Zusammenfassung des Stoffes zur Vorlesung Algorithmentechnik

Max Kramer

30. März 2009

Diese Zusammenfassung ENTSTEHT MOMENTAN als persönliche Vorbereitung auf die Klausur zur Vorlesung "Algorithmentechnik" von Prof. Dr. Dorothea Wagner im Wintersemester 08/09 an der Universität Karlsruhe (TH). Sie ist sicherlich nicht vollständig, sondern strafft bewusst ganze Kapitel und Themen wie Amortisierte Analyse, Parallele Algorithmen und viele mehr. Für Verbesserungen, Kritik und Hinweise auf Fehler oder Unstimmigkeiten an third äht web.de bin ich dankbar.

Inhaltsverzeichnis

0 Grundlagen **4**

1 Grundlegende Datenstrukturen für Operationen auf Mengen **5**

2 Aufspannende Bäume minimalen Gewichts **9**

3 Schnitte in Graphen und Zusammenhang **14**

4 Flussprobleme und Dualität **16**

5 Kreisbasen minimalen Gewichts **22**

6 Lineare Programmierung **26**

7 Approximationsalgorithmen **28**

8 Randomisierte Algorithmen **33**

9 Parallele Algorithmen **37**

10 Parametrisierte Algorithmen **42**

Liste der Algorithmen

1 MAKESET(x) . . . 5
2 weighted UNION(i, j) . . . 5
3 FIND(x) (mit Pfadkompression) . . . 5
4 Algorithmus von Kruskal . . . 6
5 OFFLINE-MIN-Initialisierung . . . 6
6 OFFLINE-MIN $\in \Theta(n)$ . . . 7
7 HEAPIFY$(A, i) \in O(\log n)$ . . . 7
8 MAKEHEAP$(A) \in \Theta(n)$ . . . 7
9 SIFT-UP$(A, i) \in O(\log n)$ . . . 8
10 DELETE$(A, i) \in O(\log n)$ . . . 8
11 INSERT$(A, x) \in \Theta(\log n)$ . . . 8
12 HEAPSORT$(A) \in O(n \log n)$ . . . 8
13 BOTTOM-UP-HEAPIFY$(A, 1) \in O(\log n)$ . . . 9
14 Färbungsmethode von Tarjan . . . 10
15 Algorithmus von Kruskal (verbal) $\in O(|E| \log |V|)$ . . . 10
16 Algorithmus von Prim (verbal) . . . 11
17 Algorithmus von Prim $\in O(|E| \log_{2+|E|/|V|}|V|)$ . . . 11
18 Greedy-Algorithmus für Optimierungsproblem über $(M, \mathcal{U})$ . . . 12
19 MIN-SCHNITT-PHASE$(G, c, a) \in O(|V| \log |V| + |E|)$ . . . 14
20 MIN-SCHNITT$(G, c, a) \in O(|V|^2 \log |V| + |V||E|)$ . . . 14
21 MAX-FLOW$(D; s, t; c)$ . . . 17
22 Algorithmus von Ford-Fulkerson . . . 18
23 PUSH(v, w) . . . 19
24 RELABEL(v) . . . 19
25 Algorithmus von Goldberg und Tarjan . . . 20
26 MCB-GREEDY-ALGORITHMUS$(\mathcal{C})$ . . . 23
27 Algorithmus von Horton $\in O(|V||E|^3)$ . . . 23
28 Algorithmus von de Pina $\in O(|E|^3 + |E||V|^2 \log |V|)$ . . . 24
29 Algorithmus von de Pina algebraisch . . . 24
30 SIMPLE MCB . . . 25
31 MCB-CHECKER . . . 25
32 GREEDY-KNAPSACK $\in O(n \log n)$ . . . 29
33 NEXT FIT $\in O(n)$ . . . 29
34 FIRST FIT $\in O(n^2)$ . . . 29
35 LIST SCHEDULING $\in O(n)$ . . . 30
36 $\mathcal{A}_l$ für MULTIPROCESSOR SCHEDULING $\in O(m^l + n)$ . . . 30
37 APAS für BIN PACKING $\in O(c_\varepsilon + n \log n)$ . . . 32
38 RANDOM MINCUT $\in O(|V|^2)$ . . . 33
39 FAST RANDOM MINCUT $\in O(|V|^2 \log |V|)$ . . . 34
40 RANDOM SAT $\in O(n)$ . . . 35
41 RANDOM MAXCUT . . . 35
42 BROADCAST $\in O(logN)$ . . . 38
43 SUMME$(a_1, \ldots, a_n) \in O(logn)$ . . . 38

44 CRCW-Oder$(x_1, \ldots, x_n) \in O(1)$ 38
45 Präfixsummen$(a_n, \ldots, a_{2n-a}) \in O(logn)$ 39
46 List Ranking$(n, h) \in O(logn)$ 39
47 Zusammenhang$(G) \in O(log^2|V|)$ 40
48 MSTg(G) 41
49 Vertex-Cover$(G, k) \in O(nk + 2^k \cdot k^2)$ 43

0 Grundlagen

0.1 Amortisierte Analyse

- **Ganzheitsmethode**: Bestimme obere Schranke $T(n)$ für n Operationen $\Rightarrow \frac{T(n)}{n}$ amortisierte Kosten je Operation
- **Buchungsmethode**: Weise Operationen "Gebühren" zu und nutze überschüssigen "Kredit" der Objekte für spätere Operationen an den Objekten.
- **Potentialmethode**: Definiere "Kredit" als Potential $\mathbb{C}(D_i)$ aller Objekte nach der i-ten Operation.
  Definiere die amortisierten Kosten: $\hat{c_i} := c_i + \mathbb{C}(D_i) - \mathbb{C}(D_{i-1})$
  Falls $\forall\ n \in \mathbb{N}$ gilt $\mathbb{C}(D_n) \geq \mathbb{C}(D_0) \Rightarrow$ amortisierte Kosten obere Schranken für Gesamtkosten

0.2 Rekursionsabschätzung

- **Substitutionsmethode**: vermute Lösung und beweise induktiv (Tricks: späterer Induktionsanfang, Vermutungen verschärfen, Variablen ersetzen)
- **Iterationsmethode**: schreibe Laufzeit durch iteratives Einsetzen als Summe und schätze diese ab
- **Master Theorem**:
  $T(n) = a \cdot T(\frac{n}{b}) + f(n), a \geq 1, b > 1$
  - $f(n) \in \Omega(n^{\log_b a+\varepsilon}), af(\frac{n}{b}) \leq cf(n)$ für $c < 1$ und $n \geq n_0 \ \Rightarrow\ T(n) \in \Theta(f(n))$
  - $f(n) \in \Theta(n^{\log_b a}) \ \Rightarrow\ T(n) \in \Theta(n^{\log_b a} \cdot \log n)$
  - $f(n) \in O(n^{\log_b a-\varepsilon}) \ \Rightarrow\ T(n) \in \Theta(n^{\log_b a})$

1 Grundlegende Datenstrukturen für Operationen auf Mengen

1.1 Union-Find-Problem

Stelle eine Datenstruktur und Operationen darauf zur Verfügung die eine Folge disjunkter Mengen möglichst effizient verwalten:

- MAKESET(x): Führe neue Menge $\{x\}$ ein.
- UNION(S_i, S_j, S_k): Vereinige S_i und S_j zu S_k und entferne S_i und S_j.
- FIND(x): Gebe die Menge M an, welche x enthält.

Repräsentiere Mengen durch Bäume indem zu jedem Element x sein Vorgänger VOR[x] in einem Array gespeichert wird. Für Wurzeln w ist VOR[w] = $-$#(Knoten im Baum w).

