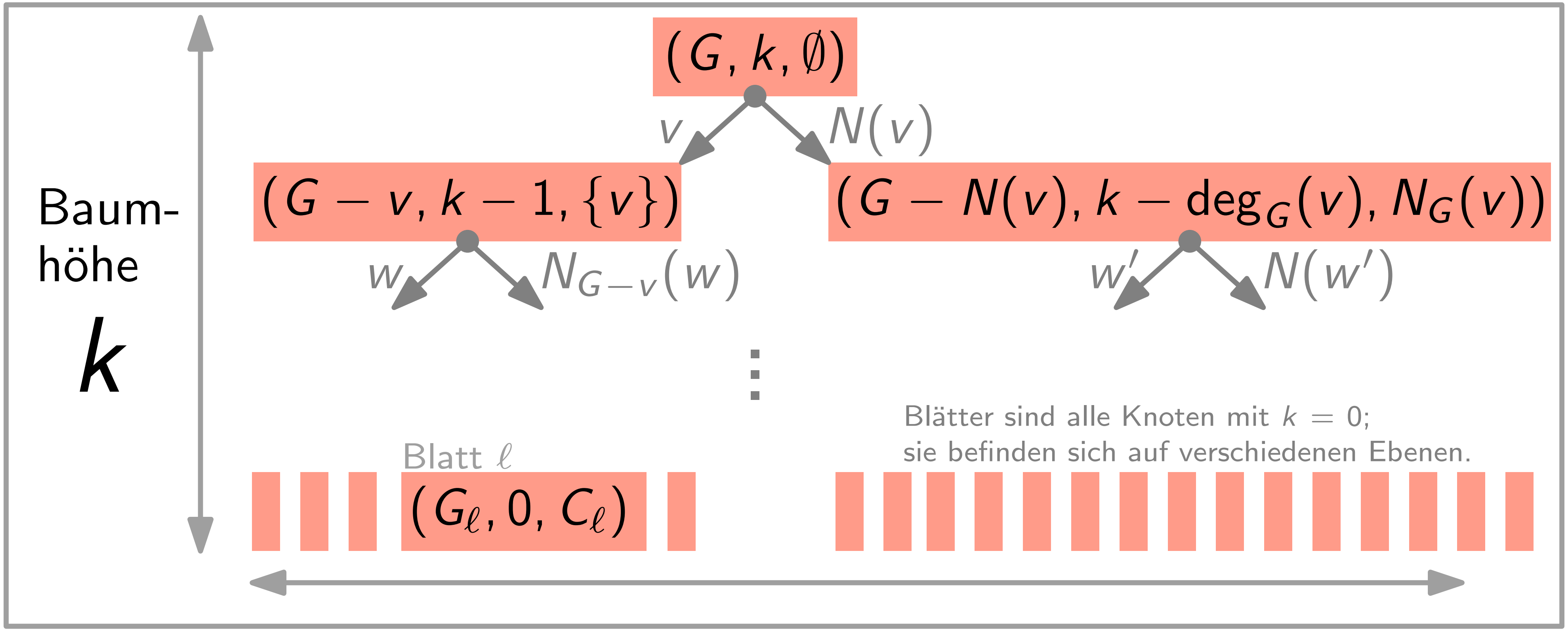


JA: Gibt es ein Blatt ℓ mit $E_\ell = \emptyset$, so ist C_ℓ ein k-VC von G.

NEIN: Gibt es kein solches Blatt, so hat G kein k-VC. (Warum?)

Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.



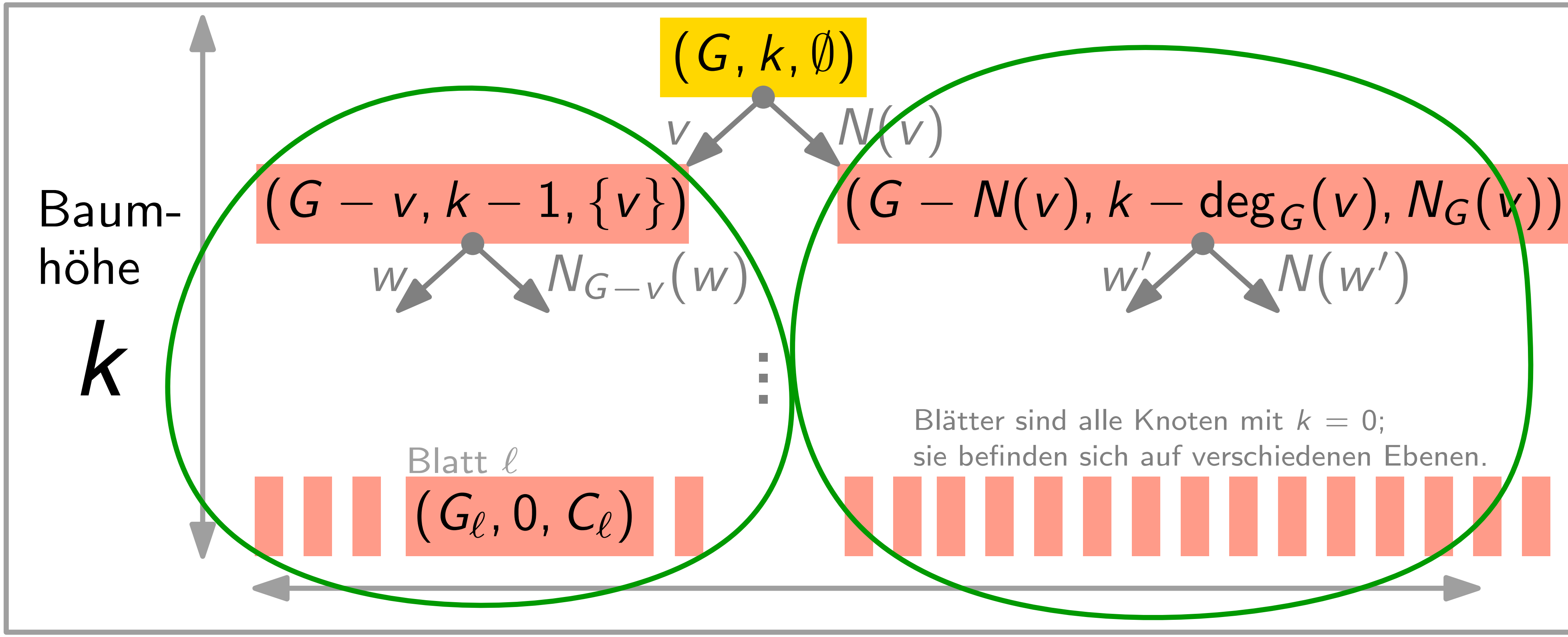
#Knoten: $T(k) \leq$

JA: Gibt es ein Blatt ℓ mit $E_\ell = \emptyset$, so ist C_ℓ ein k-VC von G.

NEIN: Gibt es kein solches Blatt, so hat G kein k-VC. (Warum?)

Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.



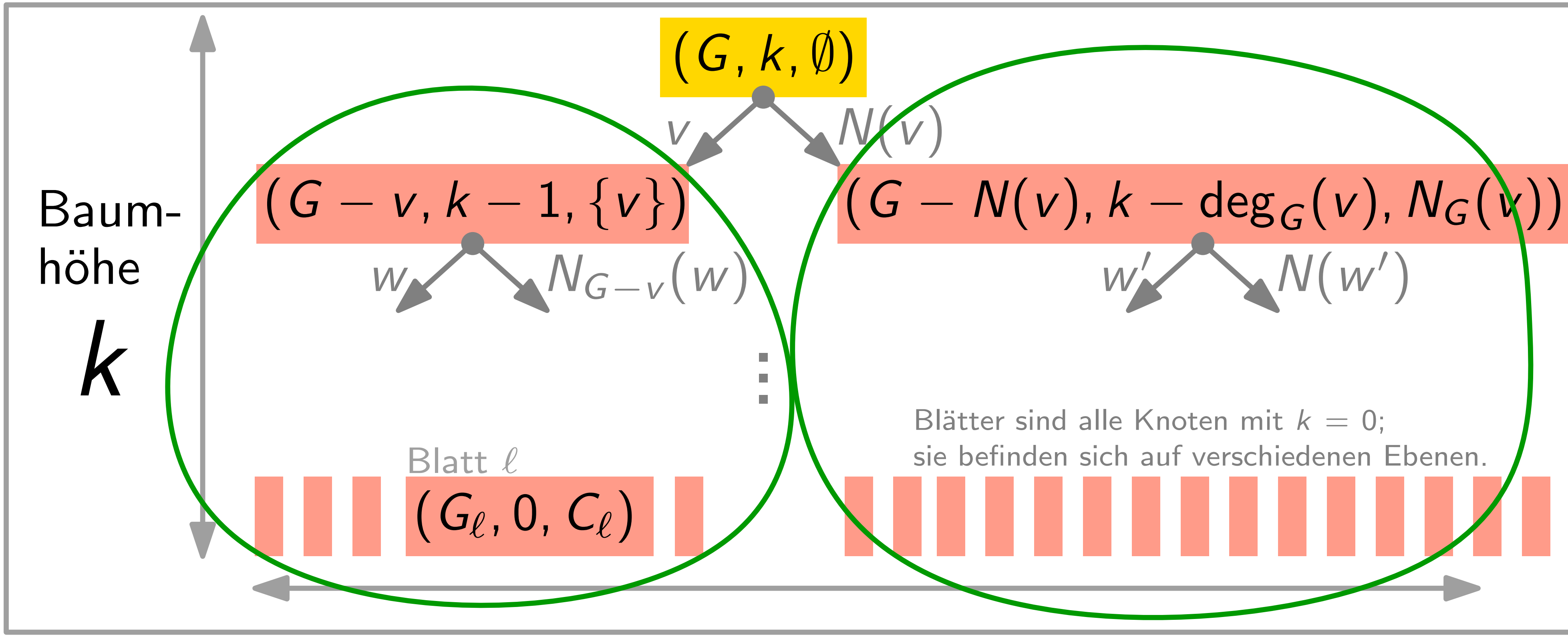
#Knoten: $T(k) \leq$

JA: Gibt es ein Blatt ℓ mit $E_\ell = \emptyset$, so ist C_ℓ ein k-VC von G.

NEIN: Gibt es kein solches Blatt, so hat G kein k-VC. (Warum?)

Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.



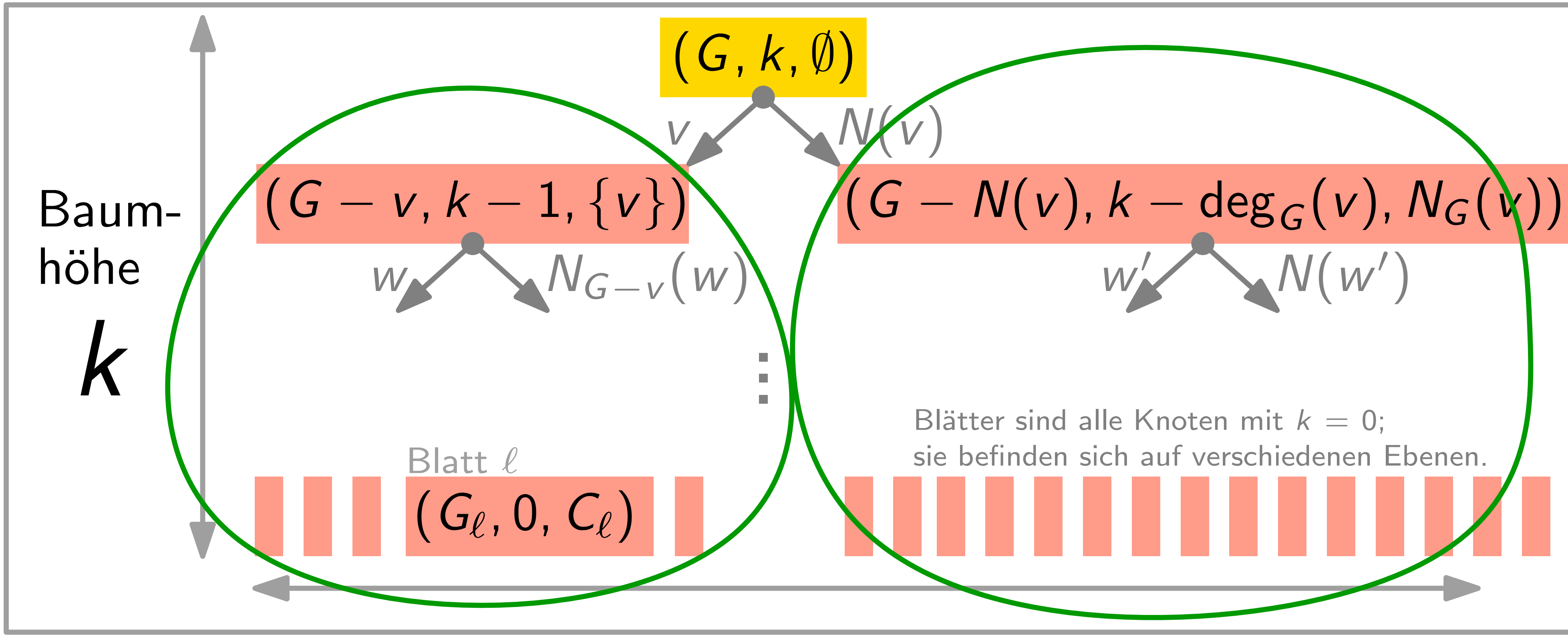
#Knoten: $T(k) \leq 2T(k-1) + \boxed{1}$

JA: Gibt es ein Blatt ℓ mit $E_\ell = \emptyset$, so ist C_ℓ ein k-VC von G.

