Computing Maximal Independent Sets on a PRAM

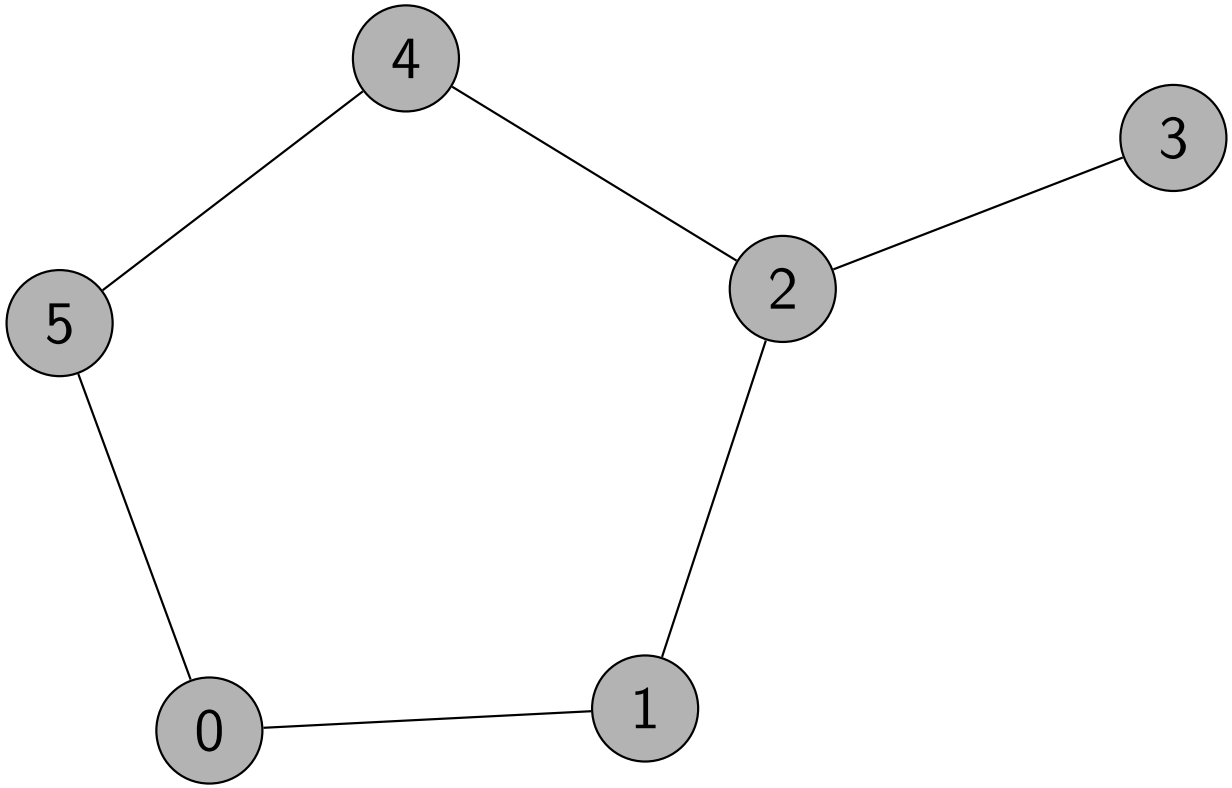
J. Schmidt-Dominé

RWTH Aachen

Proseminar Randomisierte Algorithmen 2011

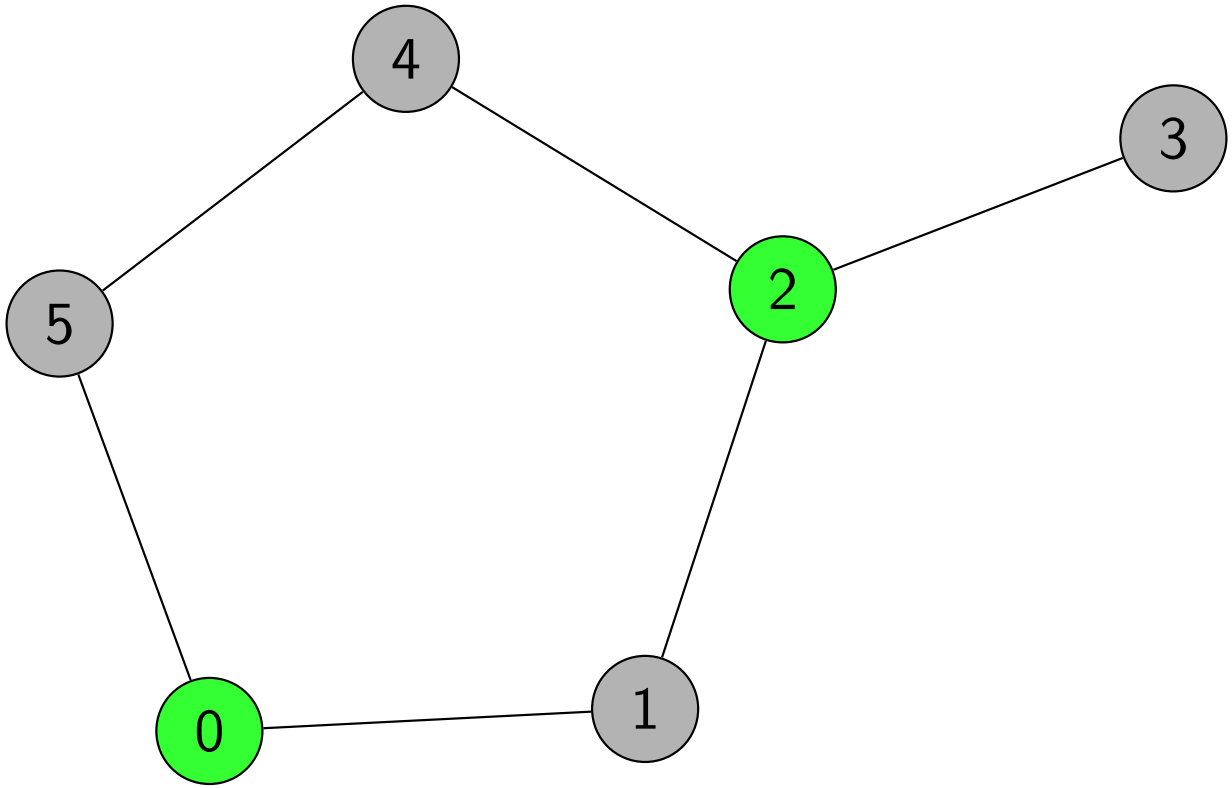
Maximal Independent Sets

Beispiel



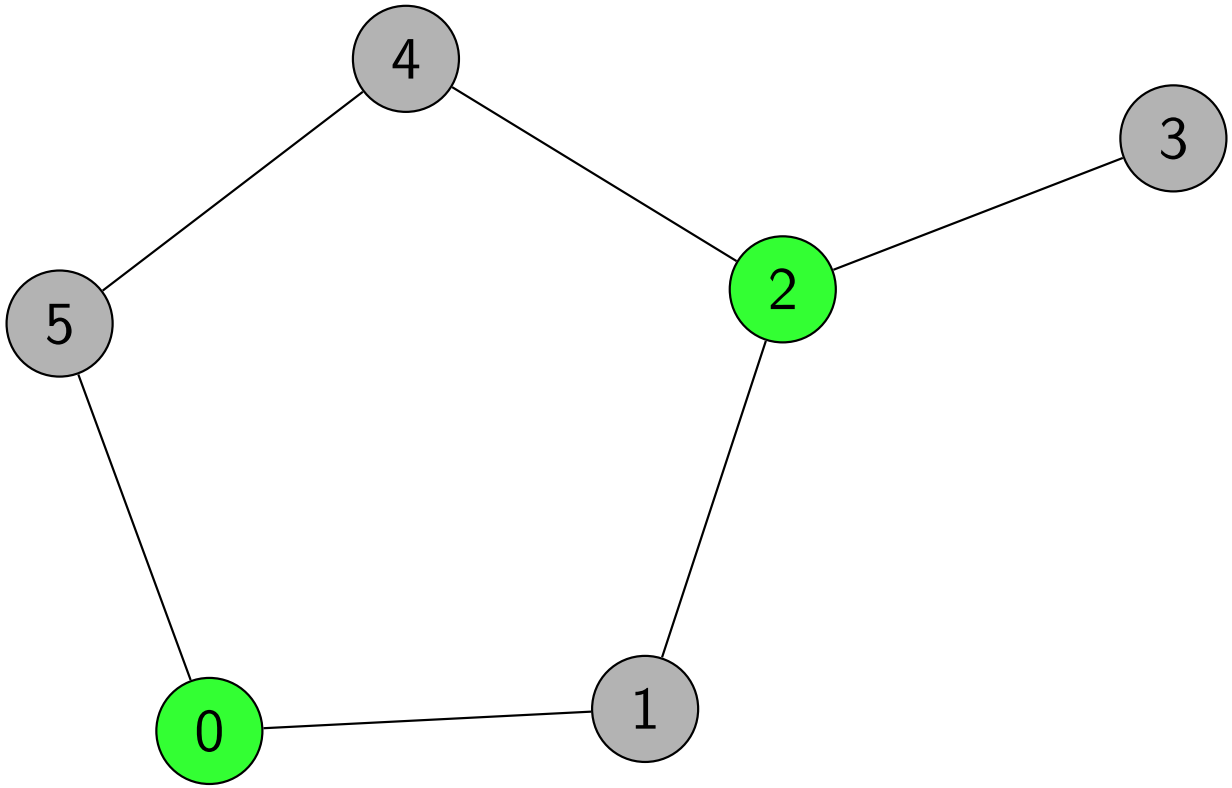
Maximal Independent Sets

Maximal Independent Set



Maximal Independent Sets

Maximal Independent Set



→ Sowohl sequentiell als auch parallel effizient berechenbar

- Maschinenmodell mit mehreren Prozessen, die auf einen Speicher zugreifen können

PRAM

Grundidee

- Maschinenmodell mit mehreren Prozessen, die auf einen Speicher zugreifen können
- Praktische Probleme der Synchronisation ausgeklammert

PRAM

High-Level-Ansatz: parallelisierte Schleifen

1: **for all** $i \in M$ **do in parallel**
2: something
3: **end for**

1: **for all** $i \in M$ **do in parallel**
2: something
3: **end for**

$\rightarrow$ Bei maximaler Parallelisierung nimmt sich jeder der $|M|$ Prozesse ein Element vor.

- Registermaschine: Realen Maschinen nachempfunden: Register, arithmetische Operationen, Jump-Anweisungen, RAM

PRAM

Low-Level-Ansatz: Registermaschine

- Registermaschine: Realen Maschinen nachempfunden: Register, arithmetische Operationen, Jump-Anweisungen, RAM
- PRAM: Register für jeden Prozess separat, parallele Ausführung, evtl. unterschiedlicher Code/unterschiedliche Parameter

PRAM

Low-Level-Ansatz: Registermaschine

- Registermaschine: Realen Maschinen nachempfunden: Register, arithmetische Operationen, Jump-Anweisungen, RAM
- PRAM: Register für jeden Prozess separat, parallele Ausführung, evtl. unterschiedlicher Code/unterschiedliche Parameter
