Computing Maximal Independent Sets on a PRAM

Jonathan Schmidt-Dominé

21. September 2011

Proseminar „Randomisierte Algorithmen“
Sommersemester 2011

Lehrstuhl für Informatik 1
Prof. Dr. Berthold Vöcking
Betreuer: Benjamin Ries
RWTH Aachen University

Inhaltsverzeichnis

Zusammenfassung

Im Folgenden werden Algorithmen vorgestellt, um die Berechnung inklusionsmaximaler unabhängiger Knotenmengen in Graphen effizient parallel zu berechnen. Zunächst werden hierzu das graphentheoretische Problem und *PRAMs* als parallele Maschinenmodelle für die Laufzeitanalyse vorgestellt. Anschließend wird ausgehend vom einfachen, sequentiellen Algorithmus mit Hilfe von Randomisierung ein effizienter paralleler Algorithmus für diese Maschinenmodelle beschrieben, der sich wiederum derandomisieren lässt. Alle Varianten des parallelen Algorithmus erreichen polylogarithmische Laufzeit mit linear vielen Prozessen.

1 Problemstellung

Sei G ein ungerichteter Graph (V, E). Ein $R \subset V$ heißt *unabhängig* bzw. *independent set* (*IS*) g.d.w.

$$\forall u, v \in R : \{u, v\} \notin E.$$

R heißt *maximal independent set* (*MIS*, d.h. inklusionsmaximal) g.d.w. R unabhängig und jede Obermenge $S \subset V$ nicht unabhängig ist, bzw.

$$\forall u \in V \setminus R : \exists v \in R : \{u, v\} \in E.$$

Solche Mengen lassen sich sowohl sequentiell als auch parallelisiert effizient berechnen, davon abzugrenzen sind *Maximum Independent Sets* (d.h. kardinalitätsmaximal), deren Berechnung NP-vollständig ist.

Abbildung 1: Maximal Independent Set

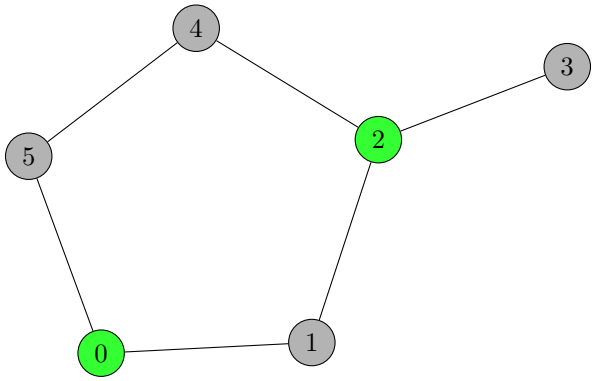
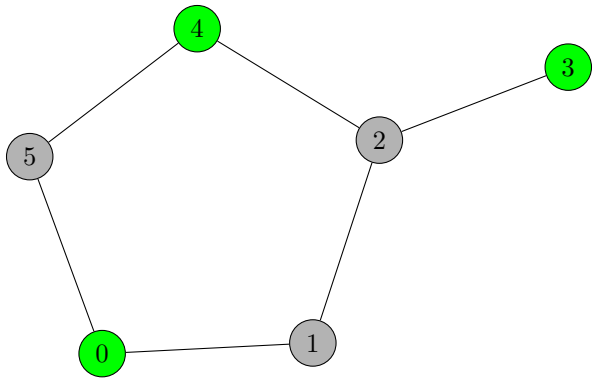


Abbildung 2: Maximum Independent Set auf demselben Graphen



2 PRAM

Für die parallelisierte Komplexitätsanalyse muss zunächst ein Maschinenmodell eingeführt werden, das Parallelisierung erlaubt. Hierzu werden verschieden Varianten des *PRAM* definiert. Diese Maschinenmodelle verfügen über eine variable Anzahl von „Prozessoren“, auf jedem kann ein Programm/Prozess ausgeführt werden, ein Zugriff auf einen gemeinsamen Speicher ist möglich, zudem hat jeder Prozess einen privaten Speicher zur Ausführung zur Verfügung. Praktische Probleme der Synchronisation werden hierbei ausgeklammert.