NEIN: Gibt es kein solches Blatt, so hat G kein k-VC. (Warum?)

Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.

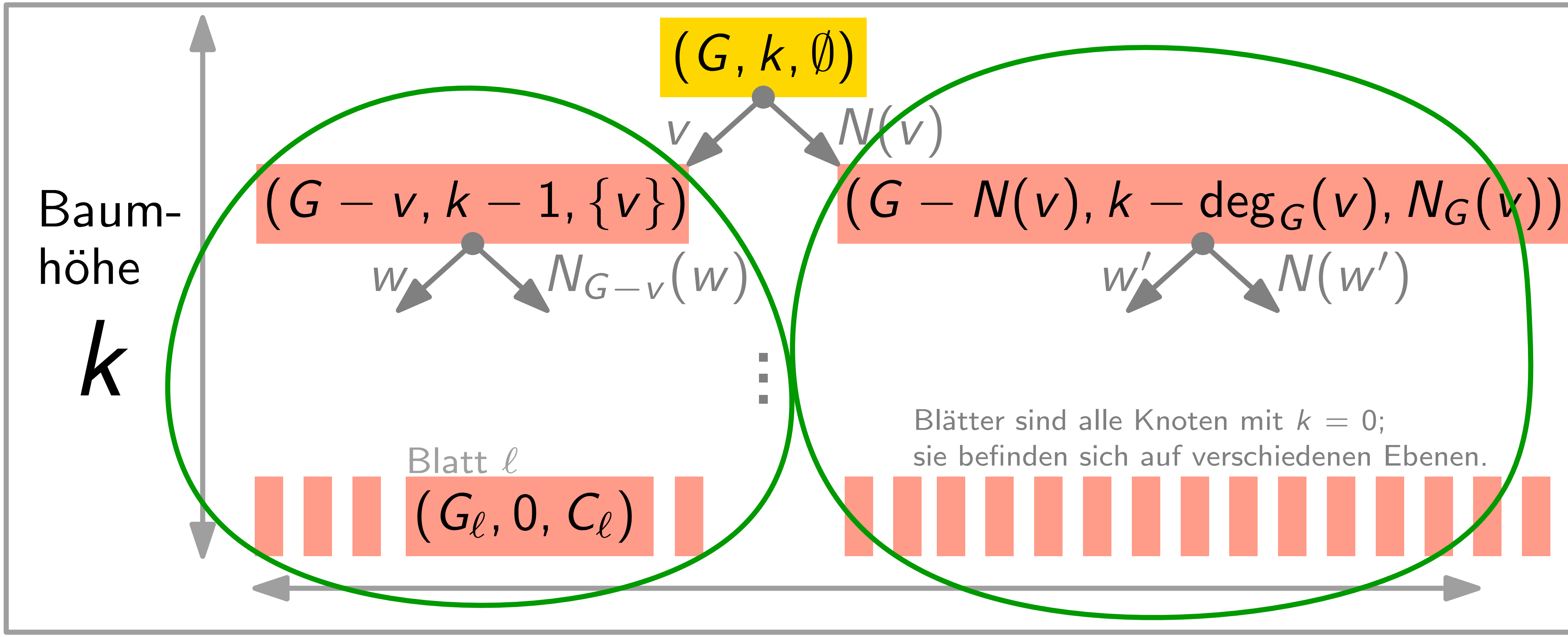


#Knoten: $T(k) \leq 2T(k-1) + \boxed{1}$, $T(0) = 1$

JA: Gibt es ein Blatt ℓ mit $E_\ell = \emptyset$, so ist C_ℓ ein k-VC von G.
NEIN: Gibt es kein solches Blatt, so hat G kein k-VC. (Warum?)

Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.



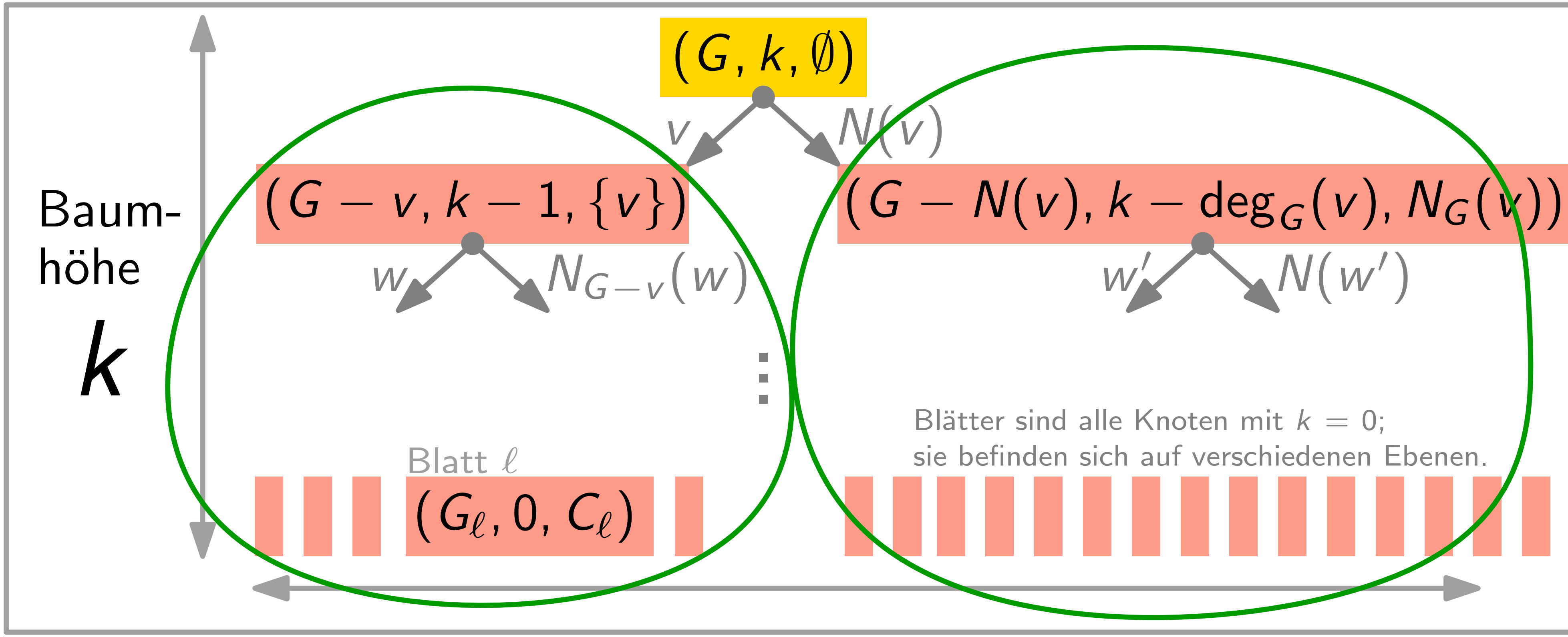
#Knoten: $T(k) \leq 2T(k-1) + \boxed{1}$, $T(0) = 1 \Rightarrow T(k) \leq 2^{k+1} - 1 \in O(2^k)$

JA: Gibt es ein Blatt ℓ mit $E_\ell = \emptyset$, so ist C_ℓ ein k-VC von G.

NEIN: Gibt es kein solches Blatt, so hat G kein k-VC. (Warum?)

Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.



#Knoten: $T(k) \leq 2\,T(k-1) + \boxed{1}$, $T(0) = 1 \;\Rightarrow\; T(k) \leq 2^{k+1} - 1 \in O(2^k)$

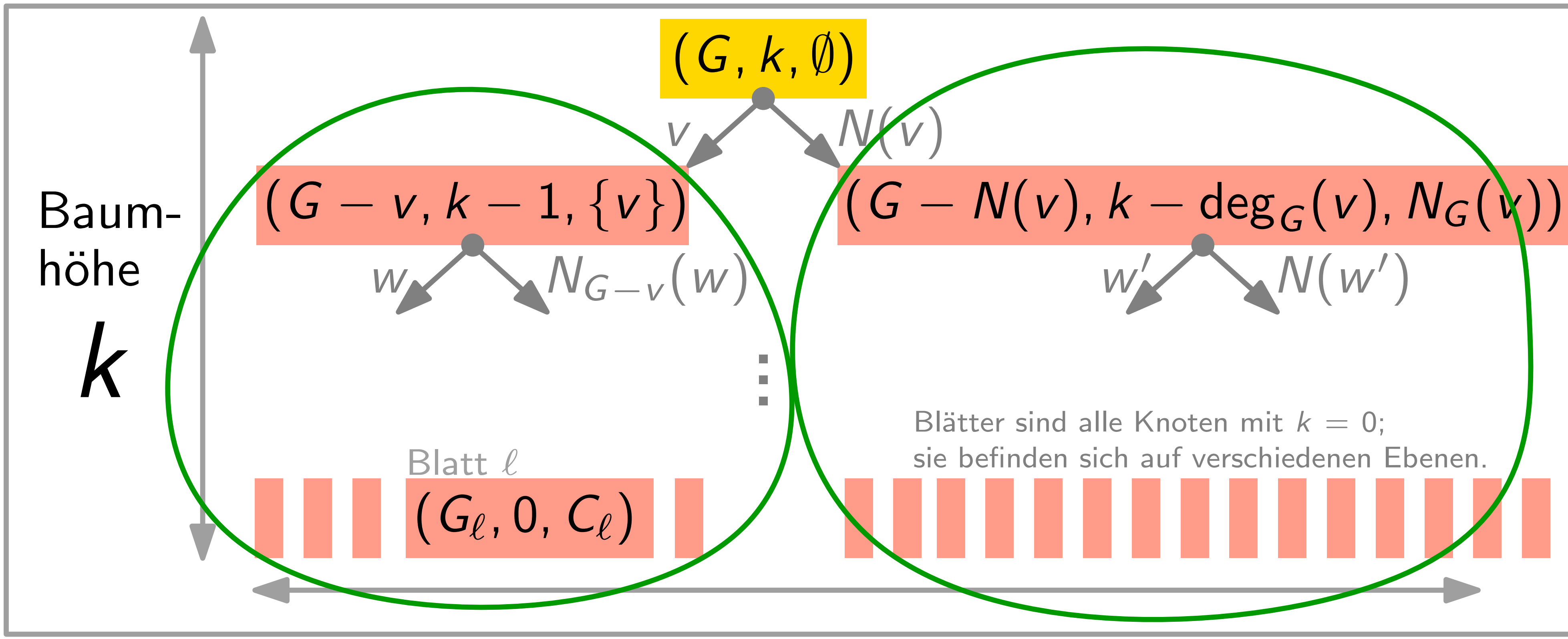
$\Rightarrow$ **Laufzeit:**

JA: Gibt es ein Blatt ℓ mit $E_\ell = \emptyset$, so ist C_ℓ ein k-VC von G.

NEIN: Gibt es kein solches Blatt, so hat G kein k-VC. (Warum?)

Suchbaum-Algorithmus

Idee. Verbessere Phase II durch Aufbau eines Suchbaums.



#Knoten: $T(k) \leq 2T(k-1) + \boxed{1}$, $T(0) = 1 \Rightarrow T(k) \leq 2^{k+1} - 1 \in O(2^k)$

$\Rightarrow$ **Laufzeit:** $O^\star(2^k)$

JA: Gibt es ein Blatt ℓ mit $E_\ell = \emptyset$, so ist C_ℓ ein k-VC von G.

NEIN: Gibt es kein solches Blatt, so hat G kein k-VC. (Warum?)

Der Grad-3-Algorithmus

Idee. Verbessere Abschätzung von $|N(v)|$.