- Der Speicher wird allerdings von allen geteilt (lokale Segmente erlaubt)s

Definition

EREW (*Exclusive Read, Exclusive Write*): Keine zwei Prozesse dürfen zugleich auf dieselbe Speicherzelle zugreifen

Definition

EREW (*Exclusive Read, Exclusive Write*): Keine zwei Prozesse dürfen zugleich auf dieselbe Speicherzelle zugreifen

Definition

CRCW (*Concurrent Read, Concurrent Write*): Prozesse dürfen gleichzeitig auf dieselbe Speicherzelle zugreifen, Schreibzugriffe müssen aber konsistent sein

Definition

EREW (*Exclusive Read, Exclusive Write*): Keine zwei Prozesse dürfen zugleich auf dieselbe Speicherzelle zugreifen

Definition

CRCW (*Concurrent Read, Concurrent Write*): Prozesse dürfen gleichzeitig auf dieselbe Speicherzelle zugreifen, Schreibzugriffe müssen aber konsistent sein

$\rightarrow$ Unterscheidung hat Auswirkungen auf die Komplexität (siehe Beispiel auf den folgenden Folien)

Erstes gesetztes Bit mit einem EREW finden

- Man wählt „turnierartig“ das kleinste aus Zweier-, Vierer-, usw. Gruppen aus.

Erstes gesetztes Bit mit einem EREW finden

- Man wählt „turnierartig“ das kleinste aus Zweier-, Vierer-, usw. Gruppen aus.
- Laufzeit $\mathcal{O}(\log n)$ mit $\mathcal{O}(n)$ Prozessoren

Erstes gesetztes Bit mit einem CRCW finden

- Für jedes Paar von Bits das größere als nicht-minimal markieren, wenn das kleinere gesetzt ist.

Erstes gesetztes Bit mit einem CRCW finden

- Für jedes Paar von Bits das größere als nicht-minimal markieren, wenn das kleinere gesetzt ist.
- Das einzige nicht als nicht-minimal markierte Bit auswählen.