2.1 Notation

Zur Beschreibung der Algorithmen wählt man zunächst einen High-Level-Ansatz, hierbei ermöglicht man die Parallelisierung einer Schleife:

Algorithmus 1 Parallelisierte Schleife

```
for all i ∈ M do in parallel
    something
end for
```

Man geht hierbei von maximaler Parallelisierung aus, d.h. jeder der $|M|$ Prozesse verarbeitet eines der Elemente von M, es muss jedoch noch betrachtet werden, ob keine Abhängigkeiten zwischen den Prozessen bestehen, evtl. ergibt sich ein zusätzlicher Faktor auf die Laufzeit für die Auflösung dieser Abhängigkeiten.

2.2 Low-Level-Ansatz: Registermaschine

Das Modell einer Registermaschine ist hinreichend bekannt, sie sind realen Maschinen nachempfunden, verfügen über Register, arithmetische Operationen, Jump-Anweisungen und einen Hauptspeicher mit wahlfreiem Zugriff. Bei einem PRAM existieren separate Sätze von Registern für jeden Prozess, jedoch ein zentraler RAM, auf den jeder Prozess zugreifen kann. Nun trifft man eine Unterscheidung danach, wie diese Zugriffe erfolgen dürfen.

Beim *EREW-PRAM* (*Exclusive Read, Exclusive Write*) darf zu einem Zeitpunkt auf jede Speicherzelle nur von höchstens einem Prozess aus zugegriffen werden. Beim *CRCW-PRAM* (*Concurrent Read, Concurrent Write*) darf auf jede Speicherzelle zu jedem Zeitpunkt von jedem Prozess aus lesend wie schreibend

zugegriffen werden. Ein Lesezugriff während eines Schreibzugriffs lässt sich ohne Verschlechterung der Komplexität auflösen, etwa durch abwechselnde Lese- und Schreibphasen. Bei Gleichzeitigen Schreibzugriffen wird gefordert, dass jeder schreibende Prozess denselben Wert in die Speicherzelle schreibt.

Die Unterscheidung zwischen diesen beiden Maschinenmodellen hat Auswirkungen auf die Laufzeitkomplexität, der weniger eingeschränkte CRCW kann bei manchen Problemen eine bessere Laufzeitkomplexität erreichen, ein EREW ist bei m Prozessen jedoch höchstens um den Faktor $\log m$ langsamer.

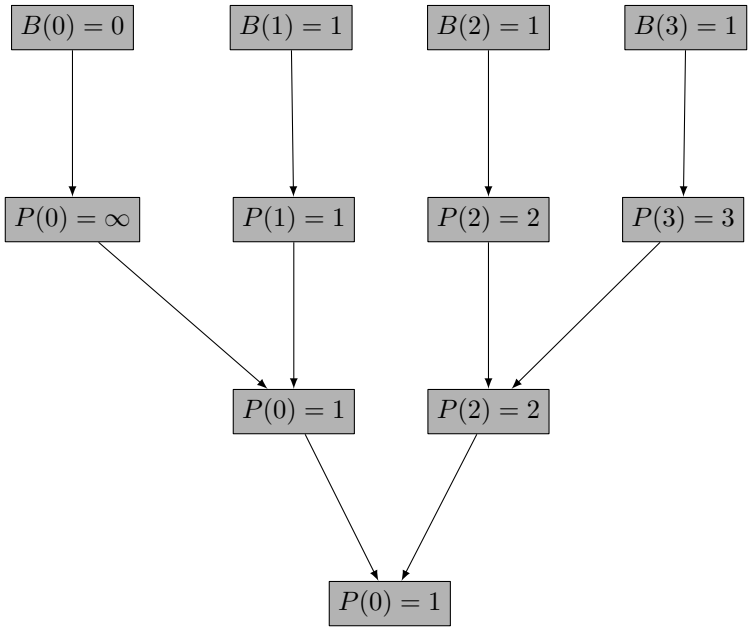
2.3 Beispiel-Problem: erstes gesetztes Bit finden

Um in einem Bitfeld die Position des ersten gesetzten Bits zu bestimmen, setzt man naheliegenderweise jeden Prozess auf eines der Bits an. Bei einem EREW müssen die Prozesse sich nun absprechen, welches das erste gesetzte Bit ist. Dies kann über eine „turnierartige“ Propagation geschehen (einfacher divide-and-conquer Ansatz), es werden je das erste gesetzte Bit in jeder Zweier-, dann Vierer-, dann Achter-, usw. Gruppe von Bits berechnet, indem jeweils das Ergebnis von zwei Prozessen zusammengeführt wird. Dies ergibt eine Laufzeit von $\mathcal{O}(\log n)$ mit $\mathcal{O}(n)$ Prozessen. Über eine solche Propagation lässt sich jeder Schreibzugriff auf einem CRCW in gültigen EREW-Code umwandeln, mit besagtem zusätzlichen Faktor $\mathcal{O}(\log n)$ (siehe Algorithmus 2).

