Julius-Maximilians-
UNIVERSITÄT WÜRZBURG

Lehrstuhl für
INFORMATIK I
Algorithmen & Komplexität

Institut für Informatik

Exact Algorithms

Sommer Term 2020

Lecture 9.1 Fixed Parameter Tractability

Based on: [Parameterized Algorithms: §1.1, 2.2.1, 3.1]

(slides by J. Spoerhase, Th. van Dijk, S. Chaplick, and A. Wolff)

Alexander Wolff

Lehrstuhl für Informatik I

Approaches to NP-Hard Problems

Approaches to NP-Hard Problems

- Exponential Algorithms , e.g. Backtracking

Approaches to NP-Hard Problems

- Exponential Algorithms , e.g. Backtracking
- Approximation Algorithms:
  Trade solution quality for runtime

Approaches to NP-Hard Problems

- **Exponential Algorithms** , e.g. Backtracking
- **Approximation Algorithms**:
  Trade solution quality for runtime
- **Heuristics** : empirically good, e.g., via benchmark instances

Approaches to NP-Hard Problems

- Exponential Algorithms , e.g. Backtracking
- Approximation Algorithms:
  Trade solution quality for runtime
- Heuristics : empirically good, e.g., via benchmark instances
- Randomization: search for a needle in a haystack

Approaches to NP-Hard Problems

- Exponential Algorithms , e.g. Backtracking
- Approximation Algorithms:
  Trade solution quality for runtime
- Heuristics : empirically good, e.g., via benchmark instances
- Randomization: search for a needle in a haystack
- Parameterized Algorithms

An Example: Vertex Cover

Def. (Recall)

Let $G = (V, E)$ be an undirectred graph.
$C \subseteq V$ is a *vertex cover* of G, when, for every edge $uv \in E$, either $u \in C$ or $v \in C$.

An Example: Vertex Cover

Def. (Recall)

Let $G = (V, E)$ be an undirectred graph.
$C \subseteq V$ is a *vertex cover* of G, when, for every edge $uv \in E$, either $u \in C$ or $v \in C$.

Prob. *Minimum Vertex Cover*

given: Graph G

Find: a minimum size vertex cover in G

An Example: Vertex Cover

Def. (Recall)

Let $G = (V, E)$ be an undirectred graph.
$C \subseteq V$ is a *vertex cover* of G, when, for every edge $uv \in E$, either $u \in C$ or $v \in C$.

Prob. *Minimum Vertex Cover* – optimization problem

given: Graph G

Find: a minimum size vertex cover in G

An Example: Vertex Cover

Def. (Recall)

Let $G = (V, E)$ be an undirectred graph.
$C \subseteq V$ is a *vertex cover* of G, when, for every edge $uv \in E$, either $u \in C$ or $v \in C$.

Prob. *Minimum Vertex Cover* – optimization problem

given: Graph G

Find: a minimum size vertex cover in G

Prob. – decision problem

An Example: Vertex Cover

Def. (Recall)

Let $G = (V, E)$ be an undirectred graph.
$C \subseteq V$ is a *vertex cover* of G, when, for every edge $uv \in E$, either $u \in C$ or $v \in C$.

Prob. *Minimum Vertex Cover* – optimization problem

given: Graph G

Find: a minimum size vertex cover in G

Prob. *k-Vertex Cover (k-VC)* – decision problem

Given: graph G, number k

Find: size $\leq k$ vertex cover in G when possible (otherwise return "NO")

History

- one of the first problems shown to be NP-hard

History

- one of the first problems shown to be NP-hard
  (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p \ldots$)

History

- one of the first problems shown to be NP-hard
  (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p$ ...) [Karp, 1972]

History

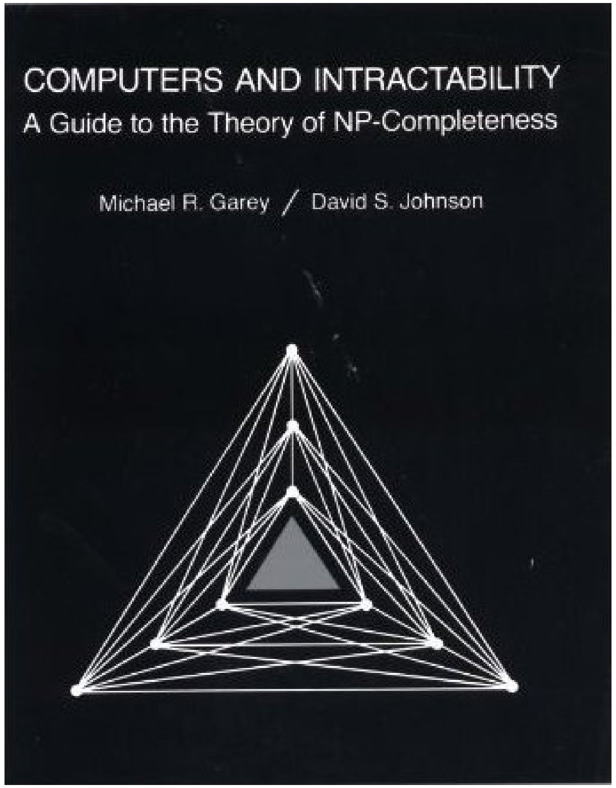
- one of the first problems shown to be NP-hard
  (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p \ldots$) [Karp, 1972]

- one of the six original NP-complete problems in the classic book [Garey & Johnson, 1979]

History

- one of the first problems shown to be NP-hard
  (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p$ . . .) [Karp, 1972]

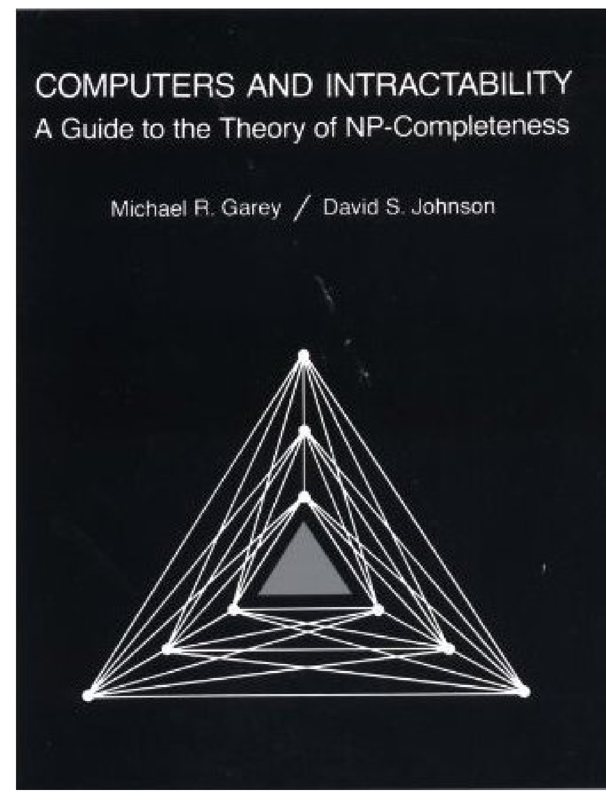
- one of the six original NP-complete problems in the classic book [Garey & Johnson, 1979]



History

- one of the first problems shown to be NP-hard
  (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p \ldots$) [Karp, 1972]

- one of the six original NP-complete problems in the classic book [Garey & Johnson, 1979]

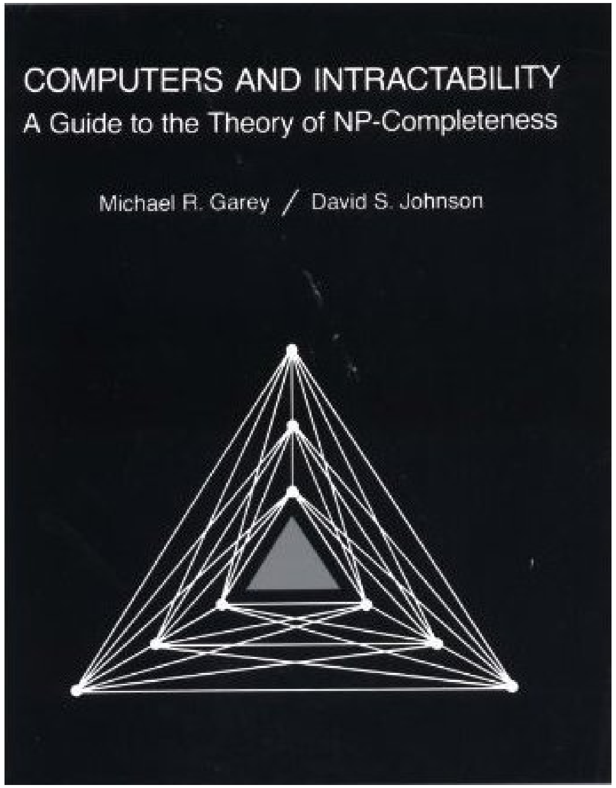


- approximation:

History

- one of the first problems shown to be NP-hard
  (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p$ ...) [Karp, 1972]

- one of the six original NP-complete problems in the classic book [Garey & Johnson, 1979]



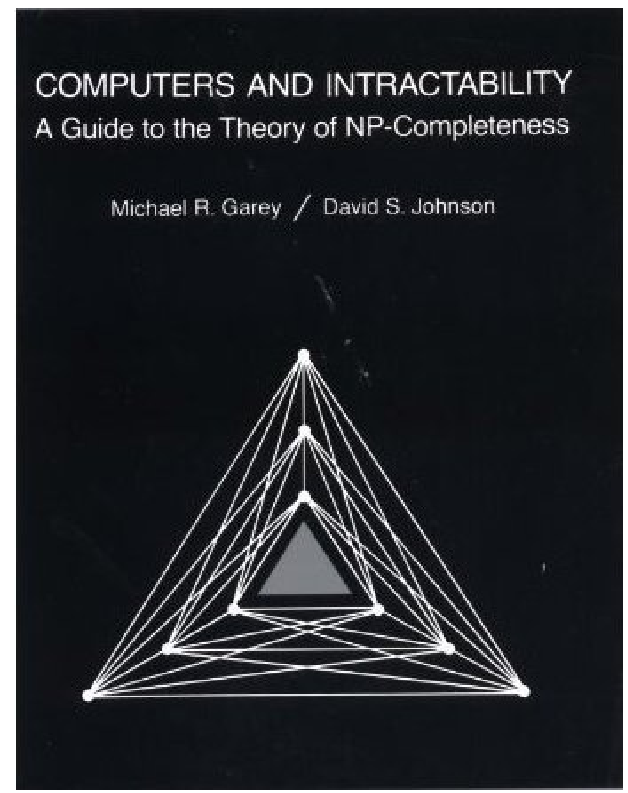
- approximation:

  maximal Matching provides a factor 2 approximation for VC.

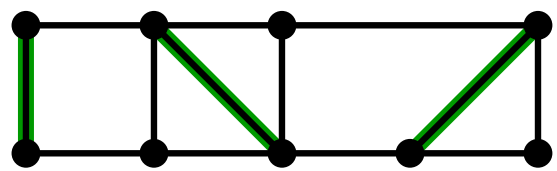
History

- one of the first problems shown to be NP-hard
  (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p$ . . .) [Karp, 1972]

- one of the six original NP-complete problems in the classic book [Garey & Johnson, 1979]



- approximation:

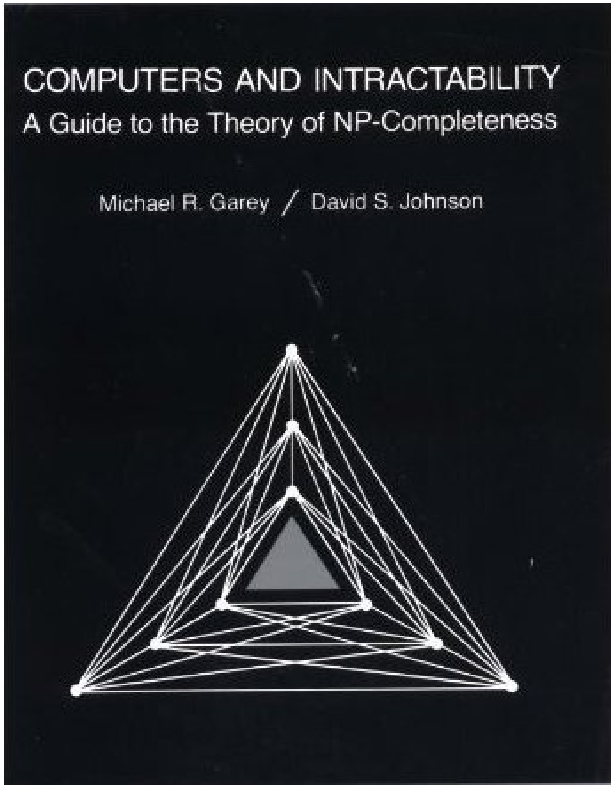


  maximal Matching provides a factor 2 approximation for VC.

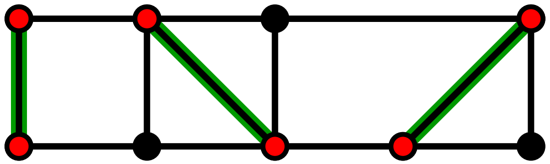
History

- one of the first problems shown to be NP-hard
  ($\text{SAT} \preceq_p \text{CLIQUE} \preceq_p \text{VC} \preceq_p \ldots$) [Karp, 1972]

- one of the six original NP-complete problems in the classic book [Garey & Johnson, 1979]



- approximation:

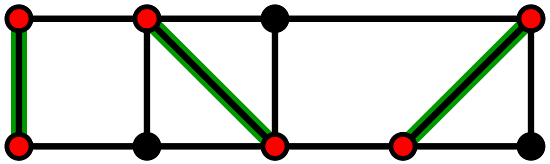


maximal Matching provides a factor 2 approximation for VC.

History

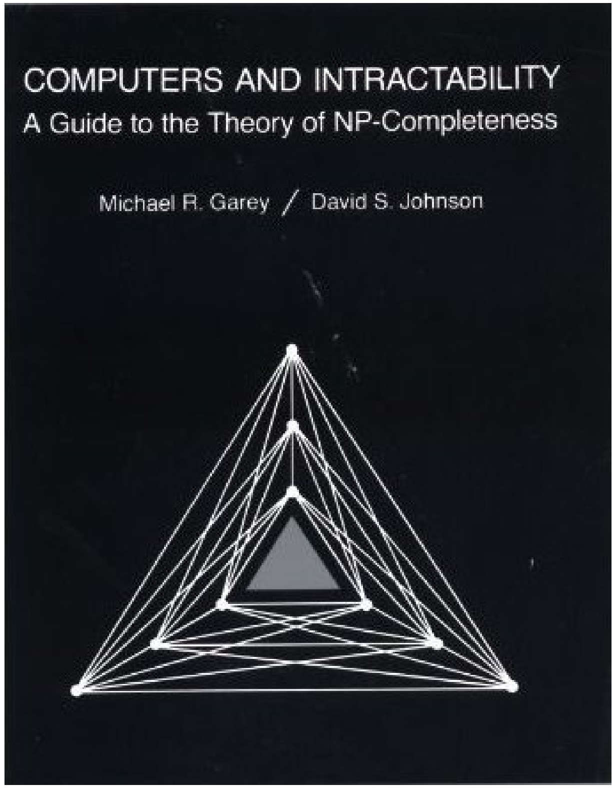
- one of the first problems shown to be NP-hard
  (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p$ ...) [Karp, 1972]

- one of the six original NP-complete problems in the classic book [Garey & Johnson, 1979]



- approximation:

  maximal Matching provides a factor 2 approximation for VC.

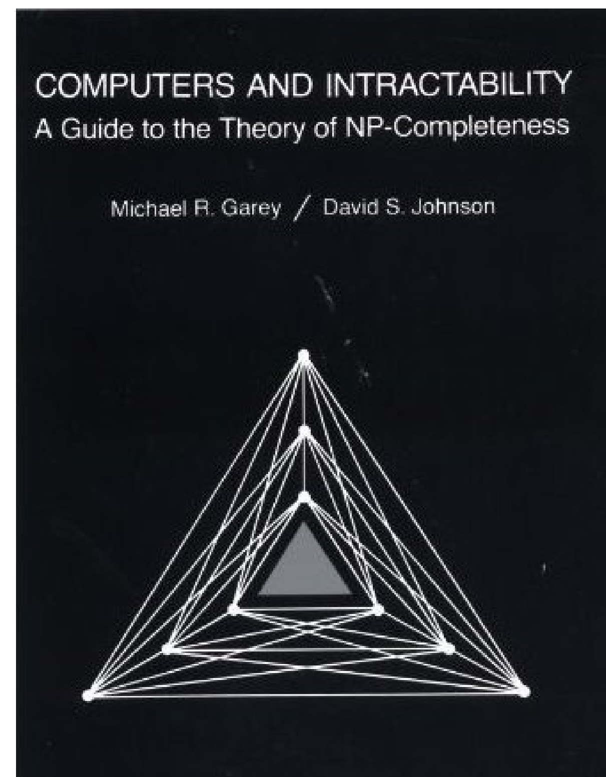


- no arbitrarily good approx. is possible:

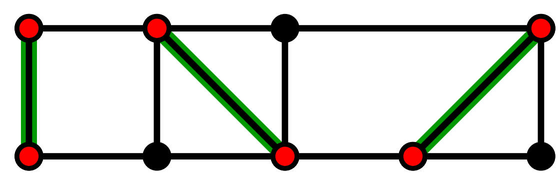
History

- one of the first problems shown to be NP-hard
  $(\text{SAT} \preceq_p \text{CLIQUE} \preceq_p \text{VC} \preceq_p \ldots)$ [Karp, 1972]

- one of the six original NP-complete problems in the classic book [Garey & Johnson, 1979]



- approximation:



  maximal Matching provides a factor 2 approximation for VC.

- no arbitrarily good approx. is possible:

  There is no factor-1.3606 approx. for vertex cover

  [Dinur & Safra, 2004]

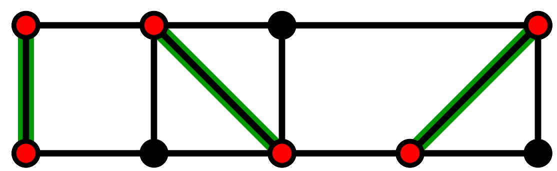
History

- one of the first problems shown to be NP-hard

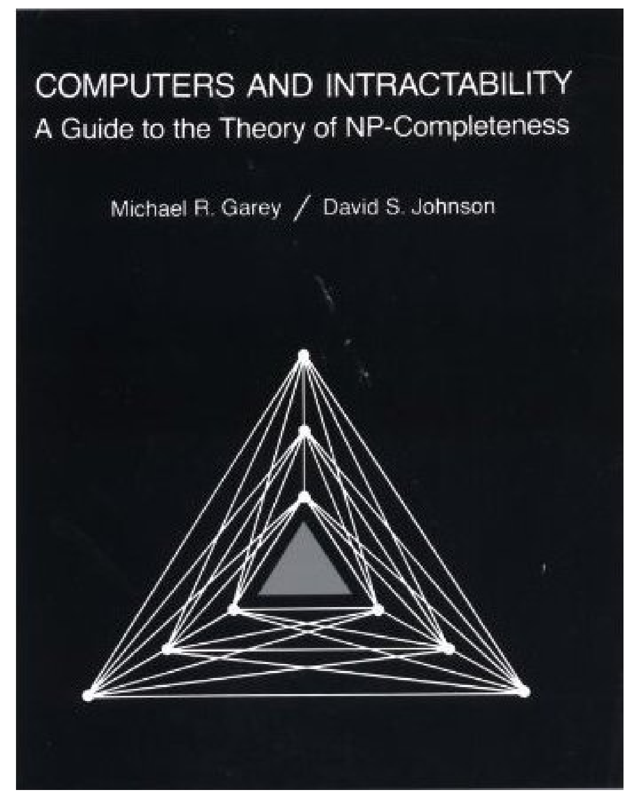
  (SAT $\preceq_p$ CLIQUE $\preceq_p$ VC $\preceq_p$ ...) [Karp, 1972]

- one of the six original NP-complete problems in the classic book [Garey & Johnson, 1979]

- approximation:



  maximal Matching provides a factor 2 approximation for VC.