Algorithmus 1 MAKESET(x)

Eingabe: Element x
Seiteneffekte: Neuer Index in VOR[]
VOR[x] $\leftarrow -1$

Algorithmus 2 weighted UNION(i, j)

Eingabe: Mengen S_i, S_j
Seiteneffekte: Die Elemente des Baumes mit weniger Elementen werden dem anderen Baum hinzugefügt
$z \leftarrow$ VOR[i] + VOR[j]
Falls |VOR[i]| < |VOR[j]|
  VOR[i] $\leftarrow j$
  VOR[j] $\leftarrow z$
Sonst
  VOR[j] $\leftarrow i$
  VOR[i] $\leftarrow z$

Lemma 1 (**Baumhöhe**). *Für die Höhe eines durch* MAKESET *und weighted* UNION *entstandenen Baumes T gilt: $h(T) \leq log\ _2|T|$*

Algorithmus 3 FIND(x) (mit Pfadkompression)

Eingabe: Element x
Ausgabe: Menge in der x enthalten ist
$w \leftarrow x$
Solange VOR[w] > 0
  $w \leftarrow$ VOR[w]
$i \leftarrow x$
Solange VOR[i] > 0
  $temp \leftarrow i$
  $i \leftarrow$ VOR[i]
  VOR[$temp$] $\leftarrow w$
Gib w aus

Satz 1 (Hopcroft & Ullman). *Die Gesamtlaufzeit von n Operationen vom Typ* MAKESET, UNION *und* FIND *mit Pfadkompression ist in $O(n \cdot G(n))$.*

Dabei ist $G(n) := min\{y \mid F(y) \geq n\}$ mit $F(0) := 1$ und $F(y) := 2^F(y-1)$ für $y > 0$. Daher ist $G(n) \leq 5$ für alle "praktisch" relevanten Werte.

Rang $r(v)$:= Höhe des Unterbaumes mit Wurzel v (ohne Pfadkompression)

Ranggruppe $\gamma_j := \{v \mid \log^{(j+1)} \cdot n < r(v) \leq \log^{j} \cdot n\}$

Eine genauere Analyse zeigt, dass m Operationen vom Typ MAKESET, UNION und FIND auf n Elementen $O(m \cdot \alpha(m,n))$ (α = Ackermannfunktion) Zeit benötigen.

1.2 Anwendungen für Union-Find-Datenstrukturen

1.2.1 Algorithmus von Kruskal

Algorithmus 4 Algorithmus von Kruskal

```
Eingabe: Graph G = (V, E) mit Kantengewichten
Ausgabe: MST in Form von grünen Kanten
GRÜN ← ∅
SORT(E) ← E aufsteigend sortiert
Für v ∈ V
  MAKESET(v)
Für {v, w} ∈ SORT(E)
  Falls FIND(v) ≠ FIND(w)
    UNION(FIND(v), FIND(w))
    GRÜN ← GRÜN ∪ {{v, w}}
```

1.2.2 Offline-Min-Problem

OFFLINE-MIN-Problem: Gebe zu einer Folge Q von n INSERT(i) und EXTRACT-MIN-Operationen alle i, die entfernt wurden, und jeweils die Operation bei der sie entfernt wurden an. (Dabei sei $i \in \{1, \ldots, n\}$)

Algorithmus 5 OFFLINE-MIN-Initialisierung

```
Eingabe: Operationenfolge Q_1, Q_2, ..., Q_n
Ausgabe: Mengen M_j := {i | INSERT(i) erfolgt zwischen j − 1-tem und j-tem EXTRACT-MIN }
j ← 1
Für i = 1 bis n
  Falls Q_i ≠ EXTRACT-MIN
    MAKESET(i)
    UNION(j, FIND(i))
  Sonst
    j ← j + 1
```

Sei k nun die Anzahl der EXTRACT-MIN-Operationen und die Mengen $M_1, \ldots, M_{k+1}$ durch PRED und SUCC doppelt verlinkt.

Algorithmus 6 OFFLINE-MIN $\in \Theta(n)$

Für $i = 1$ bis n
  $j \leftarrow$ FIND(i)
  Falls $j \leq k$
    Gebe "i ist durch die j-te EXTRACT-MIN-Operation entfernt worden" aus
    UNION$(j, \text{SUCC}[j], \text{SUCC}[j])$ // $M_{\text{SUCC}[j]} = M_j \cup M_{\text{SUCC}[j]}$
    SUCC[PRED$[j]] \leftarrow$ SUCC$[j]$ // überspringe M_j
    PRED[SUCC$[j]] \leftarrow$ PRED$[j]$

1.2.3 Priority Queues und Heaps

PRIORITY QUEUE nennt man eine Datenstruktur H welche die Operationen FINDMAX(), DELETE(H, i), INSERT(H, x) und MAKEHEAP(M) unterstützt.

Ein **HEAP** ist ein als Array A realisierter voller binärer Baum, der die HEAP-Eigenschaft erfüllt: $\forall\ i$ gilt $A[i] \geq A[2i]$ und $A[i] \geq A[2i+1]$

Algorithmus 7 HEAPIFY$(A, i) \in O(\log\ n)$

Eingabe: Vollst. binärer Baum als Array A, Index i
Ausgabe: Array A mit Unterbaum von i als HEAP
Vorbedingungen: Unterbäume von $A[2i]$ und $A[2i+1]$ sind bereits HEAPS
Falls $2i \leq |A|$ und $A[2i] > A[i]$
  $m \leftarrow 2i$
Sonst
  $m \leftarrow i$
Falls $2i + 1 \leq |A|$ und $A[2i+1] > A[m]$
  $m \leftarrow 2i + 1$
Falls $m \neq i$
  Vertausche $A[i]$ und $A[m]$
  HEAPIFY(A, m)

Algorithmus 8 MAKEHEAP$(A) \in \Theta(n)$

Eingabe: Vollst. binärer Baum als Array A
Ausgabe: Array A als HEAP
Für $i = \lfloor \frac{|A|}{2} \rfloor, \ldots, 1$
  HEAPIFY(A, i)

Um HEAPSORT zu beschleunigen kann man eine Operation BOTTOM-UP-HEAPIFY$(A, 1)$ implementieren, die mit $\log\ n + \varepsilon$ (im Mittel $\varepsilon \leq 2$) statt $2 \cdot \log\ n$ Vergleichen auskommt, und daher schneller ist, falls das "Hochtauschen zur Wurzel" schnell implementiert wird.

Algorithmus 9 SIFT-UP$(A, i) \in O(\log n)$

Eingabe: Array A, bis evtl. auf $A[i]$ HEAP
Ausgabe: Array A als HEAP
$l \leftarrow i$
Solange $\lfloor \frac{l}{2} \rfloor > 0$ und $A[l] > A[\lfloor \frac{l}{2} \rfloor]$
  Vertausche $A[l]$ und $A[\lfloor \frac{l}{2} \rfloor]$
  $l \leftarrow \lfloor \frac{l}{2} \rfloor$

Algorithmus 10 DELETE$(A, i) \in O(\log n)$

Eingabe: HEAP A, Index i des zu löschenden Elements
Ausgabe: HEAP $A \setminus A[i]$
$A[i] \leftarrow A[|A|]$
$|A| \leftarrow |A| - 1$
Falls $A[i] \leq A[\lfloor \frac{i}{2} \rfloor]$
  HEAPIFY(A, i)
Sonst
  SIFT-UP(A, i)

Algorithmus 11 INSERT$(A, x) \in \Theta(\log n)$

Eingabe: HEAP A, einzufügendes Element x
Ausgabe: HEAP $A \cup \{x\}$
$|A| \leftarrow |A| + 1$
$A[|A|] \leftarrow x$
SIFT-UP$(A, |A|)$

Algorithmus 12 HEAPSORT$(A) \in O(n \log n)$

Eingabe: Array A
Ausgabe: Array A aufsteigend sortiert
MAKEHEAP(A)
$n \leftarrow |A|$
Für $i = n, \ldots, 2$
  Vertausche $A[1]$ und $A[i]$
  $|A| \leftarrow |A| - 1$
  HEAPIFY$(A, 1)$
$|A| \leftarrow n$

Algorithmus 13 BOTTOM-UP-HEAPIFY$(A, 1) \in O(\log n)$

Eingabe: Array A, bis auf $A[1]$ HEAP
Ausgabe: HEAP A
$j \leftarrow 1$
// Sinke entlang größerer Nachfolger bis zu einem Blatt ab
Solange $2j > |A|$
  Falls $A[2j] \geq A[2j+1]$
    $j \leftarrow 2j$
  Sonst
    $j \leftarrow 2j+1$
// Steige bis zur korrekten Position auf
Solange $A[1] \geq A[j]$
  $j \leftarrow \lfloor \frac{j}{2} \rfloor$
$temp \leftarrow A[j]$
$A[j] \leftarrow A[1]$
$j \leftarrow \lfloor \frac{j}{2} \rfloor$
Solange $j > 0$
  vertausche $temp$ und $A[j]$
  $j \leftarrow \lfloor \frac{j}{2} \rfloor$

2 Aufspannende Bäume minimalen Gewichts

2.1 Grundbegriffe der Graphentheorie

Ein **Weg** der Länge l in einem Graphen $G = (V, E)$ ist eine Folge von Knoten $v_1, \ldots, v_{l-1}$, in der aufeinanderfolgende Knoten durch Kanten verbunden sind.