Algorithmus 2 Finden des ersten gesetzten Bits auf einem EREW mit n Prozessen (der Einfachheit halber sei n eine Zweierpotenz), „turnierartige“ Propagation

```
procedure FirstBit(B)
    for all i ∈ [n] do in parallel
        if B(i) then
            P(i) ← i
        else
            P(i) ← ∞
        end if
    end for
    w ← 2
    while w ≤ n do
        for all i ∈ [n] do in parallel
            if i ≡ 0 (mod w) then
                P(i) ← min(P(i), P(i + w/2))
            end if
        end for
        w ← 2 · w
    end while
    return P(0)
end procedure
```

Abbildung 3: Beispiel zum Finden der Position des ersten gesetzten Bits



Die Verwendung eines CRCW ermöglicht hier tatsächlich die Reduzierung der Komplexität um diesen Faktor. Hierzu verwende man $\mathcal{O}(n^2)$ Prozesse, einen für jedes Paar von Bits. In dem Falle, dass beide markiert sind, wird das letztere der beiden als nicht-minimal markiert. Dies ergibt eine Laufzeit von $\mathcal{O}(1)$. Die Anzahl der Prozesse lässt sich jedoch reduzieren. Zunächst stelle man fest, dass man auf einem CRCW in $\mathcal{O}(1)$ mit n Prozessen feststellen kann, ob eines von n Bits gesetzt ist, hierfür setzt einfach jeder Prozess, der ein gesetztes Bit vorfindet, das Ergebnis auf 1. Unterteilt man das Bitfeld in $\sqrt{n}$ Segmente, so kann für jedes davon bestimmt werden, ob in ihm ein Bit gesetzt ist – mit insgesamt n Prozessen. Mit dem Algorithmus mit $\mathcal{O}(n^2)$ Prozessen kann man nun feststellen, welches der $\sqrt{n}$ Segmente, das erste ist, das ein gesetztes Bit enthält, und in diesem Segment mit dem Algorithmus das erste gesetzte Bit ausfindig machen, für diese beiden Schritte benötigt man $\mathcal{O}(\sqrt{n}^2) = \mathcal{O}(n)$ Prozesse, und es ergibt sich eine Laufzeit von $\mathcal{O}(1)$[1].

3 Der Algorithmus

Ausgehend von einem einfachen, sequentiellen Algorithmus sollen in der Folge parallelisierbare Teilschritte herausextrahiert und diese mit Hilfe von Randomisierung parallelisiert werden, sodass eine polylogarithmische Laufzeit erreicht werden kann.

3.1 Definitionen

Definition 1. Die *Nachbarschaftsfunktion* zu einem Knoten x bzw. einer Knotenmenge R sei definiert als $nh(x) := \{y \in V | \{x, y\} \in E\}$ bzw. für eine Menge von Knoten $nh(R) := \bigcup_{x \in R} nh(x)$.

Definition 2. Der *Knotengrad* eines Knotens x sei notiert als $d(x) := |\{\{x, y\} \in E\}|$

3.2 Sequentieller Algorithmus

Für das MIS-Problem existiert ein sehr einfacher sequentieller Greedy-Algorithmus. Dieser wählt so lange wie möglich unabhängige Knoten aus:

Algorithmus 3 Sequentieller Greedy-Algorithmus

```
procedure GreedyMIS(V, E)
    R ← ∅
    while V ≠ ∅ do
        x ← chooseNextFrom(V)
        V ← V \ ({x} ∪ nh(x))
        R ← R ∪ {x}
    end while
    return R
end procedure
```

Dieser Algorithmus hat eine Laufzeit von $\mathcal{O}(|E|)$, da bei der Entfernung der Nachbarschaft schlussendlich jede Kante genau einmal berücksichtigt werden muss. Eine einzelne Entfernung der Nachbarschaft lässt sich parallelisieren, so kommt man auf eine Laufzeit von $\mathcal{O}(|V| \cdot \log |V|)$ (EREW) bzw. $\mathcal{O}(|V|)$ (CRCW) bei $\mathcal{O}(|V|)$ Prozessen. Die äußere Schleife ist jedoch nicht parallelisierbar, da Durchläufe von vorherigen abhängen können. Sollte etwa ein Knoten zum IS hinzugefügt werden, läge er womöglich in der Nachbarschaft eines parallel hinzugefügten, und führte so zur Verletzung der IS-Eigenschaft. Das Hinzufügen einer *unabhängigen* Menge von Knoten ist dagegen stets parallel möglich. Dies ist die Grundidee des parallelen Algorithmus.