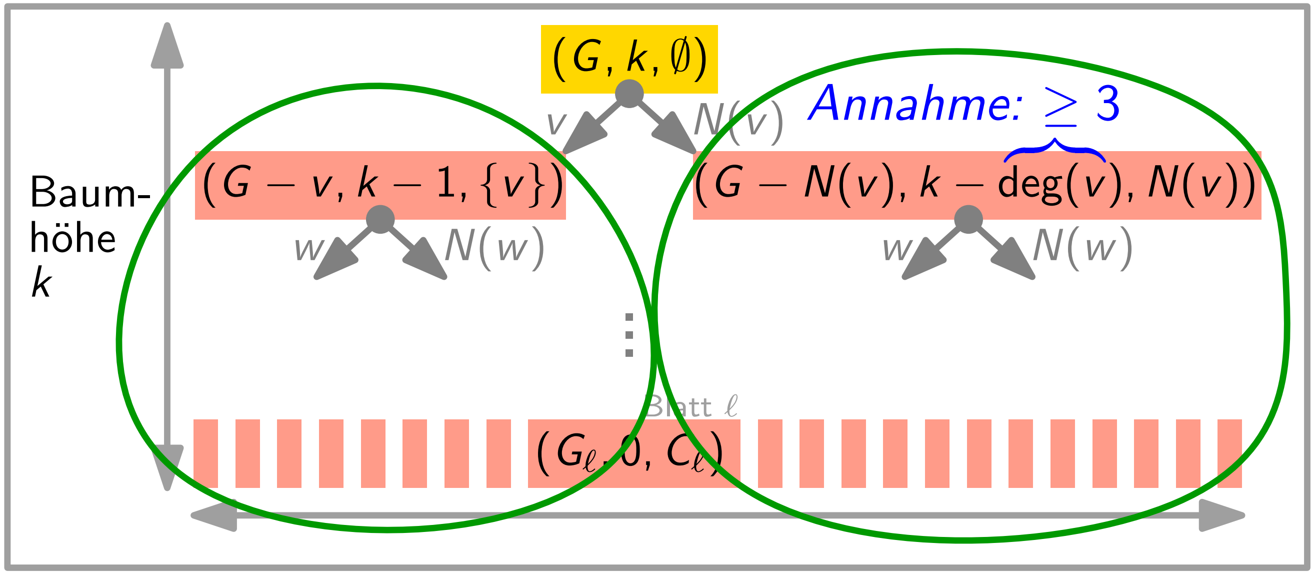
Der Grad-3-Algorithmus

Idee. Verbessere Abschätzung von $|N(v)|$.

Was wäre, wenn wir garantieren könnten, dass wir immer mit einem Knoten v verzweigen, dessen Grad mindestens 3 ist?

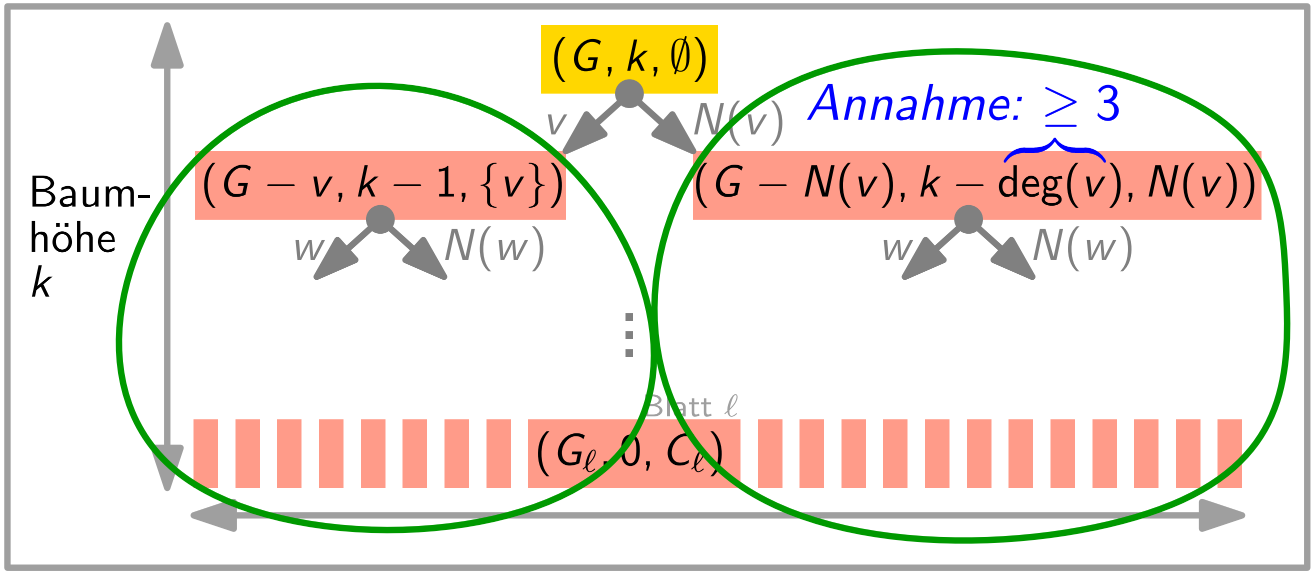
Der Grad-3-Algorithmus

Idee. Verbessere Abschätzung von $|N(v)|$.



Der Grad-3-Algorithmus

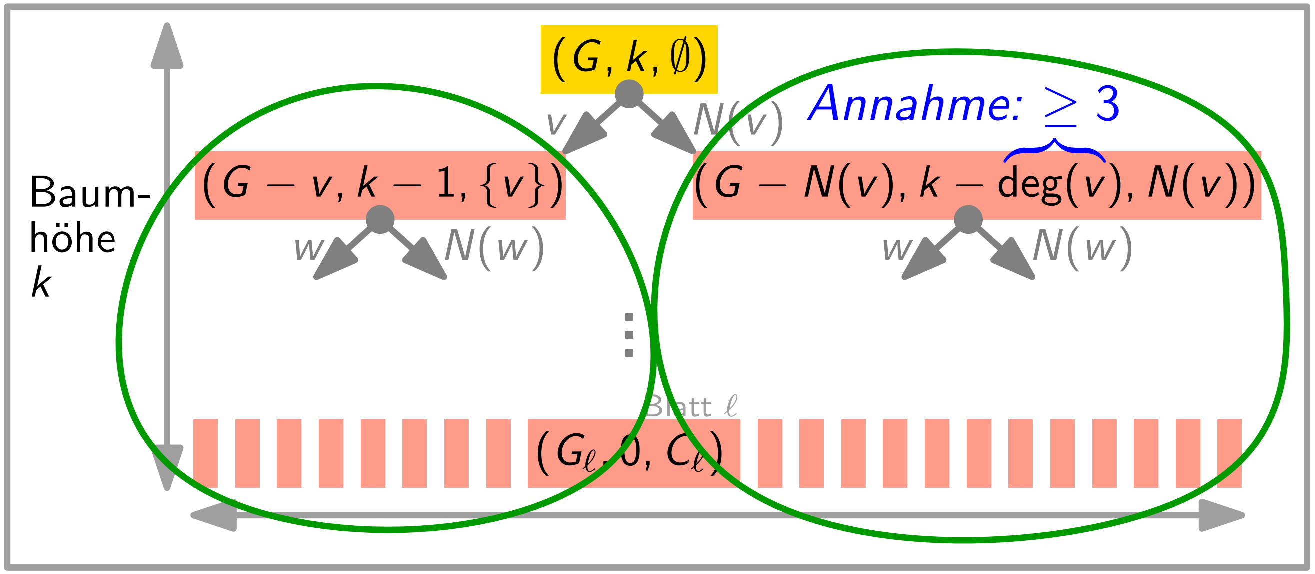
Idee. Verbessere Abschätzung von $|N(v)|$.



$\Rightarrow \; T(k) =$

Der Grad-3-Algorithmus

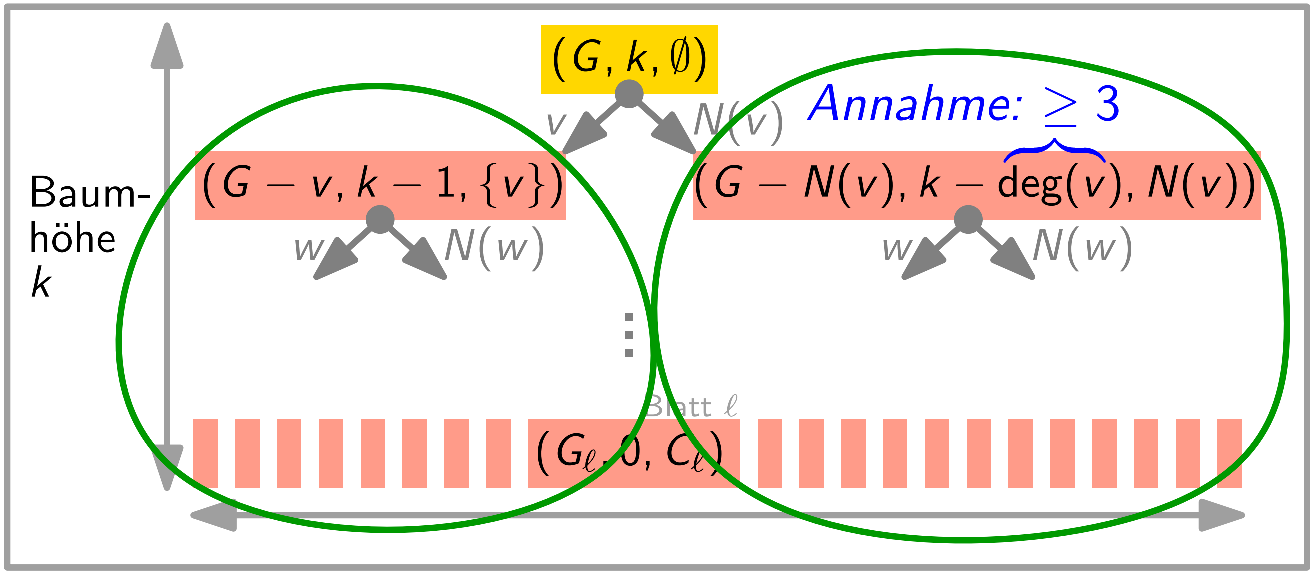
Idee. Verbessere Abschätzung von $|N(v)|$.



$$\Rightarrow \; T(k) = T(k-3) + T(k-1) + 1, \quad T(\leq 3) = const.$$

Der Grad-3-Algorithmus

Idee. Verbessere Abschätzung von $|N(v)|$.

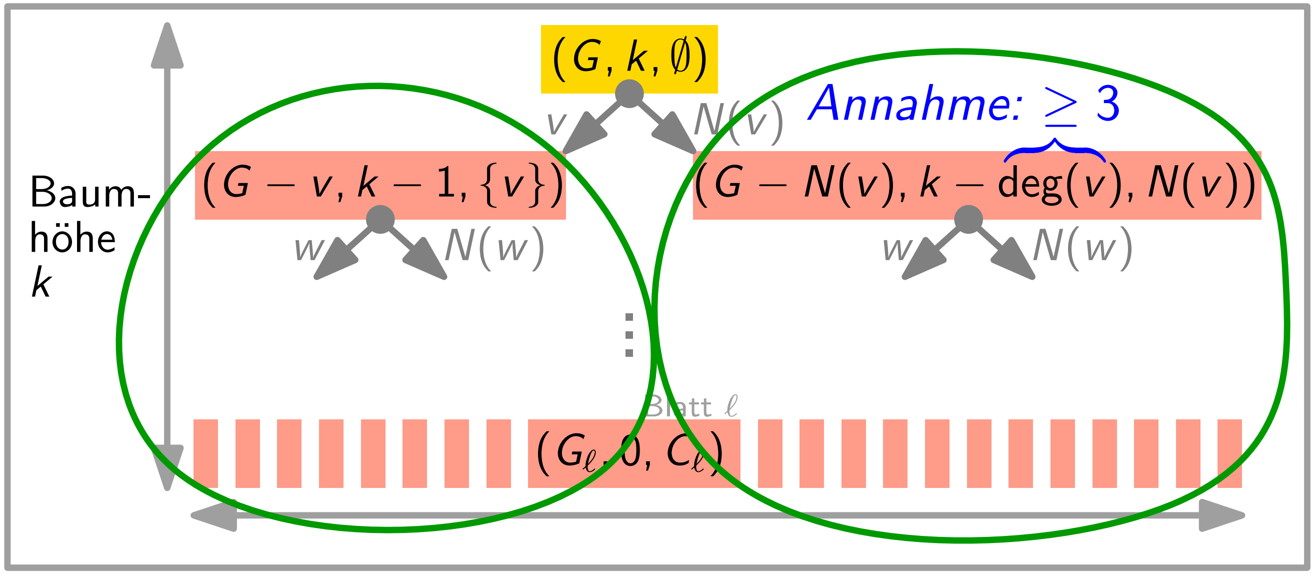


$$\Rightarrow \quad T(k) = T(k - 3) + T(k - 1) + 1, \quad T(\leq 3) = const.$$

Verzweigungsvektor $(3, 1)$

Der Grad-3-Algorithmus

Idee. Verbessere Abschätzung von $|N(v)|$.