Ein **Pfad** ist ein Weg in dem jeder Knoten nur einmal vorkommt.

Ein **Baum** ist ein Graph $G = (V, E)$, in dem es zwischen je 2 Knoten genau einen Pfad gibt.

Ein Teilgraph $G' = (V', E')$ von $G = (V, E)$ mit $E' \subseteq E$ heißt **aufspannend**, wenn $V' = V$.

2.2 Das MST-Problem

Finde zu einem zusammenhängenden Graphen $G = (V, E)$ mit der Gewichtsfunktion $c : E \to \mathbb{R}$ einen aufspannenden Teilbaum $B = (V, E')$ mit minimalem Gewicht $c(B) = \sum_{\{u,v\} \in E'} c(\{u, v\})$

2.3 Die Färbungsmethode von Tarjan

Ein **Schnitt** in einem Graphen $G = (V, E)$ ist eine Partition $(S, V \setminus S)$ von V.

Eine Kante $\{u, v\}$ **kreuzt** den Schnitt falls $u \in S$ und $v \in V \setminus S$. (Ein Schnitt wird oft mit der Menge der kreuzenden Kanten identifiziert.)

Ein **Kreis** in einem Graphen $G = (V, E)$ ist eine Folge $v_1, \ldots, v_k = v_1$ mit $k > 3$, in der aufeinanderfolgende Knoten durch Kanten verbunden sind und außer v_1 kein Knoten zweimal vorkommt.

Grüne Regel: Färbe die farblose Kante *minimalen* Gewichts eines *Schnittes* ohne grüne Kanten grün.

Rote Regel: Färbe die farblose Kante *maximalen* Gewichts eines *Kreises* ohne rote Kanten rot.

Algorithmus 14 Färbungsmethode von Tarjan

Eingabe: gewichteter Graph
Ausgabe: MST in Form von grünen Kanten
Solange eine der beiden Regeln anwendbar
  Wende eine der beiden Regeln an

Satz 2 (**Färbungsinvariante**). *Die Färbungsmethode erhält die Invariante, dass es einen MST gibt, der alle grünen Kanten und keine rote Kante enthält.*

Der Satz kann mit Hilfe einer Fallunterscheidung induktiv bewiesen werden:

- Grüne Regel auf $e \in E_B$ angewendet ✓
- Grüne Regel auf $e = \{u, v\} \notin E_B$ angwendet. B aufspannend $\Rightarrow \exists$ kreuzende Kante $e' \in E_B$ auf dem Weg von u nach v. $e' \in E_B$, grüne Regel $\Rightarrow e'$ ungefärbt und $c(e') \geq c(e) \Rightarrow E'_B = E_B \setminus e' \cup e$ MST ✓
- Rote Regel auf $e \notin E_B$ angewendet ✓
- Rote Regel auf $e \in E_B$ angewendet. $B \setminus e$ zerfällt in 2 Teilbäume die einen Schnitt induzieren. Auf dem Kreis der roten Regel liegt eine kreuzende, nicht-rote Kante $e' \neq e$ mit $c(e) \geq c(e')$. B Baum $\Rightarrow e'$ nicht grün $\Rightarrow E'_B = E_B \setminus e \cup e'$ MST ✓

2.4 Der Algorithmus von Kruskal

Algorithmus 15 Algorithmus von Kruskal (verbal) $\in O(|E| \log |V|)$

Eingabe: Graph mit Kantengewichten
Ausgabe: MST in Form von grünen Kanten
Sortiere Kanten nicht-absteigend
Für alle Kanten
  Falls beide Endknoten liegen in demselben grünen Baum
    Färbe Kante rot
  Sonst
    Färbe Kante grün

Der Algorithmus von Kruskal ist eine Spezialisierung der Färbungsmethode die durch Union-Find implementiert werden kann und deren Laufzeit durch das Sortieren bestimmt wird.

2.5 Der Algorithmus von Prim

Der Algorithmus von Prim ist eine weitere Spezialisierung der Färbungsmethode und besonders für dichte Graphen geeignet. Falls die Kanten bereits sortiert sind ist er dem Algorithmus von Kruskal unterlegen.

Algorithmus 16 Algorithmus von Prim (verbal)

Eingabe: Graph $G = (V, E)$ mit Kantengewichten
Ausgabe: MST in Form von grünen Kanten
Färbe einen beliebigen Knoten grün.
Für $i = 1, \ldots, |V| - 1$
  Wähle eine farblos Kante minimalen Gewichts mit genau einem grünen Endknoten und färbe sie und den anderen Endknoten grün.

Algorithmus 17 Algorithmus von Prim $\in O(|E| \log_{2+|E|/|V|}|V|)$

Eingabe: Graph $G = (V, E)$ mit Kantengewichten, Startknoten $s \in V$
Ausgabe: MST in Form von grünen Kanten
Für alle $v \in V$
  KEY$[v] \leftarrow \infty$
$v \leftarrow s$
Solange v definiert
  KEY$[v] \leftarrow -\infty$
  Für alle $\{v, w\} \in E$
    Falls KEY$[w] = \infty$
      KEY$[w] \leftarrow c(\{v, w\})$
      GRÜN$[w] \leftarrow \{v, w\}$
      INSERT(H, w)
    Sonst
      Falls $c(\{v, w\} <$ KEY$[w]$
        KEY$[w] \leftarrow c(\{v, w\})$
        GRÜN$[w] \leftarrow \{v, w\}$
        DECREASEKEY$(H, w, c(\{v, w\}))$

2.6 Greedy-Verfahren und Matroide

Ein **Unabhängigkeitsystem** ist ein Tupel $(M,\mathcal{U})$ mit $\mathcal{U} \subset 2^M$, M endlich für das gilt:

- $\emptyset \in \mathcal{U}$ und
- $\forall\ I_1 \in \mathcal{U}$: $I_2 \subseteq I_1 \Rightarrow I_2 \in \mathcal{U}$

$I \in \mathcal{U}$ heißen **unabhängig**, alle anderen $I \subseteq M$ mit $I \notin \mathcal{U}$ **abhängig**.

$B \in \mathcal{U}$ mit $B \subseteq F$ **Basis** von $F \subseteq M \ \Leftrightarrow\ \forall\ B' \in \mathcal{U} : B \subseteq B' \subseteq F \Rightarrow B = B'$ (maximal bzgl. $\subseteq$)

Eine **Basis eines Unabhängigkeitsystem** $(M,\mathcal{U})$ ist eine Basis von M.

Die Menge aller Basen von $(M,\mathcal{U})$ heißt **Basissystem** von $(M,\mathcal{U})$.

Rang von $F \subseteq M$ $r(F) := max\{|B| \mid B \text{ Basis von } F\}$ $(r((M,\mathcal{U})) = r(M))$

Ein **Kreis** in $(M,\mathcal{U})$ ist eine bzgl. $\subseteq$ minimale, abhängige Menge.

Optimierungsproblem über dem Unabhängigkeitssystem $(M,\mathcal{U})$ mit der Gewichtsfunktion w: Finde ein $U \in \mathcal{U}$ mit minimalem $w(U)$.

Optimierungsproblem über dem Basisssystem $\mathcal{B}$: Finde ein $B \in \mathcal{B}$ mit minimalem $w(B)$.

MST = Optimierungsproblem über dem Basissystem der aufspannenden Bäume.

Algorithmus 18 Greedy-Algorithmus für Optimierungsproblem über $(M,\mathcal{U})$

---

sortiere M auf- bzw. absteigend (Mini- bzw. Maximierung) zu $M = l_1, \ldots, l_n$
$I \leftarrow \emptyset$
Für $i = 1, \ldots, n$
  Falls $I \cup \{l_i\} \in \mathcal{U}$
    $I \leftarrow I \cup \{l_i\}$

Ein **Matroid** ist ein Unabhängigkeitssystem $(M,\mathcal{U})$ für das gilt:
$\forall\ I, J \in \mathcal{U}$ mit $|I| < |J|\ \exists\ e \in J \setminus I$ mit $I \cup \{e\} \in \mathcal{U}$

Satz 3 (**Matroid-Äquivalenz**). *Für ein Unabhängigkeitssystem $(M,\mathcal{U})$ sind folgende Aussagen äquivalent:*

(*a*) *Ein Greedy-Algorithmus liefert bei bel. Gewichtsfkt. w eine Optimallösung $max\{w(U) \mid U \in \mathcal{U}\}$*

(*b*) *$(M,\mathcal{U})$ ist ein Matroid*

(*c*) *Für bel. $F \subseteq M$ und bel. inklusionsmaximale, unabhängige $I_1, I_2 \subseteq F$ gilt $|I_1| = |I_2|$*

Beweis:
$(a) \Rightarrow (b)$:
Annahme $(a) \wedge \neg(b) \Rightarrow \exists\ U, W \in \mathcal{U}$ mit $|U| = |W| + 1$ und $\forall\ e \in U \setminus W$ gilt $W \cup \{e\} \notin \mathcal{U}$.

$$w(m) := \begin{cases} |W|+2 & \text{falls } e \in W \\ |W|+1 & \text{falls } e \in U \setminus W \\ -1 & \text{falls } e \notin U \setminus W \end{cases} \Rightarrow\ w(U) \geq |U|(|W|+1) = \overset{\text{2x Binomi}}{\cdots} > w(W)$$

$\Rightarrow$ Greedy-Algorithmus wählt alle $w \in W$ und kann dann nichts mehr hinzunehmen. Widerspruch zur Optimalität.