3.3 Paralleler Algorithmus

Der parallele Algorithmus (4) berechnet wiederholt unabhängige Mengen, sodass sich schließlich ein MIS als Vereinigung dieser ergibt. Er muss effizient ein IS konstruieren, das trotz der Effizienzforderung groß genug ist, dass im Erwartungswert nur logarithmisch viele dieser Konstruktionen notwendig sind, um ein MIS erreichen.

Algorithmus 4 Struktur des parallelen Algorithmus

```
R ← ∅
while V ≠ ∅ do
    S ← chooseIndependentSet(V)
    R ← R ∪ S
    V ← V \ (S ∪ nh(S))
end while
```

3.4 Finden eines IS

Die Auswahl eines IS findet durch die zufällige Auswahl von Knoten statt. Stehen ausgewählte Knoten im *Konflikt*, sind also adjazent, wird der mit dem niedrigeren Grad entfernt. Zunächst werden alle *Waisen*, d.h. Knoten mit Grad 0 zur Auswahl hinzugefügt. Diese Waisen lassen sich mit $|V|$ Prozessen in $\mathcal{O}(1)$ auswählen (Algorithmus 5). Etwaige Überprüfungen oder eine Entfernung der Nachbarschaft sind für diese Knoten nicht notwendig. Die Entfernung der Waisen sorgt dafür, dass im folgenden in der Laufzeitanalyse der Grad eines jeden Knoten als strikt positiv angenommen werden kann. Dieser Schritt verschlechtert nicht die Komplexität, da die restlichen Schritte bei der Auswahl des IS eine Laufzeit von $\Omega(1)$ benötigen.

Algorithmus 5 Waisen ($d(v) = 0$) zum IS hinzufügen

```
for all v ∈ V do in parallel
    if d(v) = 0 then
        R ← R ∪ {v}
        V ← V \ {v}
    end if
end for
```

Bei der zufälligen Auswahl weiterer Knoten (Algorithmus 6) werden bevorzugt Knoten mit niedrigem Grad ausgewählt, dies ist plausibel, da die Auswahl von Knoten mit höherem Grad eher zu Konflikten führt.

Algorithmus 6 Auswahl, nachdem Waisen entfernt sind

```
S ← ∅
for all v ∈ V do in parallel
    with probability 1/(2·d(v)) do
        S ← S ∪ {v}
    end do
end for
```

Wenn zwei adjazente Knoten ausgewählt worden sind, wird derjenige mit dem niedrigeren Grad aus der Auswahl entfernt (derjenige mit dem höheren Grad vergrößert schließlich tendenziell die zu entfernende Nachbarschaft und führt somit zu früherer Konvergenz). Im Falle gleichen Knotengrades wird nach einer willkürlichen Ordnung der Knoten ein Knoten aus der Auswahl entfernt:

Algorithmus 7 Behebung der Konflikte

```
for all (u, v) ∈ E do in parallel
    if u ∈ S ∧ v ∈ S then
        if d(u) < d(v) then
            S ← S \ {u}
        else if d(v) < d(u) then
            S ← S \ {v}
        else
            if u < v then
                S ← S \ {u}
            else
                S ← S \ {v}
            end if
        end if
    end if
end for
```

4 Laufzeitanalyse

Zur Untersuchung der Laufzeit wird im Folgenden bewiesen, dass in jedem Durchgang der äußeren Schleife im Erwartungswert ein gewisser Anteil der verbleibenden Kanten entfernt wird. Hierfür betrachtet man eine Klasse von für die Laufzeit gutartigen Knoten.

4.1 Abschätzung der Wahrscheinlichkeit der Auswahl guter Knoten

Definition 3. 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 4. Eine Kante heißt *gut*, genau dann wenn sie mindestens einen guten inzidenten Knoten hat.

Zunächst wird gezeigt, dass jeder gute Knoten mit konstanter Wahrscheinlichkeit die Auswahl zumindest eines Knotens garantiert.