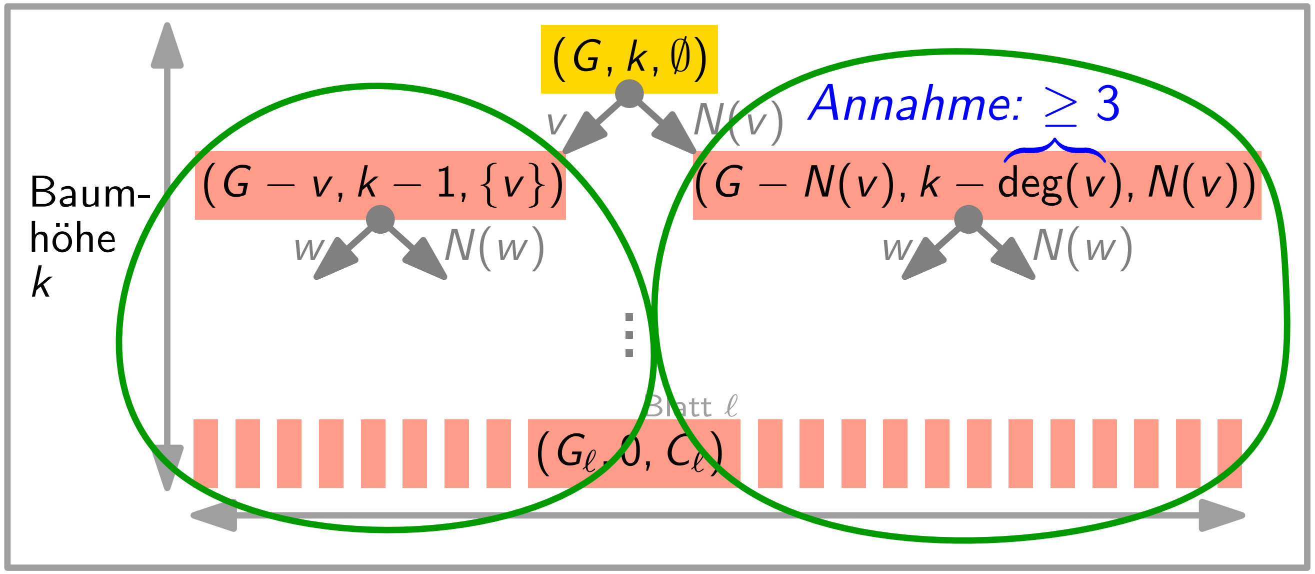
$\Rightarrow \; T(k) = T(k - 3) + T(k - 1) + 1, \quad T(\leq 3) = const.$

Verzweigungsvektor $(3, 1)$

Teste $T(k) = z^k - 1$

Der Grad-3-Algorithmus

Idee. Verbessere Abschätzung von $|N(v)|$.



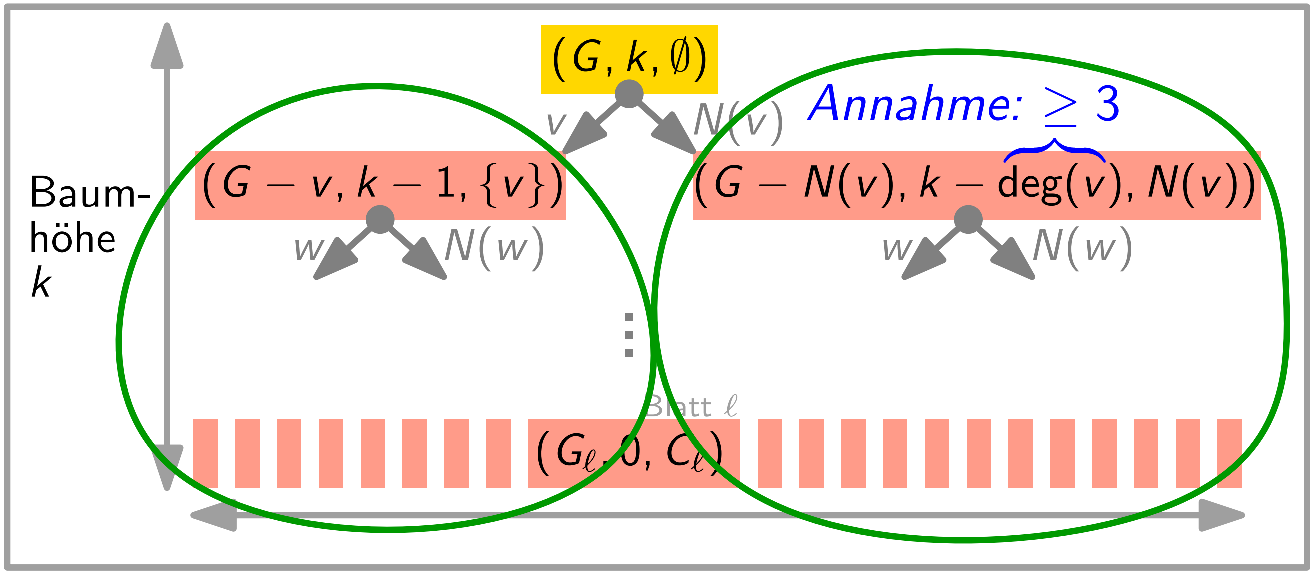
$\Rightarrow \; T(k) = T(k - \boxed{3}) + T(k - \boxed{1}) + 1, \;\; T(\leq 3) = const.$

Verzweigungsvektor $(3, 1)$

Teste $T(k) = z^k - 1 \quad \Rightarrow \quad z^k =$

Der Grad-3-Algorithmus

Idee. Verbessere Abschätzung von $|N(v)|$.



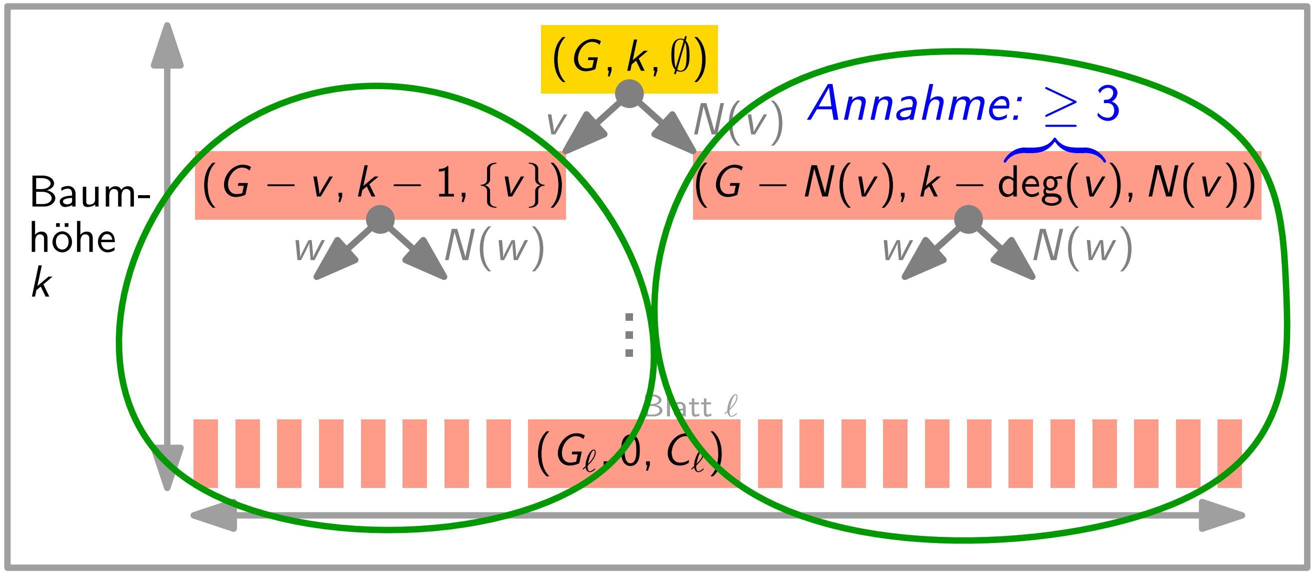
$$\Rightarrow \; T(k) = T(k - \boxed{3}) + T(k - \boxed{1}) + 1, \quad T(\leq 3) = const.$$

Verzweigungsvektor $(3, 1)$

Teste $T(k) = z^k - 1 \quad \Rightarrow \quad z^k = z^{k-3} + z^{k-1}$

Der Grad-3-Algorithmus

Idee. Verbessere Abschätzung von $|N(v)|$.



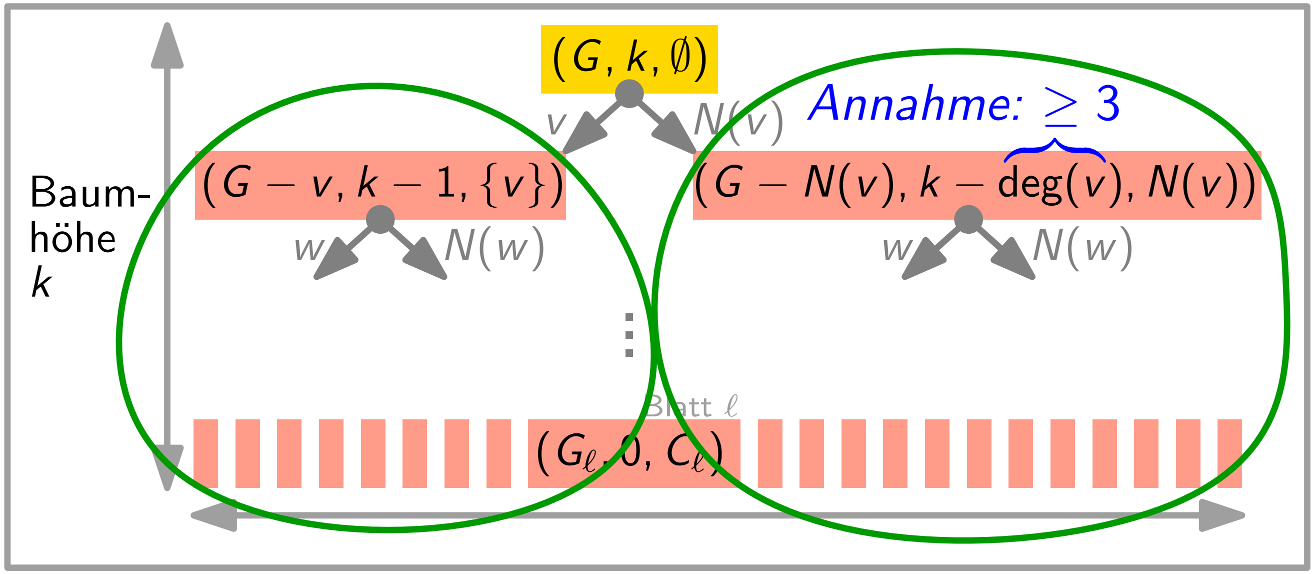
$\Rightarrow \quad T(k) = T(k - \boxed{3}) + T(k - \boxed{1}) + 1, \quad T(\leq 3) = const.$

Verzweigungsvektor $(3, 1)$

Teste $T(k) = z^k - 1 \quad \Rightarrow \quad z^k = z^{k-3} + z^{k-1}$

Der Grad-3-Algorithmus

Idee. Verbessere Abschätzung von $|N(v)|$.