Erstes gesetztes Bit mit einem CRCW finden

- Für jedes Paar von Bits das größere als nicht-minimal markieren, wenn das kleinere gesetzt ist.
- Das einzige nicht als nicht-minimal markierte Bit auswählen.
- Laufzeit $\mathcal{O}(1)$ mit $\mathcal{O}(n^2)$ Prozessoren

Erstes gesetztes Bit mit einem CRCW finden

- Für jedes Paar von Bits das größere als nicht-minimal markieren, wenn das kleinere gesetzt ist.
- Das einzige nicht als nicht-minimal markierte Bit auswählen.
- Laufzeit $\mathcal{O}(1)$ mit $\mathcal{O}(n^2)$ Prozessoren
- Auch mit $\mathcal{O}(n)$ Prozessoren möglich

Hilfsfunktionen

Definition

Nachbarschaftsfunktion:

$$NH(x) := \{y \in V | \{x, y\} \in E\}$$

Hilfsfunktionen

Definition

Nachbarschaftsfunktion:

$$NH(x) := \{y \in V | \{x, y\} \in E\}$$

Definition

Knotengrad:

$$d(x) := |\{\{x, y\} \in E\}|$$

Sequentieller Greedy-Algorithmus

1: **procedure** GreedyMIS(V, E)
2: $R \leftarrow \emptyset$
3: **while** $V \neq \emptyset$ **do**
4: $x \leftarrow chooseNextFrom(V)$
5: $V \leftarrow V \setminus (\{x\} \cup NH(x))$
6: $R \leftarrow R \cup \{x\}$
7: **end while**
8: **return** R
9: **end procedure**

Eigenschaften

- Laufzeit:

$$\mathcal{O}(|E|)$$

Eigenschaften

- Laufzeit:

$$\mathcal{O}(|E|)$$

- Einzelner Schleifendurchlauf parallelisierbar (Nachbarn entfernen)

Eigenschaften

- Laufzeit:

$$\mathcal{O}(|E|)$$

- Einzelner Schleifendurchlauf parallelisierbar (Nachbarn entfernen)
- Schleife nicht parallelisierbar

Eigenschaften

- Laufzeit:

$$\mathcal{O}(|E|)$$

- Einzelner Schleifendurchlauf parallelisierbar (Nachbarn entfernen)
- Schleife nicht parallelisierbar
- Schleifendurchlauf hängt von vorherigen ab, wenn dort bereits ein Nachbar ausgewählt wurde

Eigenschaften

- Laufzeit:

$$\mathcal{O}(|E|)$$

- Einzelner Schleifendurchlauf parallelisierbar (Nachbarn entfernen)
- Schleife nicht parallelisierbar
- Schleifendurchlauf hängt von vorherigen ab, wenn dort bereits ein Nachbar ausgewählt wurde
- Das Hinzufügen einer Menge *unabhängiger* Knoten dagegen ist parallel möglich (hier setzt die Parallelisierung an)

Finden eines IS

Waisen ($d(v) = 0$) zum IS hinzufügen

for all $v \in V$ **do in parallel**
    if $d(v) = 0$ **then**
        $R \leftarrow R \cup \{v\}$
        $V \leftarrow V \setminus \{v\}$
    end if
end for

Finden eines IS

Waisen ($d(v) = 0$) zum IS hinzufügen

for all $v \in V$ **do in parallel**
  if $d(v) = 0$ **then**
    $R \leftarrow R \cup \{v\}$
    $V \leftarrow V \setminus \{v\}$
  end if
end for

$\rightarrow$ Nichts weiter notwendig für Waisen:

$$\mathcal{O}(1)$$

(mit $|V|$ Prozessen) $\rightarrow$ Sie werden in der Analyse vernachlässigt, da sie den Algorithmus nur beschleunigen können

Finden eines IS

Auswahl, nachdem Waisen entfernt sind

$S \leftarrow \emptyset$
for all $v \in V$ **do in parallel**
  with probability $\frac{1}{2 \cdot d(v)}$ **do**
    $S \leftarrow S \cup \{v\}$
  end do
end for

Finden eines IS

Auswahl, nachdem Waisen entfernt sind

$S \leftarrow \emptyset$
for all $v \in V$ **do in parallel**
  with probability $\frac{1}{2 \cdot d(v)}$ **do**
    $S \leftarrow S \cup \{v\}$
  end do
end for

$\rightarrow$ Mit $|V|$ Prozessen:

$$\mathcal{O}(1)$$

Finden eines IS

Auswahl, nachdem Waisen entfernt sind

$S \leftarrow \emptyset$
for all $v \in V$ **do in parallel**
  with probability $\frac{1}{2 \cdot d(v)}$ **do**
    $S \leftarrow S \cup \{v\}$
  end do
end for

$\rightarrow$ Mit $|V|$ Prozessen:

$$\mathcal{O}(1)$$

$\rightarrow$ Anschließend müssen adjazente Knoten wieder teilweise entfernt werden.