Lemma 1. *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 2. *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 $d(w) \leq d(v)$. Für die Wahrscheinlichkeit $Q = 1 - P$, dass keiner von ihnen ausgewählt wird, gilt aufgrund der Unabhängigkeit

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

□

Da somit zu jedem guten Knoten zumindest mit konstanter Wahrscheinlichkeit ein adjazenter Knoten zum IS hinzugefügt wird, wird in jedem Schleifendurchlauf einjeder guter Knoten ebenfalls mit zumindest ebenjener Wahrscheinlichkeit aus dem Graphen entfernt, da am Ende des Schleifenkörpers auch die Nachbarschaft des gewählten IS entfernt wird.

4.2 Häufigkeit guter Kanten

In diesem Beweis entscheidend für die Laufzeit des Algorithmus ist die Entfernung guter Kanten, deren Häufigkeit nun abgeschätzt werden muss.

Definition 5. $d_{\leq}(v)$ bzw. $d_{>}(v)$: Anzahl der adjazenten Knoten mit niedrigerem oder gleich großen bzw. größerem Grad

$$d_{\leq}(v) := |\{u \in V | \{u, v\} \in E \wedge d(u) \leq d(v)\}|, \; d_{>}(v) := d(v) - d_{\leq}(v).$$

Definition 6. $e_{b/g,b/g}$: Anzahl der Kanten von einem bösen/guten Knoten zu einem bösen/guten Knoten mit höherem Grad.

Definition 7. $e_{b/g}$: Anzahl der bösen/guten Kanten.

Lemma 3. *Für jeden bösen Knoten v gilt*

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

Beweis. Folgt unmittelbar aus der Definition der Bosheit, $d_{\leq}(v) < \frac{1}{3} \cdot d(v)$ und da $d_{\leq}(v) + d_{>}(v) = d(v)$, gilt auch $d_{>}(v) > \frac{2}{3} \cdot d(v)$. □

Lemma 4. *Es gibt in einem Graphen höchstens so viele böse wie gute Kanten.*

Beweis. Hierzu betrachte man die Summe der Grade aller bösen Knoten X:

$$X := \sum_{v \text{ böse}} d(v) \leq 3 \cdot \sum_{v \text{ böse}} (d_{>}(v) - d_{\leq}(v))$$

Da in der Summe auf der rechten Seite jedoch alle Kanten zwischen bösen Knoten genau einmal positiv und genau einmal negativ auftreten und somit nur, ist diese Summe genau gleich $e_{b,g} - e_{g,b}$.

$$\Rightarrow X \leq 3 \cdot (e_{b,g} - e_{g,b}) \leq 3 \cdot (e_{b,g} + e_{g,b})$$

Es gilt zudem

$$X = 2 \cdot e_b + e_{b,g} + e_{g,b} \Rightarrow 2 \cdot e_b + e_{b,g} + e_{g,b} \leq 3 \cdot e_{b,g} + 3 \cdot e_{g,b}$$
$$\Rightarrow e_b \leq e_{b,g} + e_{g,b} \leq e_g.$$

□

4.3 Ergebnis

Da jede gute Kante einen adjazenten guten Knoten hat, welcher zumindest mit konstanter Wahrscheinlichkeit aus dem Graphen entfernt wird, wird auch jede gute Kante in einem Schleifendurchlauf zumindest mit dieser konstanten Wahrscheinlichkeit entfernt. Somit wird in jedem Schleifendurchlauf zumindest ein konstanter Anteil der Kanten (nämlich die guten) mit konstanter Wahrscheinlichkeit entfernt und es folgt:

Theorem 5. *Der MIS-Algorithmus benötigt im Mittel* $\mathcal{O}(\log |E|) = \mathcal{O}(\log |V|)$ *Schleifendurchläufe[2].*

4.4 Laufzeit auf einem EREW

Der Schleifenkörper lässt sich auf einem EREW nicht in konstanter Laufzeit durchführen, denn die Konfliktbehebung benötigt eine Synchronisation, damit nicht mehrere Prozesse denselben Knoten entfernen. Wie zuvor bereits ausgeführt ist diese durch „tunierartiges“ Propagieren jeweils zu entfernender Knoten möglich. Dadurch wird auch sicher gestellt, dass gespeicherte Knotengrade aktualisiert werden können, denn es kann so auch sichergestellt werden, dass keine zwei Prozesse den Grad eines Knoten dekrementieren. Dies führt zu einer Laufzeit des Schleifenkörpers von $\mathcal{O}(\log |V|)$.