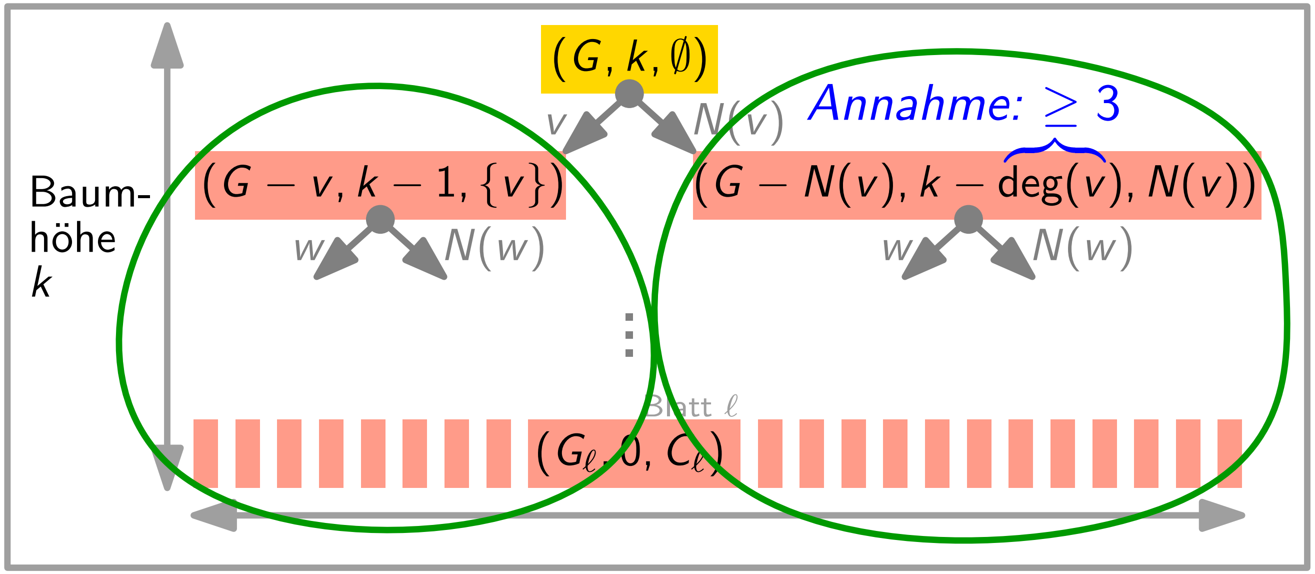
$\Rightarrow \quad T(k) = T(k - \boxed{3}) + T(k - \boxed{1}) + 1, \quad T(\leq 3) = const.$

Verzweigungsvektor $(3, 1)$

Teste $T(k) = z^k - 1 \quad \Rightarrow \quad z^k = z^{k-3} + z^{k-1}$ $\quad \cdot \frac{1}{z^{k-3}}$

Der Grad-3-Algorithmus

Idee. Verbessere Abschätzung von $|N(v)|$.



$\Rightarrow \quad T(k) = T(k - 3) + T(k - 1) + 1, \quad T(\leq 3) = const.$

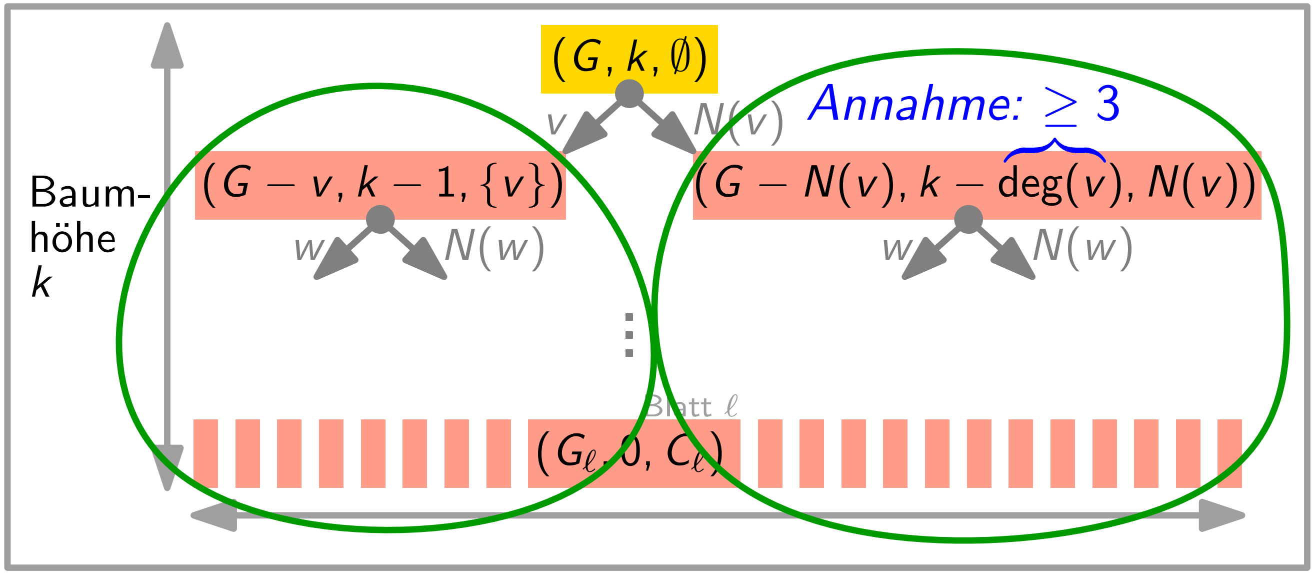
Verzweigungsvektor $(3, 1)$

Teste $T(k) = z^k - 1 \quad \Rightarrow \quad z^k = z^{k-3} + z^{k-1} \quad \cdot \frac{1}{z^{k-3}}$

$\Rightarrow$ Charakteristisches Polynom: $z^3 = 1 + z^2$

Der Grad-3-Algorithmus

Idee. Verbessere Abschätzung von $|N(v)|$.



$\Rightarrow$ $T(k) = T(k - 3) + T(k - 1) + 1, \quad T(\leq 3) = const.$

Verzweigungsvektor $(3, 1)$

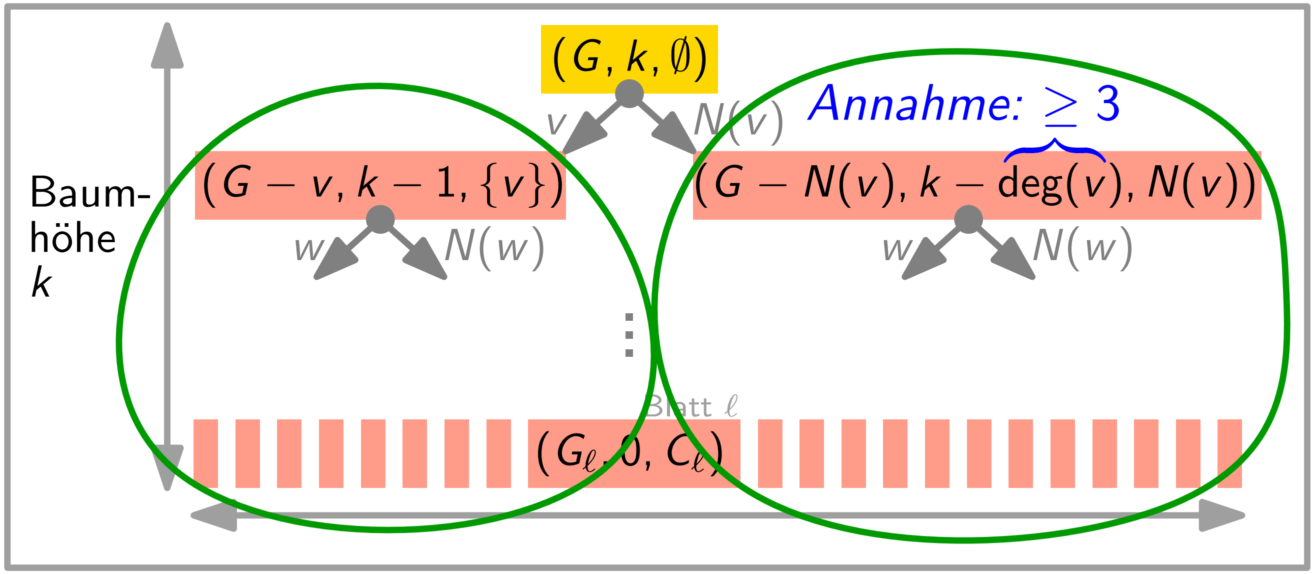
Teste $T(k) = z^k - 1 \quad \Rightarrow \quad z^k = z^{k-3} + z^{k-1}$ $\quad \cdot \frac{1}{z^{k-3}}$

$\Rightarrow$ Charakteristisches Polynom: $z^3 = 1 + z^2$

$\Rightarrow$ Größte positive Lösung:

Der Grad-3-Algorithmus

Idee. Verbessere Abschätzung von $|N(v)|$.



$\Rightarrow \; T(k) = T(k - 3) + T(k - 1) + 1, \quad T(\leq 3) = const.$

Verzweigungsvektor $(3, 1)$

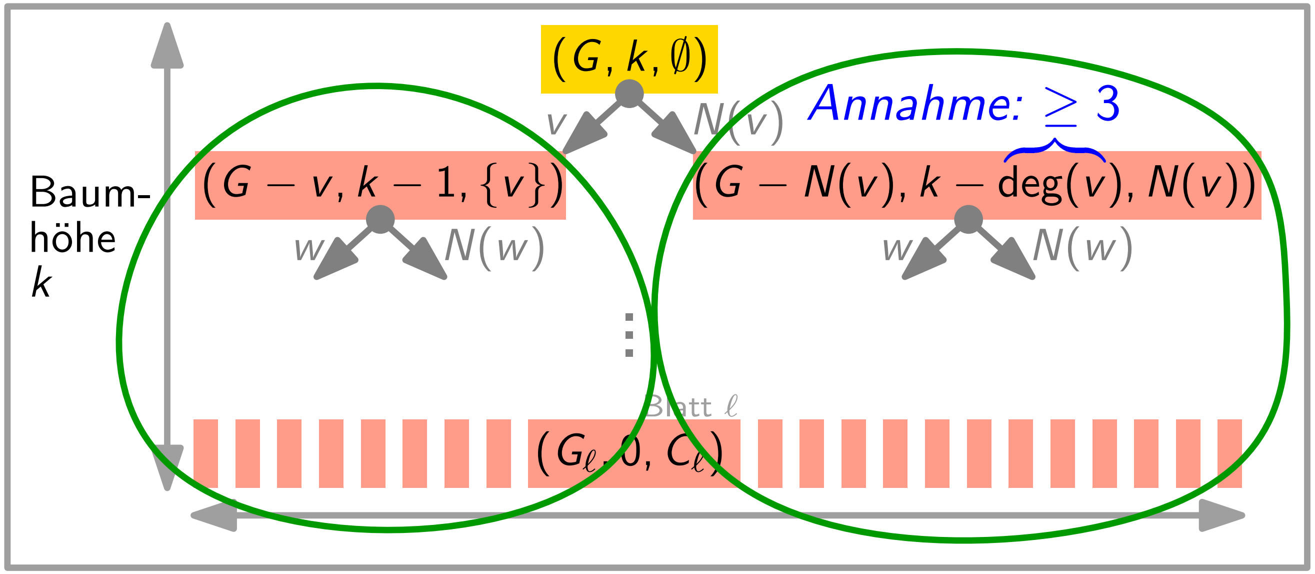
Teste $T(k) = z^k - 1 \quad \Rightarrow \quad z^k = z^{k-3} + z^{k-1}$ $\quad \cdot \frac{1}{z^{k-3}}$

$\Rightarrow$ Charakteristisches Polynom: $z^3 = 1 + z^2$

$\Rightarrow$ Größte positive Lösung: $z \approx 1{,}47$ (*Verzweigungszahl*)

Der Grad-3-Algorithmus

Idee. Verbessere Abschätzung von $|N(v)|$.