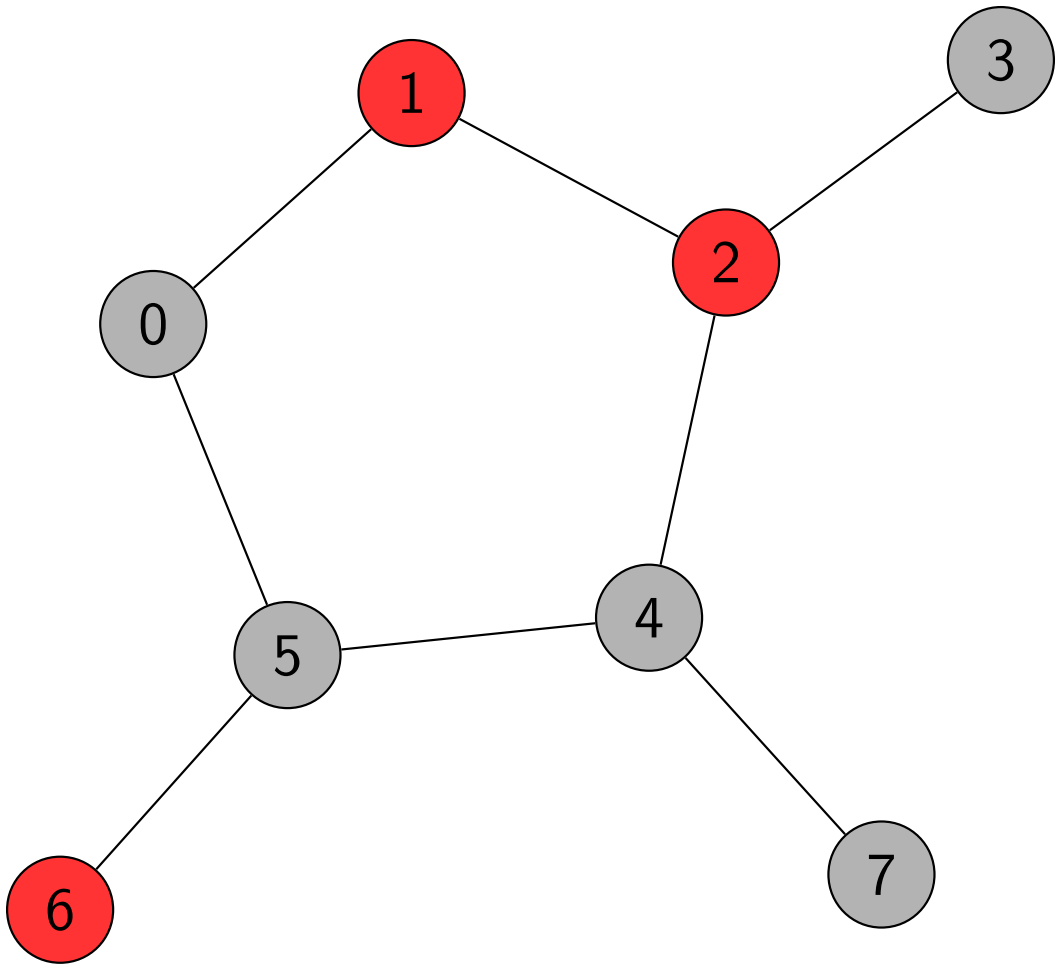
Finden eines IS

Entfernen der Konflikte

for all $(u, v) \in E$ **do in parallel**
  if $u \in S \wedge v \in S$ **then**
    ▷ Höherer Grad wird behalten $\Rightarrow$ Knoten schneller entfernt
    if $d(u) < d(v)$ **then**
      $S \leftarrow S \setminus \{u\}$
    else if $d(v) < d(u)$ **then**
      $S \leftarrow S \setminus \{v\}$
    else
      if $u < v$ **then**
        $S \leftarrow S \setminus \{u\}$
      else
        $S \leftarrow S \setminus \{v\}$
      end if
    end if
  end if
end for