- no arbitrarily good approx. is possible:

  There is no factor-1.3606 approx. for vertex cover , assuming $\mathcal{P} \neq \mathcal{NP}$.
  [Dinur & Safra, 2004]

An Exact Algorithm for k-VC

BruteForceVC(Graph G, Integer k)

An Exact Algorithm for k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V choose k) do
      // check if C is a VC
      vc = true

      if vc then
          return ("YES", C)
  return ("NO", ∅)
```

BruteForceVC(Graph G, Integer k)

foreach $C \in \binom{V}{k}$ **do**

// check if C is a VC

vc $= \mathit{true}$

if vc **then**

return (“YES”, C)

return (“NO”, $\emptyset$)

An Exact Algorithm for k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V choose k) do
      // check if C is a VC
      vc = true
      foreach uv ∈ E do
          if {u, v} ∩ C = ∅ then
              vc = false
      if vc then
          return ("YES", C)
  return ("NO", ∅)
```

An Exact Algorithm for k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V choose k) do
      // check if C is a VC
      vc = true
      foreach uv ∈ E do
          if {u, v} ∩ C = ∅ then
              vc = false
      if vc then
          return ("YES", C)
  return ("NO", ∅)
```

Runtime.

An Exact Algorithm for k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V choose k) do
      // check if C is a VC
      vc = true
      foreach uv ∈ E do
          if {u, v} ∩ C = ∅ then
              vc = false
      if vc then
          return ("YES", C)
  return ("NO", ∅)
```

Runtime.

An Exact Algorithm for k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V choose k) do
    // check if C is a VC
    vc = true
    foreach uv ∈ E do
      if {u, v} ∩ C = ∅ then
        vc = false
    if vc then
      return ("YES", C)
  return ("NO", ∅)
```

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

Runtime.

An Exact Algorithm for k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V choose k) do
    // check if C is a VC
    vc = true
    foreach uv ∈ E do
      if {u, v} ∩ C = ∅ then
        vc = false
    if vc then
      return ("YES", C)
  return ("NO", ∅)
```

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

Runtime.

An Exact Algorithm for k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V choose k) do
      // check if C is a VC
      vc = true
      foreach uv ∈ E do
          if {u, v} ∩ C = ∅ then
              vc = false
      if vc then
          return ("YES", C)
  return ("NO", ∅)
```

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

Runtime.

An Exact Algorithm for k-VC

BruteForceVC(Graph G, Integer k)
  foreach $C \in \binom{V}{k}$ **do**
    // check if C is a VC
    vc $= \mathit{true}$
    foreach $uv \in E$ **do**
      if $\{u, v\} \cap C = \emptyset$ **then**
        vc $= \mathit{false}$
    if vc **then**
      return ("YES", C)
  return ("NO", $\emptyset$)

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

$O(E) = O(m)$

Runtime.

An Exact Algorithm for k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V choose k) do
      // check if C is a VC
      vc = true
      foreach uv ∈ E do
          if {u, v} ∩ C = ∅ then
              vc = false
      if vc then
          return ("YES", C)
  return ("NO", ∅)
```

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

$O(E) = O(m)$

Runtime. $O(n^k m)$

An Exact Algorithm for k-VC

```
BruteForceVC(Graph G, Integer k)
  foreach C ∈ (V choose k) do
      // check if C is a VC
      vc = true
      foreach uv ∈ E do
          if {u, v} ∩ C = ∅ then
              vc = false
      if vc then
          return ("YES", C)
  return ("NO", ∅)
```

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

$O(E) = O(m)$

Runtime. $O(n^k m)$ – This is **not** polynomial in the input size ($|G| = n + m; k$) — k is not constant as it is part of the input.

A New Goal

Find an algorithm for k-VC with runtime:

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

A New Goal

Find an algorithm for k-VC with runtime:

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

where $f : \mathbb{N} \to \mathbb{N}$ is a computable function (indep. of I),
I is a given instance, and c is a constant (indep. of I)

A New Goal

Find an algorithm for k-VC with runtime:

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

where $f\colon \mathbb{N} \to \mathbb{N}$ is a computable function (indep. of I),
I is a given instance, and c is a constant (indep. of I)

i.e., the runtime depends
– “somehow” on the Parameter k,
– polynomially on the size $|I|$ of the input I.

A New Goal

Find an algorithm for k-VC with runtime:

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

where $f\colon \mathbb{N} \to \mathbb{N}$ is a computable function (indep. of I),
I is a given instance, and c is a constant (indep. of I)

i.e., the runtime depends
– "somehow" on the Parameter k, *difficulty of the instance*
– polynomially on the size $|I|$ of the input I.

A New Goal

Find an algorithm for k-VC with runtime:

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

where $f : \mathbb{N} \to \mathbb{N}$ is a computable function (indep. of I),
I is a given instance, and c is a constant (indep. of I)

i.e., the runtime depends
– "somehow" on the Parameter k, *difficulty of the instance*
– polynomially on the size $|I|$ of the input I.

A problem admitting algorithms with this type of runtime is called **fixed-parameter tractable** with respect to k.

A New Goal

Find an algorithm for k-VC with runtime:

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

where $f\colon \mathbb{N} \to \mathbb{N}$ is a computable function (indep. of I),
I is a given instance, and c is a constant (indep. of I)

i.e., the runtime depends
– "somehow" on the Parameter k, *difficulty of the instance*
– polynomially on the size $|I|$ of the input I.

A problem admitting algorithms with this type of runtime is called **fixed-parameter tractable** with respect to k.

$\mathcal{FPT}$ = class of fixed-parameter tractable problems.

Remark.
The class $\mathcal{FPT}$ implicitly allows us to replace $+$ by $\cdot$ here.

A New Goal

Find an algorithm for k-VC with runtime:

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

where $f\colon \mathbb{N} \to \mathbb{N}$ is a computable function (indep. of I),
I is a given instance, and c is a constant (indep. of I)

i.e., the runtime depends
– "somehow" on the Parameter k, *difficulty of the instance*
– polynomially on the size $|I|$ of the input I.

A problem admitting algorithms with this type of runtime is called **fixed-parameter tractable** with respect to k.

$\mathcal{FPT}$ = class of fixed-parameter tractable problems.

A New Goal

Remark.
The class $\mathcal{FPT}$ implicitly allows us to replace $+$ by $\cdot$ here.

Find an algorithm for k-VC with runtime:

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

where $f\colon \mathbb{N} \to \mathbb{N}$ is a computable function (indep. of I),
I is a given instance, and c is a constant (indep. of I)

i.e., the runtime depends
– "somehow" on the Parameter k, *difficulty of the instance*
– polynomially on the size $|I|$ of the input I.

A problem admitting algorithms with this type of runtime is called **fixed-parameter tractable** with respect to k.

$\mathcal{FPT}$ = class of fixed-parameter tractable problems.

A New Goal

Remark.
The class $\mathcal{FPT}$ implicitly allows us to replace $+$ by $\cdot$ here.

Find an algorithm for k-VC with runtime:

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

ignore polynomial factors!

where $f\colon \mathbb{N} \to \mathbb{N}$ is a computable function (indep. of I),
I is a given instance, and c is a constant (indep. of I)

i.e., the runtime depends
– "somehow" on the Parameter k, *difficulty of the instance*
– polynomially on the size $|I|$ of the input I.

A problem admitting algorithms with this type of runtime is called **fixed-parameter tractable** with respect to k.

$\mathcal{FPT}$ = class of fixed-parameter tractable problems.

A New Goal

Remark.
The class $\mathcal{FPT}$ implicitly allows us to replace $+$ by $\cdot$ here.

Find an algorithm for k-VC with runtime:

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

ignore polynomial factors!

where $f\colon \mathbb{N} \to \mathbb{N}$ is a computable function (indep. of I),
I is a given instance, and c is a constant (indep. of I)

i.e., the runtime depends
– "somehow" on the Parameter k, *difficulty of the instance*
– polynomially on the size $|I|$ of the input I.

A problem admitting algorithms with this type of runtime is called **fixed-parameter tractable** with respect to k.

$\mathcal{FPT}$ = class of fixed-parameter tractable problems.

Remark. BruteForceVC does not have this runtime.

Some Observations

For a graph G, VC C of G, and a vertex v outside of C, which which vertices must be in C?

Some Observations

For a graph G, VC C of G, and a vertex v outside of C, which which vertices must be in C?

Obs. 1. For a graph G, VC C of G and vertex v, Either $v \in C$ or $N(v) \subseteq C$.

Some Observations

For a graph G, VC C of G, and a vertex v outside of C, which which vertices must be in C?

Obs. 1. For a graph G, VC C of G and vertex v,
Either $v \in C$ or $N(v) \subseteq C$.

For the decision problem k-VC,
what happens with vertices of degree $> k$?

Some Observations

For a graph G, VC C of G, and a vertex v outside of C, which which vertices must be in C?

Obs. 1. For a graph G, VC C of G and vertex v, Either $v \in C$ or $N(v) \subseteq C$.

For the decision problem k-VC, what happens with vertices of degree $> k$?

Obs. 2. Each vertex of degree $> k$ is in every k-VC.

Some Observations

For a graph G, VC C of G, and a vertex v outside of C, which which vertices must be in C?

Obs. 1. For a graph G, VC C of G and vertex v,
Either $v \in C$ or $N(v) \subseteq C$.

For the decision problem k-VC,
what happens with vertices of degree $> k$?

Obs. 2. Each vertex of degree $> k$ is in every k-VC.

What if $|E| > k^2$ and every vertex has degree $\leq k$?

Some Observations

For a graph G, VC C of G, and a vertex v outside of C, which which vertices must be in C?

Obs. 1. For a graph G, VC C of G and vertex v,
Either $v \in C$ or $N(v) \subseteq C$.

For the decision problem k-VC,
what happens with vertices of degree $> k$?

Obs. 2. Each vertex of degree $> k$ is in every k-VC.

What if $|E| > k^2$ and every vertex has degree $\leq k$?

Obs. 3. If $|E| > k^2$ and $\Delta(G) := \max_{v \in V} \deg v \leq k$,
then G has no k-VC.

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

II) solve the reduced problem exactly

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

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

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** ("NO", $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** ("NO", $\emptyset$)

II) solve the reduced problem exactly

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

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

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** (“NO”, $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** (“NO”, $\emptyset$)

II) solve the reduced problem exactly

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

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

Runtime.

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** ("NO", $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** ("NO", $\emptyset$)

II) solve the reduced problem exactly

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

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

Runtime.

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** (“NO”, $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** (“NO”, $\emptyset$)

$O(n+m)$ time

II) solve the reduced problem exactly

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

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

Runtime.

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** ("NO", $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** ("NO", $\emptyset$)

$O(n+m)$ time

II) solve the reduced problem exactly

(vc, C') = BruteForceVC(G', k')
return (vc, $C \cup C'$)

Runtime.

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** (“NO”, $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** (“NO”, $\emptyset$)

$O(n + m)$ time

II) solve the reduced problem exactly

(vc, C') = BruteForceVC(G', k')
return (vc, $C \cup C'$)

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

Runtime.

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** ("NO", $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** ("NO", $\emptyset$)

$O(n+m)$ time

II) solve the reduced problem exactly

(vc, C') = BruteForceVC(G', k')
return (vc, $C \cup C'$)

$O(m' \cdot (n')^{k'})$ time
where $m' := |E'| \leq k^2$

Runtime.

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** ("NO", $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** ("NO", $\emptyset$)

$O(n+m)$ time

II) solve the reduced problem exactly

(vc, C') = BruteForceVC(G', k')
return (vc, $C \cup C'$)

$O(m' \cdot (n')^{k'})$ time
where $m' := |E'| \leq k^2$
$\Rightarrow \; n' := |V'| \leq 2k^2$

Runtime.

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** (“NO”, $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** (“NO”, $\emptyset$)

$O(n + m)$ time

II) solve the reduced problem exactly

(vc, C') = BruteForceVC(G', k')
return (vc, $C \cup C'$)

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

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

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** ("NO", $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** ("NO", $\emptyset$)

$O(n+m)$ time

II) solve the reduced problem exactly

$(\text{vc}, C') = \text{BruteForceVC}(G', k')$
return $(\text{vc}, C \cup C')$

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

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

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** ("NO", $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** ("NO", $\emptyset$)

$O(n + m)$ time

II) solve the reduced problem exactly

(vc, C') = BruteForceVC(G', k')
return (vc, $C \cup C'$)

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

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

Also:

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel of the instance

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** (“NO”, $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** (“NO”, $\emptyset$)

$O(n + m)$ time

II) solve the reduced problem exactly

(vc, C') = BruteForceVC(G', k')
return (vc, $C \cup C'$)

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

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

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

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** ("NO", $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** ("NO", $\emptyset$)

$O(n+m)$ time

II) solve the reduced problem exactly

$(\text{vc}, C') = \text{BruteForceVC}(G', k')$
return $(\text{vc}, C \cup C')$

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

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

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

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** (“NO”, $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** (“NO”, $\emptyset$)

$O(n+m)$ time

II) solve the reduced problem exactly

(vc, C') = BruteForceVC(G', k')
return (vc, $C \cup C'$)

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

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

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

Algorithm [Buss 1993]

BussVC(Graph G, Integer k)

I) Reduce to the kernel *of the instance*

$C = \{v \in V \mid \deg v > k\}$
if $|C| > k$ **then return** (“NO”, $\emptyset$)

$G' = (V', E') := G[V \setminus (C \cup L)]$ ($L =$ isolated vertices)
$k' = k - |C|$
if $|E'| > k^2$ **then return** (“NO”, $\emptyset$)

$O(n + m)$ time

II) solve the reduced problem exactly

$(\mathrm{vc}, C') = \mathrm{BruteForceVC}(G', k')$
return $(\mathrm{vc}, C \cup C')$

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

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

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

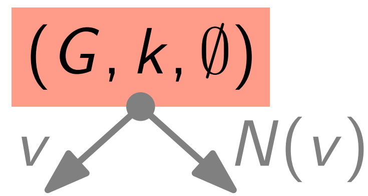
Search Tree Algorithm

Idea. Improve phase II using a search tree.

$(G, k, \emptyset)$

Search Tree Algorithm

Idea. Improve phase II using a search tree.



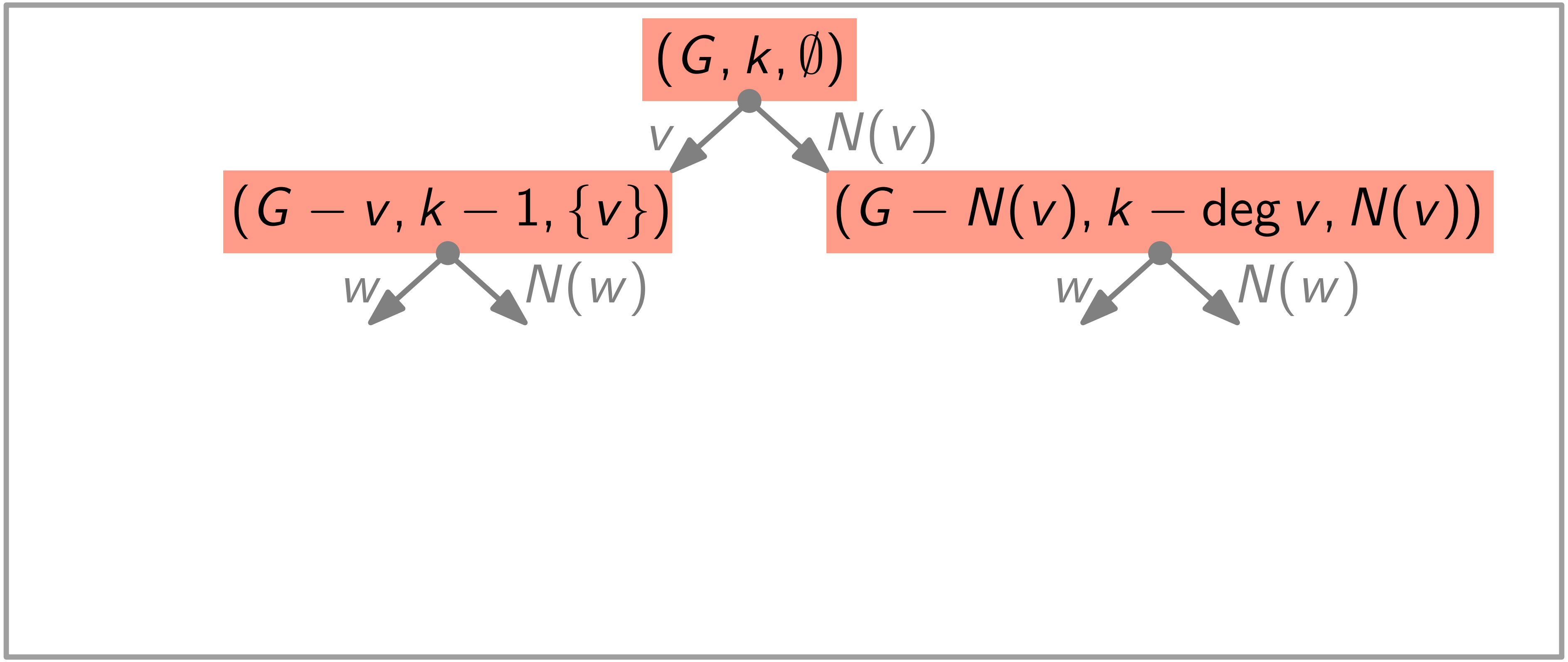
Search Tree Algorithm

Idea. Improve phase II using a search tree.

$$
\begin{tikzcd}
& (G, k, \emptyset) \arrow[dl, "v"'] \arrow[dr, "N(v)"] & \\
(G - v, k - 1, \{v\}) & & (G - N(v), k - \deg v, N(v))
\end{tikzcd}
$$

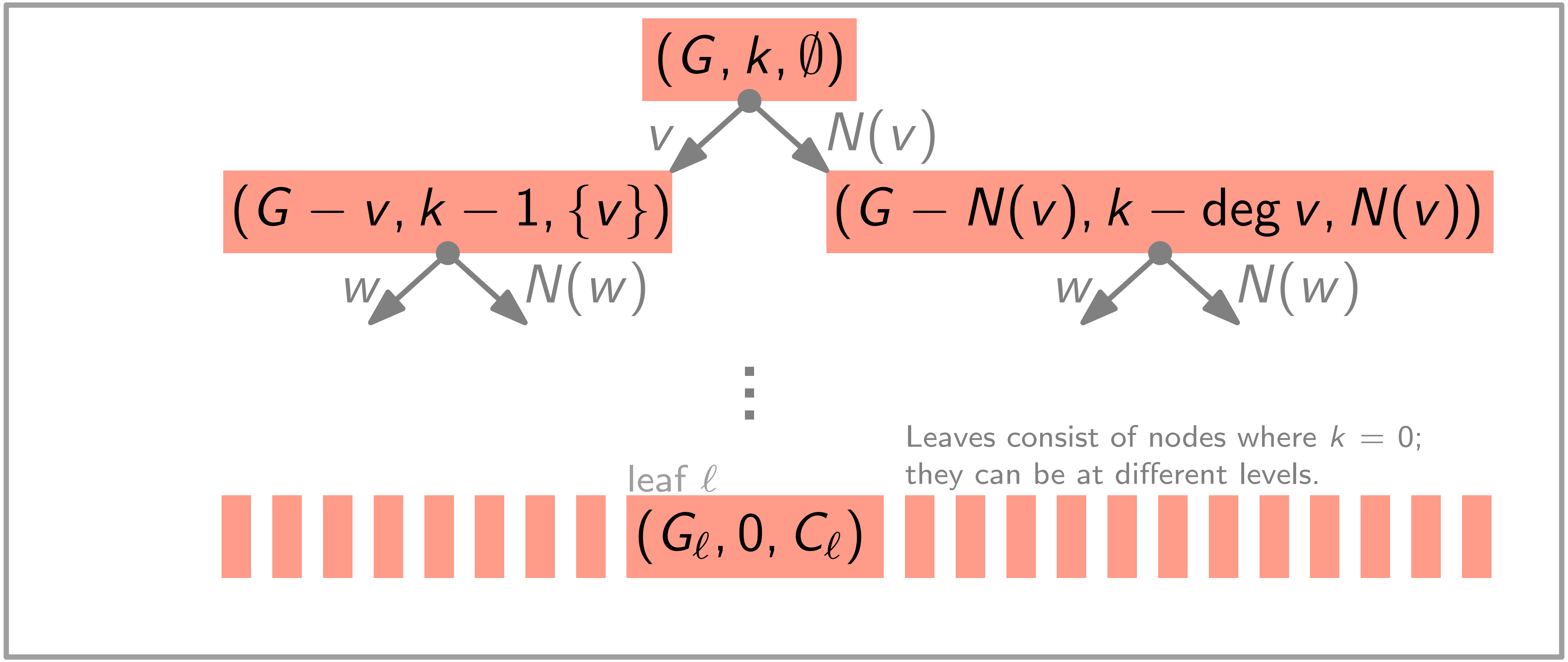
Search Tree Algorithm

Idea. Improve phase II using a search tree.