$(b) \Rightarrow (c)$:
Annahme $(a) \wedge \neq (c) \Rightarrow \exists I_1, I_2 \subseteq F \subseteq M$ mit I_1, I_2 maximal unabhängig in F und o.B.d.A. $|I_1| < |I_2|$
Konstruiere $I' \subseteq I_2$ mit $|I'| = |I_1| + 1$ durch streichen von $|I_2| - |I_1| - 1$ Elementen aus I_2.
$\mathcal{U}$ bzgl. $\subseteq$ n.u. abgeschl. $\Rightarrow I' \in \mathcal{U} \overset{(M,\mathcal{U})\ \text{Matroid}}{\Longrightarrow} \exists\ e \in I' \setminus I_1$ mit $\mathcal{U} \ni I_1 \cup \{e\} \subseteq F$. Widerspruch zu I_1 maximal unabhängig.

$(c) \Rightarrow (a)$:
Sei $I \in \mathcal{U}$ Lösung des Greedy-Algorithmus und $J \in \mathcal{U}$ Optimallösung.
I, J maximal unabhängig in $F = \{e \in M \mid w(e) > 0\} \overset{(c)}{\Rightarrow} |I| = |J|$
Ordne I und J absteigend nach Gewicht zu $i_1, \ldots, i_n$ und $j_1, \ldots, j_n$
Zeige induktiv $\forall k = 1, \ldots, n$ gilt $w(i_k) \geq w(j_k)$: IA: Greedy ✓
IS: Annahme: $w(i_k) < w(j_k) \Rightarrow \{i_1, \ldots, i_{k-1}\}$ maximal unabhängig in $F' = \{e \in M \mid w(e) \geq w(j_k)\}$ da die Greedy-Methode sonst e mit $\{i_1, \ldots, i_{k-1}, e\} \in \mathcal{U}$ gewählt hätte. Widerspruch zu (c)

da unabhängige $\{j_1, \ldots, j_k-\} \subseteq F'$ mit größerer Kardinalität.
$\Rightarrow w(I) \geq w(J)$ also I optimal.

3 Schnitte in Graphen und Zusammenhang

3.1 Schnitte minimalen Gewichts

Die **Kapazität eines Schnittes** $(S, V \setminus S)$ ist $c(S, V \setminus S) := \sum_{\{u,v\} \in E \cap S \times V \setminus S} c(\{u, v\})$.

Ein **Schnitt** $(S, V \setminus S)$ kann kürzer durch eine Menge $S \subset V$, welche den Schnitt **induziert**, angegeben werden.

Ein **minimaler Schnitt** ist ein Schnitt $(S, V \setminus S)$ für den $c(S, V \setminus S) \leq c(S', V \setminus S')$ für alle nichttrivialen $S' \subsetneq V$ ist.

Min-Cut-Problem: Finde zu einem Graphen $G = (V, E)$ mit Gewichtsfkt. $c : E \to \mathbb{R}_0^+$ einen minimalen Schnitt.

Zu $S \subseteq V$ nennen wir den **am stärksten mit** S **verbundenen** Knoten, denjenigen Knoten $v \in V \setminus S$ für den $c(S, v) := \sum_{\{u,v\} \in E \cap S \times V} c(\{u, v\})$ maximal wird.

Wir **verschmelzen 2 Knoten** $s, t \in V$ indem wir sie durch einen neuen Knoten $x_{s,t}$ ersetzen und ihre Kanten durch Kantenmit $x_{s,t}$ ersetzen und dabei gegebenenfalls Gewichte addieren.

3.2 Der Algorithmus von Stoer & Wagner

Algorithmus 19 Min-Schnitt-Phase$(G, c, a) \in O(|V| \log |V| + |E|)$

Eingabe: Graph $G_i = (V_i, E_i)$, Gewichtsfkt. c, Startknoten $a \in V$
Ausgabe: Graph G_{i+1} und Schnitt der Phase $(V_i \setminus \{t\}, \{t\})$
$S \leftarrow \{a\}$
$t \leftarrow a$
Solange $S \neq V_i$
  Bestimme am stärksten verbundenen $v \in V_i \setminus S$ // $c(S, v)$ maximal
  $S \leftarrow S \cup \{v\}$
  $s \leftarrow t$
  $t \leftarrow v$
Speichere $(V_i \setminus \{t\}, \{t\})$ als Schnitt der Phase
$G_{i+1} \leftarrow G_i$ mit verschmolzenem s, t

Algorithmus 20 Min-Schnitt$(G, c, a) \in O(|V|^2 \log |V| + |V||E|)$

Eingabe: Graph $G = (V, E)$, Gewichtsfkt. c, Startknoten $a \in V$
Ausgabe: Minimaler Schnitt
$G_1 \leftarrow G$
Für $i = 1, \ldots, |V| - 1$
  Min-Schnitt-Phase(G_i, c, a)
  Falls Schnitt der Phase $i <$ Min-Schnitt
    Min-Schnitt $\leftarrow$ Schnitt der Phase i
Gebe Min-Schnitt aus

Implementiere Min-Schnitt-Phase mit Hilfe eines Fibonacci-Heaps dessen Schlüssel immer die aktuellen Werte $c(S, v)$ zur aktuellen Menge S sind. Damit gelingt die Knotenbestimmung

in $O(\log |V|)$ und die Verschmelzung in $O(|E|)$ (da INCREASE-KEY $\in O(1)$). Somit ist der Algorithmus von Stoer & Wagner etwas effizienter als $|V| - 1$-maliges Anwenden des effizientesten Flussalgorithmus.

Korrektheitsbeweis: FOLGT NOCH ....

4 Flussprobleme und Dualität

4.1 Grundlagen

Ein **Netzwerk** ist ein Tupel $(D; s, t; c)$, wobei $D = (V, E)$ ein gerichteter Graph mit Kantenkapazitäten $c : E \to \mathbb{R}_0^+$ und einer Quelle $s \in V$ und einer Senke $t \in V$ ist.

$f : E \to \mathbb{R}_0^+$ heißt **Fluss** falls $\forall\ (v, w) \in E$ die Kapazitätsbedingung $0 \leq f(v, w) \leq c(v, w)$ und $\forall\ v \in V \setminus \{s, t\}$ die Flusserhaltungsbedingung $\sum\limits_{\{w|(v,w)\in E\}} f(v, w) \ - \sum\limits_{\{w|(w,v)\in E\}} f(w, v) = 0$ gilt.

Lemma 2 (**Quellen-Senken-Fluss**). *Für einen Fluss f in einem Netwerk $(D; s, t; c)$ gilt:*
$\sum\limits_{(s,i)\in E} f(s, i) - \sum\limits_{(i,s)\in E} f(i, s) = \sum\limits_{(i,t)\in E} f(i, t) - \sum\limits_{(t,i)\in E} f(t, i)$

Der **Wert** eines Flusses f ist $w(f) := \sum\limits_{(s,i)\in E} f(s, i) - \sum\limits_{(i,s)\in E} f(i, s)$

Ein **Maximalfluss** in $(D; s, t; c)$ ist ein Fluss f für den $w(f) \geq w(f')$ für alle Flüsse f' in $(D; s, t; c)$ ist.

Max-Flow-Problem: Finde in einem Netzwerk $(D; s, t; c)$ einen Maximalfluss.

Ein Schnitt $(S, V \setminus S)$ heißt **s-t-Schnitt** falls $s \in S$ und $t \in V \setminus S$.