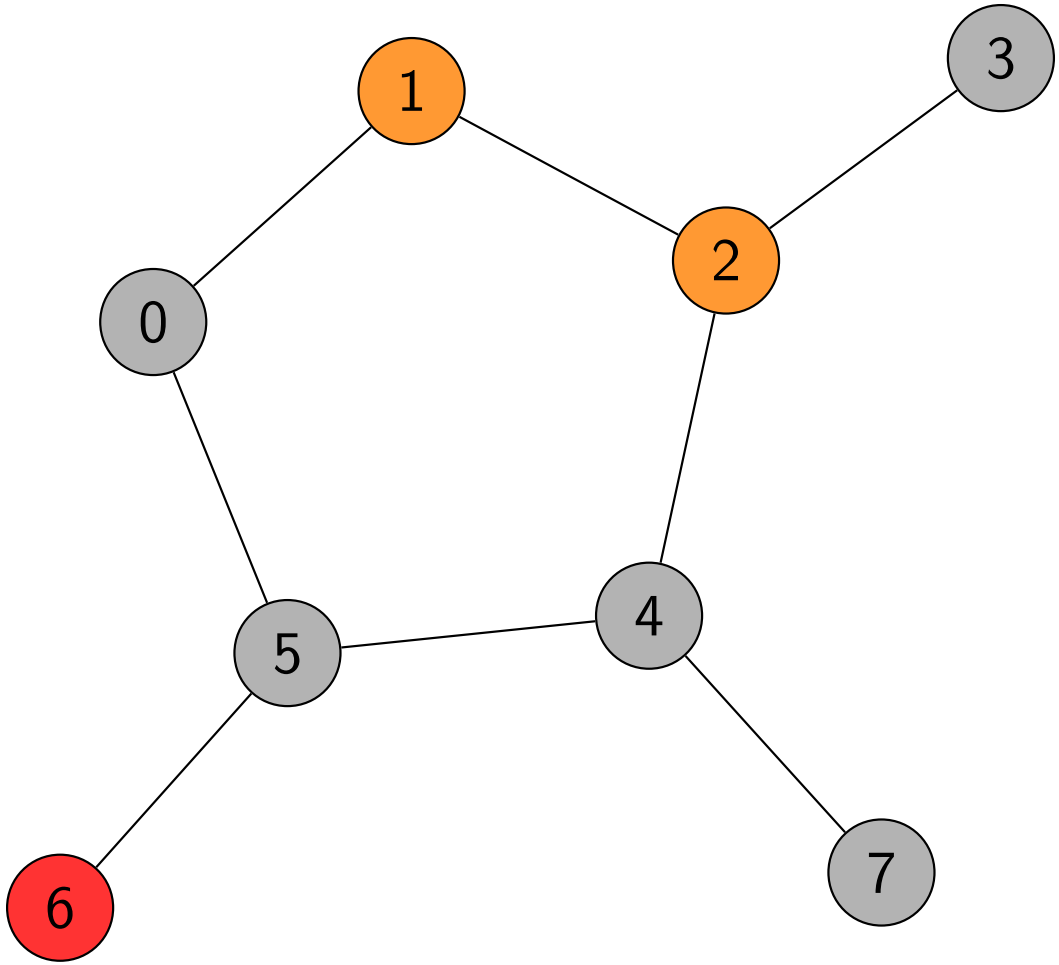
Beispiel-Ablauf des Algorithmus

Zufällige Auswahl



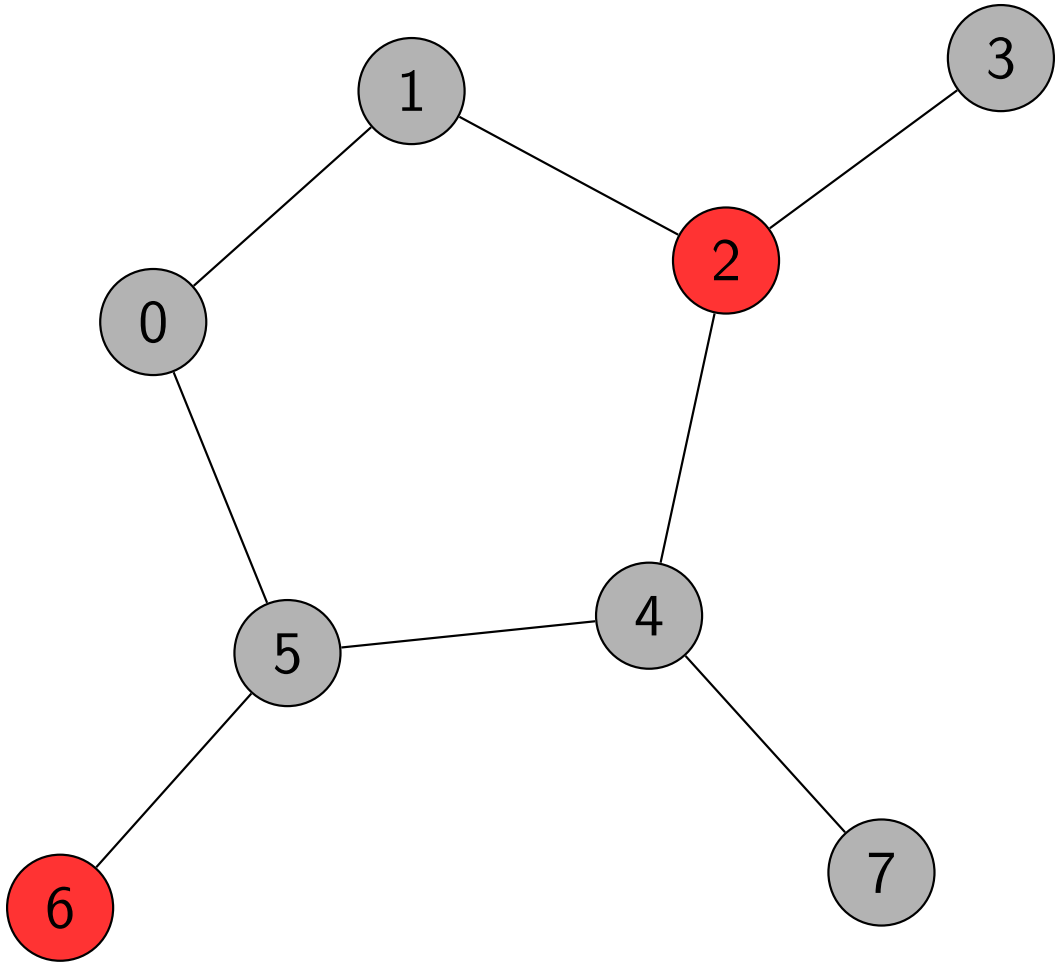
Beispiel-Ablauf des Algorithmus

Konfliktbehebung



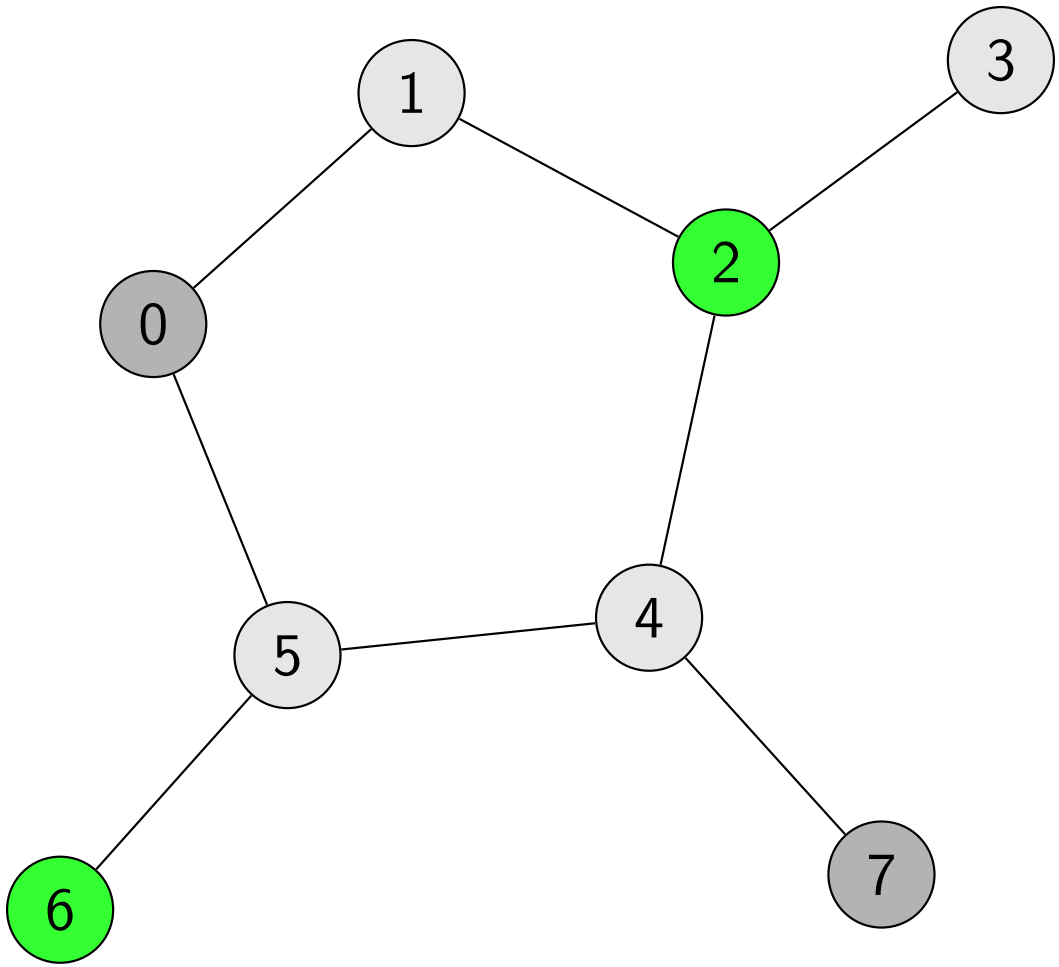
Beispiel-Ablauf des Algorithmus

Konfliktbehebung



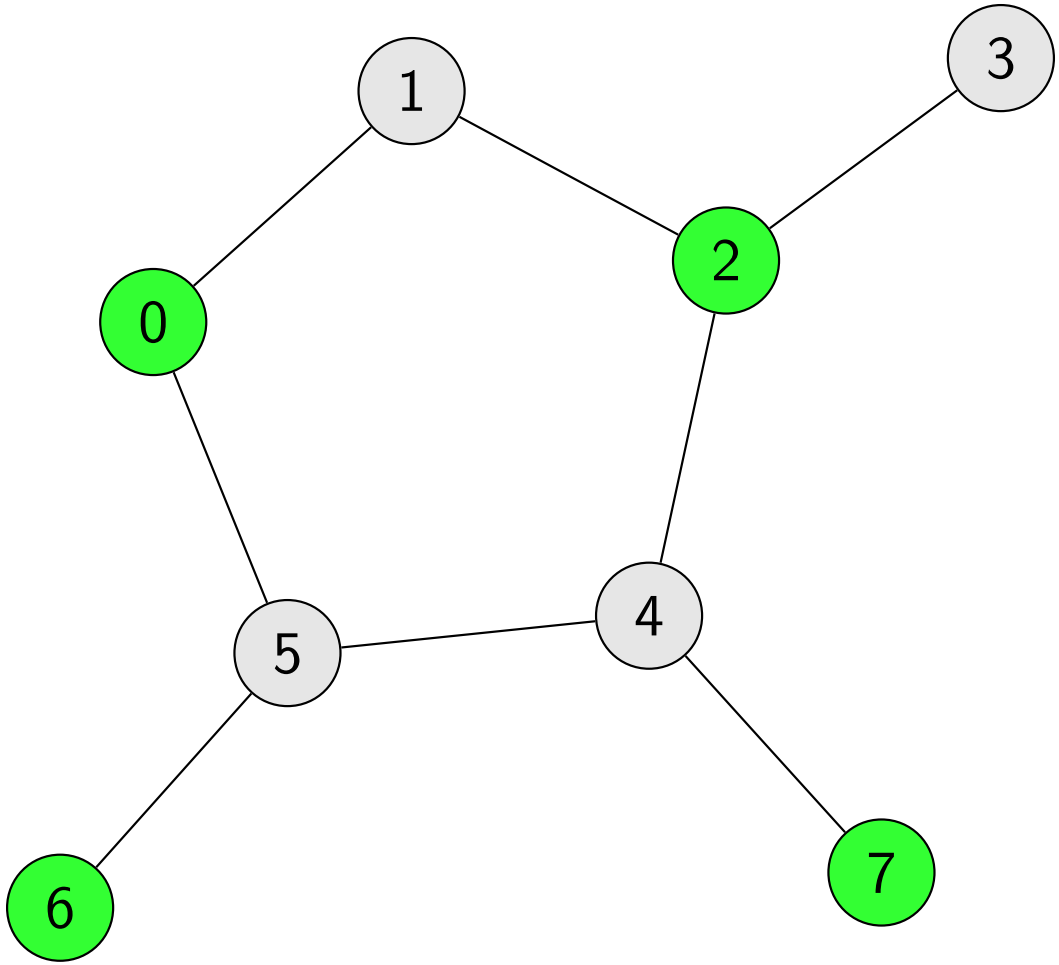
Beispiel-Ablauf des Algorithmus

Auswahl abschließen und Nachbarschaft entfernen



Beispiel-Ablauf des Algorithmus

Nächster Durchgang: nur noch Waisen



Definition

Ein Knoten v heißt **gut**, genau dann wenn er mindestens $\frac{d(v)}{3}$ adjazente Knoten mit einem Grad kleiner oder gleich $d(v)$ hat, andernfalls heißt er **böse**.

Definition

Ein Knoten v heißt **gut**, genau dann wenn er mindestens $\frac{d(v)}{3}$ adjazente Knoten mit einem Grad kleiner oder gleich $d(v)$ hat, andernfalls heißt er **böse**.

Definition

Eine Kante heißt **gut**, genau dann wenn sie mindestens einen guten inzidenten Knoten hat.

Lemma

Die Auswahl eines Knoten v wird mit $P \geq \frac{1}{2}$ nicht wieder aufgehoben.

Lemma

Die Auswahl eines Knoten v wird mit $P \geq \frac{1}{2}$ nicht wieder aufgehoben.

Beweis

Die Auswahl kann nur entfernt werden, wenn ein adjazenter Knoten w mit $d(w) \geq d(v)$ ausgewählt ist. Ein solcher ist mit Wahrscheinlichkeit $Q = \frac{1}{2 \cdot d(w)} \leq \frac{1}{2 \cdot d(v)}$ ausgewählt, womit $P \geq 1 - Q \cdot d(v) \geq \frac{1}{2}$.

Lemma

Zu einem guten Knoten v wird mit $P \geq 1 - e^{-\frac{1}{6}}$ ein Knoten $w \in NH(v)$ ausgewählt.

Lemma

Zu einem guten Knoten v wird mit $P \geq 1 - e^{-\frac{1}{6}}$ ein Knoten $w \in NH(v)$ ausgewählt.

Beweis