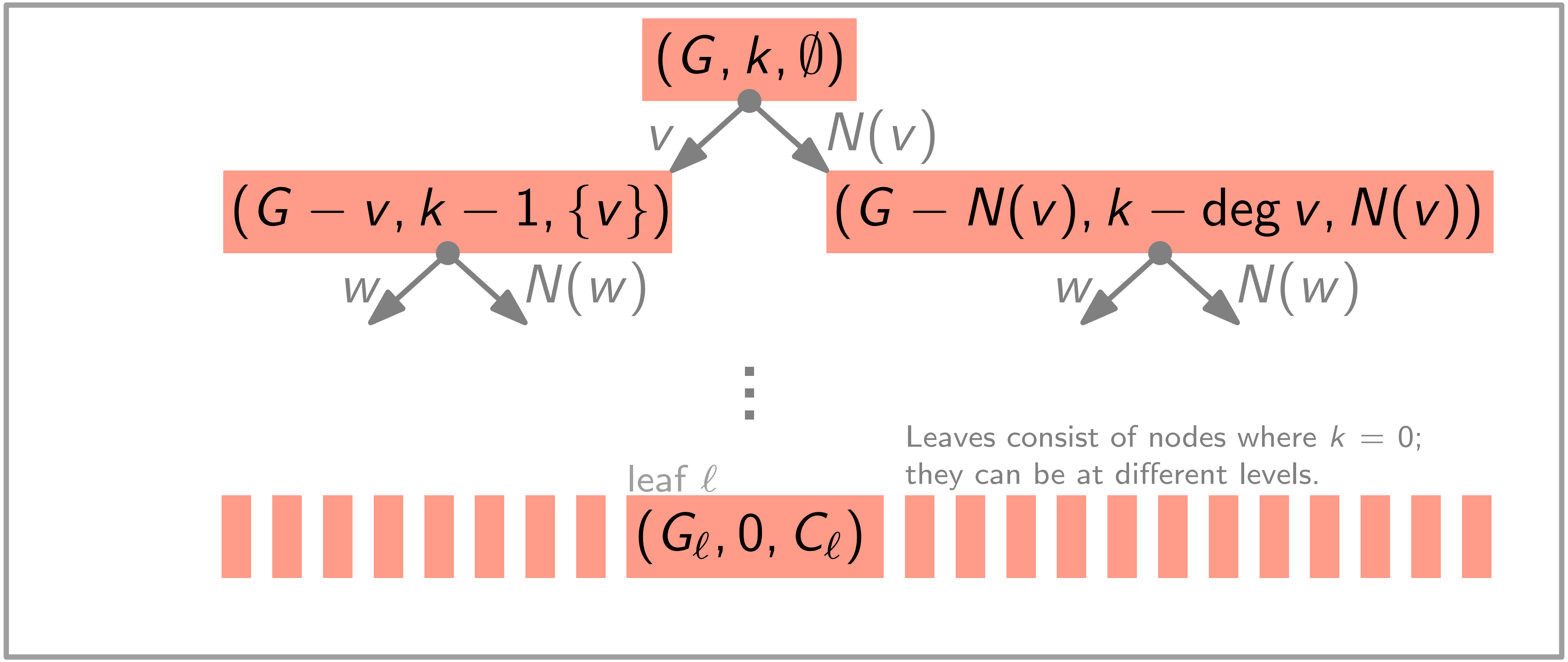
Search Tree Algorithm

Idea. Improve phase II using a search tree.



Search Tree Algorithm

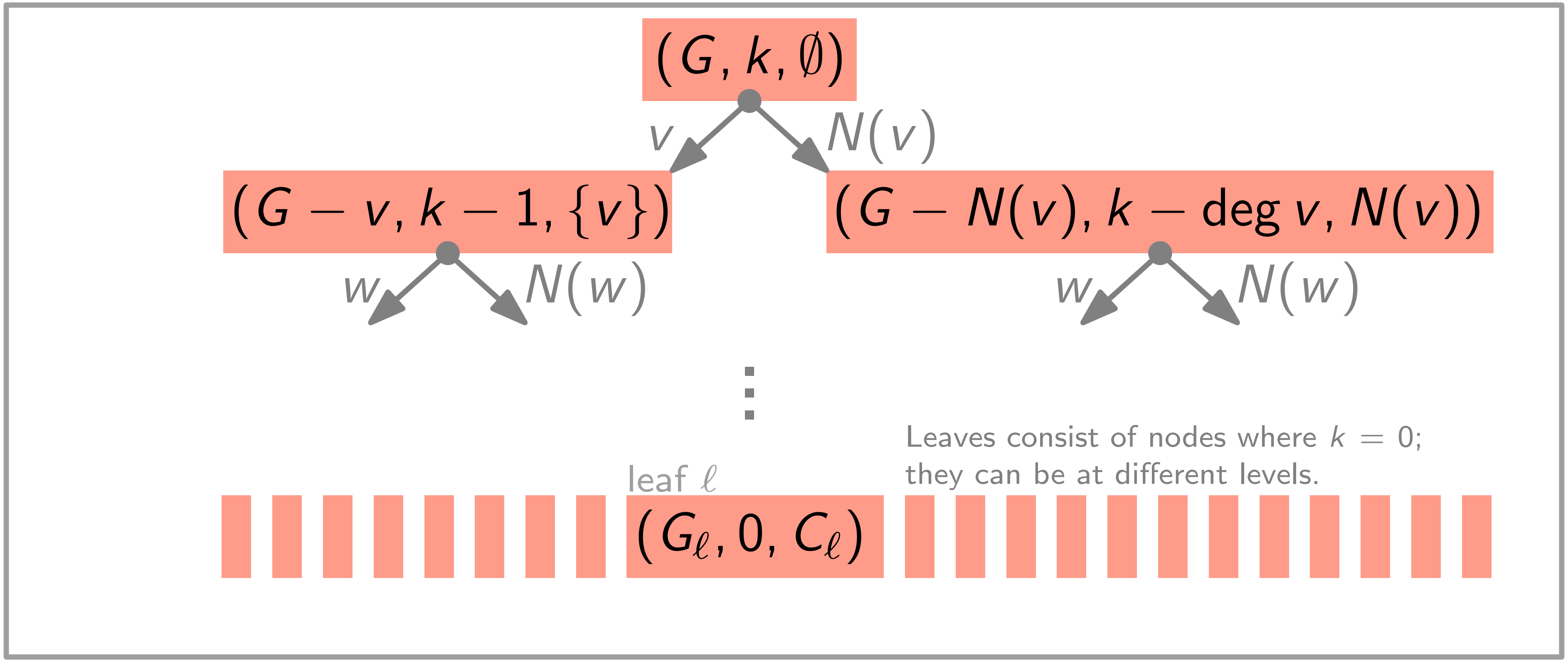
Idea. Improve phase II using a search tree.



YES:

Search Tree Algorithm

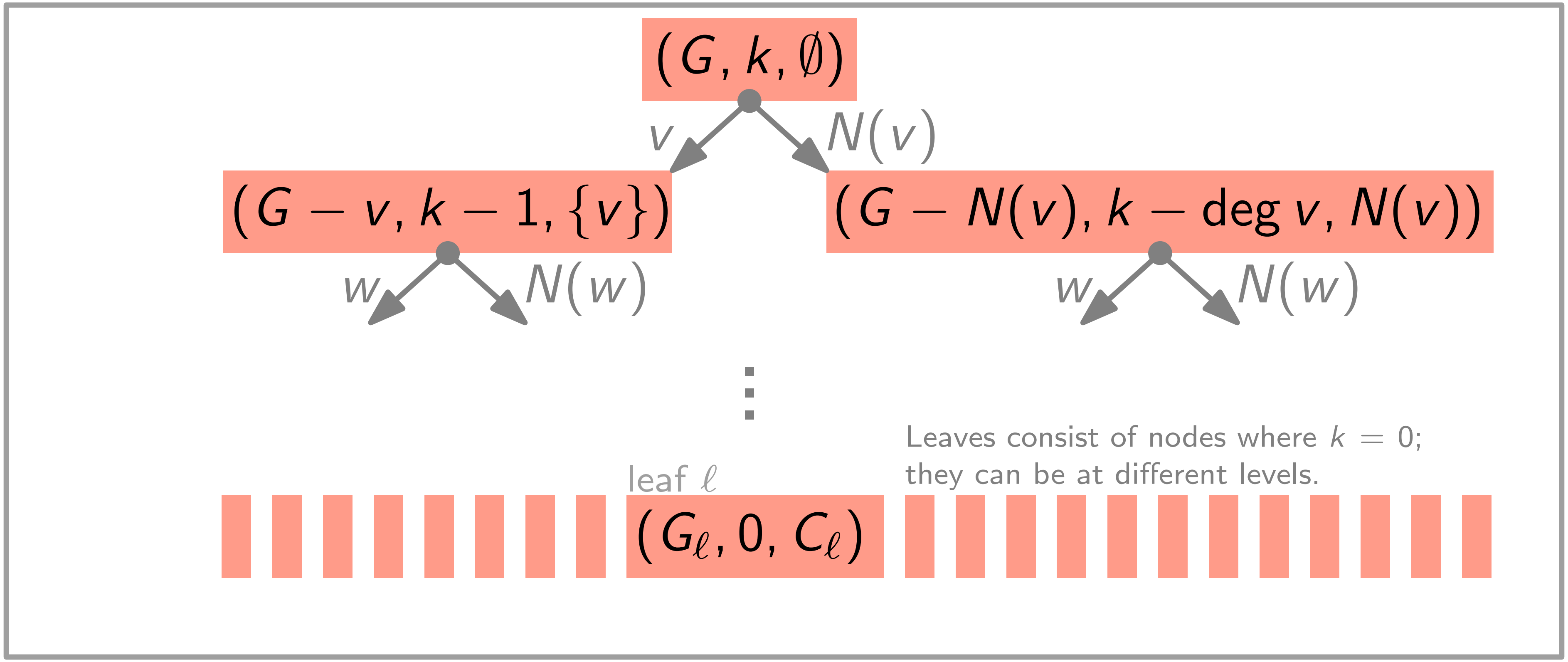
Idea. Improve phase II using a search tree.



YES: If there is a leaf ℓ where $E_\ell = \emptyset$, then C_ℓ is a k-VC of G.

Search Tree Algorithm

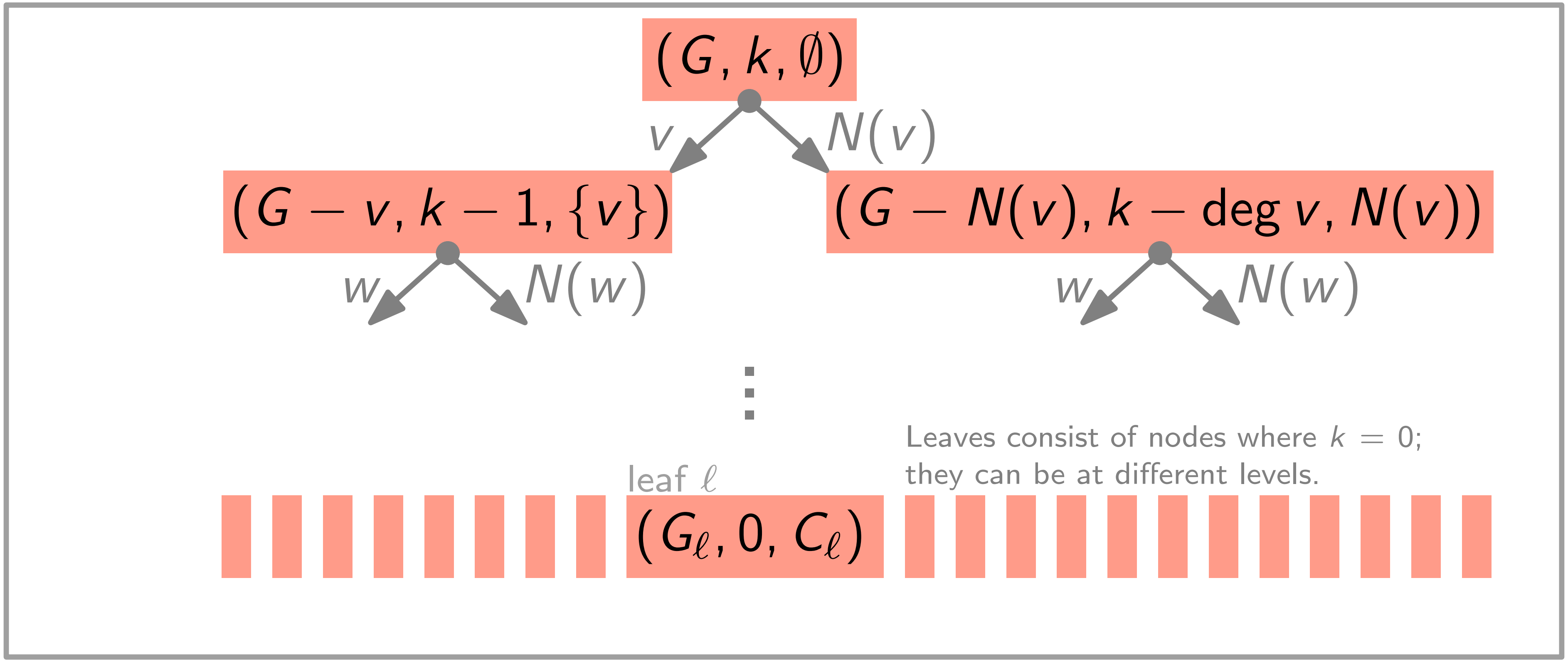
Idea. Improve phase II using a search tree.



YES: If there is a leaf ℓ where $E_\ell = \emptyset$, then C_ℓ is a k-VC of G.
NO:

Search Tree Algorithm

Idea. Improve phase II using a search tree.

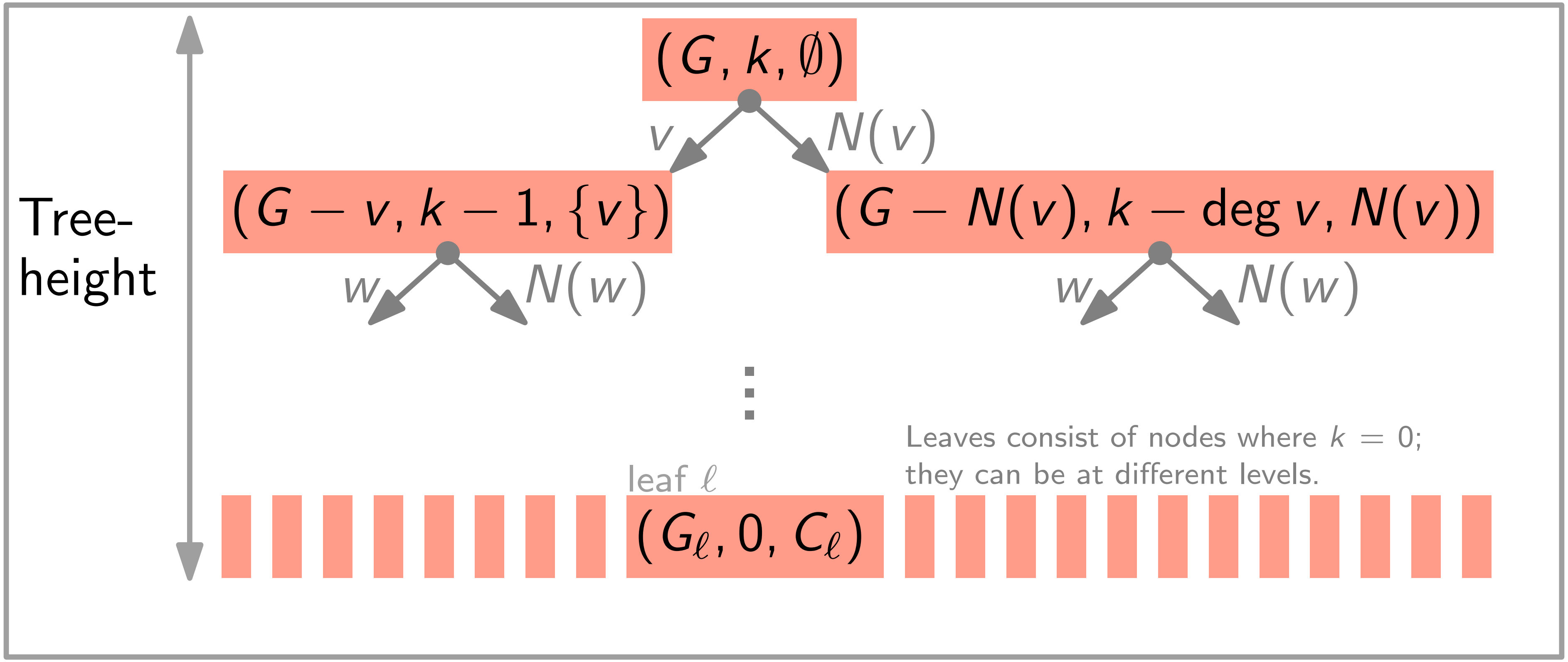


YES: If there is a leaf ℓ where $E_\ell = \emptyset$, then C_ℓ is a k-VC of G.

NO: If there is no such leaf, then G has no k-VC.

Search Tree Algorithm

Idea. Improve phase II using a search tree.

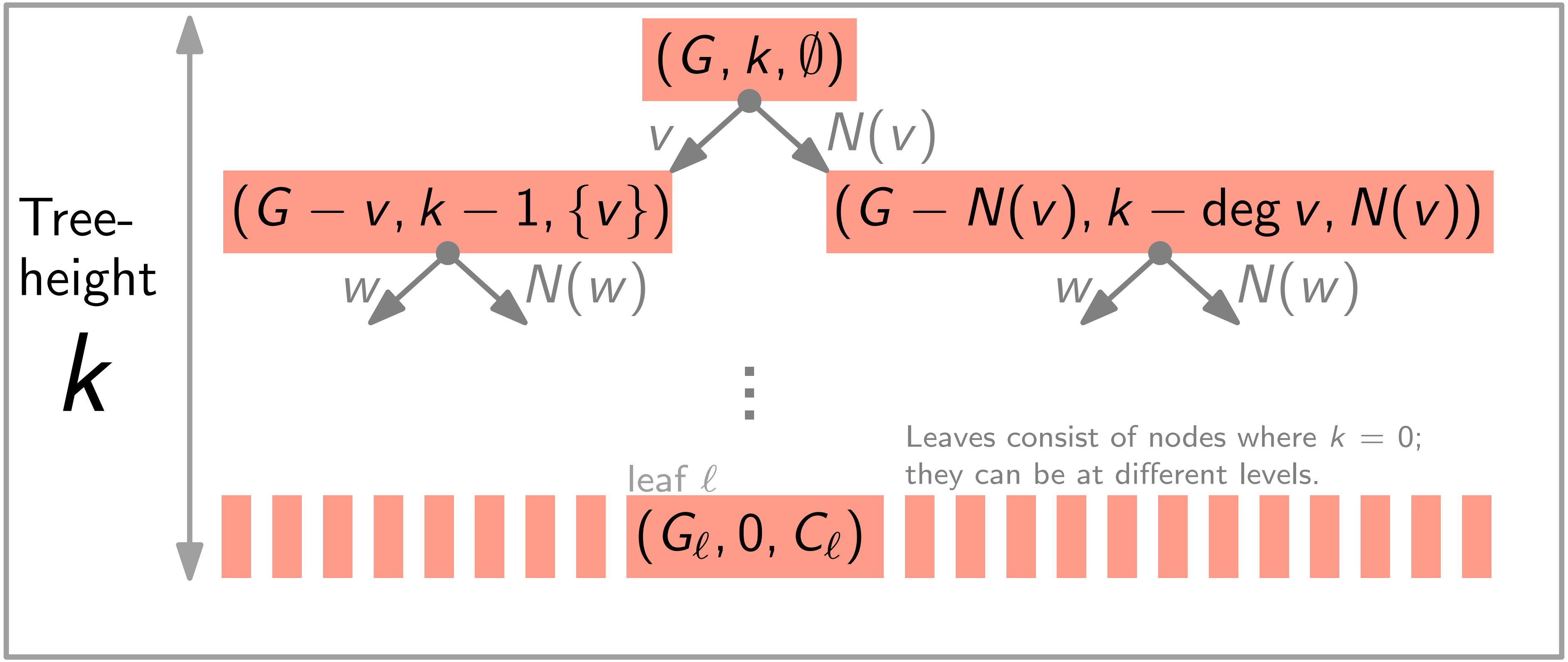


YES: If there is a leaf ℓ where $E_\ell = \emptyset$, then C_ℓ is a k-VC of G.