Lemma 3 (**Schnitt-Lemma**). *$\forall$ s-t-Schnitte $(S, V \setminus S)$ in einem Netzwerk $(D; s, t; c)$ gilt für jeden Fluss f: $w(f) = \sum\limits_{(i,j)\in E\cap S\times V\setminus S} f(i, j) - \sum\limits_{(i,j)\in E\cap V\setminus S\times S} f(i, j) \leq c(S, V \setminus S)$*

Beweis durch Addition aller Flussdifferenzen $= 0$ (Flusserhaltung) und Abschätzung durch Kapazitätsbedingung.

Zu einem Fluss f in einem Netzwerk $(D; s, t; c)$ heißen alle von s nach t gerichteten Kanten eines Weges von s nach t **Vorwärtskanten** und alle anderen Kanten des Weges **Rückwärtskanten**.

Ein **erhöhender Weg** ist ein Weg von s nach t für den für jede Vorwärtskante $f(i, j) < c(i, j)$ und für jede Rückwärtskante $f(i, j) > 0$ gilt.

Satz 4 (**vom erhöhenden Weg**). *Ein Fluss f in einem Netzwerk $(D; s, t; c)$ ist genau dann ein Maximalfluss, wenn es keinen erhöhenden Weg gibt.*

$\Rightarrow$: Annahme: $\exists$ erhöhender Weg W, erhöhe f um kleinste Kapazität auf W. Widerspruch zur Maximalität.

$\Leftarrow$: $\nexists$ erhöhender Weg $\Rightarrow$ Menge aller Knoten, zu denen erhöhende Wege existieren, induziert Schnitt, dessen kreuzende Vorwärtskanten saturiert, und dessen kreuzende Rückwärtskanten leer sind $\overset{Schnitt-Lemma}{\Longrightarrow} w(f) = c(S, V \setminus S)$ also maximal.

Satz 5 (**Max-Flow Min-Cut Theorem**). *In einem Netzwerk $(D; s, t; c)$ ist der Wert eines Maximalflusses gleich der Kapazität eines minimalen s-t-Schnittes.*

Beweis: Für einen Maximalfluss f und die Menge S aller Knoten, zu denen erhöhende Wege existieren, und alle Flüsse f' und alle $S' \subset V$ gilt:

$$w(f') \overset{f \text{ min.}}{\leq} w(f) \overset{\text{Satz 4}}{=} c(S, V \setminus S) \overset{\text{Schnitt-Lemma}}{\leq} c(S', V \setminus S')$$

Satz 6 (Ganzzahligkeitssatz). *In jedem Netzwerk $(D; s, t; c)$ mit ganzzahligen Kantenkapazitäten $c : E \to \mathbb{N}_0$ gibt es einen ganzzahligen Maximalfluss ($\forall\ (i,j) \in E : f(i,j) \in \mathbb{N}_0 \Rightarrow w(f) \in \mathbb{N}_0$*

Beweis durch Iteration über erweiternde Pfade, da jeder erweiternde Pfad ganzzahlig ist.

4.2 Bestimmung maximaler Flüsse

Algorithmus 21 MAX-FLOW$(D; s, t; c)$

Eingabe: Netzwerk $(D; s, t; c)$
Ausgabe: Maximalfluss f von s nach t
Für alle $(i, j) \in E$
  $f(i, j) \leftarrow 0$
Solange erhöhender Weg $W = e_1, \ldots, e_k$ existiert
  $\Delta \leftarrow min(\{c(e_i) - f(e_i) \mid e_i \text{ VwK in } W\} \cup \{f(e_i) \mid e_i \text{ RwK in } W\})$
  Für alle $e_i \in W$
    Falls e_i VwK
      $f(e_i) \leftarrow f(e_i) + \Delta$
    Sonst
      $f(e_i) \leftarrow f(e_i) - \Delta$

4.2.1 Algorithmus von Ford-Fulkerson

Die Laufzeit des Algorithmus von Ford Fulkerson hängt von der Wahl von v und dem maximalen Kantengewicht ab. Bei nicht-rationalen Kantengewichten ist es möglich, dass er nicht terminiert.

4.2.2 Der Algorithmus von Edmonds und Karp

Der Algorithmus von Edmonds und Karp ist eine Optimierung des Algorithmus von Ford Fulkerson: Er wählt unter allen unbesuchten $v \in S$ den Knoten, welcher am längsten in S ist (Breitensuche) und kann daher in $O(|V||E|^2)$ implementiert werden.

4.2.3 Der Algorithmus von Goldberg und Tarjan

Der Algorithmus von Goldberg und Tarjan ist der effizienteste bekannte Algorithmus zur Maximalfluss-Bestimmung.

Vereinfache das Netzwerk durch die Antisymmetrie-Forderung $\forall (v, w) \in V \times V : f(v, w) = -f(w, v)$ und führe dazu vorhandene Rückwärtskanten mit ihren Vorwärtskanten zusammen und vervollständige E zu E' um bisher nicht vorhandene Rückwärtskanten mit Kantengewicht 0.

Algorithmus 22 Algorithmus von Ford-Fulkerson

Eingabe: Netzwerk $(D; s, t; c)$
Ausgabe: Maximalfluss f von s nach t und minimaler s-t-Schnitt $(S, V \setminus S)$
Für alle $(i, j) \in E$
  $f(i, j) \leftarrow 0$
$S \leftarrow \{s\}$
Für alle $v \in V$
  $\Delta[v] \leftarrow \infty$
  BESUCHT$[v] \leftarrow false$
Solange $\exists\ v \in S$ mit BESUCHT$[v] = false$
  // Bestimme erhöhenden Pfad
  Für alle $(v, w) \in E$ mit $w \notin S$
    Falls $f(v, w) < c(v, w)$
      VOR$[w] \leftarrow +v$ // VwK-Vorgänger
      $\Delta[w] \leftarrow min\{c(v, w) - f(v, w), \Delta[v]\}$
      $S \leftarrow S \cup \{w\}$
  Für alle $(w, v) \in E$ mit $w \notin S$
    Falls $f(w, v) > 0$
      VOR$[w] \leftarrow -v$ // RwK-Vorgänger
      $\Delta[w] \leftarrow min\{f(w, v), \Delta[v]\}$
      $S \leftarrow S \cup \{w\}$
  BESUCHT$[v] = true$
  Falls $t \in S$
    // Erhöhe Fluss entlang Pfad
    $w \leftarrow t$
    Solange $w \neq s$
      Falls VOR$[w] > 0$
        $f(\text{VOR}[w], w) \leftarrow f(\text{VOR}[w], w) + \Delta[t]$
      Sonst
        $f(w, -\text{VOR}[w]) \leftarrow f(w, -\text{VOR}[w]) - \Delta[t]$
    $S \leftarrow \{s\}$
    Für alle $v \in V$
      $\Delta[v] \leftarrow \infty$
      BESUCHT$[v] \leftarrow false$
Gebe f und $(S, V \setminus S)$ aus

$f : E' \to \mathbb{R}$ heißt dadurch nun **Fluss** falls es die Antisymmetrie-Bedingung erfüllt, und falls $\forall\ (v,w) \in E'$ die Kapazitätsbedingung $f(v,w) \leq c(v,w)$ und $\forall\ v \in V \setminus \{s,t\}$ die Flusserhaltungsbedingung $\sum\limits_{u \in V} f(u,v) = 0$ gilt.

Der **Wert** eines Flusses f ist nun $w(f) := \sum\limits_{v \in V} f(s,v) = \sum\limits_{v \in V} f(v,t)$

$f : E' \to \mathbb{R}$ heißt ein **Präfluss** wenn es die Kapazitäts- und Antisymmetriebedingung erfüllt und $\forall\ v \in V \setminus \{s\}$ gilt $\sum\limits_{u \in V} f(u,v) \geq 0$ (es fließt mindestens soviel in v hinein wie heraus).

(Alle folgenden Definitionen sind immer bzgl. eines Präflusses f zu verstehen.)

Der **Flussüberschuss** eines Knoten $v \in V \setminus \{t\}$ ist $e(v) := \sum\limits_{u \in V} f(u,v)$.

Als **Restkapazität** definieren wir $r_f : E' \to \mathbb{R}$ mit $\forall (v,w) \in E' : r_f(v,w) := c(v,w) - f(v,w)$

Die **Residualkanten** sind $E_f := \{(v,w) \in E' \mid r_f(v,w) > 0\}$

Als **Residualgraph** bezeichnen wir $D_f(V, E_f)$

$d : V \to \mathbb{N}_0 \cup \{\infty\}$ heißt **zulässige Markierung** falls $d(s) = |V|, d(t) = 0$ und $\forall v \in V \setminus \{t\}, (v,w) \in E_f$ gilt $d(v) \leq d(w) + 1$

$v \in V \setminus \{t\}$ heißt **aktiv** wenn $e(v) > 0$ und $d(v) < \infty$

Push(v,w) ist zulässig falls v aktiv ist, $r_f(v,w) > 0$ und $d(v) = d(w) + 1$.