Mindestens $\frac{d(v)}{3}$ Knoten aus der Nachbarschaft haben einen Grad $d(w) \leq d(v)$. Die Wahrscheinlichkeit, dass keiner von ihnen ausgewählt wird, ist aufgrund der Unabhängigkeit:

$$Q \leq \left(1 - \frac{1}{2 \cdot d(v)}\right)^{\frac{d(v)}{3}} \leq e^{-\frac{1}{6}}$$

Definition

$d_{\leq}(v)$: Anzahl der adjazenten Knoten mit niedrigerem oder gleich großen Grad

$d_{>}(v)$: Anzahl der adjazenten Knoten mit größerem Grad

Definition

$d_{\leq}(v)$: Anzahl der adjazenten Knoten mit niedrigerem oder gleich großen Grad
$d_{>}(v)$: Anzahl der adjazenten Knoten mit größerem Grad

Definition

$e_{b/g,b/g}$: Anzahl der Kanten von einem bösen/guten Knoten zu einem bösen/guten Knoten mit höherem Grad.

Lemma

Für jeden bösen Knoten v gilt:

$$d_{>}(v) - d_{\leq}(v) \geq \frac{2}{3} \cdot d(v) - \frac{1}{3} \cdot d(v) = \frac{1}{3} \cdot d(v)$$

Lemma

Es gibt in einem Graphen stets weniger böse als gute Kanten.

Lemma

Es gibt in einem Graphen stets weniger böse als gute Kanten.

Beweis

$$2 \cdot e_{b,b} + e_{b,g} + e_{g,b} = \sum_{v \text{ böse}} d(v)$$

$$\leq 3 \cdot \sum_{v \text{ böse}} (d_{>}(v) - d_{\leq}(v)) = 3 \cdot (e_{b,g} - e_{g,b}) \leq 3 \cdot (e_{b,g} + e_{g,b})$$

$$\Rightarrow e_{b,b} \leq e_{b,g} + e_{g,b}$$

Laufzeitanalyse

Ergebnis

Da zumindest ein konstanter Anteil der Kanten je mit einer konstanten Wahrscheinlichkeit ausgewählt und dann entfernt wird, folgt:

Theorem

Der MIS-Algorithmus benötigt im Mittel $\mathcal{O}(\log n)$ Schleifendurchläufe.

Derandomisierung

Idee

- Braucht ein Algorithmus n Zufallsbits, so lässt er sich mit 2^n Prozessen deterministisch simulieren.