NO: If there is no such leaf, then G has no k-VC.

Search Tree Algorithm

Idea. Improve phase II using a search tree.

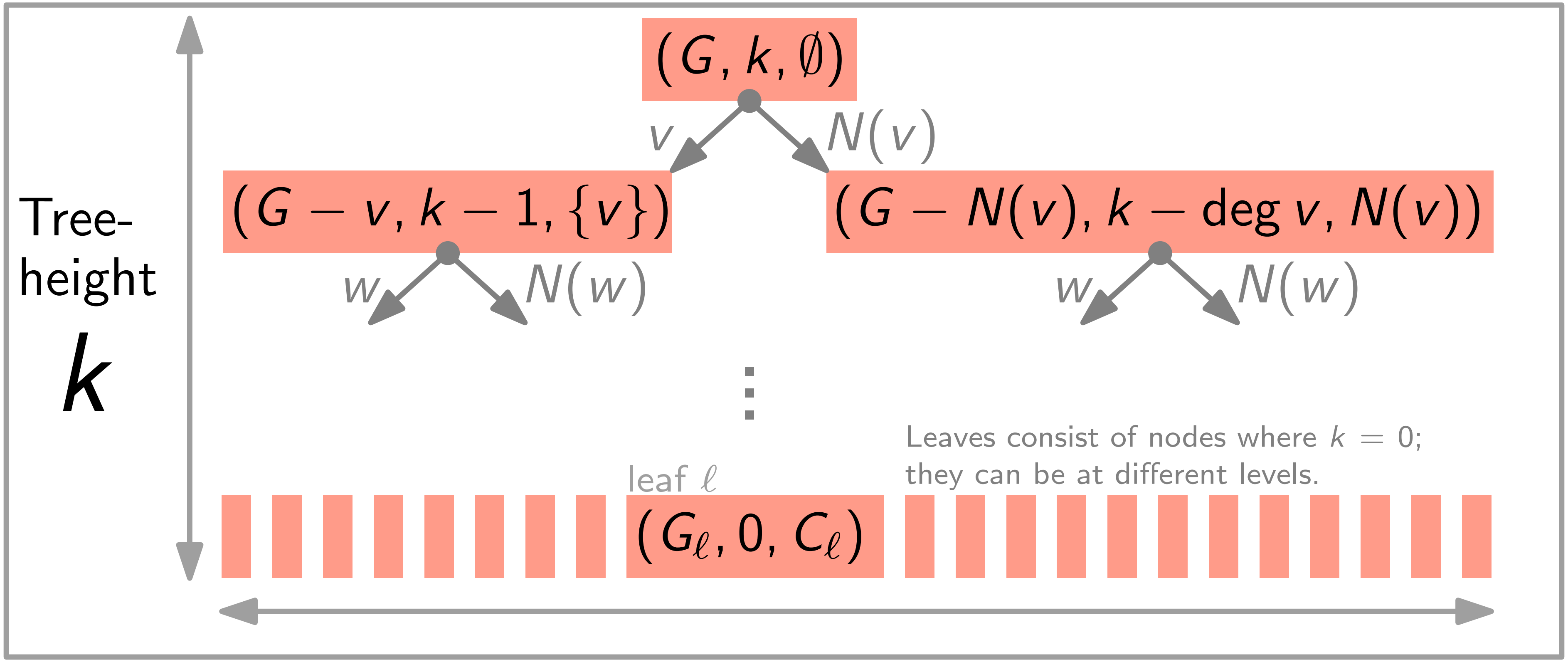


YES: If there is a leaf ℓ where $E_\ell = \emptyset$, then C_ℓ is a k-VC of G.

NO: If there is no such leaf, then G has no k-VC.

Search Tree Algorithm

Idea. Improve phase II using a search tree.

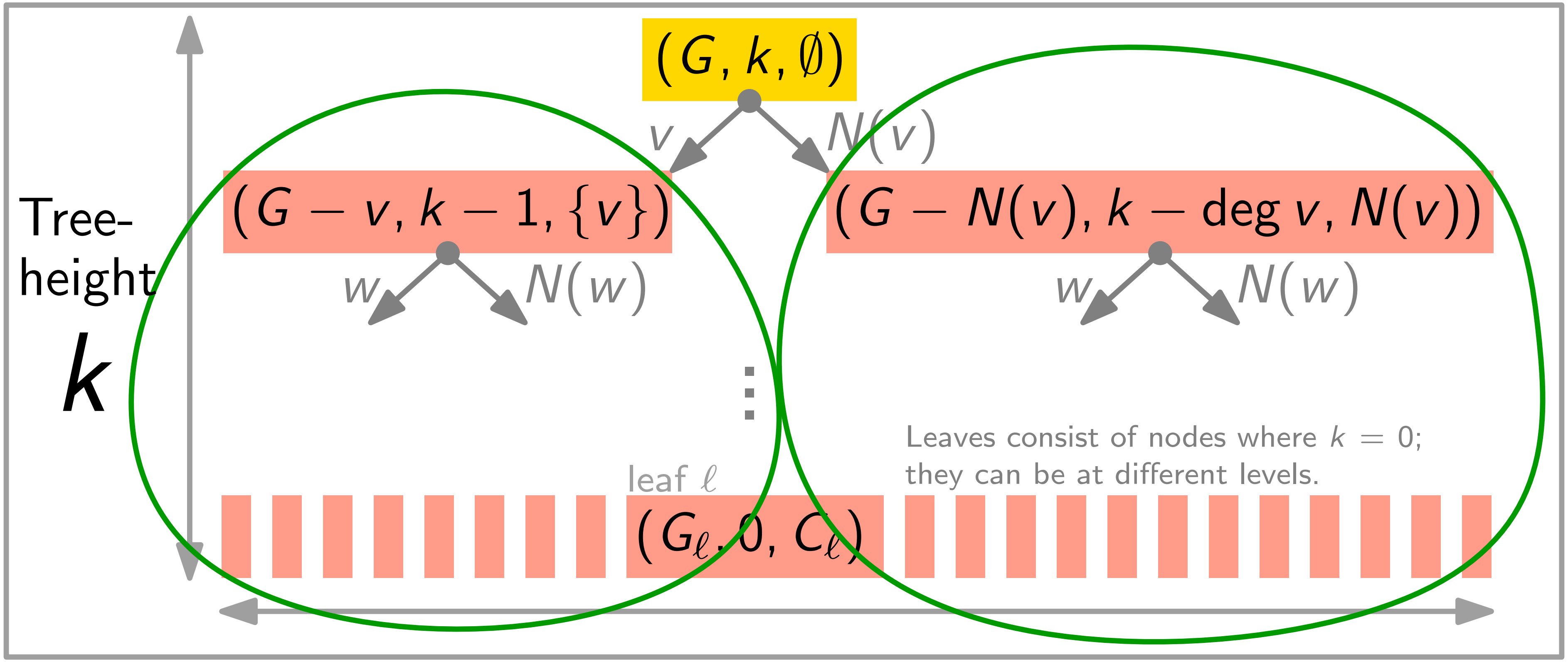


#nodes: $T(k) \leq 2$

YES: If there is a leaf ℓ where $E_\ell = \emptyset$, then C_ℓ is a k-VC of G.
NO: If there is no such leaf, then G has no k-VC.

Search Tree Algorithm

Idea. Improve phase II using a search tree.



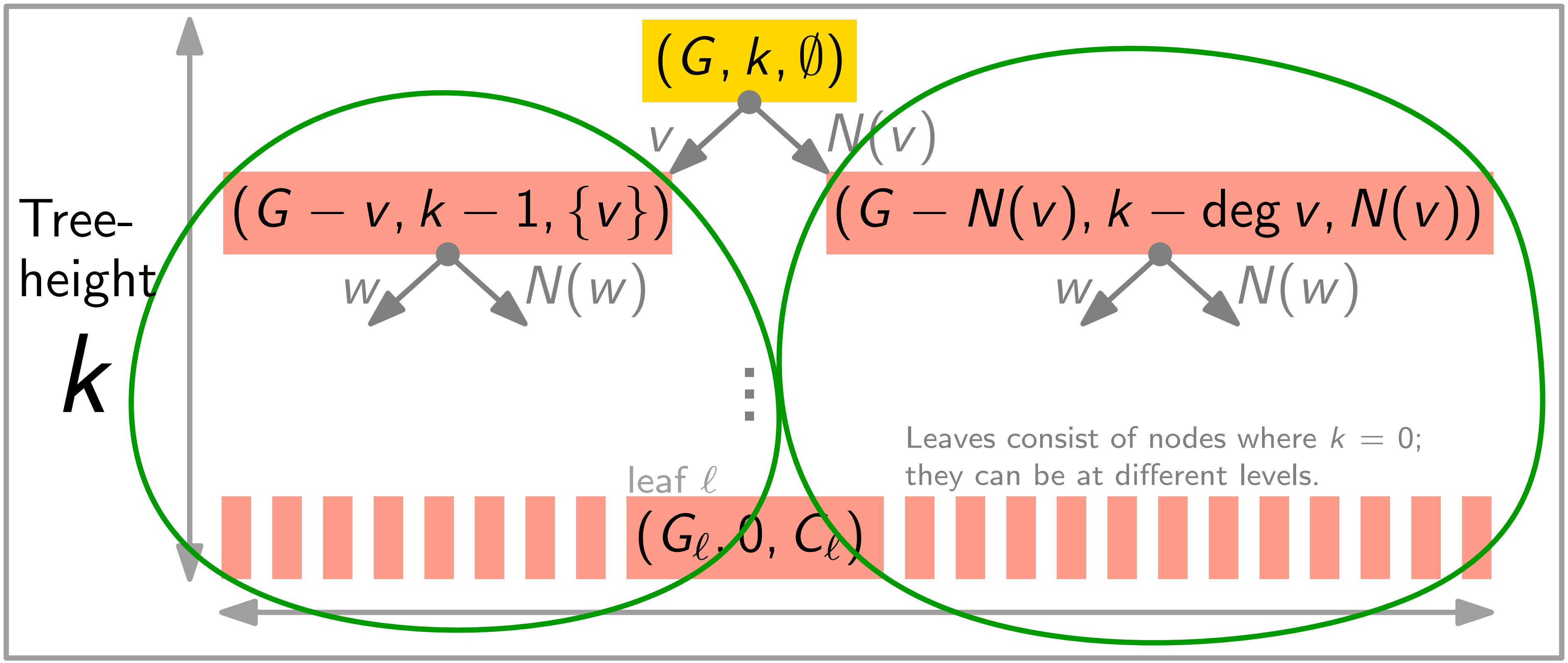
#nodes: $T(k) \leq 2$

YES: If there is a leaf ℓ where $E_\ell = \emptyset$, then C_ℓ is a k-VC of G.

NO: If there is no such leaf, then G has no k-VC.

Search Tree Algorithm

Idea. Improve phase II using a search tree.



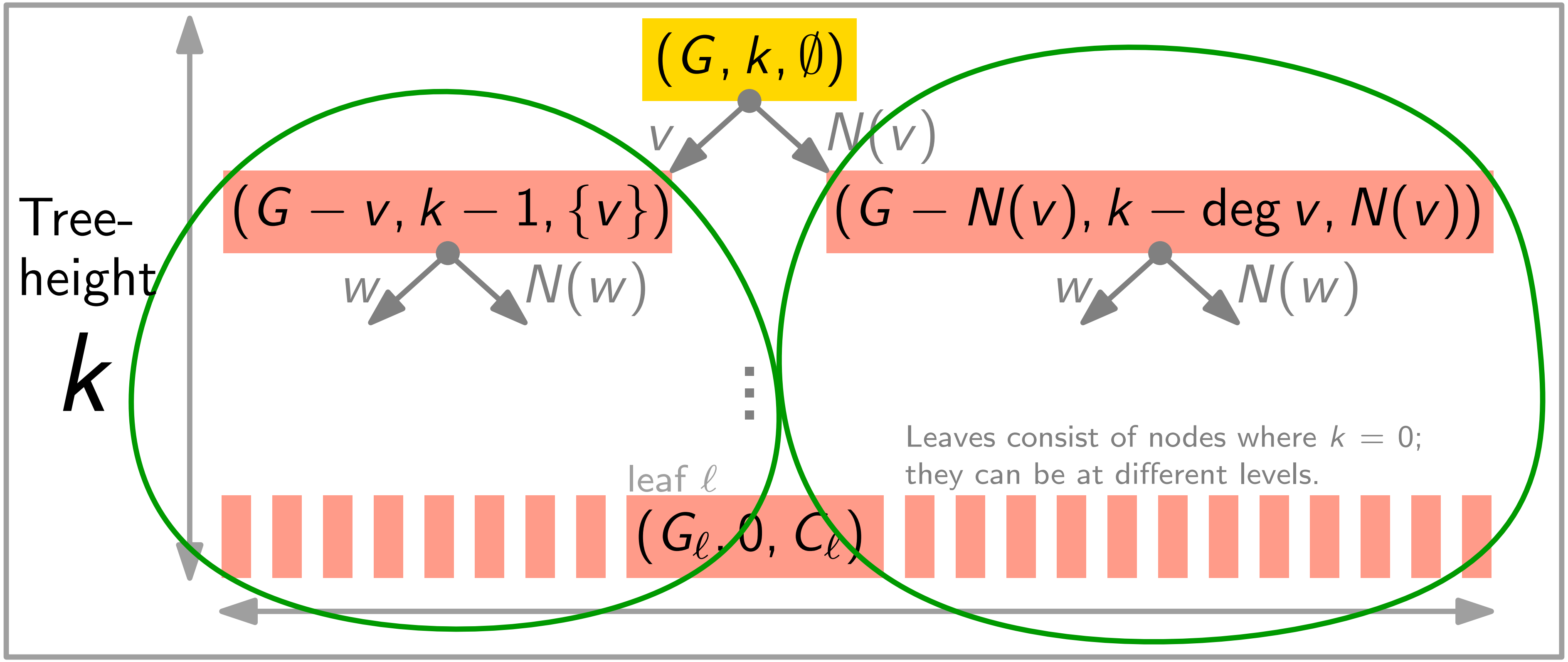
#nodes: $T(k) \leq 2T(k-1) + \boxed{1}$, $T(0) = 1$

YES: If there is a leaf ℓ where $E_\ell = \emptyset$, then C_ℓ is a k-VC of G.

NO: If there is no such leaf, then G has no k-VC.

Search Tree Algorithm

Idea. Improve phase II using a search tree.



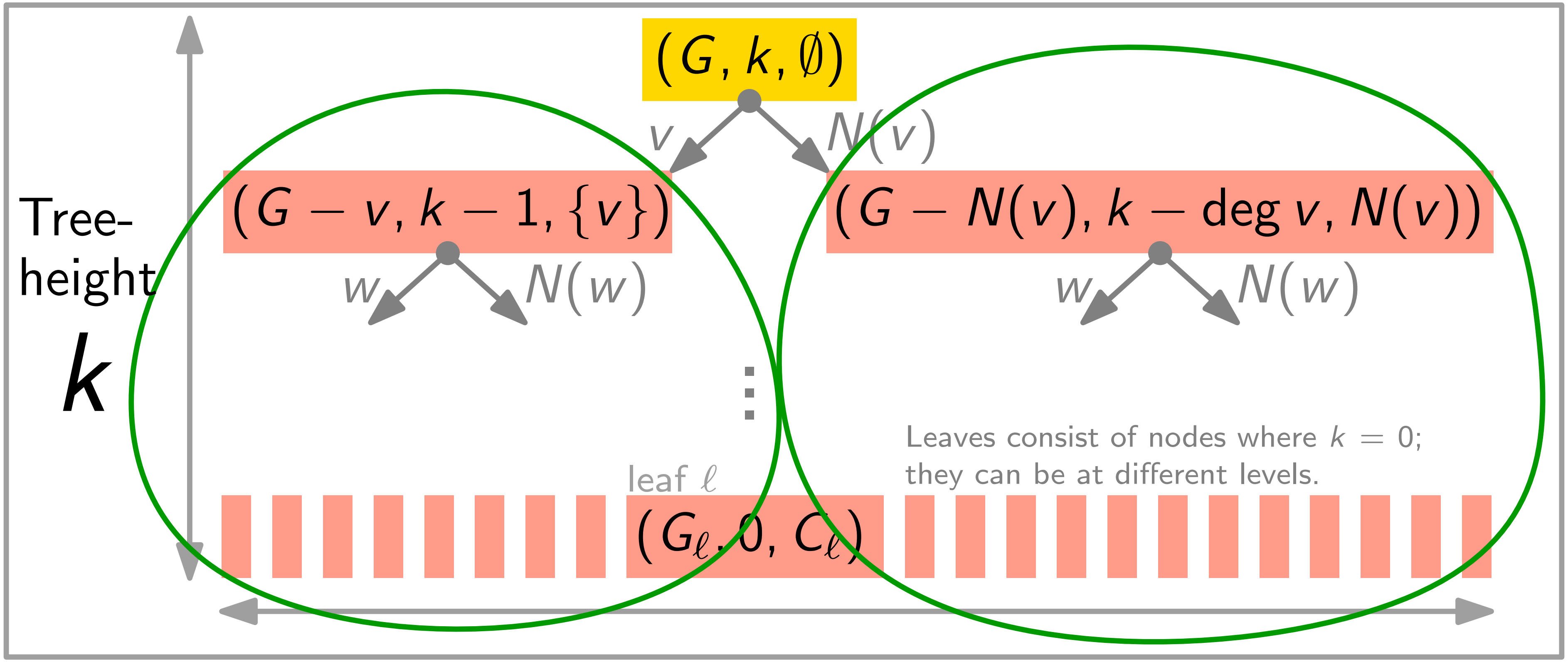
#nodes: $T(k) \leq 2T(k-1) + \boxed{1}$, $T(0) = 1 \quad \Rightarrow T(k) \leq 2^{k+1} - 1 \in O(2^k)$

YES: If there is a leaf ℓ where $E_\ell = \emptyset$, then C_ℓ is a k-VC of G.

NO: If there is no such leaf, then G has no k-VC.

Search Tree Algorithm

Idea. Improve phase II using a search tree.



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

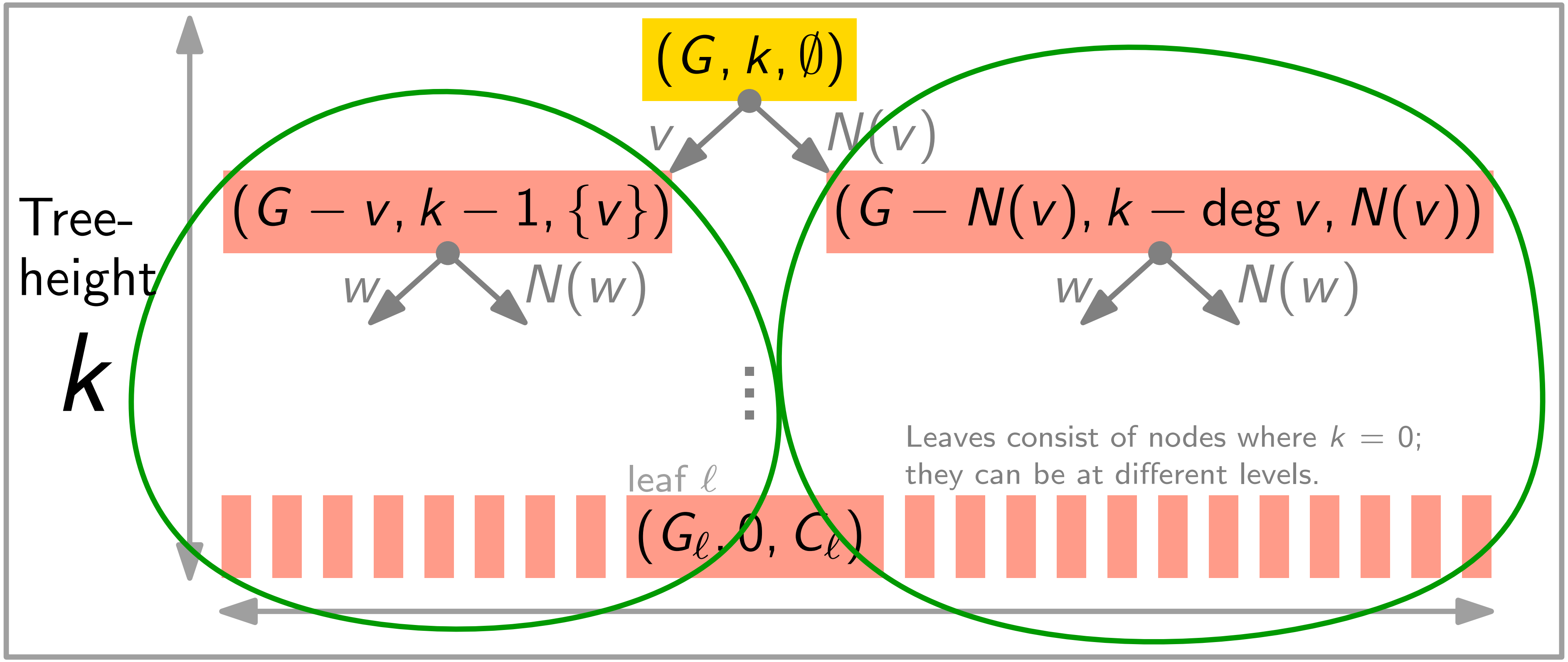
$\Rightarrow$ **Runtime:**

YES: If there is a leaf ℓ where $E_\ell = \emptyset$, then C_ℓ is a k-VC of G.