Algorithmus 23 Push(v,w)

Eingabe: $v, w \in V$ mit v aktiv, $r_f(v,w) > 0$ und $d(v) = d(w) + 1$
Seiteneffekte: Flussüberschuss von v wird über (v,w) zu w geleitet
$\Delta \leftarrow min\{e(v), r_f(v,w)\}$
$f(v,w) \leftarrow f(v,w) + \Delta$
$f(w,v) \leftarrow f(w,v) - \Delta$
(Berechne Restkapazitäten und Überschuss neu)

Relabel(v) ist zulässig falls v aktiv ist und falls *kein* $w \in V$ mit $r_f(v,w) > 0$ und $d(v) = d(w)+1$ *existiert.*

Algorithmus 24 Relabel(v)

Eingabe: $v \in V$ mit v aktiv und $\forall w \in V$ mit $r_f(v,w) > 0$ gilt $d(v) \leq d(w)$
Seiteneffekte: $d(v)$ wird erhöht

$$d(v) := \begin{cases} \infty & \text{falls } \{w \mid r_f(v,w) > 0\} = \emptyset \\ min\{d(w) + 1 \mid r_f(v,w) > 0\} & \text{sonst} \end{cases}$$

Für den Algorithmus von Goldberg und Tarjan gilt, dass falls für $v \in V$ $d(v) < |V|$ ist, $d(v)$ eine untere Schranke für den Abstand von v zu t in D_f ist. Analog ist für $d(v) > |V|$ t in D_f von v aus unerreichbar und $d(v) - |V|$ eine untere Schranke für den Abstand von v zu s in D_f.

Algorithmus 25 Algorithmus von Goldberg und Tarjan

Eingabe: Netzwerk $(D; s, t; c)$
Ausgabe: Maximalfluss f von s nach t
Für alle $(v, w) \in E'$
  $f(v, w) \leftarrow 0$
  $r_f(v, w) \leftarrow c(v, w)$
$d(v) \leftarrow |V|$
Für alle $v \in V \setminus \{s\}$
  $f(s, v) \leftarrow c(s, v), r_f(s, v) \leftarrow 0$
  $f(v, s) \leftarrow -c(s, v), r_f(v, s) \leftarrow c(v, s) - f(v, s)$
  $d(v) \leftarrow 0$
  $e(v) \leftarrow c(s, v)$
Solange aktiver Knoten $v \in V$ existiert
  Führe zulässiges PUSH(v,w) oder RELABEL(v) aus
Gebe f aus

Lemma 4. *Für einen aktiven Knoten $v \in V$ zu einem Präfluss f und einer zulässigen Markierung d ist entweder* PUSH *oder* RELABEL *zulässig.*

Beweis: Die Zulässigkeits-Bedingung von RELABEL ist das Negat der Zulässigkeits-Bedingung von PUSH.

Lemma 5. *Während des Algorithmus von Goldberg und Tarjan ist f stets ein Präfluss und d stehts eine zulässige Markierung.*

Beweis: Das Lemma folgt durch Induktion über die Anzahl der Operationen und Fallunterscheidung zwischen den Operationen unmittelbar wegen der Forderungen für zulässige Operationen.

Lemma 6. *In einem Residualgraph D_f zu einem Präfluss f und zulässigem d ist t von s aus unerreichbar.*

Beweis: Jeder Weg von s nach t würde wegen $d(v) \leq d(w)$ für Kanten (v, w) auf dem Weg und $d(t) = 0$ zu einem Widerspruch zu $d(s) = |V|$ führen.

Satz 7 (**Maximalität des Algorithmus von Goldberg und Tarjan**). *Falls der Algorithmus von Goldberg und Tarjan mit endlichen Markierungen terminiert ist der konstruierte Präfluss ein Maximalfluss.*

Beweis: Algorithmus terminiert $\overset{\text{Lemma 4}}{\Longrightarrow}$ keine Knoten mehr aktiv. Alle Markierungen endlich $\Rightarrow$ alle Überschüsse $0 \Rightarrow f$ Fluss. Lemma 6 $\Rightarrow$ kein erhöhender s-t-Weg.

Lemma 7. *Falls für einen Präfluss f und ein $v \in V$ $e(v) > 0$ gilt ist s in D_f von v aus erreichbar.*

Beweis: Sei S_v die Menge aller von v in D_f erreichbaren Knoten. Von unerreichbaren $u \in V \setminus S_v$ zu erreichbaren $w \in S_v$ fließt nichts positives: $0 = r_f(w, u) = c(w, u) - f(w, u) \geq 0 + f(u, w)$ $(*)$

$$\sum_{w \in S_v} e(w) = \sum_{w \in S_v, u \in V} f(u, w) = \sum_{w \in S_v, u \in V \setminus S_v} f(u, w) + \underbrace{\sum_{u, w \in S_v} f(u, w)}_{=0} \overset{(*)}{\leq} 0$$

f Präfluss $\Rightarrow \sum_{w \in S_v \setminus \{s\}} e(w) \geq 0 \overset{e(v) > 0}{\Longrightarrow} s \in S_v$

Lemma 8. *Während des Algorithmus gilt* $\forall\ v \in V\ d(v) \leq 2|V| - 1$.

Beweis: Induktion über RELABEL-Operationen: IA: ✓
IS: RELABEL(v) zulässig $\Rightarrow e(v) > 0 \overset{\text{Lemma 7}}{\Longrightarrow} \exists$ v-s-Weg der Länge $l \leq |V| - 1 \overset{\text{d zulässig}}{\Longrightarrow} d(v) \leq d(s) + l \leq 2|V| - 1$

Lemma 9. *Es werden je* $v \in V$ *höchstens* $2|V| - 1$ RELABEL(v) *also insgesamt höchstens* $2|V|^2$ RELABEL *ausgeführt.*

Beweis: RELABEL erhöht $d(v) \overset{\text{Lemma 8}}{\Longrightarrow}$ Behauptung

Ein PUSH(v, w) heißt **saturierend** falls danach $r_f(v, w) = 0$ gilt.

Lemma 10. *Es werden höchstens* $2|V||E|$ *saturierende* PUSH *ausgeführt.*

Beweis: Zulässigkeit $\Rightarrow$ zwischen einem saturierendem und dem nächsten PUSH(v, w) muss $d(w)$ um mindestens 2 erhöht werden. Nach dem letzten PUSH(v, w) gilt nach Lemma 8 $d(v) + d(w) \leq 4|V| - 2$. Also können höchstens $2|V|$ saturierende PUSH(v, w) stattgefunden haben. Also insgesamt höchstens $2|V||E|$ saturierende PUSH.

Lemma 11. *Es werden höchstens* $4|V|^2|E|$ *nicht saturierende* PUSH *ausgeführt.*

Beweis: Da für jedes nicht-saturierende PUSH(v, w) v inaktiv und evtl. w mit $d(w) = d(v) - 1$ aktiv wird erniedrigt es $D := \sum_{v \in V \setminus \{s,t\} \text{aktiv}} d(v)$ um mindestens 1. Da das aktivierte w Lemma 8 erfüllt erhöht jedes saturierende PUSH(v, w) D um höchstens $2|V| - 1 \overset{\text{Lemma 10}}{\Longrightarrow}$ saturierenden PUSH erhöhen D um höchstens $(2|V| - 1)(2|V||E|)$. Lemma 9 $\Rightarrow D$ kann durch RELABEL um höchstens $(2|V| - 1)|V|$ erhöht werden. Da D nur so stark verringert wie erhöht werden kann ergeben sich insgesamt höchstens $4|V|^2|E|$ nicht-saturierende PUSH.

Satz 8 (**Termination des Algorithmus von Goldberg und Tarjan**). *Der Algorithmus von Goldberg und Tarjan terminiert nach höchstens* $O(|V|^2|E|)$ *zulässigen* PUSH *oder* RELABEL*-Operationen.*

Beweis: Vorangehende Lemmata.