Derandomisierung

Idee

- Braucht ein Algorithmus n Zufallsbits, so lässt er sich mit 2^n Prozessen deterministisch simulieren.
- *n paarweise unabhängige* Zufallsbits lassen sich aus $\mathcal{O}(\log n)$ *unabhängige* Zufallsbits erzeugen.

Derandomisierung

Idee

- Braucht ein Algorithmus n Zufallsbits, so lässt er sich mit 2^n Prozessen deterministisch simulieren.
- *n paarweise unabhängige* Zufallsbits lassen sich aus $\mathcal{O}(\log n)$ *unabhängige* Zufallsbits erzeugen.
- Erzeugung von Zufallsbits über *XOR*-Kombination einer nicht-leeren Teilmenge.

Derandomisierung

Idee

- Braucht ein Algorithmus n Zufallsbits, so lässt er sich mit 2^n Prozessen deterministisch simulieren.
- *n paarweise unabhängige* Zufallsbits lassen sich aus $\mathcal{O}(\log n)$ *unabhängige* Zufallsbits erzeugen.
- Erzeugung von Zufallsbits über *XOR*-Kombination einer nicht-leeren Teilmenge.
- In jedem der $\log |V|$ Schritte werden $V \cdot \log |V|$ Zufallsbits benötigt.

Derandomisierung

Idee

- Braucht ein Algorithmus n Zufallsbits, so lässt er sich mit 2^n Prozessen deterministisch simulieren.
- *n paarweise unabhängige* Zufallsbits lassen sich aus $\mathcal{O}(\log n)$ *unabhängige* Zufallsbits erzeugen.
- Erzeugung von Zufallsbits über *XOR*-Kombination einer nicht-leeren Teilmenge.
- In jedem der $\log |V|$ Schritte werden $V \cdot \log |V|$ Zufallsbits benötigt.
- Ergibt linearen Faktor für die Anzahl der Prozesse.

Derandomisierung

Beweis, dass paarweise Unabhängigkeit genügt

- Zu zeigen: konstante untere Schranke für die Wahrscheinlichkeit P der Auswahl eines Knotens aus $NH(v)$ mit gutem v (Unabhängigkeit wurde bislang vorausgesetzt).

Derandomisierung

Beweis, dass paarweise Unabhängigkeit genügt

- Zu zeigen: konstante untere Schranke für die Wahrscheinlichkeit P der Auswahl eines Knotens aus $NH(v)$ mit gutem v (Unabhängigkeit wurde bislang vorausgesetzt).
- Sei p_u die Wahrscheinlichkeit, dass u ausgewählt wird und Q die Wahrscheinlichkeit, dass genau einer der Knoten aus $NH(v)$ ausgewählt wird.

Derandomisierung

Beweis, dass paarweise Unabhängigkeit genügt

- Zu zeigen: konstante untere Schranke für die Wahrscheinlichkeit P der Auswahl eines Knotens aus $NH(v)$ mit gutem v (Unabhängigkeit wurde bislang vorausgesetzt).
- Sei p_u die Wahrscheinlichkeit, dass u ausgewählt wird und Q die Wahrscheinlichkeit, dass genau einer der Knoten aus $NH(v)$ ausgewählt wird.
- Nun gilt:

$$P \geq Q \geq \sum_{u \in NH(v)} p_u - \sum_{u<w} p_u \cdot p_w \geq \sum_u p_u \cdot \left(1 - \frac{1}{2} \cdot \sum_w p_w\right)$$

$$\geq \frac{1}{2} \cdot \sum_u p_u \geq \frac{1}{2} \cdot \frac{d(v)}{3} \cdot \frac{1}{2 \cdot d(v)} = \frac{1}{12}, \text{ falls } \sum_u p_u \leq 1$$

Derandomisierung

Beweis, dass paarweise Unabhängigkeit genügt

- Zu zeigen: konstante untere Schranke für die Wahrscheinlichkeit P der Auswahl eines Knotens aus $NH(v)$ mit gutem v (Unabhängigkeit wurde bislang vorausgesetzt).
- Sei p_u die Wahrscheinlichkeit, dass u ausgewählt wird und Q die Wahrscheinlichkeit, dass genau einer der Knoten aus $NH(v)$ ausgewählt wird.
- Nun gilt:

$$P \geq Q \geq \sum_{u \in NH(v)} p_u - \sum_{u<w} p_u \cdot p_w \geq \sum_u p_u \cdot \left(1 - \frac{1}{2} \cdot \sum_w p_w\right)$$

$$\geq \frac{1}{2} \cdot \sum_u p_u \geq \frac{1}{2} \cdot \frac{d(v)}{3} \cdot \frac{1}{2 \cdot d(v)} = \frac{1}{12}, \text{ falls } \sum_u p_u \leq 1$$

- Andernfalls wähle man nur die mit dem geringsten Grad, sodass die Summe zwischen $\frac{1}{2}$ und 1 liegt, dies ergibt eine Schranke von $\frac{1}{4}$.

Laufzeit auf einem EREW

- Auf den ersten Blick lässt sich der Schleifenkörper perfekt parallelisieren.

Laufzeit auf einem EREW

- Auf den ersten Blick lässt sich der Schleifenkörper perfekt parallelisieren.
- Problem: Die Konfliktbehebung benötigt eine Synchronisierung, damit nicht mehrere Prozesse denselben Knoten entfernen.

Laufzeit auf einem EREW

- Auf den ersten Blick lässt sich der Schleifenkörper perfekt parallelisieren.
- Problem: Die Konfliktbehebung benötigt eine Synchronisierung, damit nicht mehrere Prozesse denselben Knoten entfernen.
- Dies führt zu einer Laufzeit von lediglich $\mathcal{O}(\log n)$.

Laufzeit auf einem EREW

- Auf den ersten Blick lässt sich der Schleifenkörper perfekt parallelisieren.
- Problem: Die Konfliktbehebung benötigt eine Synchronisierung, damit nicht mehrere Prozesse denselben Knoten entfernen.
- Dies führt zu einer Laufzeit von lediglich $\mathcal{O}(\log n)$.
- Möglich über „turnierartiges“ Propagieren der Entfernung eines Knotens.

Laufzeit auf einem CRCW

- Es stellt sich das Problem, den Grad eines Knotens zählen zu müssen.

Laufzeit auf einem CRCW

- Es stellt sich das Problem, den Grad eines Knotens zählen zu müssen.
- Möchte man einen *concurrent write* zum Aufheben der Auswahl erlauben ($\mathcal{O}(1)$), lässt sich dabei der Grad nicht berechnen.