NO: If there is no such leaf, then G has no k-VC.

Search Tree Algorithm

Idea. Improve phase II using a search tree.



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

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

YES: If there is a leaf ℓ where $E_\ell = \emptyset$, then C_ℓ is a k-VC of G.

NO: If there is no such leaf, then G has no k-VC.

Degree-4 Algorithm

Idea. Better analysis based on $|N(v)|$.

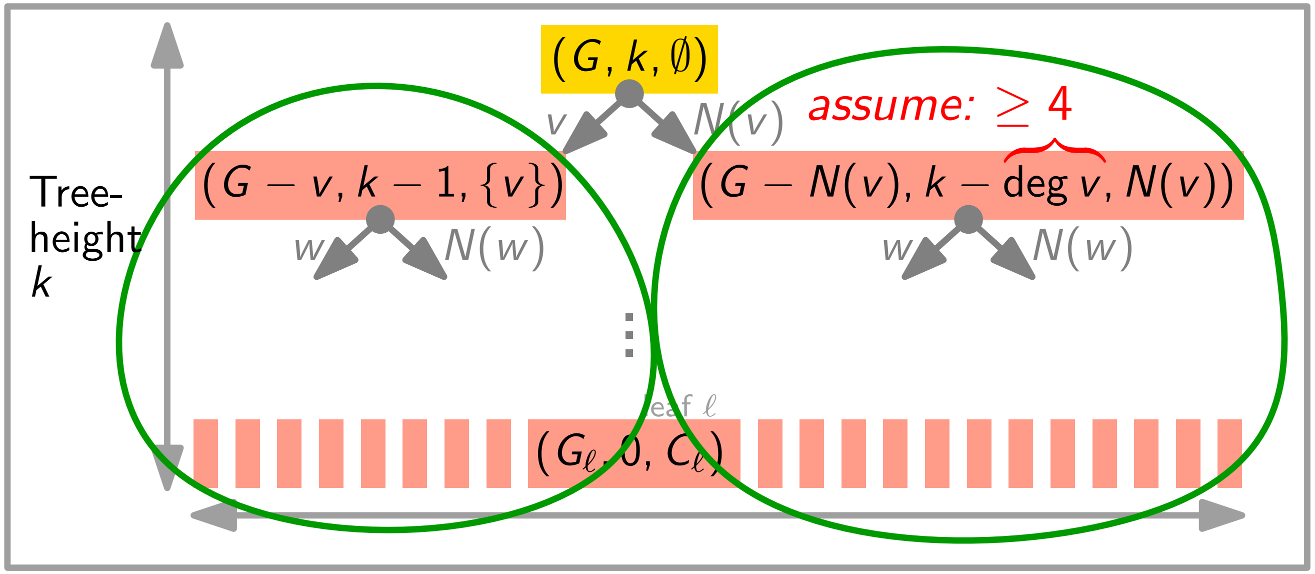
Degree-4 Algorithm

Idea. Better analysis based on $|N(v)|$.

What if we could always branch on a vertex v whose degree is at least 4?

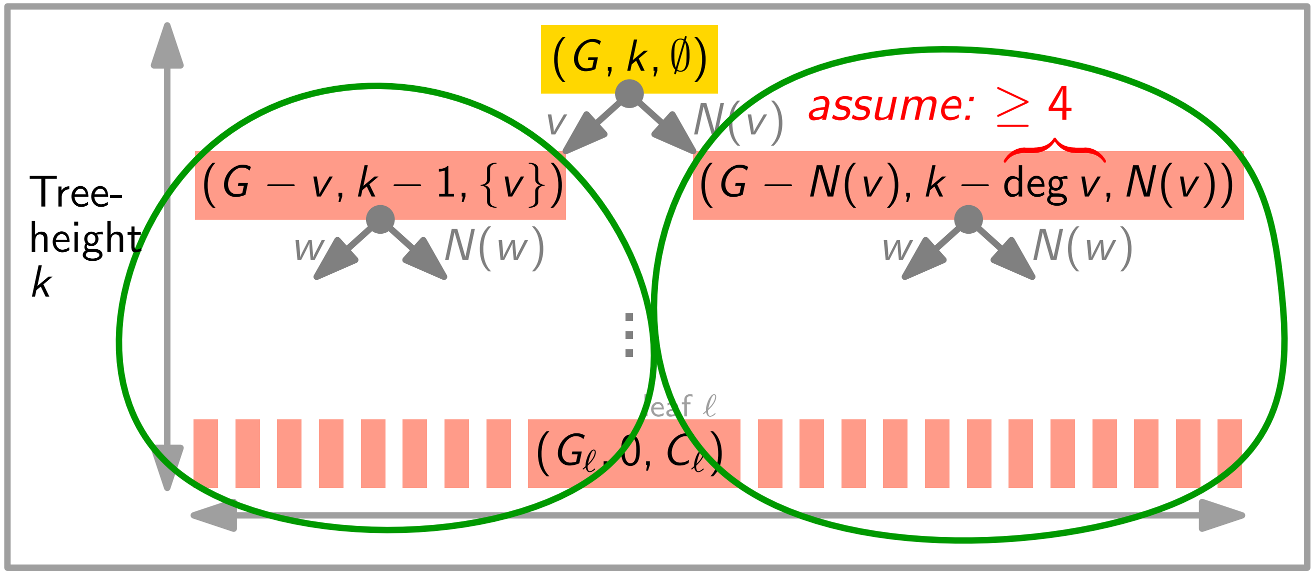
Degree-4 Algorithm

Idea. Better analysis based on $|N(v)|$.



Degree-4 Algorithm

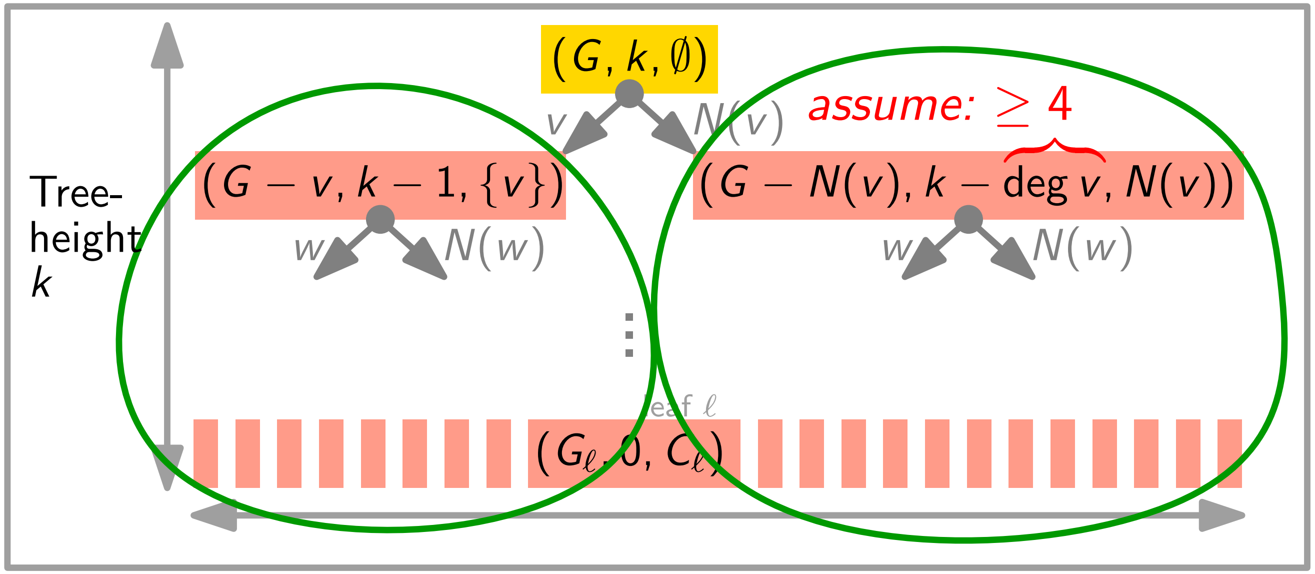
Idea. Better analysis based on $|N(v)|$.



$\Rightarrow \; T(k) =$

Degree-4 Algorithm

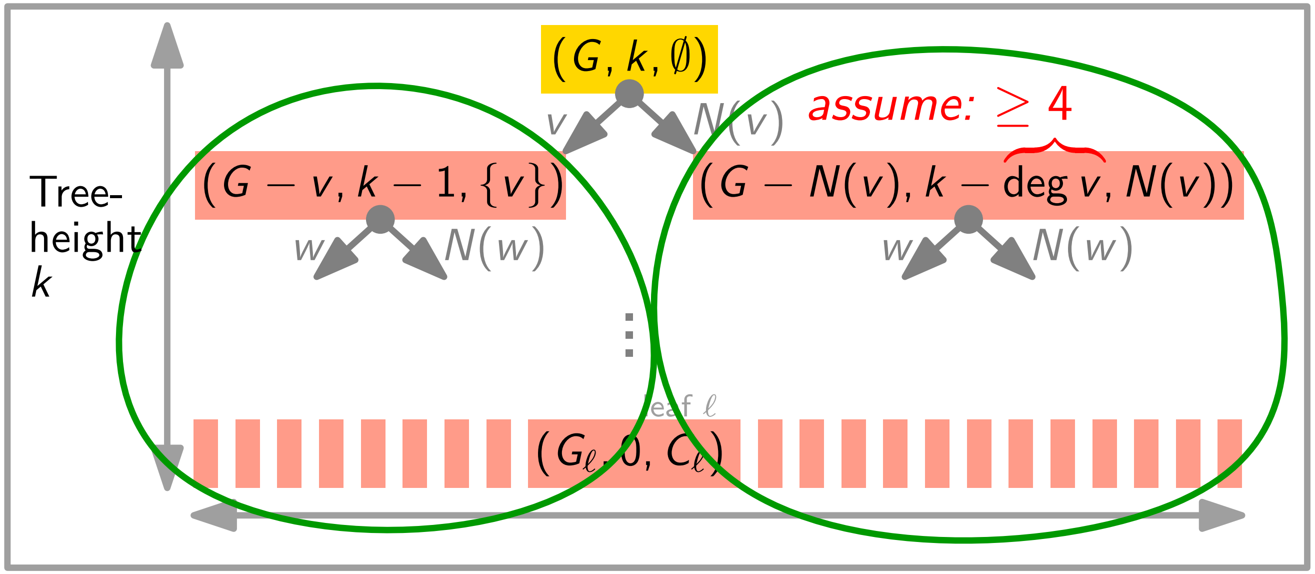
Idea. Better analysis based on $|N(v)|$.



$$\Rightarrow \ T(k) = T(k-4) + T(k-1) + 1, \quad T(\leq 4) = const.$$

Degree-4 Algorithm

Idea. Better analysis based on $|N(v)|$.

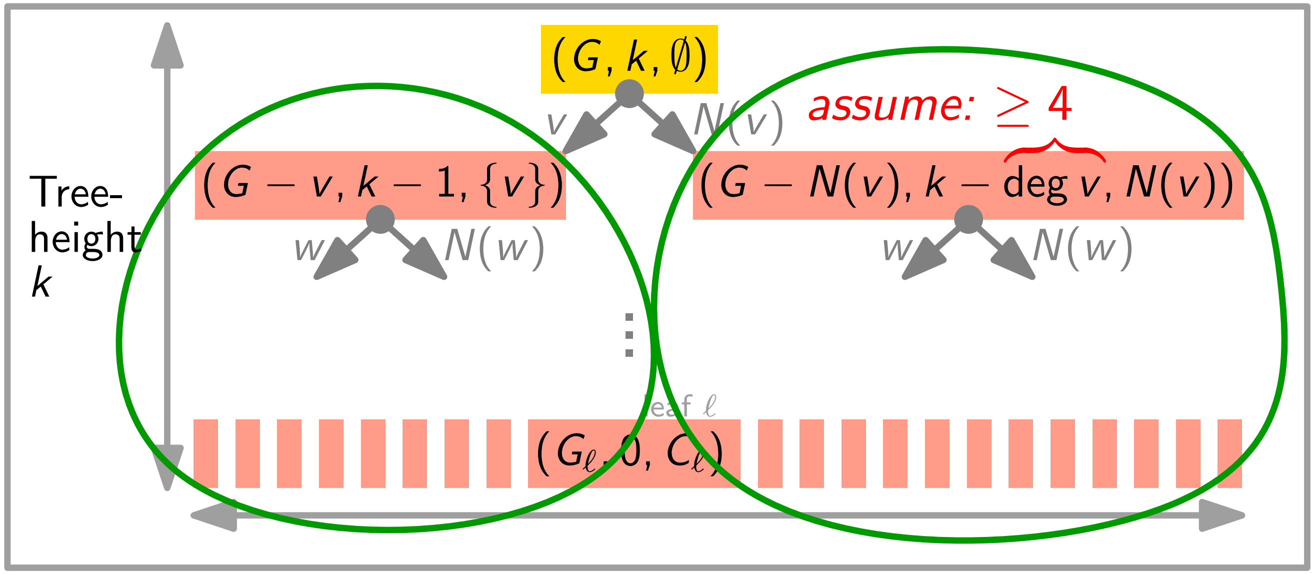


$$\Rightarrow \; T(k) = T(k - 4) + T(k - 1) + 1, \quad T(\leq 4) = const.$$

branching vector $(4, 1)$

Degree-4 Algorithm

Idea. Better analysis based on $|N(v)|$.



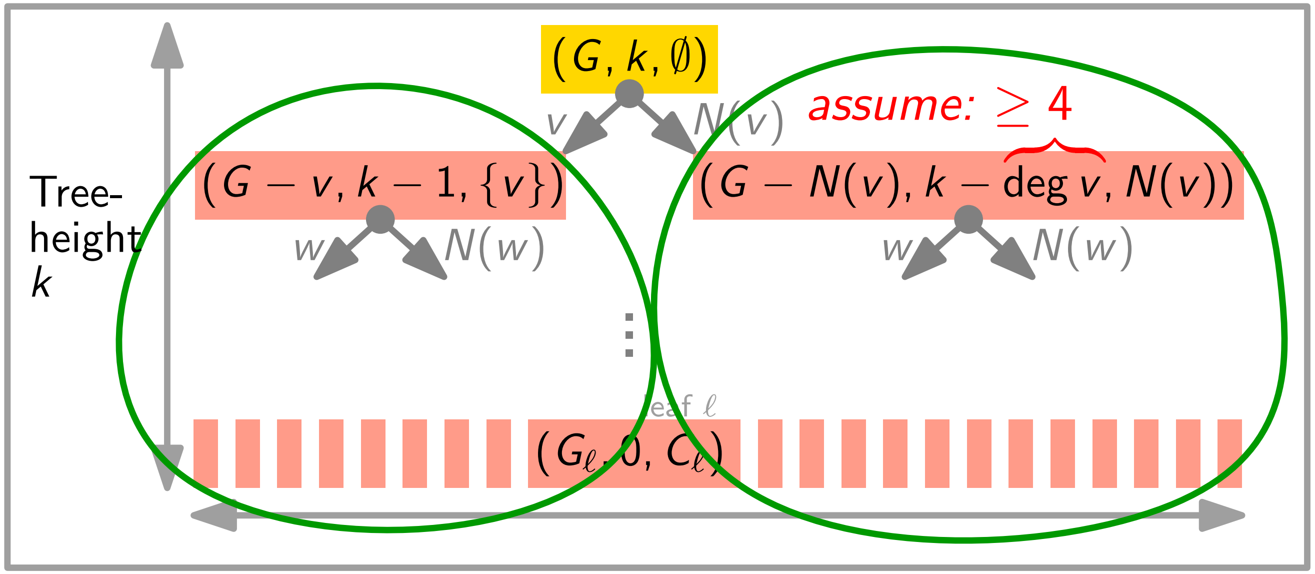
$$\Rightarrow \quad T(k) = T(k - \boxed{4}) + T(k - \boxed{1}) + 1, \quad T(\leq 4) = const.$$

branching vector $(4, 1)$

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

Degree-4 Algorithm

Idea. Better analysis based on $|N(v)|$.



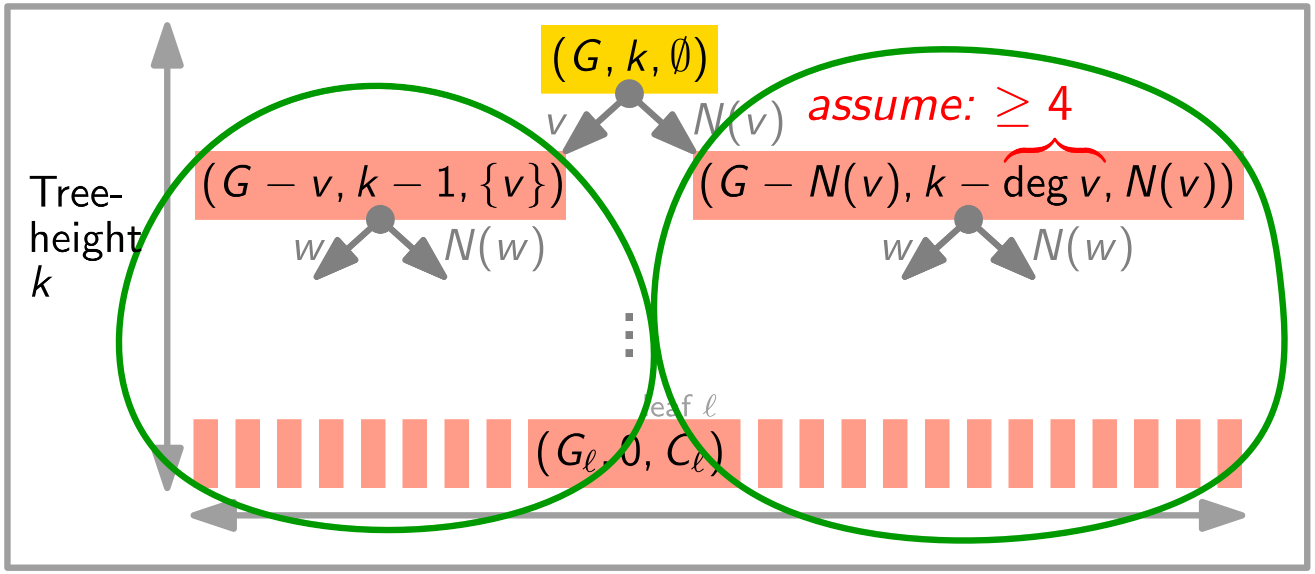
$\Rightarrow \; T(k) = T(k - \boxed{4}) + T(k - \boxed{1}) + 1, \; T(\leq 4) = const.$

branching vector $(4, 1)$

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

Degree-4 Algorithm

Idea. Better analysis based on $|N(v)|$.