$\Rightarrow$ $T(k) = T(k - \boxed{3}) + T(k - \boxed{1}) + 1$, $T(\leq 3) = const.$

Verzweigungsvektor $(3, 1)$

Teste $T(k) = z^k - 1$ $\Rightarrow$ $z^k = z^{k-3} + z^{k-1}$ $\cdot \frac{1}{z^{k-3}}$

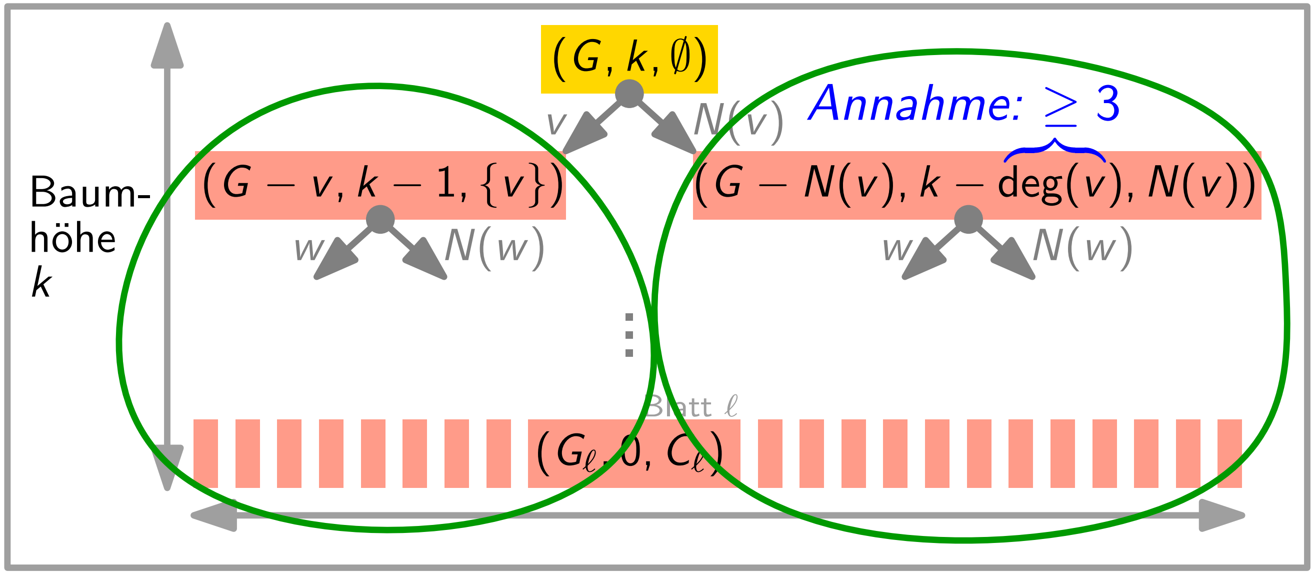
$\Rightarrow$ Charakteristisches Polynom: $z^3 = 1 + z^2$

$\Rightarrow$ Größte positive Lösung: $z \approx 1{,}47$ (*Verzweigungszahl*)

$\Rightarrow$ $T(k) \in O(1{,}47^k)$.

Der Grad-3-Algorithmus

Idee. Verbessere Abschätzung von $|N(v)|$.



$\Rightarrow\ T(k) = T(k - \boxed{3}) + T(k - \boxed{1}) + 1, \quad T(\leq 3) = const.$

Verzweigungsvektor $(3, 1)$

Teste $T(k) = z^k - 1 \quad \Rightarrow \quad z^k = z^{k-3} + z^{k-1}$ $\quad \cdot \frac{1}{z^{k-3}}$

$\Rightarrow$ Charakteristisches Polynom: $z^3 = 1 + z^2$

$\Rightarrow$ Größte positive Lösung: $z \approx 1{,}47$ (*Verzweigungszahl*)

$\Rightarrow\ T(k) \in O(1{,}47^k)$. Aber wie stellen wir $\deg(v) \geq 3$ sicher?

Kernbildung II

Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$

Kernbildung II

Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$
Regel 0: Eliminiere Knoten mit Grad 0

Kernbildung II

Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$

Regel 0: Eliminiere Knoten mit Grad 0

Verbesserte Kernbildung:

Regel 1:

Regel 2:

Kernbildung II

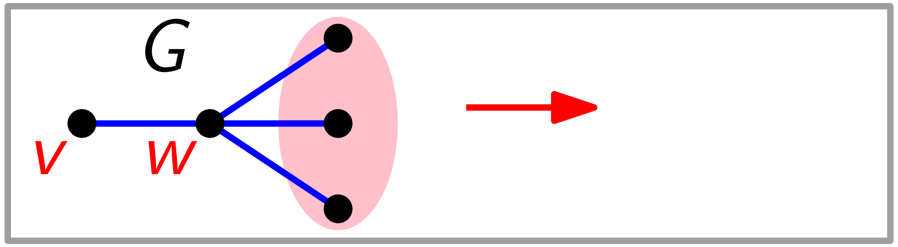
Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$

Regel 0: Eliminiere Knoten mit Grad 0

Verbesserte Kernbildung:

Regel 1: Eliminiere Knoten mit Grad 1



Regel 2:

Kernbildung II

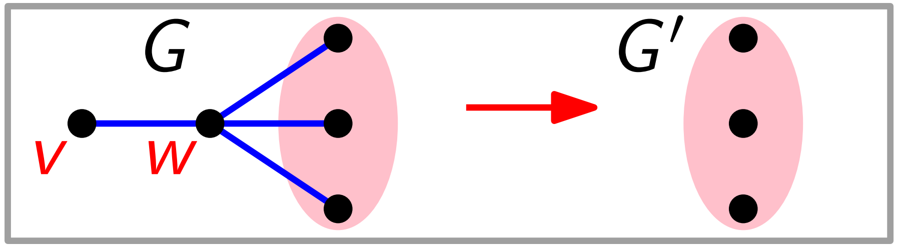
Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$

Regel 0: Eliminiere Knoten mit Grad 0

Verbesserte Kernbildung:

Regel 1: Eliminiere Knoten mit Grad 1



Regel 2:

Kernbildung II

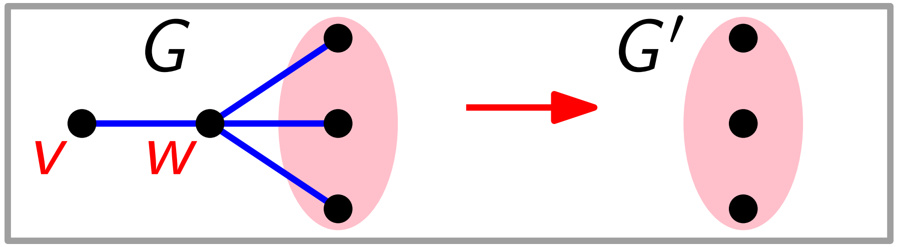
Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$

Regel 0: Eliminiere Knoten mit Grad 0

Verbesserte Kernbildung:

Regel 1: Eliminiere Knoten mit Grad 1



Setze $k' = k - 1$.

Berechne C' in G'.

Regel 2:

Kernbildung II

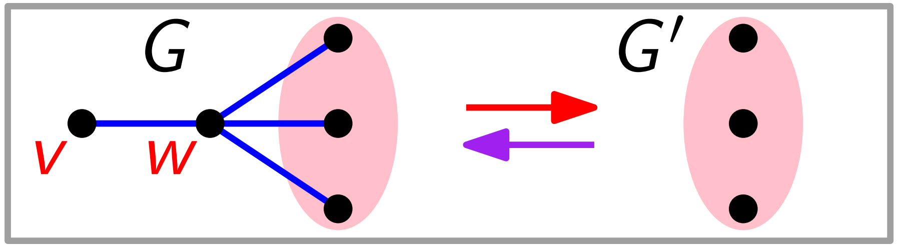
Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$

Regel 0: Eliminiere Knoten mit Grad 0

Verbesserte Kernbildung:

Regel 1: Eliminiere Knoten mit Grad 1



Setze $k' = k - 1$.
Berechne C' in G'.

Regel 2:

Kernbildung II

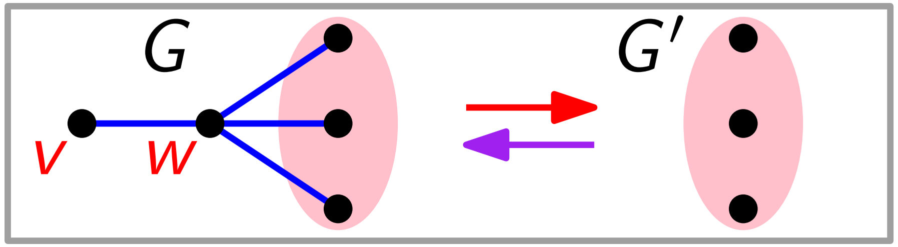
Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$

Regel 0: Eliminiere Knoten mit Grad 0

Verbesserte Kernbildung:

Regel 1: Eliminiere Knoten mit Grad 1



Setze $k' = k - 1$.
Berechne C' in G'.

$C = C' \cup \{w\}$

Regel 2:

Kernbildung II

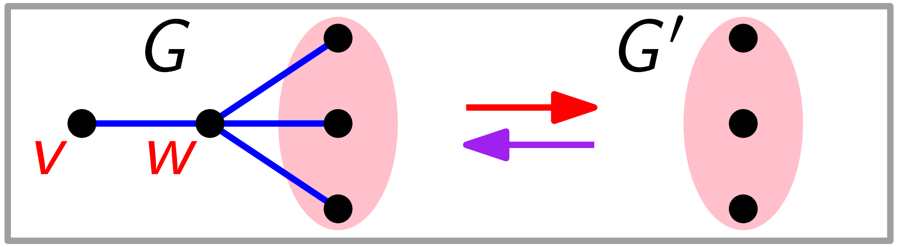
Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$

Regel 0: Eliminiere Knoten mit Grad 0

Verbesserte Kernbildung:

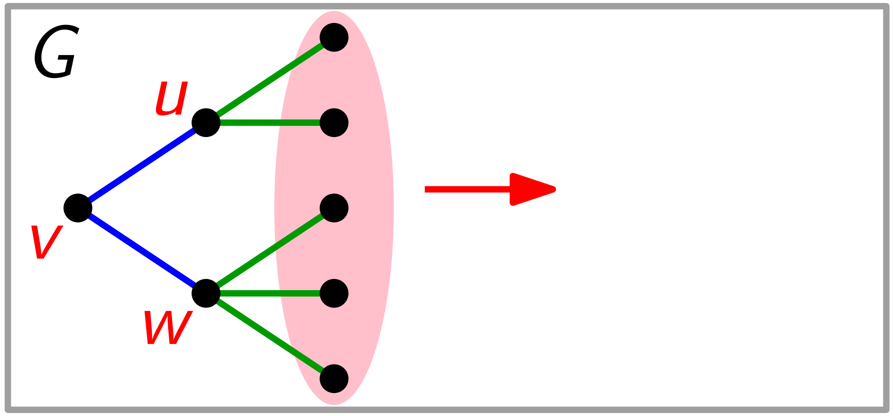
Regel 1: Eliminiere Knoten mit Grad 1



Setze $k' = k - 1$.
Berechne C' in G'.

$C = C' \cup \{w\}$

Regel 2: Eliminiere Knoten mit Grad 2



Kernbildung II

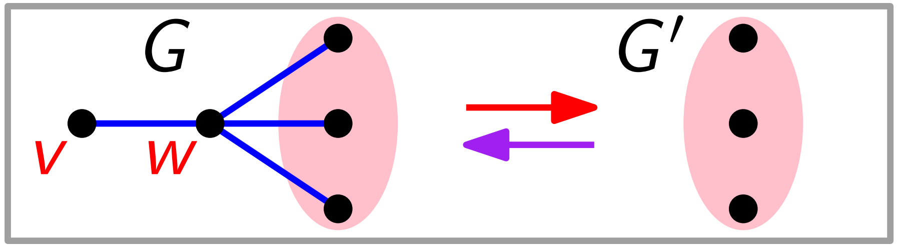
Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$

Regel 0: Eliminiere Knoten mit Grad 0

Verbesserte Kernbildung:

Regel 1: Eliminiere Knoten mit Grad 1

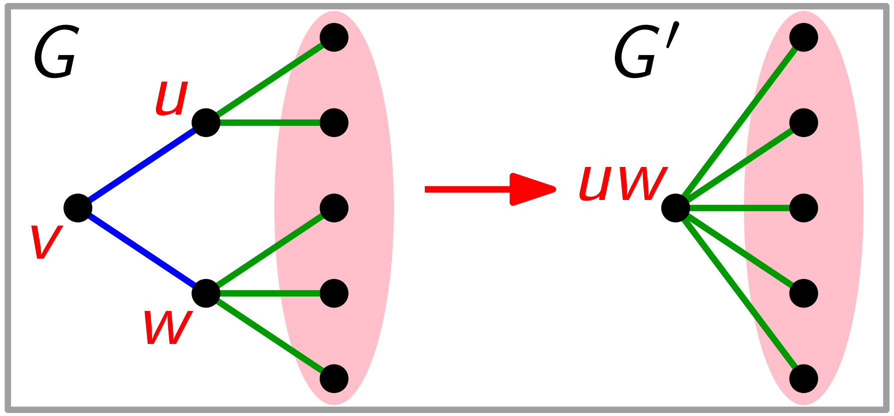


Setze $k' = k - 1$.

Berechne C' in G'.