4.5 Laufzeit auf einem CRCW

Bei einem CRCW können problemlos mehrere Prozesse gleichzeitig in konstanter Zeit einen Knoten als nicht mehr ausgewählt markieren. Das Mitzählen der Knotengrade ist dann jedoch nicht mehr möglich, da dies eine Synchronisation erfordern würde, lediglich die Information, welche Knoten und Kanten noch im Graphen vorhanden sind, steht zur Verfügung. Man kann die konstante Zeit jedoch dennoch gewährleisten, indem man die zufällige Auswahl, bei der der Knotengrad verwendet wird, anders implementiert. Hierbei soll die Auswahl mit einer ähnlichen Wahrscheinlichkeit stattfinden, jedoch ohne Kenntnis des Grades.

4.5.1 Auswahl mit einem CRCW

Für jeden Knoten wird nun parallel durch jeweils D Prozesse, entschieden, ob dieser ausgewählt wird, wobei D der Knotengrad zu Beginn der Ausführung ist. Jeder der Prozesse zieht nun einen zufälligen Knotenindex. Mit dem Algorithmus, um parallel das erste gesetzte Bit zu finden, bestimmt man nun den ersten dieser Knoten, der noch nicht entfernt ist (diese Information liegt als Bitfeld vor). Befindet sich in der zufälligen Ziehung kein solcher Knoten, so findet die Auswahl nicht statt. Die Auswahl des Knoten findet ansonsten genau dann statt, wenn dieser gezogene Knoten gleich dem minimalen noch nicht entfernten adjazenten Knoten ist, der sich ebenso bestimmen lässt. Sei nun P die Wahrscheinlichkeit, dass diese Auswahl stattfindet. Es gilt sicherlich $P = \frac{1}{d(v)}$ für den Fall, dass ein noch nicht entfernter adjazenter Knoten gezogen worden ist, da im ersten Schritt jeder adjazente Knoten mit derselben Wahrscheinlichkeit

ausgewählt wird. Für die Wahrscheinlichkeit Q, dass kein gezogener adjazenter Knoten noch nicht entfernt worden ist, gilt

$$Q \leq \left(\frac{D-1}{D}\right)^D = \left(1 - \frac{1}{D}\right)^D \leq \frac{1}{e} \leq \frac{1}{2}.$$

Hierfür hat man den Fall betrachtet, dass nur noch ein adjazenter Knoten noch nicht entfernt worden ist, wären alle entfernt, läge ein Waise vor, und der Knoten selbst wäre schon zuvor entfernt worden. Somit gilt für P

$$\frac{1}{d(v)} \geq P = (1-Q) \cdot \frac{1}{d(v)} \geq \frac{1}{2 \cdot d(v)}.$$

Ich verweise auf die Literatur, dass auch für ein P aus diesem Intervall eine Abschätzung gelingt, dass stets ein konstanter Anteil der Knoten entfernt wird[3]. Zur randomisierten Auswahl bedarf es nun $\mathcal{O}(|E|)$ Prozesse, um die zufällige Entscheidung für jeden Knoten durchzuführen.

5 Derandomisierung

5.1 Idee

Jeder randomisierte Algorithmus, der n Zufallsbits benötigt, lässt sich mit 2^n Prozessen deterministisch simulieren. Oftmals genügen jedoch statt n *unabhängigen* Zufallsbits n *paarweise unabhängige* Zufallsbits. n paarweise unabhängige Zufallsbits lassen sich aus $\log(n+1)$ unabhängigen Zufallsbits erzeugen, indem man je nicht-leere Teilmengen dieser Zufallsbits per XOR kombiniert. Ein Beispiel mit drei Zufallsbits, genannt b_0, b_1 und b_2, aus ihnen lassen sich folgende sieben paarweise unabhängige Bits erzeugen:

$$b_0,\ b_1,\ b_2,\ b_0 \veebar b_1,\ b_0 \veebar b_2,\ b_1 \veebar b_2,\ b_0 \veebar b_1 \veebar b_2$$

Offensichtlich ist jedes dieser Zufallsbits gleichverteilt. Die paarweise Unabhängigkeit ergibt sich dadurch, dass die bedingte Wahrscheinlichkeit zweier dieser Zufallsbits wiederum die Negation der XOR-Kombination der Bits ist, um die sie sich unterscheiden und somit $\frac{1}{2}$ ist, denn $P(B \veebar b = 1 | B = 1) = P(b = 0) = P(B = 1 | B \veebar b = 1)$, z.B. $P(b_0 \veebar b_1 = 1 | b_0 \veebar b_2 = 1) = P(b_1 \veebar b_2 = 0) = \frac{1}{2}$. Offensichtlich sind diese Bits allerdings nicht vollständig unabhängig, da sich aus einer Teilmenge alle weitern berechnen lassen.