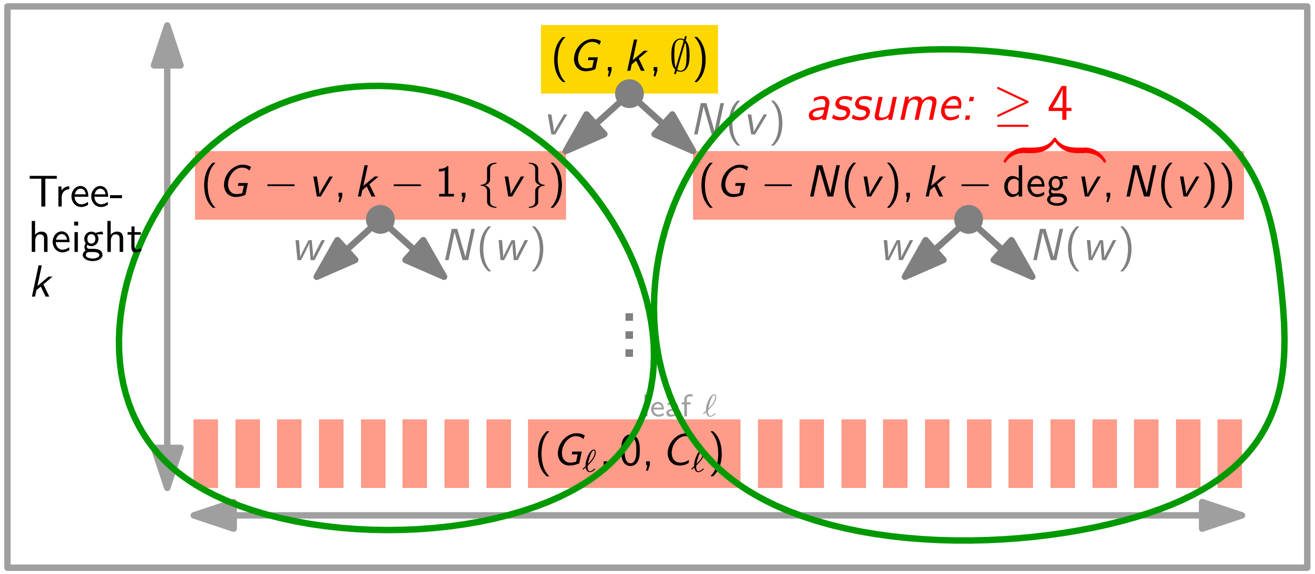
$\Rightarrow \; T(k) = T(k - 4) + T(k - 1) + 1, \; T(\leq 4) = const.$

branching vector $(4, 1)$

solve $T(k) = z^k - 1 \quad \Rightarrow \quad z^k = z^{k-4} + z^{k-1}$

Degree-4 Algorithm

Idea. Better analysis based on $|N(v)|$.



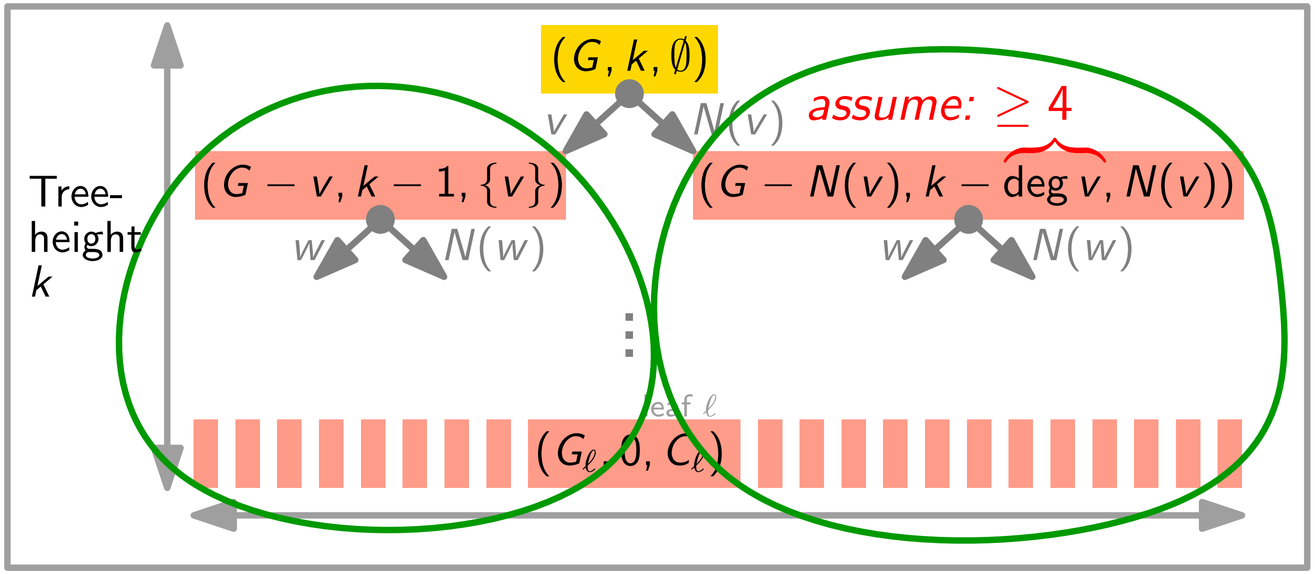
$$\Rightarrow \quad T(k) = T(k - 4) + T(k - 1) + 1, \quad T(\leq 4) = const.$$

branching vector $(4, 1)$

solve $T(k) = z^k - 1 \quad \Rightarrow \quad z^k = z^{k-4} + z^{k-1}$

Degree-4 Algorithm

Idea. Better analysis based on $|N(v)|$.



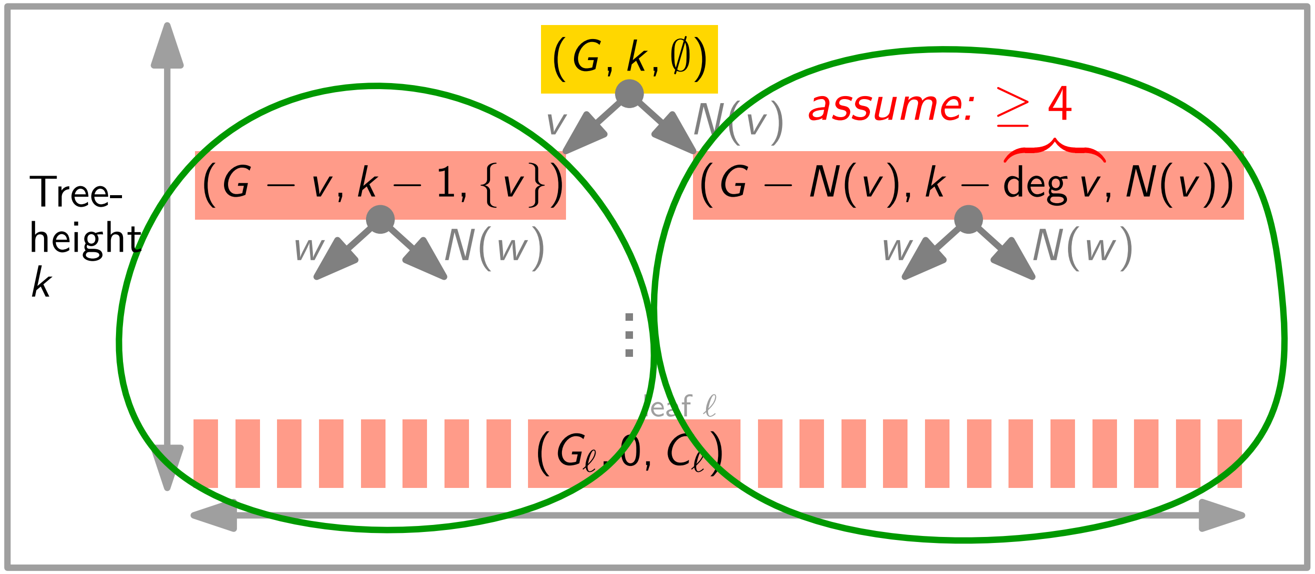
$\Rightarrow \ T(k) = T(k - \boxed{4}) + T(k - \boxed{1}) + 1, \ T(\leq 4) = const.$

branching vector $(4, 1)$

solve $T(k) = z^k - 1 \quad \Rightarrow \quad z^k = z^{k-4} + z^{k-1}$ $\cdot \frac{1}{z^{k-4}}$

Degree-4 Algorithm

Idea. Better analysis based on $|N(v)|$.



$\Rightarrow \; T(k) = T(k - 4) + T(k - 1) + 1, \; T(\leq 4) = const.$

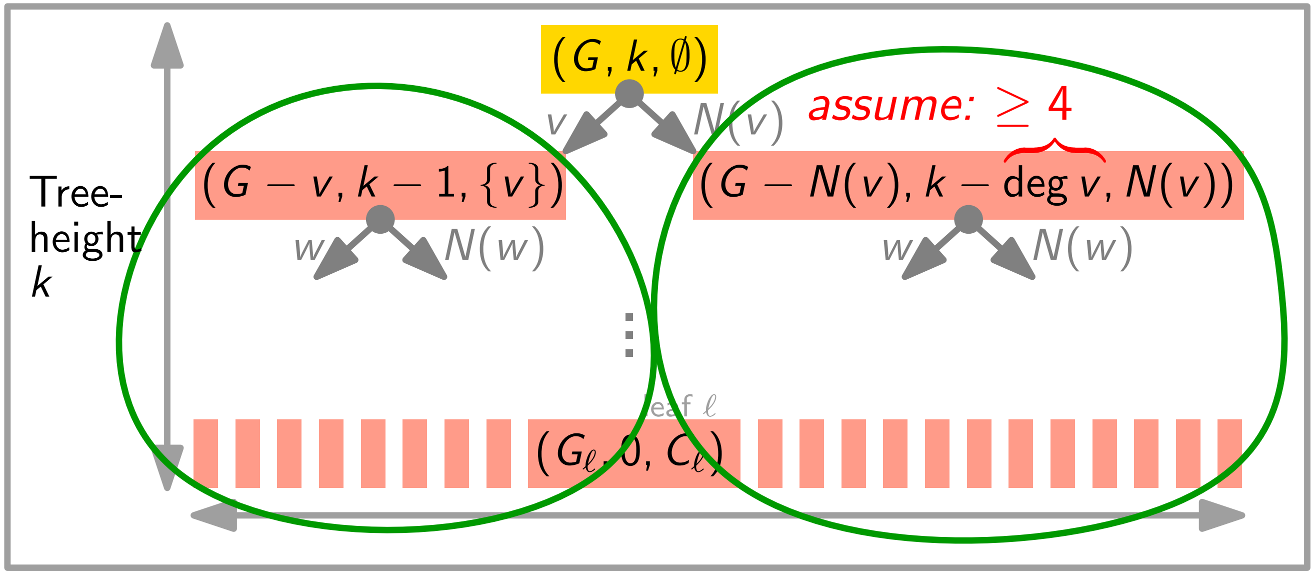
branching vector $(4, 1)$

solve $T(k) = z^k - 1 \quad \Rightarrow \quad z^k = z^{k-4} + z^{k-1}$ $\quad \cdot \frac{1}{z^{k-4}}$

$\Rightarrow$ Characteristic polynomial: $z^4 = 1 + z^3$

Degree-4 Algorithm

Idea. Better analysis based on $|N(v)|$.



$\Rightarrow$ $T(k) = T(k - \boxed{4}) + T(k - \boxed{1}) + 1,\ T(\leq 4) = const.$

branching vector $(4, 1)$

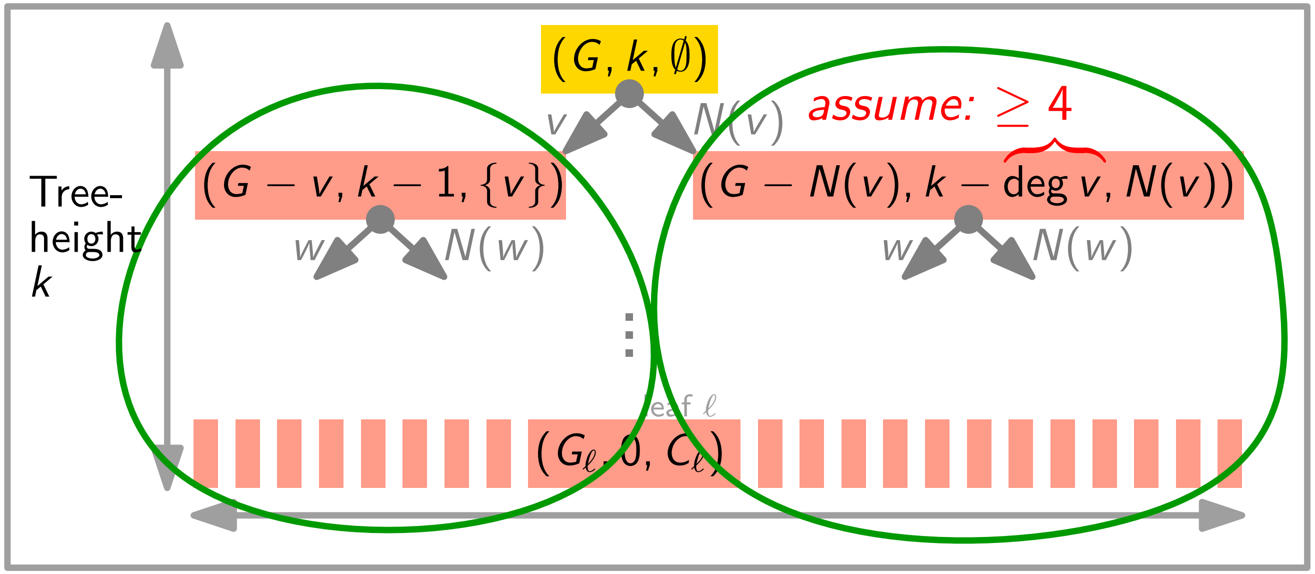
solve $T(k) = z^k - 1 \quad \Rightarrow \quad z^k = z^{k-4} + z^{k-1}$ $\cdot \frac{1}{z^{k-4}}$

$\Rightarrow$ Characteristic polynomial: $z^4 = 1 + z^3$

$\Rightarrow$ largest positive solution:

Degree-4 Algorithm

Idea. Better analysis based on $|N(v)|$.



$\Rightarrow \; T(k) = \; T(k - \boxed{4}) + T(k - \boxed{1}) + 1, \; T(\leq 4) = const.$

branching vector $(4, 1)$

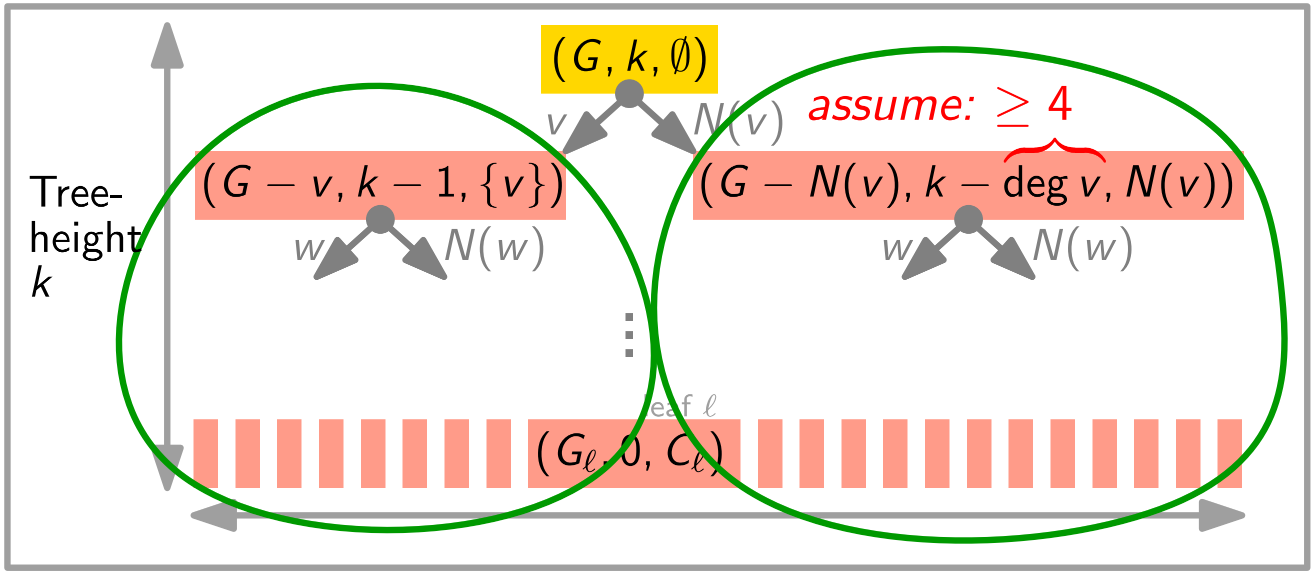
solve $T(k) = z^k - 1 \quad \Rightarrow \quad z^k = z^{k-4} + z^{k-1}$ $\quad \cdot \frac{1}{z^{k-4}}$

$\Rightarrow$ Characteristic polynomial: $z^4 = 1 + z^3$

$\Rightarrow$ largest positive solution: $z \approx 1.38 \quad$ (*branching value*)

Degree-4 Algorithm

Idea. Better analysis based on $|N(v)|$.



$\Rightarrow$ $T(k) = T(k - \boxed{4}) + T(k - \boxed{1}) + 1$, $T(\leq 4) = const.$

branching vector $(4, 1)$

solve $T(k) = z^k - 1$ $\Rightarrow$ $z^k = z^{k-4} + z^{k-1}$ $\quad \cdot \frac{1}{z^{k-4}}$

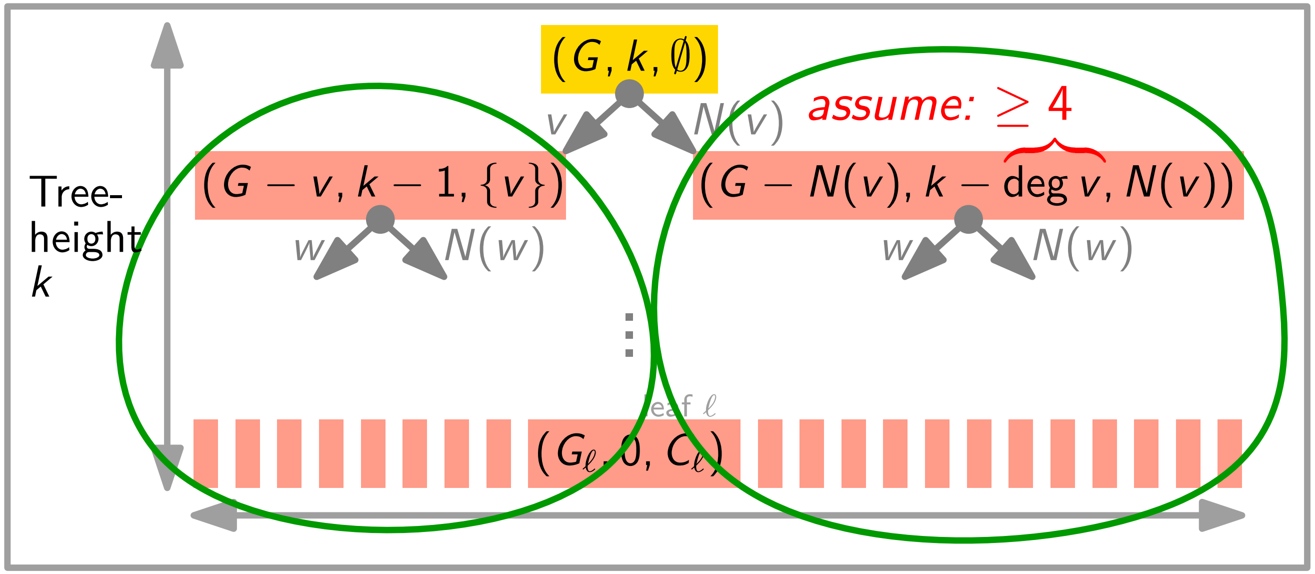
$\Rightarrow$ Characteristic polynomial: $z^4 = 1 + z^3$

$\Rightarrow$ largest positive solution: $z \approx 1.38$ (*branching value*)

$\Rightarrow$ $T(k) \in O(1.38^k)$.

Degree-4 Algorithm

Idea. Better analysis based on $|N(v)|$.



$\Rightarrow \quad T(k) = T(k - \boxed{4}) + T(k - \boxed{1}) + 1, \quad T(\leq 4) = const.$

branching vector $(4, 1)$

solve $T(k) = z^k - 1 \quad \Rightarrow \quad z^k = z^{k-4} + z^{k-1} \quad \cdot \frac{1}{z^{k-4}}$

$\Rightarrow$ Characteristic polynomial: $z^4 = 1 + z^3$

$\Rightarrow$ largest positive solution: $z \approx 1.38$ (*branching value*)

$\Rightarrow \quad T(k) \in O(1.38^k)$. How can we ensure $\deg v \geq 4$?