$C = C' \cup \{w\}$

Regel 2: Eliminiere Knoten mit Grad 2



Kernbildung II

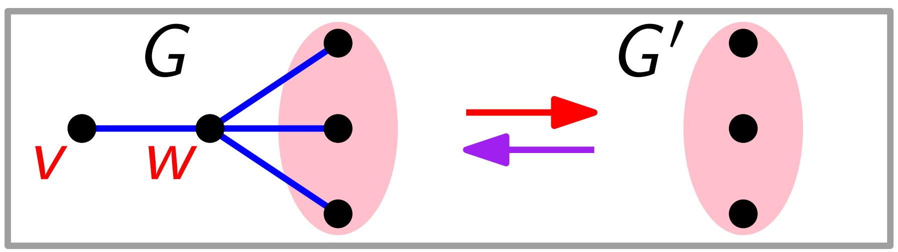
Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$

Regel 0: Eliminiere Knoten mit Grad 0

Verbesserte Kernbildung:

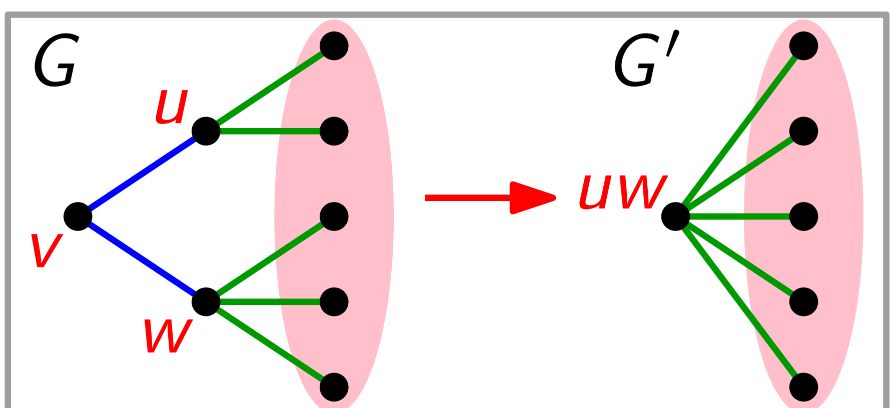
Regel 1: Eliminiere Knoten mit Grad 1



Setze $k' = k - 1$.
Berechne C' in G'.

$C = C' \cup \{w\}$

Regel 2: Eliminiere Knoten mit Grad 2



Setze $k' = k - 1$.

Kernbildung II

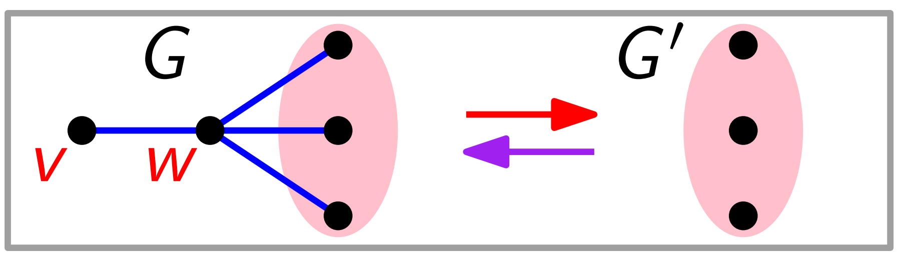
Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$

Regel 0: Eliminiere Knoten mit Grad 0

Verbesserte Kernbildung:

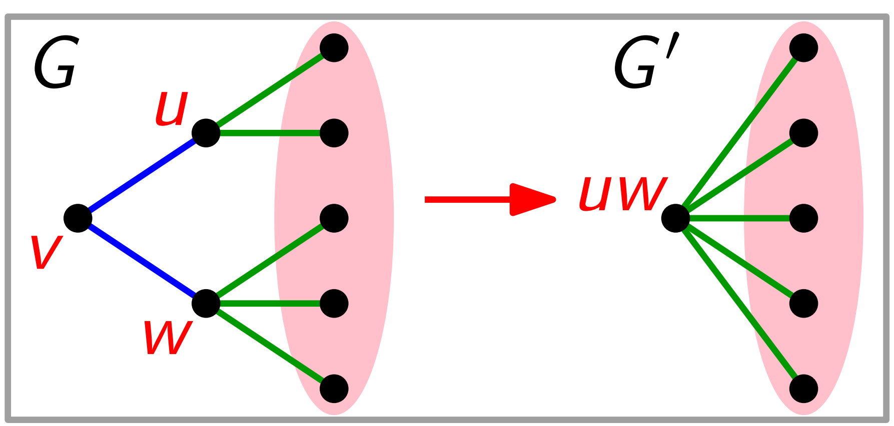
Regel 1: Eliminiere Knoten mit Grad 1



Setze $k' = k - 1$.
Berechne C' in G'.

$C = C' \cup \{w\}$

Regel 2: Eliminiere Knoten mit Grad 2



Setze $k' = k - 1$.
Berechne C' in G'.

Kernbildung II

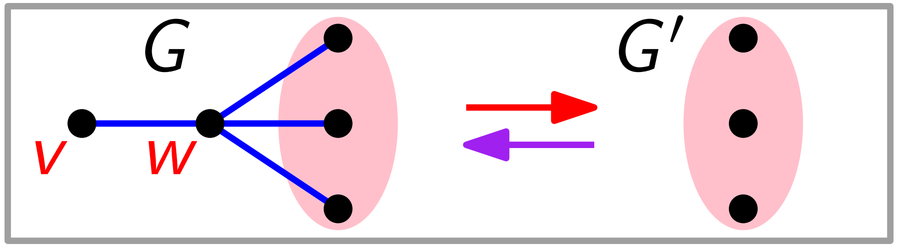
Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$

Regel 0: Eliminiere Knoten mit Grad 0

Verbesserte Kernbildung:

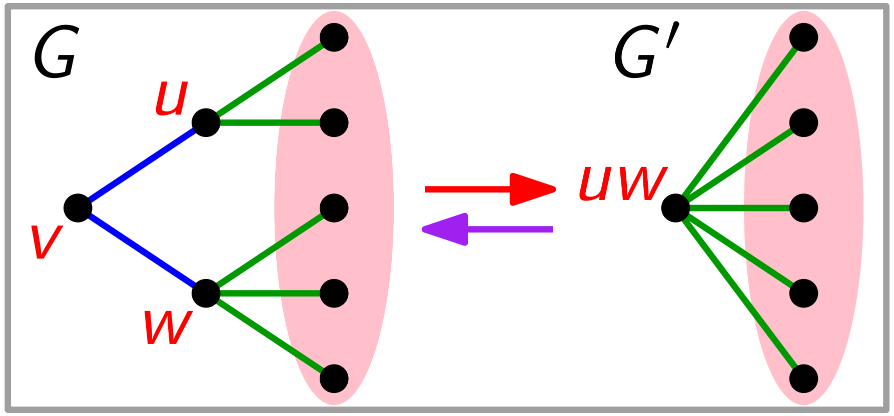
Regel 1: Eliminiere Knoten mit Grad 1



Setze $k' = k - 1$.
Berechne C' in G'.

$C = C' \cup \{w\}$

Regel 2: Eliminiere Knoten mit Grad 2



Setze $k' = k - 1$.
Berechne C' in G'.

Kernbildung II

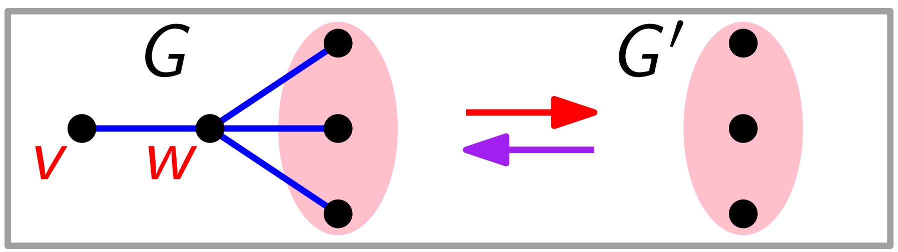
Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$

Regel 0: Eliminiere Knoten mit Grad 0

Verbesserte Kernbildung:

Regel 1: Eliminiere Knoten mit Grad 1

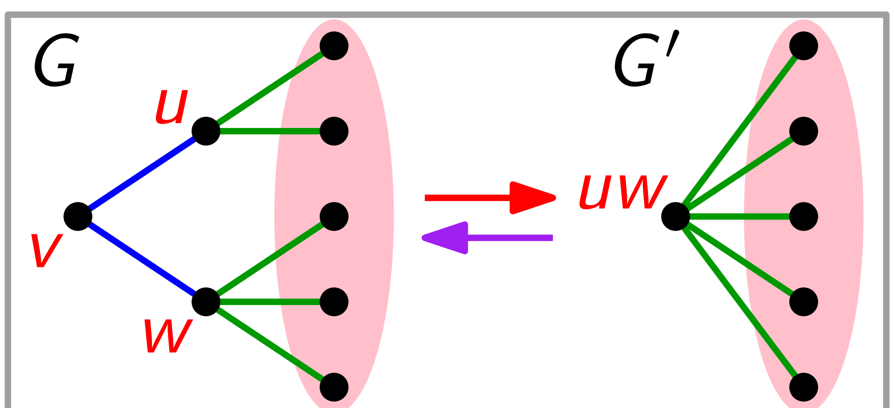


Setze $k' = k - 1$.

Berechne C' in G'.

$C = C' \cup \{w\}$

Regel 2: Eliminiere Knoten mit Grad 2



Setze $k' = k - 1$.

Berechne C' in G'.

Falls $uw \in C'$,
nimm u und w in C,
sonst v.

Kernbildung II

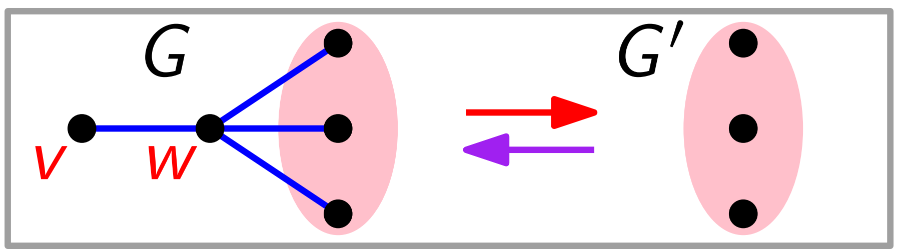
Bisherige Kernbildung:

Regel K: Eliminiere Knoten mit Grad $> k$

Regel 0: Eliminiere Knoten mit Grad 0

Verbesserte Kernbildung:

Regel 1: Eliminiere Knoten mit Grad 1

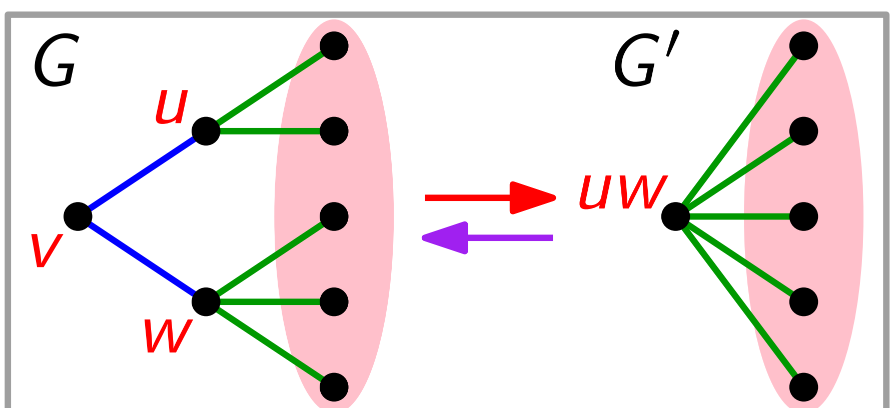


Setze $k' = k - 1$.

Berechne C' in G'.

$C = C' \cup \{w\}$

Regel 2: Eliminiere Knoten mit Grad 2



Setze $k' = k - 1$.

Berechne C' in G'.

Falls $uw \in C'$,
nimm u und w in C,
sonst v. $(\Rightarrow k = k'+1)$

Der Grad-3-Algorithmus

Idee: Wende die verbesserte Kernbildung *in jedem Knoten* des Suchbaums *erschöpfend* an!

Der Grad-3-Algorithmus

Idee: Wende die verbesserte Kernbildung *in jedem Knoten* des Suchbaums *erschöpfend* an!

$\Rightarrow$ **Laufzeit**:

Der Grad-3-Algorithmus

Idee: Wende die verbesserte Kernbildung *in jedem Knoten* des Suchbaums *erschöpfend* an!

$\Rightarrow$ **Laufzeit**: $O(\underbrace{\square}_{\text{Vorverarbeitung}} + \underbrace{\square}_{\text{Kernbildung in jedem Knoten des Suchbaums}} \cdot 1{,}47^k)$

Der Grad-3-Algorithmus

Idee: Wende die verbesserte Kernbildung *in jedem Knoten* des Suchbaums *erschöpfend* an!

$\Rightarrow$ **Laufzeit**: $O(\boxed{nk} + \boxed{} \cdot 1{,}47^k)$

Vorverarbeitung

Kernbildung in jedem Knoten des Suchbaums

Der Grad-3-Algorithmus

Idee: Wende die verbesserte Kernbildung *in jedem Knoten* des Suchbaums *erschöpfend* an!