Die exhaustive Betrachtung aller möglichen Belegungen logarithmisch vieler unabhängiger Zufallsbits, aus denen sich die paarweise unabhängigen ergeben, benötigt nun nur noch einen linearen Faktor auf die Anzahl der benötigten Prozesse und macht auch ein Zusammenführen der Ergebnisse aller Prozesse zu einem Ergebnis in logarithmischer Zeit möglich. Es stellt sich heraus, dass für den betrachteten MIS-Algorithmus auf einem EREW tatsächlich paarweise unabhängige Zufallsbits genügen. Ihre Zahl liegt in $\mathcal{O}\left(|V| \cdot \log^2 |V|\right)$ (in logarithmisch vielen äußeren Schleifendurchläufen wird für jeden Knoten eine Zufallszahl mit logarithmisch vielen Bits benötigt, asymptotisch werden somit $\mathcal{O}(\log |V|)$ unabhängige Zufallsbits benötigt).

5.2 Beweis, dass paarweise Unabhängigkeit genügt

Lemma 6. *Auch bei nur paarweise unabhängigen Zufallsbits benötigt der Algorithmus im Erwartungswert nur logarithmisch viele äußere Schleifendurchläufe.*

Beweis. Zu zeigen ist eine konstante untere Schranke für die Wahrscheinlichkeit P der Auswahl mindestens eines Knotens aus $nh(v)$ mit gutem v, dies ist die einzige Stelle, an der (im Falle eines EREW) bislang Unabhängigkeit vorausgesetzt wurde. 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 aufgrund der paarweisen Unabhängigkeit

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

Falls nun $\sum_w p_w \leq 1$, folgt

$$P \geq \frac{1}{2} \cdot \sum_u p_u \geq \frac{1}{2} \cdot \frac{d(v)}{3} \cdot \frac{1}{2 \cdot d(v)} = \frac{1}{12}.$$

Andernfalls betrachte man die Knoten der Nachbarschaft mit dem geringsten Grad, sodass die Summe der p_u gerade nicht 1 übersteigt. Diese Summe ist größer oder gleich $\frac{1}{2}$, da $\frac{1}{2 \cdot d(w)}$ stets kleiner oder gleich $\frac{1}{2}$ ist. Dann gilt

$$P \geq \sum_u p_u \cdot \left(1 - \frac{1}{2} \cdot \sum_w p_w\right) \geq \frac{1}{2} \cdot \left(1 - \frac{1}{2} \cdot 1\right) = \frac{1}{4}.$$

Insgesamt ergibt sich also als untere Schranke $\frac{1}{12}$, diese ist sogar etwas größer als $1 - e^{-\frac{1}{6}}$[4]. □

6 Ergebnisse

Aus den Betrachtungen ergeben sich folgende obere Laufzeitschranken:

- Deterministisch sequentiell lässt sich ein Maximal Independent Set in $\mathcal{O}(|E|)$ finden.
- Deterministisch auf einem EREW mit $\mathcal{O}(|V|)$ Prozessen in $\mathcal{O}(|V| \cdot \log |V|)$.
- Deterministisch auf einem CRCW mit $\mathcal{O}(|V|)$ Prozessen in $\mathcal{O}(|V|)$.
- Randomisiert auf einem EREW mit $\mathcal{O}(|E|)$ Prozessen in $\mathcal{O}(\log^2 |V|)$.
- Deterministisch (durch Derandomisierung) auf einem EREW mit $\mathcal{O}(|V| \cdot |E| \cdot \log^2 |V|)$ Prozessen in $\mathcal{O}(\log^2 |V|)$.
- Randomisiert auf einem CRCW mit $\mathcal{O}(|E|)$ Prozessen in $\mathcal{O}(\log |V|)$.
- Es ist bislang kein deterministischer paralleler Algorithmus mit einer Laufzeit von $\mathcal{O}(\log |V|)$ bekannt.