Kernel Construction II

Previous version:

Rule K: Reduce vertices of degree $> k$

Kernel Construction II

Previous version:

Rule K: Reduce vertices of degree $> k$
Rule 0: Delete isolated (degree 0) vertices

Kernel Construction II

Previous version:

Rule K: Reduce vertices of degree $> k$

Rule 0: Delete isolated (degree 0) vertices

New rules:

Rule 1:

Rule 2:

Kernel Construction II

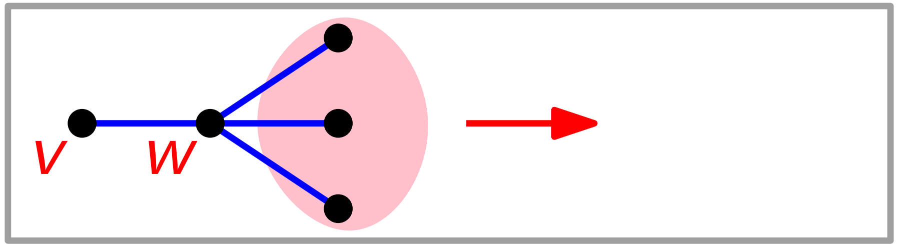
Previous version:

Rule K: Reduce vertices of degree $> k$

Rule 0: Delete isolated (degree 0) vertices

New rules:

Rule 1: Reduce degree 1 vertices



Rule 2:

Kernel Construction II

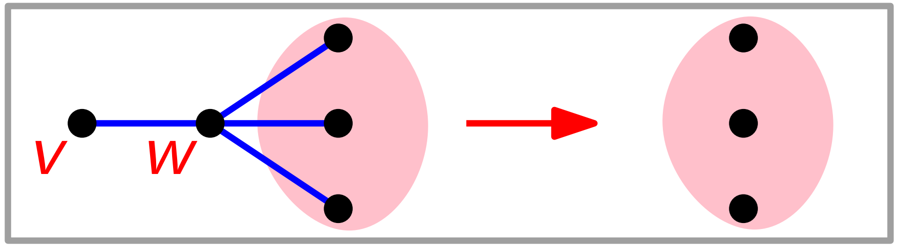
Previous version:

Rule K: Reduce vertices of degree $> k$

Rule 0: Delete isolated (degree 0) vertices

New rules:

Rule 1: Reduce degree 1 vertices



Rule 2:

Kernel Construction II

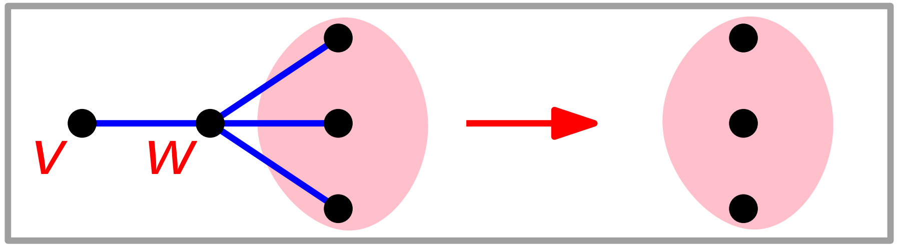
Previous version:

Rule K: Reduce vertices of degree $> k$

Rule 0: Delete isolated (degree 0) vertices

New rules:

Rule 1: Reduce degree 1 vertices



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

Rule 2:

Kernel Construction II

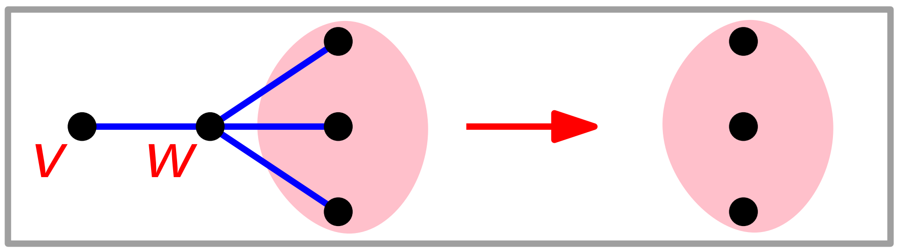
Previous version:

Rule K: Reduce vertices of degree $> k$

Rule 0: Delete isolated (degree 0) vertices

New rules:

Rule 1: Reduce degree 1 vertices



$C = C' \cup \{w\}$
$k' = k - 1$

Rule 2:

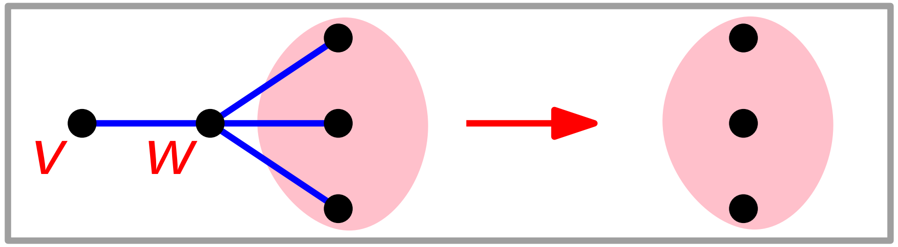
Kernel Construction II

Previous version:

Rule K: Reduce vertices of degree $> k$
Rule 0: Delete isolated (degree 0) vertices

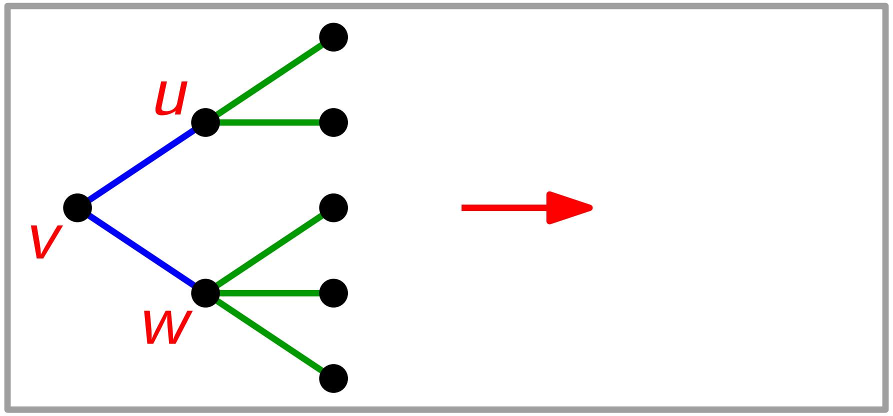
New rules:

Rule 1: Reduce degree 1 vertices



$C = C' \cup \{w\}$
$k' = k - 1$

Rule 2: Reduce degree 2 vertices



Kernel Construction II

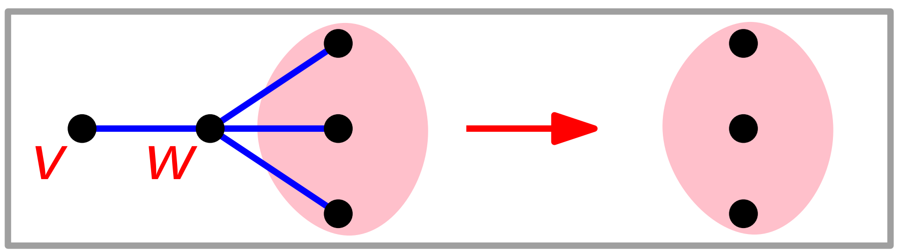
Previous version:

Rule K: Reduce vertices of degree $> k$

Rule 0: Delete isolated (degree 0) vertices

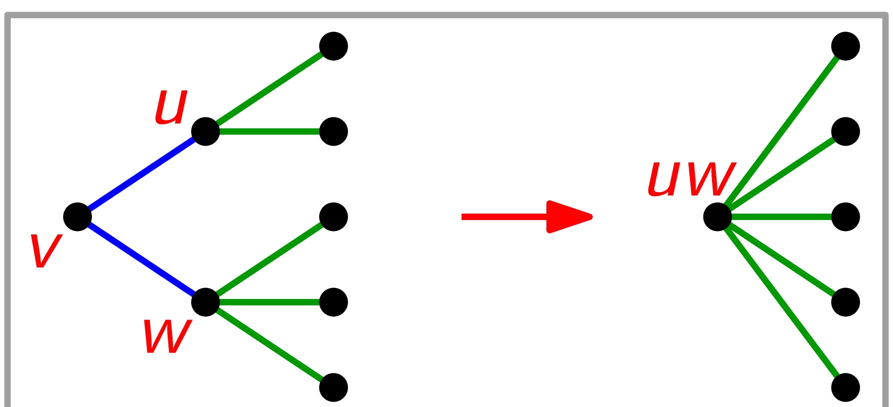
New rules:

Rule 1: Reduce degree 1 vertices



$C = C' \cup \{w\}$
$k' = k - 1$

Rule 2: Reduce degree 2 vertices



Kernel Construction II

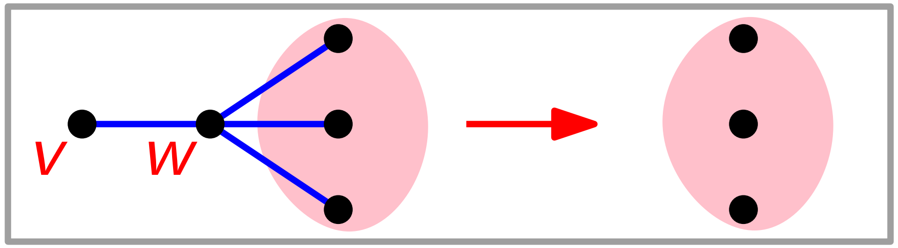
Previous version:

Rule K: Reduce vertices of degree $> k$

Rule 0: Delete isolated (degree 0) vertices

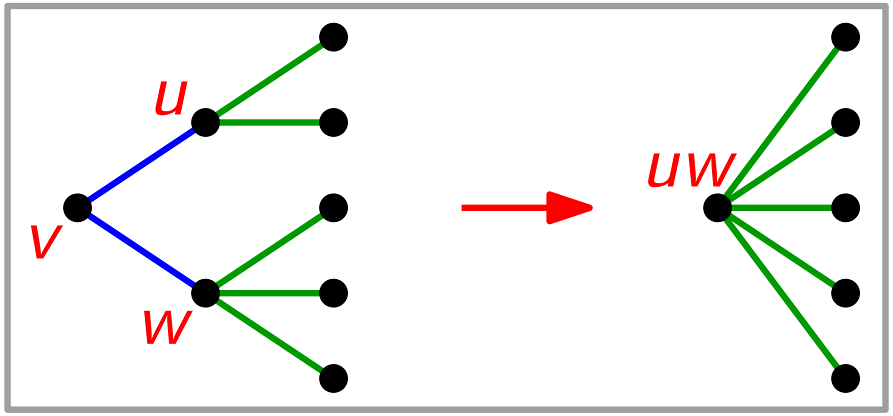
New rules:

Rule 1: Reduce degree 1 vertices



$C = C' \cup \{w\}$
$k' = k - 1$

Rule 2: Reduce degree 2 vertices



when uw is an edge, put u and w in C, otherwise $v \in C$, and merge u, w.

Kernel Construction II

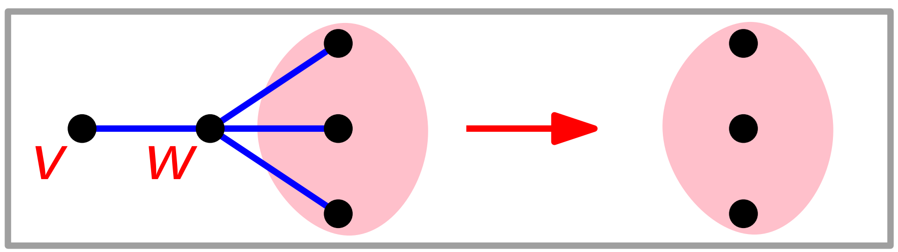
Previous version:

Rule K: Reduce vertices of degree $> k$

Rule 0: Delete isolated (degree 0) vertices

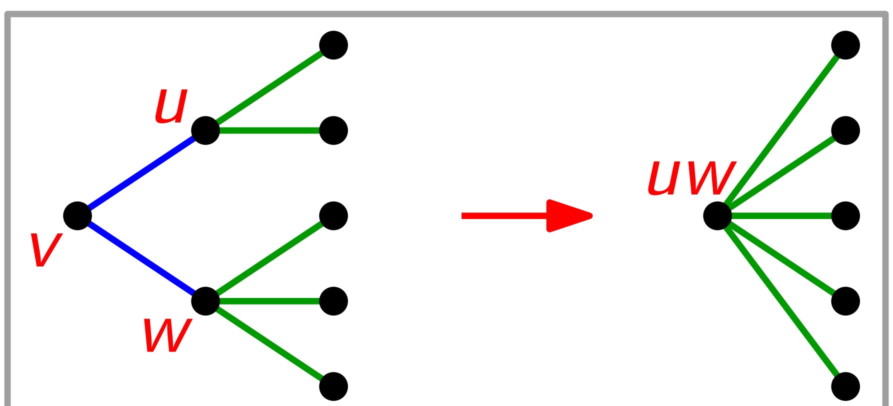
New rules:

Rule 1: Reduce degree 1 vertices



$C = C' \cup \{w\}$
$k' = k - 1$

Rule 2: Reduce degree 2 vertices



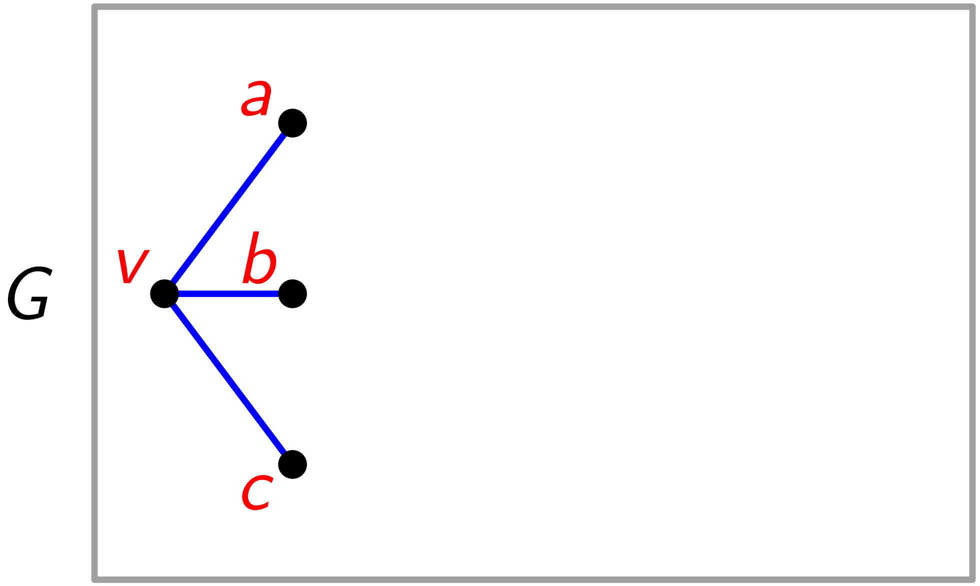
$k' = k - 2$

when uw is an edge, put u and w in C, otherwise $v \in C$, and merge u, w.