$\Rightarrow$ **Laufzeit**: $O(\underbrace{nk}_{\text{Vorverarbeitung}} + \underbrace{k^2}_{\text{Kernbildung in jedem Knoten des Suchbaums}} \cdot 1{,}47^k)$

Der Grad-3-Algorithmus

Idee: Wende die verbesserte Kernbildung *in jedem Knoten* des Suchbaums *erschöpfend* an!

$\Rightarrow$ **Laufzeit**: $O(\underbrace{nk}_{\text{Vorverarbeitung}} + \underbrace{k^2}_{\text{Kernbildung in jedem Knoten des Suchbaums}} \cdot 1{,}47^k) \quad \subseteq \quad O^\star(1{,}47^k)$

Der Grad-3-Algorithmus

Idee: Wende die verbesserte Kernbildung *in jedem Knoten* des Suchbaums *erschöpfend* an!

$\Rightarrow$ **Laufzeit**: $O(\underbrace{nk}_{\text{Vorverarbeitung}} + \underbrace{k^2}_{\text{Kernbildung in jedem Knoten des Suchbaums}} \cdot 1{,}47^k) \subseteq O^\star(1{,}47^k)$

Der Grad-3-Algorithmus ist also ein Fest-Parameter-Algorithmus.

Fazit

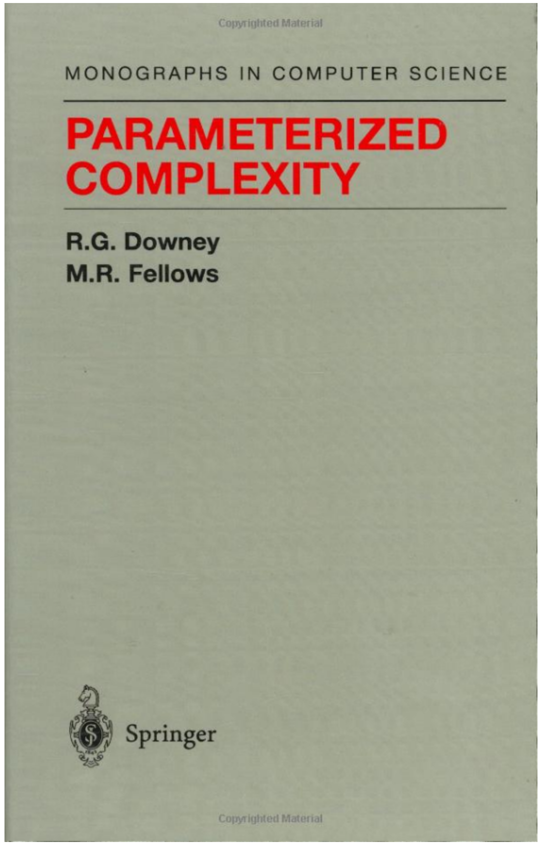
- k-VC kann in $O(nk + 1{,}47^k k^2)$ Zeit gelöst werden.
  Der momentan beste FPT-Algo läuft in $O^\star(1{,}27^k)$ Zeit.
  [Chen, Kanj, Jia, 2001]

- Parametrisierte Komplexität $=$
  neuer Werkzeugkasten für schwere Probleme:
  Kernbildung, Suchbäume, dynamische Programmierung, ...

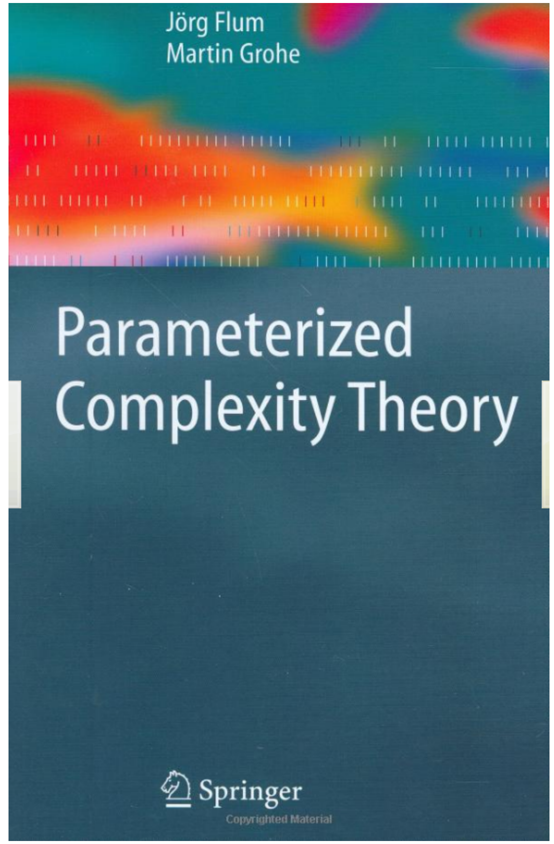
- Es ist immer sinnvoll, beschränkte Parameter zu identifizieren – FPT nutzt sie!

- Hoffnung:
  „natürliches" Problem $P \in \mathcal{FPT} \;\Rightarrow\; f(k)$ erträglich.

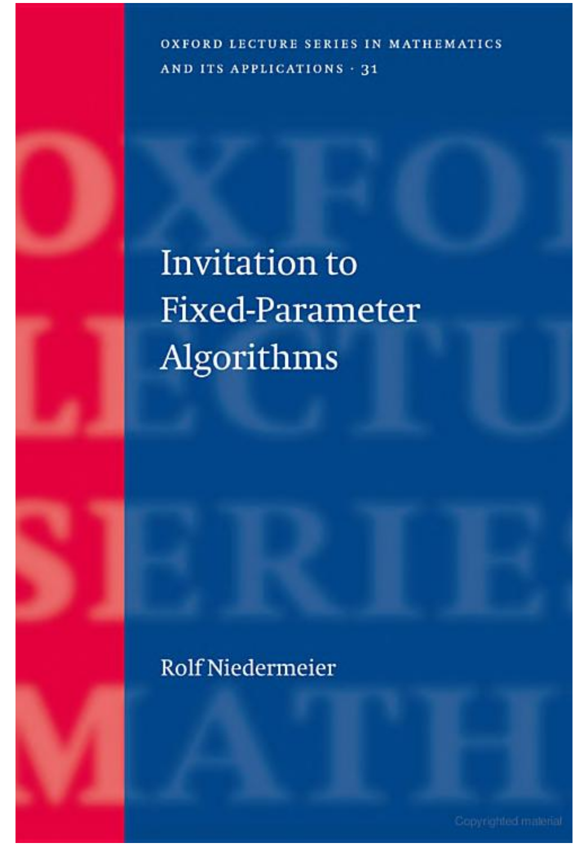
Bücher zum Thema



1999

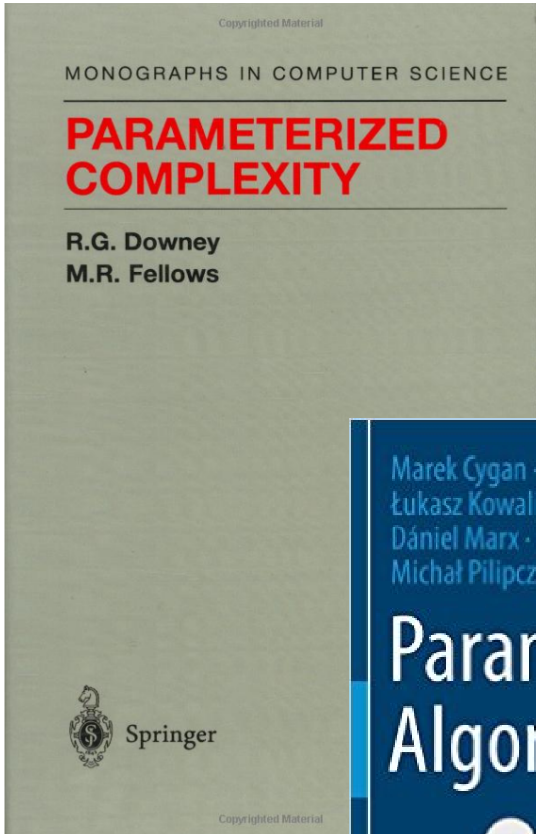


2006



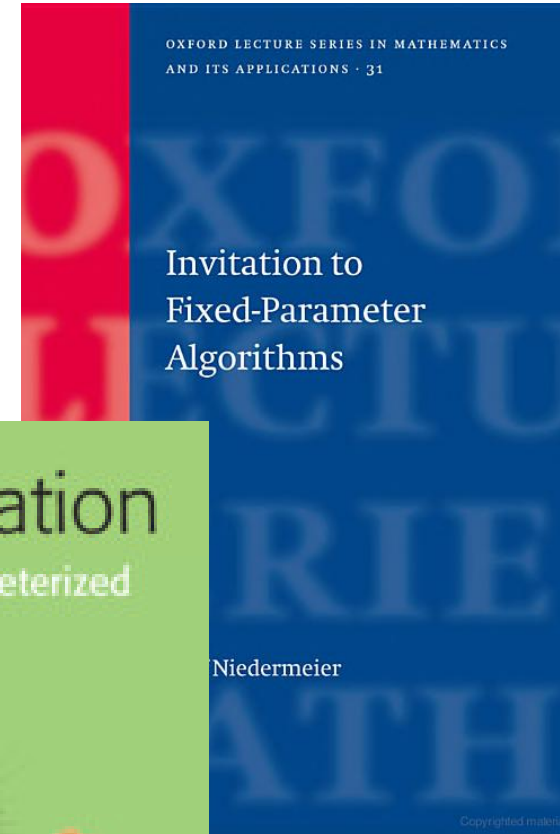
2006

Bücher zum Thema

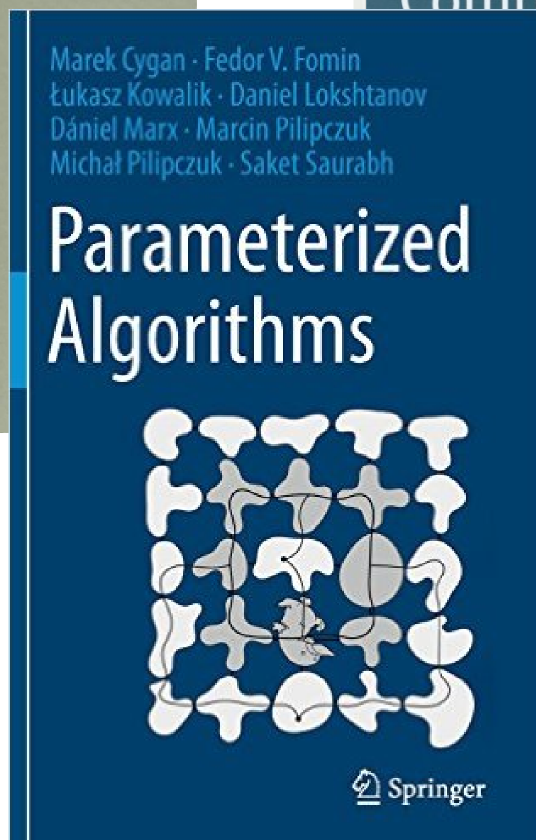


1999

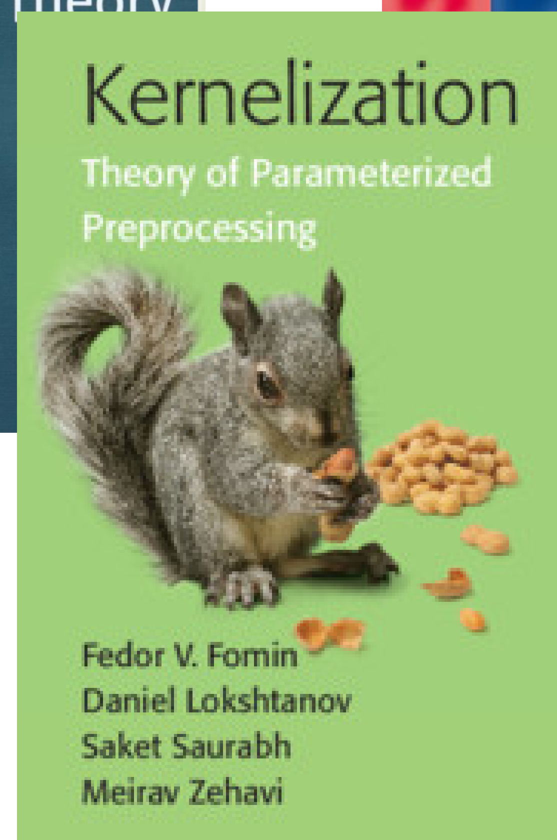
2006



2006



2015



2019