Literatur

[1] I. Potapov, "Lecture on efficient parallel algorithms." University of Liverpool.

[2] R. Motwani and P. Raghavan, *Randomized Algorithms.* Cambridge University Press, 1995.

[3] N. Alon, L. Babai, and A. Itai, "A fast and simple randomized parallel algorithm for the maximal independent set problem," *Journal of Algorithms*, vol. 7, no. 4, pp. 567–583, 1986.

[4] E. Vigoda, "Lecture notes on a parallel algorithm for generating a maximal independent set." Georgia Institute of Technology.

7 Anhang: Beispiel-Ablauf des Algorithmus

Abbildung 4: Zufällige Auswahl

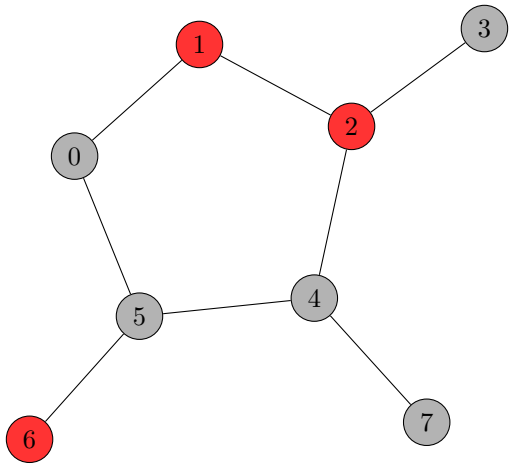


Abbildung 5: Konfliktbehebung (Konflikt in Orange)

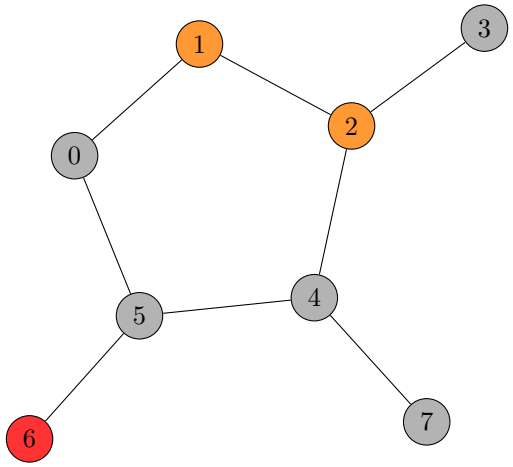


Abbildung 6: Konfliktbehebung (ein Knoten wurde wieder aus der Auswahl entfernt)

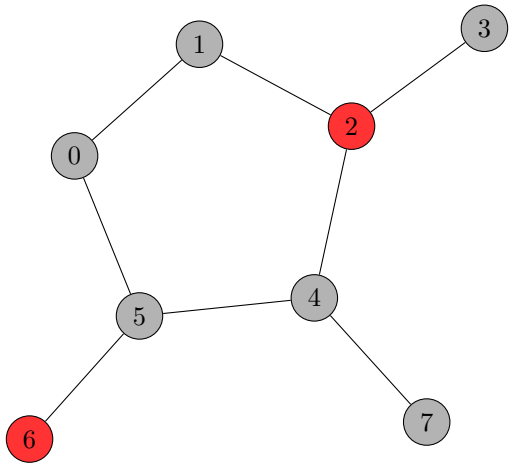


Abbildung 7: Auswahl abschließen und Nachbarschaft entfernen

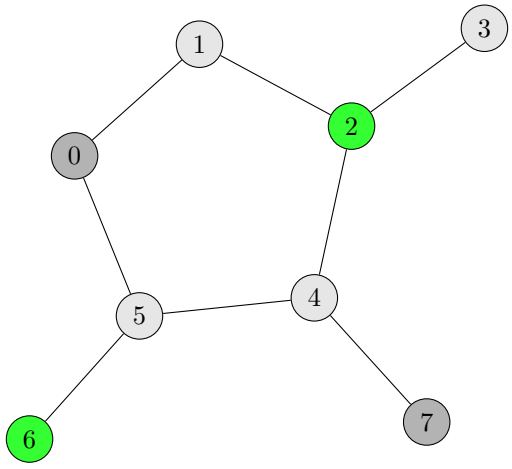


Abbildung 8: Nächster Durchgang: nur noch Waisen