Die tatsächliche Laufzeit des Algorithmus von Goldberg und Tarjan ist stark von der Wahl der aktiven Knoten und der "zu pushenden" Kanten abhängig:

FIFO-Implementation $\in O(|V|^3)$ (mit dynam. Bäumen $O(|V||E| \log \frac{|V|^2}{|E|})$)

Highest-Label-Implementation $\in O(|V|^2|E|^{1/2})$

Excess-Scaling-Implementation $\in O(|E| + |V|^2 \log C)$ (mit C maximales Kantengewicht und für PUSH(v, w) $e(v)$ "groß" und $e(w)$ klein)

5 Kreisbasen minimalen Gewichts

5.1 Kreisbasen

Ein **Kreis** ist ein Teilgraph $C = (V_C, E_C)$ von $G = (V, E)$ in dem alle Knoten geraden Grad haben.

Ein **einfacher Kreis** ist ein zusammenhängender Teilgraph $C = (V_C, E_C)$ von $G = (V, E)$ in dem alle Knoten Grad 2 haben.

Wir identifizieren einen Kreis mit seiner Kantenmenge und schreiben ihn als $|E|$-dimensionalen **Vektor** über $\{0, 1\}$.

Die Menge aller Kreise induziert einen **Kreisraum** genannten Vektorraum $\mathcal{C}$ dessen Addition die **symmetrische Differenz** $\oplus$ der Kantenmengen ist.

Die Begriffe **Dimension, linear (un)abhängig** und **Basis** ergeben sich vollkommen kanonisch.

Eine **Fundamentalbasis** eines zusammenhängenden Graphen kann aus einem aufspannenden Baum T konstruiert werden indem jede Nichtbaumkante $\{v, w\}$ um den eindeutigen Weg in T von v nach w zu einem **Fundamentalkreis** ergänzt wird.

$\dim(\mathcal{C}) = |E| - |V| + \mathcal{K}(G)$ mit $\mathcal{K}(G) =$ Anzahl der Zusammenhangskomponenten von G

Gewicht einer Kreisbasis $\mathcal{B}$ ist $w(\mathcal{B}) := \sum_{C \in \mathcal{B}} w(C) = \sum_{C \in \mathcal{B}} \sum_{e \in C} w(w)$

Minimum Cycle Basis-Problem: Finde zu einem Graphen $G = (V, E)$ und einer Gewichtsfunktion $c : E \to \mathbb{R}_0^+$ eine Kreisbasis von G minimalen Gewichts.

Jede MCB von G enthält zu jeder Kante einen Kreis minimalen Gewichts, der diese Kante enthält.

5.2 Das Kreismatroid

$(\mathcal{C},\mathcal{U})$ mit $\mathcal{U} := \{U \subseteq C \mid U \text{ linear unabhängig}\}$ ist ein Unabhängigkeitssystem.

Satz 9 (**Austauschsatz von Steinitz**). *Eine Basis B eines endlichen Vektorraums V ist nach dem Austausch von beliebig vielen ihrer Vektoren mit geeignet gewählten Vektoren einer linearen unabhängigen Teilmenge von V immer noch eine Basis.*

Kein Beweis im Skript angegeben.

Satz 10 (**Kreismatroid**). $(\mathcal{C},\mathcal{U})$ *ist ein Kreismatroid von G genannter Matroid.*

Beweis: Der Austauschsatz von Steinitz liefert direkt die nötige Austauscheigenschaft.

Da die Anzahl der Kreise in einem Graphen exponentiell in $|V| + |E|$ sein kann ist dieser Greedy-Algorithmus nicht polynomiell.

Algorithmus 26 MCB-Greedy-Algorithmus($\mathcal{C}$)

Eingabe: Menge $\mathcal{C}$ aller Kreise in $G = (V, E)$
Ausgabe: MCB von G
Sortiere $\mathcal{C}$ aufsteigend nach Gewicht
$\mathcal{B} \leftarrow \emptyset$
Für $i = 1, \ldots, |C|$
  Falls $\mathcal{B} \cup \{C_i\}$ linear unabhängig
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{C_i\}$

5.3 Der Algorithmus von Horton

Lemma 12. *Falls $C = C_1 \oplus C_2 \in \mathcal{B}$ und $\mathcal{B}$ eine Kreisbasis ist, dann ist entweder $\mathcal{B} \setminus \{C\} \cup \{C_1\}$ oder $\mathcal{B} \setminus \{C\} \cup \{C_2\}$ auch eine Kreisbasis.*

Beweis: Falls C_1 durch $\mathcal{B} \setminus \{C\}$ darstellbar ist, ist $\mathcal{B} \setminus \{C\} \cup \{C_2\}$ eine Basis. Andernfalls ist C_2 durch $\mathcal{B} \setminus \{C\}$ darstellbar und damit $\mathcal{B} \setminus \{C\} \cup \{C_1\}$ eine Basis.

Lemma 13. *Für $x, y \in V$ und einen Weg P von x nach y kann jeder Kreis C einer Kreisbasis $\mathcal{B}$ von G, der x und y enthält, durch einen Kreis C', der P enthält, ersetzt werden.*

Beweis: Folgt direkt aus dem vorherigen Lemma 12 und der Tatsache, dass $C_1 = C_2 - C = C_2 \oplus C$.

Lemma 14. *Jeder Kreis C einer MCB $\mathcal{B}$, der zwei Knoten $x, y \in V$ enthält, enthält auch einen kürzesten Weg zwischen x und y.*

Beweis: Annahme: Beide Wege zwischen x und y in C sind keine kürzesten Wege sondern P. Lemma 13 $\Rightarrow$ Erzeugung einer Basis geringeren Gewichts durch einen Kreis C' der P enthält möglich. Widerspruch zur Minimalität von $\mathcal{B}$.

Satz 11 (**Satz von Horton**). *Für jeden Kreis C einer MCB und jeden Knoten v auf C existiert eine Kante $\{u, w\} \in C$ so dass $C = SP(u, v) + SP(w, v) + \{u, w\}$.*

Beweis: Indiziere die Knoten eines beliebigen Kreises C entlang ihrer Kanten zu $v, v_2, \ldots, v_n, v$. Sei Q_i jeweils der Weg auf C von v über v_2 usw. nach vi und P_i jeweils der Weg auf C von v_i über v_{i+1} usw. nach v. Lemma 14 $\Rightarrow$ Q_i oder P_i kürzester Weg. Sei nun k der größte Index für den Q_k kürzester Weg von v nach v_k ist $\Rightarrow$ $C = Q_k \oplus \{v_k, v_{k+1}\} \oplus P_{k+1}$

Algorithmus 27 Algorithmus von Horton $\in O(|V||E|^3)$

Eingabe: $G = (V, E)$
Ausgabe: MCB von G
$\mathcal{H} \leftarrow \emptyset$
Für alle $v \in V$ und $\{u, w\} \in E$
  Berechne $C_v^{uw} = SP(u, v) + SP(w, v) + \{u, w\}$
  Falls C_v^{uw} einfacher Kreis
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{C_v^{uw}\}$
Gebe das Ergebnis von MCB-Greedy-Algorithmus($\mathcal{H}$) zurück

Durch schnelle Matirzen-Multiplikation kann die Laufzeit des Algorithmus von Horton auf $O(|V||E|^\omega)$ mit $\omega < 2,376$ reduziert werden.

5.4 Der Algorithmus von de Pina

Algorithmus 28 Algorithmus von de Pina $\in O(|E|^3 + |E||V|^2 \log |V|)$

Eingabe: $G = (V, E)$, Nichtbaumkanten $e_1, \ldots, e_N$
Ausgabe: MCB von G
Für $j = 1, \ldots, N$
  $S_{1,j} \leftarrow \{e_j\}$
Für $k = 1, \ldots, N$
  $C_k \leftarrow$ kürzester Kreis, der eine ungerade Anzahl von Kanten aus $S_{k,k}$ enthält
  Für $j = k+1, \ldots, N$

$$S_{k+1,j} \leftarrow \begin{cases} S_{k,j} & \text{falls } C_k \text{ eine gerade Anzahl Kanten aus } S_{k,j} \text{ enthält} \\ S_{k,j} \oplus S_{k,k} & \text{sonst} \end{cases}$$

Gebe $\{C_1, \ldots, C_N\}$ aus

Wir formulieren eine algebraische Version des Algorithmus um ihn schneller auszuführen und seine Korrektheit zu beweisen.

Dazu stellen wir Kreise nun durch ihre Nichtbaumkanten, also als $|E| - |V| + \mathcal{K}(G)$-dimensionale Vektoren über $\{0, 1\}$ dar.

Auch die "Zeugen" $S_{k,k}$ stellen wir also solche Vektoren dar, was uns die Verwendung der Bilinearform $\langle C, S\rangle = \bigoplus_{i=1}^{N} (c_i \odot s_i)$ erlaubt.