Laufzeit auf einem CRCW

- Es stellt sich das Problem, den Grad eines Knotens zählen zu müssen.
- Möchte man einen *concurrent write* zum Aufheben der Auswahl erlauben ($\mathcal{O}(1)$), lässt sich dabei der Grad nicht berechnen.
- Idee: Mit ähnlicher Wahrscheinlichkeit auswählen, aber ohne Kenntnis des Grades.

Auswahl mit einem CRCW

- n Prozesse wählen jeweils einen zufälligen Knotenindex.

Auswahl mit einem CRCW

- n Prozesse wählen jeweils einen zufälligen Knotenindex.
- Wähle nun den ersten dieser Knoten (in der zufälligen Reihenfolge), der noch nicht entfernt ist.

Auswahl mit einem CRCW

- n Prozesse wählen jeweils einen zufälligen Knotenindex.
- Wähle nun den ersten dieser Knoten (in der zufälligen Reihenfolge), der noch nicht entfernt ist.
- Ist dieser gleich dem minimalen adjazenten Knoten, so findet die Auswahl statt.

Auswahl mit einem CRCW

- n Prozesse wählen jeweils einen zufälligen Knotenindex.
- Wähle nun den ersten dieser Knoten (in der zufälligen Reihenfolge), der noch nicht entfernt ist.
- Ist dieser gleich dem minimalen adjazenten Knoten, so findet die Auswahl statt.

Auswahl mit einem CRCW

- n Prozesse wählen jeweils einen zufälligen Knotenindex.
- Wähle nun den ersten dieser Knoten (in der zufälligen Reihenfolge), der noch nicht entfernt ist.
- Ist dieser gleich dem minimalen adjazenten Knoten, so findet die Auswahl statt.

$\rightarrow$ Auswahl findet mit Wahrscheinlichkeit $\frac{1}{2 \cdot d(v)} \leq P \leq \frac{1}{d(v)}$ statt.

Ergebnisse

- Deterministisch sequentiell lässt sich ein Maximal Independent Set in $\mathcal{O}(|E|)$ finden.

Ergebnisse

- Deterministisch sequentiell lässt sich ein Maximal Independent Set in $\mathcal{O}(|E|)$ finden.
- Deterministisch auf einem EREW mit $\mathcal{O}(|V|)$ Prozessoren in $\mathcal{O}(|V| \cdot \log |V|)$.

Ergebnisse

- Deterministisch sequentiell lässt sich ein Maximal Independent Set in $\mathcal{O}(|E|)$ finden.
- Deterministisch auf einem EREW mit $\mathcal{O}(|V|)$ Prozessoren in $\mathcal{O}(|V| \cdot \log |V|)$.
- Deterministisch auf einem CRCW mit $\mathcal{O}(|V|)$ Prozessoren in $\mathcal{O}(|V|)$.

Ergebnisse

- Deterministisch sequentiell lässt sich ein Maximal Independent Set in $\mathcal{O}(|E|)$ finden.
- Deterministisch auf einem EREW mit $\mathcal{O}(|V|)$ Prozessoren in $\mathcal{O}(|V| \cdot \log |V|)$.
- Deterministisch auf einem CRCW mit $\mathcal{O}(|V|)$ Prozessoren in $\mathcal{O}(|V|)$.
- Randomisiert auf einem EREW mit $\mathcal{O}(|E|)$ Prozessoren in $\mathcal{O}(\log^2 |V|)$.

Ergebnisse

- Deterministisch sequentiell lässt sich ein Maximal Independent Set in $\mathcal{O}(|E|)$ finden.
- Deterministisch auf einem EREW mit $\mathcal{O}(|V|)$ Prozessoren in $\mathcal{O}(|V| \cdot \log |V|)$.
- Deterministisch auf einem CRCW mit $\mathcal{O}(|V|)$ Prozessoren in $\mathcal{O}(|V|)$.
- Randomisiert auf einem EREW mit $\mathcal{O}(|E|)$ Prozessoren in $\mathcal{O}(\log^2 |V|)$.
- Deterministisch (z.B. durch Derandomisierung) auf einem EREW mit $\mathcal{O}(|V| \cdot |E|)$ Prozessoren in $\mathcal{O}(\log^2 |V|)$.

Ergebnisse

- Deterministisch sequentiell lässt sich ein Maximal Independent Set in $\mathcal{O}(|E|)$ finden.
- Deterministisch auf einem EREW mit $\mathcal{O}(|V|)$ Prozessoren in $\mathcal{O}(|V| \cdot \log |V|)$.
- Deterministisch auf einem CRCW mit $\mathcal{O}(|V|)$ Prozessoren in $\mathcal{O}(|V|)$.
- Randomisiert auf einem EREW mit $\mathcal{O}(|E|)$ Prozessoren in $\mathcal{O}(\log^2 |V|)$.
- Deterministisch (z.B. durch Derandomisierung) auf einem EREW mit $\mathcal{O}(|V| \cdot |E|)$ Prozessoren in $\mathcal{O}(\log^2 |V|)$.
- Auf einem CRCW mit $\mathcal{O}(|V|^2)$ Prozessoren geht es randomisiert sogar in $\mathcal{O}(\log n)$ (wobei die Auswahl anders implementiert werden muss).

Ergebnisse

- Deterministisch sequentiell lässt sich ein Maximal Independent Set in $\mathcal{O}(|E|)$ finden.
- Deterministisch auf einem EREW mit $\mathcal{O}(|V|)$ Prozessoren in $\mathcal{O}(|V| \cdot \log |V|)$.
- Deterministisch auf einem CRCW mit $\mathcal{O}(|V|)$ Prozessoren in $\mathcal{O}(|V|)$.
- Randomisiert auf einem EREW mit $\mathcal{O}(|E|)$ Prozessoren in $\mathcal{O}(\log^2 |V|)$.
- Deterministisch (z.B. durch Derandomisierung) auf einem EREW mit $\mathcal{O}(|V| \cdot |E|)$ Prozessoren in $\mathcal{O}(\log^2 |V|)$.
- Auf einem CRCW mit $\mathcal{O}(|V|^2)$ Prozessoren geht es randomisiert sogar in $\mathcal{O}(\log n)$ (wobei die Auswahl anders implementiert werden muss).
- Deterministisch parallelisiert in $\mathcal{O}(\log n)$ unbekannt.

Referenzen


N. Alon, L. Babai, and A. Itai.
A fast and simple randomized parallel algorithm for the maximal independent set problem.
Journal of Algorithms, 7(4):567–583, 1986.


Rajeev Motwani and Prabhakar Raghavan.
Randomized Algorithms.
Cambridge University Press, 1995.


Igor Potapov.
Lecture on efficient parallel algorithms.
University of Liverpool.


Eric Vigoda.
Lecture notes on a parallel algorithm for generating a maximal independent set.
Georgia Institute of Technology.