$k' = k - 1$

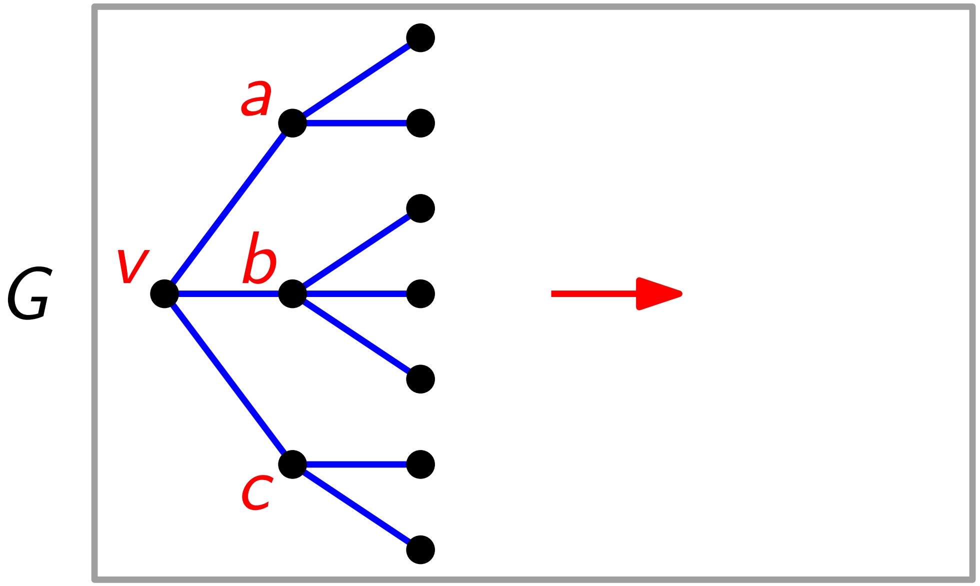
Rule 3: Reduce Degree-3 Vertices

Rule 3.1: $G[N(v)]$ has no edges



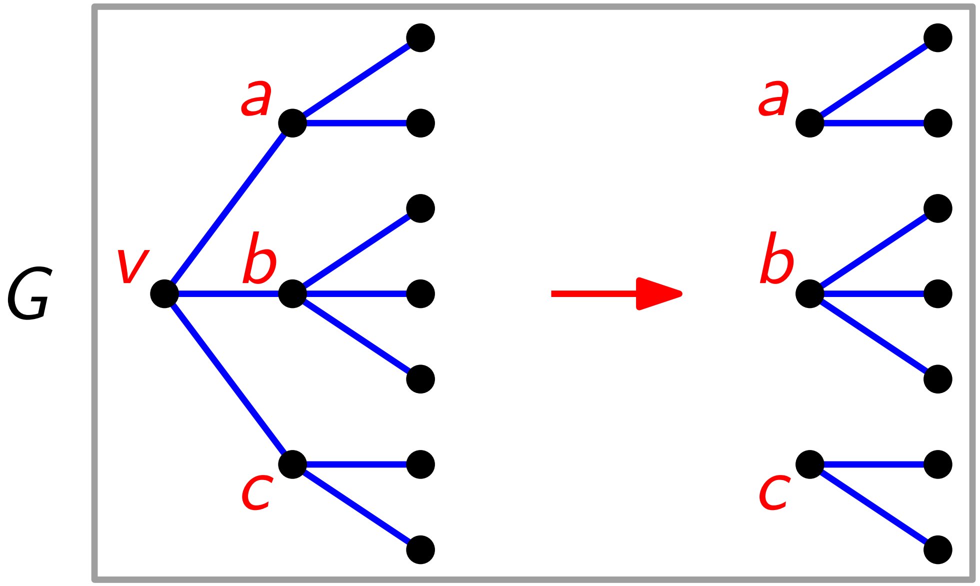
Rule 3: Reduce Degree-3 Vertices

Rule 3.1: $G[N(v)]$ has no edges



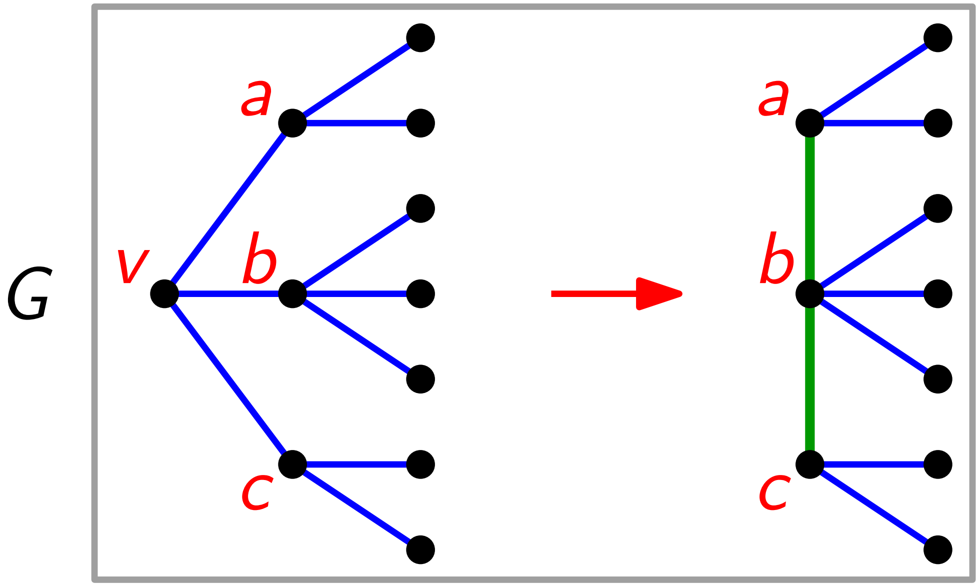
Rule 3: Reduce Degree-3 Vertices

Rule 3.1: $G[N(v)]$ has no edges



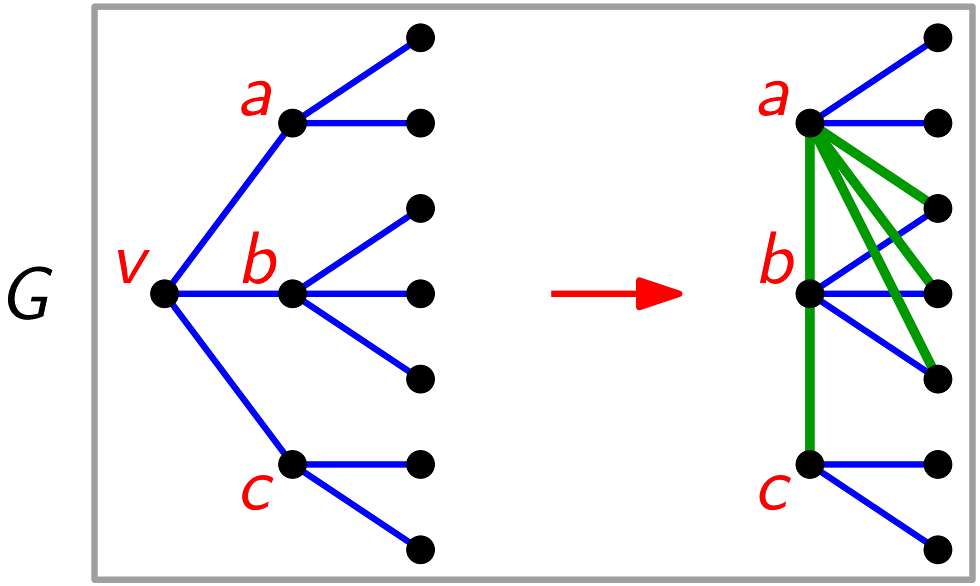
Rule 3: Reduce Degree-3 Vertices

Rule 3.1: $G[N(v)]$ has no edges



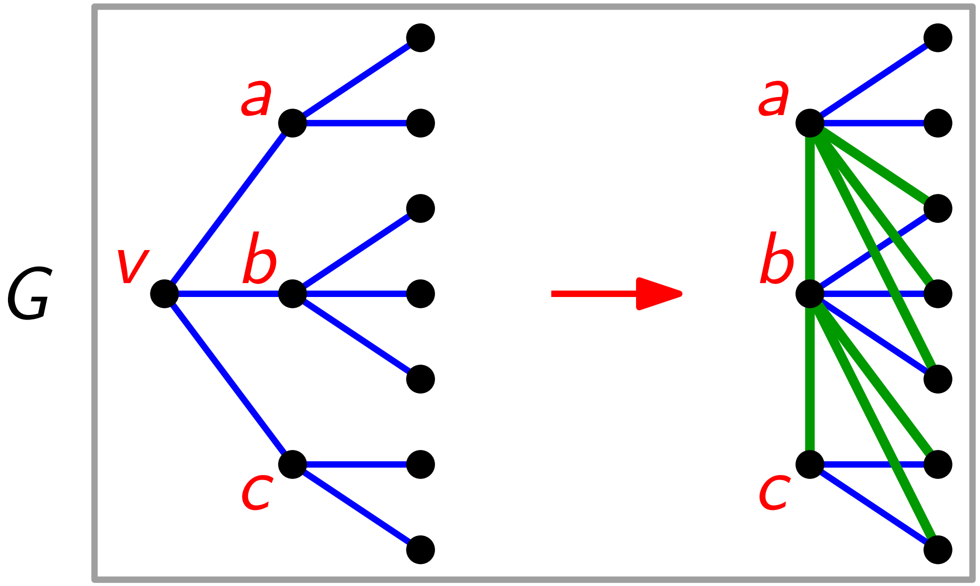
Rule 3: Reduce Degree-3 Vertices

Rule 3.1: $G[N(v)]$ has no edges



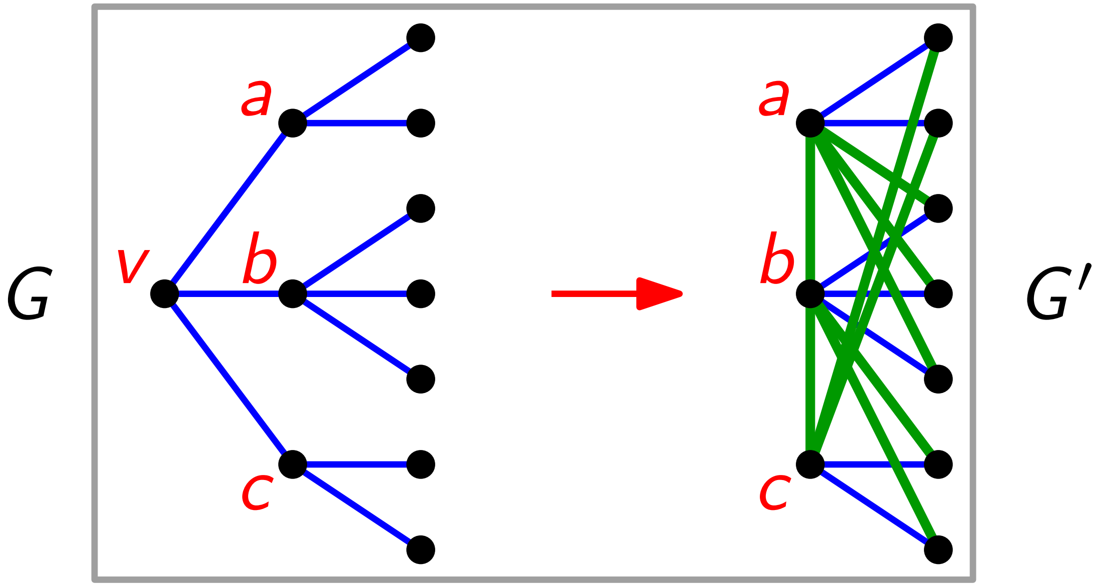
Rule 3: Reduce Degree-3 Vertices

Rule 3.1: $G[N(v)]$ has no edges



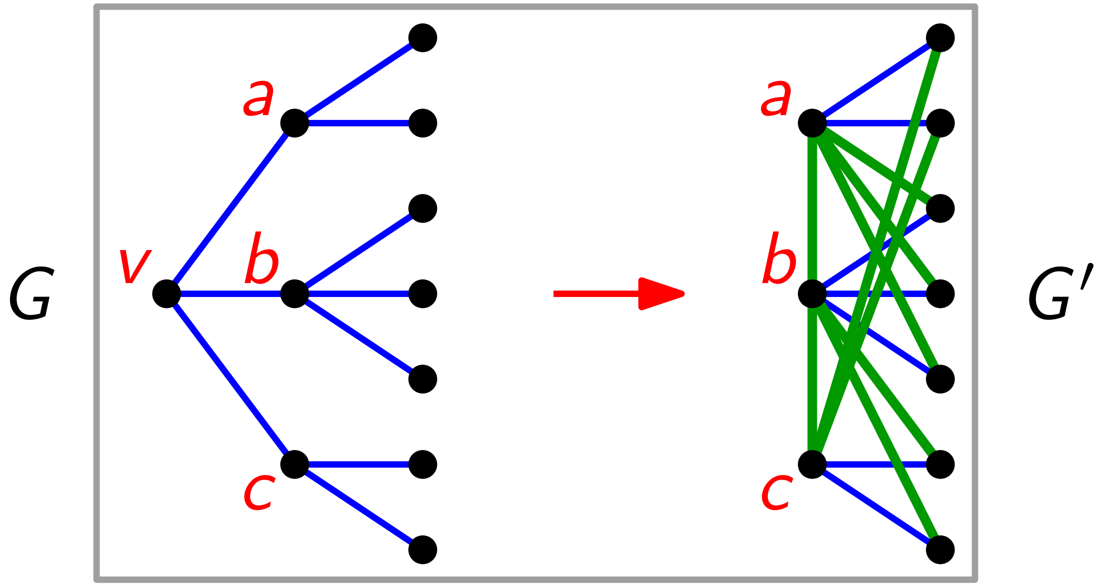
Rule 3: Reduce Degree-3 Vertices

Rule 3.1: $G[N(v)]$ has no edges



Rule 3: Reduce Degree-3 Vertices

Rule 3.1: $G[N(v)]$ has no edges

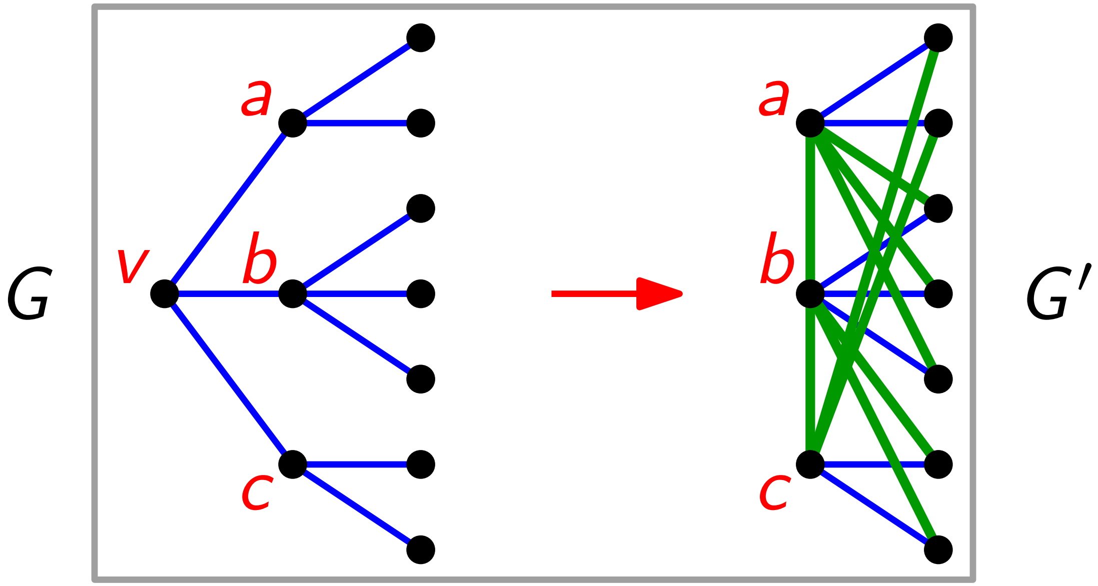


Note. k-VC in $G \Leftrightarrow k$-VC in G'.

[proof omitted]

Rule 3: Reduce Degree-3 Vertices

Rule 3.1: $G[N(v)]$ has no edges



Note. k-VC in G $\Leftrightarrow$ k-VC in G'.

[proof omitted]

Rule 3.2: $G[N(v)]$ contains an edge

$\dots$

Degree-4 Algorithm

Idea: Apply the improved kernelization approach at each node of the search tree.

Degree-4 Algorithm

Idea: Apply the improved kernelization approach at each node of the search tree.

$\Rightarrow$ **Runtime**:

Degree-4 Algorithm

Idea: Apply the improved kernelization approach at each node of the search tree.

$\Rightarrow$ **Runtime**: $O(\boxed{} + \boxed{} \cdot 1.38^k)$

Preprocessing   Kernelization in each node

Degree-4 Algorithm

Idea: Apply the improved kernelization approach at each node of the search tree.

$\Rightarrow$ **Runtime**: $O(\underbrace{nk}_{\text{Preprocessing}} + \underbrace{k^2}_{\text{Kernelization in each node}} \cdot 1.38^k)$

Degree-4 Algorithm

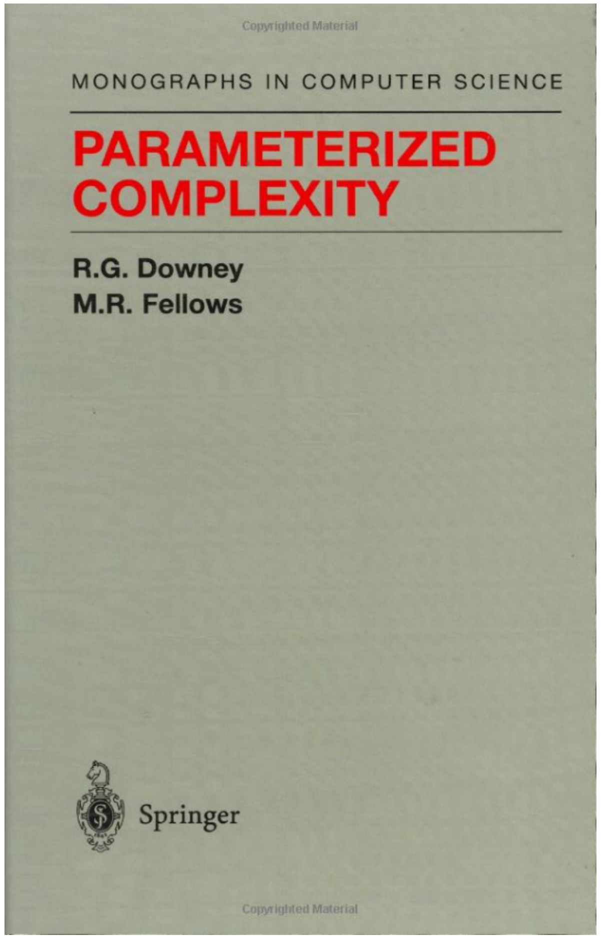
Idea: Apply the improved kernelization approach at each node of the search tree.

$\Rightarrow$ **Runtime**: $O(\underbrace{nk}_{\text{Preprocessing}} + \underbrace{k^2}_{\text{Kernelization in each node}} \cdot 1.38^k) \subseteq O^\star(1.38^k)$

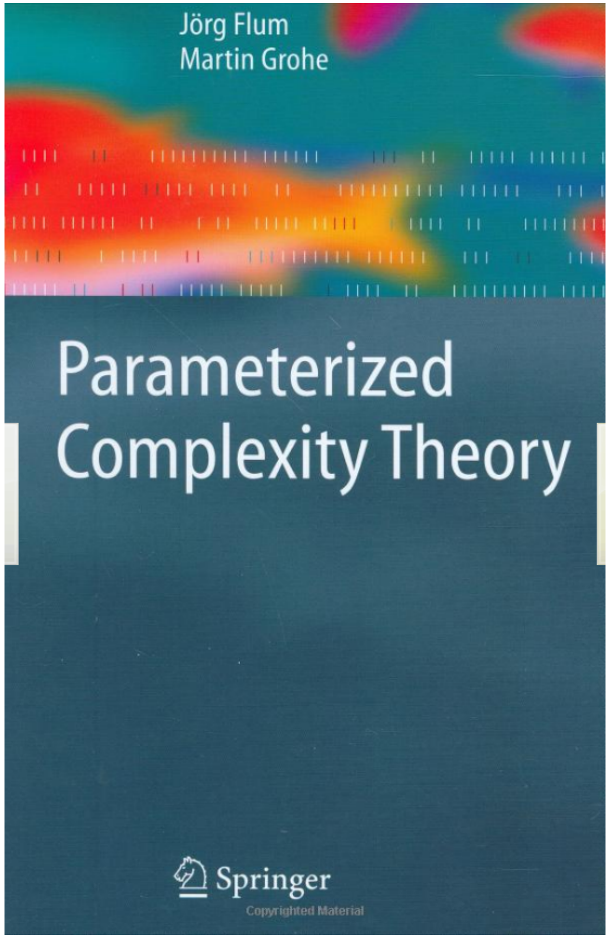
Summary

- k-VC can be solved in $O(nk + 1.38^k k^2)$ time.
- parameterized complexity =
  new approach to hard problems: kernelization, search trees,
  ...
- always a good idea look for parameterized analysis
  as in FPT !
- Ideally:
  "natural" problem $P \in \mathcal{FPT} \Rightarrow$ reasonable $f(k)$.

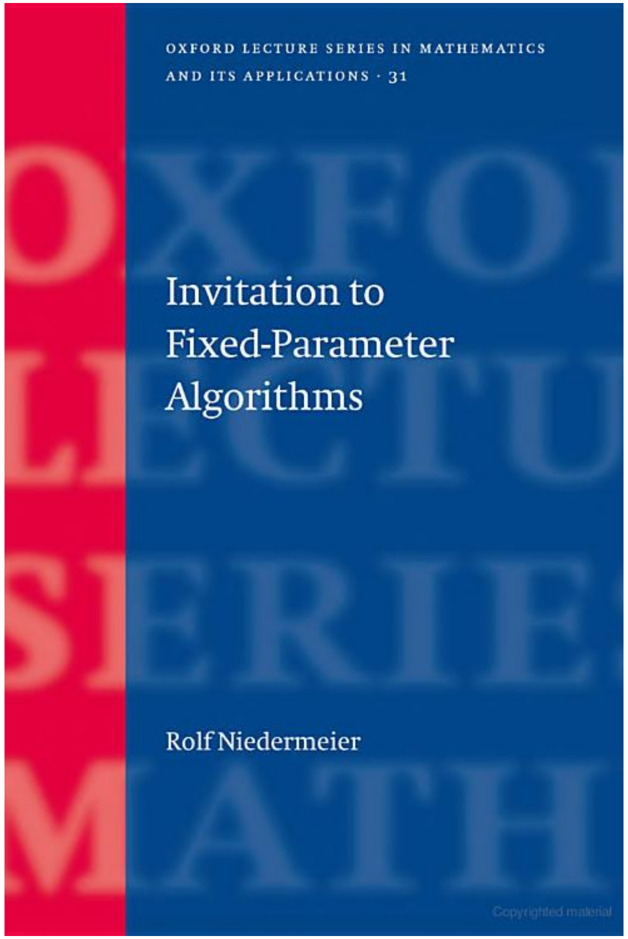
Books on the Topic



1999



2006

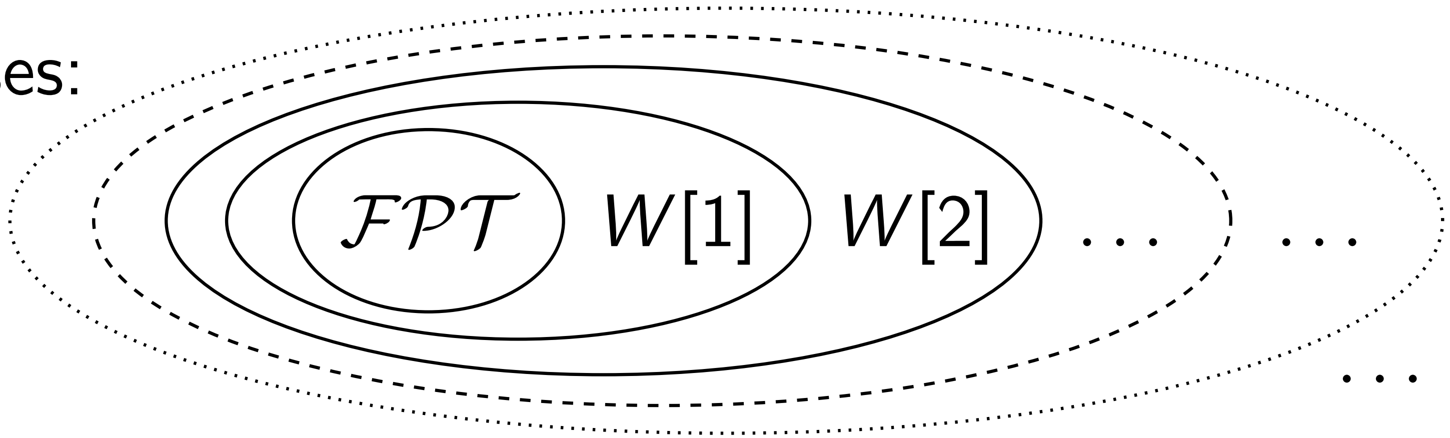


2006

Also, the textbook we are using:
Parameterized Algorithms

Computational Complexity

- FPT-reduction
- Decision circuits: weft and depth
- Problem Classes:



- Example $W[1]$-complete problems
  - k-IndependentSet
  - k-Clique
- Example of a $W[2]$-complete problem:
  - k-DominatingSet

Exercise: Show that these problems are in W[1]/W[2]