Da diese Bilinearform nicht postitiv definit ($\langle x, x\langle \not\Rightarrow x = 0$) ist sie kein Skalarprodukt. Da wir das gar nicht benötigen sprechen wir dennoch von Orthogonalität.

$$\langle C, S\rangle = \begin{cases} 0 & \Leftrightarrow C \text{ und } S \text{ orthogonal} \\ 1 & \Leftrightarrow C \text{ hat eine ungerade Anzahl Einträge mit } S \text{ gemeinsam} \end{cases}$$

Algorithmus 29 Algorithmus von de Pina algebraisch

Eingabe: $G = (V, E)$, Nichtbaumkanten $e_1, \ldots, e_N$
Ausgabe: MCB von G
Für $j = 1, \ldots, N$
  $S_j \leftarrow \{e_j\}$
Für $k = 1, \ldots, N$
  $C_k \leftarrow$ kürzester Kreis mit $\langle C_k, S_k\rangle = 1$
  Für $j = k+1, \ldots, N$
    Falls $\langle C_k, S_j\rangle = 1$
      $S_j \leftarrow S_j \oplus S_k$
Gebe $\{C_1, \ldots, C_N\}$ aus

Lemma 15. *Der Algorithmus von de Pina erhält die Invariante* $\forall\ 1 \le i \le j \le N : \langle C_i, S_{j+1}\rangle = 0$.

Beweis: Induktive Fallunterscheidung unter Ausnutzung der Bilinearität.

Das vorangegange Lemma erlaubt es uns den Algorithmus von de Pina weiter zu vereinfachen zum Simple MCB-Algorithmus.

Algorithmus 30 SIMPLE MCB

Eingabe: $G = (V, E)$, Nichtbaumkanten $e_1, \ldots, e_N$
Ausgabe: MCB von G
$S_1 \leftarrow \{e_1\}$
$C_1 \leftarrow$ kürzester Kreis mit $\langle C_1, S_1 \rangle = 1$
Für $k = 2, \ldots, N$
  $S_k \leftarrow$ beliebige, nichttriviale Lsg. des Systems $\langle C_i, X \rangle_{i=1,\ldots,k-1} = 0$
  $C_k \leftarrow$ kürzester Kreis mit $\langle C_k, S_k \rangle = 1$
Gebe $\{C_1, \ldots, C_N\}$ aus

Satz 12 (**Korrektheit des Algorithmus von de Pina**). *Der* SIMPLE MCB*-Algorithmus, und damit auch der Algorithmus von de Pina, berechnen eine MCB.*

Beweis: Lemma 15 $\Rightarrow \{C_1, \ldots, C_N\}$ linear unabhängig und damit Basis.
Angenommen $\{C_1, \ldots, C_N\}$ sei keine MCB, aber $\mathcal{B}$. Sei i minimaler Index mit $\{C_1, \ldots, C_i\} \subseteq \mathcal{B}$ und für alle MCB $\mathcal{B}'$ $\{C_1, \ldots, C_i, C_{i+1}\} \nsubseteq \mathcal{B}'$. $\mathcal{B}$ Basis $\Rightarrow \exists\, D_1, \ldots, D_l \in \mathcal{B}$ mit $C_{i+1} = D_1 \oplus \ldots \oplus D_l$. $\langle C_{i+1}, S_{i+1} \rangle = 1 \Rightarrow \exists\, D_j \in \{D_1, \ldots, D_l\}$ mit $\langle D_j, S_i \rangle = 1$. C_i+1 kürzester Kreis mit $\langle C_{i+1}, S_{i+1} \rangle = 1 \Rightarrow w(C_{i+1}) \leq w(D_j)$. $\mathcal{B}^* := \mathcal{B} \setminus \{D_j\} \cup \{D_{i+1}\}$ MCB da $w(\mathcal{B}^*) \leq w(\mathcal{B})$. Invariante aus Lemma 15 $\Rightarrow D_j \notin \{C_1, \ldots, C_i\} \Rightarrow \{C_1, \ldots, C_i, C_{i+1}\} \subseteq \mathcal{B}^*$ im Widerspruch zur Wahl von i.

Die Laufzeit des Algorithmus von de Pina kann u.A. durch eine verzögerte Aktualisierung der S_j auf $O(|E|^2|V| + |E||V|^2 \log |V|)$ reduziert werden.

Die Laufzeit kann durch eine Beschränkung der Wahl von C_k auf $\mathcal{H}$ aus dem Algorithmus von Horton empirisch weiter reduziert werden.

5.5 Ein MCB-Zertifikat

Algorithmus 31 MCB-CHECKER

Eingabe: $G = (V, E)$, Kreise $C_1, \ldots, C_N$
Ausgabe: Zertifikat zur Prüfung ob $\{C_1, \ldots, C_N\}$ eine MCB von G ist
$\{e_1, \ldots, e_N\} \leftarrow$ Nichtbaumkanten eines aufspannenden Waldes
$C \leftarrow (\tilde{C_1} \ldots \tilde{C_N})$ wobei $\tilde{C_i}$ der Inzidenzvektor von C_i mit $\{e_1, \ldots, e_N\}$ ist
Gebe C^{-1} aus

C invertierbar $\Leftrightarrow \{C_1, \ldots, C_N\}$ linear unabhängig, also Basis.

Lemma 16. *Für den Unterraum $S = (S_1 \ldots S_k)$ mit $S_1, \ldots, S_k \in \{0,1\}^N$ linear unabhängig gilt $S^\perp = \mathcal{L}(\langle S_i, X \rangle_{i=1,\ldots,k} = 0)$ mit $dim(S^\perp) = N - k$.*

Lemma 17. *Für linear unabhängige $S_1, \ldots, S_n$ und $C_1, \ldots, C_N$ mit C_i jeweils kürzester Kreis mit $\langle S_i, C_i \rangle = 1$ ist $\{C_1, \ldots, C_N\}$ eine MCB.*

Lemma 18. *Für ein $A = (A_1 \ldots A_N)$ mit $A_1, \ldots, A_N \in \{0,1\}^N$, $\begin{pmatrix} S_1 \\ \vdots \\ S_N \end{pmatrix} = A^{-1}$ und A_i jeweils kürzester Kreis mit $\langle S_i, A_i \rangle = 1$ gilt für jede Kreisbasis $\mathcal{B}$ $\sum_{i=1}^{N} w(A_i) \leq w(\mathcal{B})$.*

6 Lineare Programmierung

Standardform-Bestimmung eines LP: Bestimme zu gegebenen $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n$ ein $x \in \mathbb{R}^n$, so dass $\sum_{i=1}^{n} c_i \cdot x_i = c^T x$ minimiert (bzw. maximiert) wird unter den Nebenbedingungen $Ax \geq b$ (bzw. $Ax \leq b$) und $x \geq 0$.

Überführung in Standardform: $= \rightsquigarrow \leq$ und $\geq$. $\geq \rightsquigarrow \ - \leq -$. $x_i < 0$ kann durch $x_i' - x_i''$ mit $x_i' \geq 0$ und $x_i'' \geq 0$ erreicht werden.

Zu einem **primalen Programm** $P : \min(c^T x)$ unter $Ax \geq b$ und $x \geq 0$ mit $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n$ heißt $D : \max(y^T b)$ unter $y^T A \leq c^T$ und $y \geq 0$ mit $y \in \mathbb{R}^m$ das **duale Programm**.

Satz 13 (**Schwacher Dualitätssatz**). *Für alle $x_0 \in \mathbb{R}^n, y_0 \in \mathbb{R}^m$ welche die Nebenbedingungen des primalen Programms P und des zugehörigen dualen Programms D erfüllen gilt: $y_0^T b \leq c^T x_0$.*

Beweis: $y_0^T b \leq y_0^T (Ax_0) = (y_0^T A) x_0 \leq c^T x_0$

6.1 Geometrische Repräsentation von LPs

$P \subseteq \mathbb{R}^n$ **konvex** falls $\forall\ s, t \in P, 0 < \lambda < 1$ auch die **Konvexkombination** $\lambda s + (1 - \lambda) t$ in P ist.

Ein $p \in P$ mit P konvex heißt **Extrempunkt** falls es keine Konvexkombination zweier Punkte in P gibt die gleich p ist.

Die **konvexe Hülle** von $P \subseteq \mathbb{R}^n$ ist die kleinste konvexe Menge $P' \subseteq \mathbb{R}^n$ mit $P \subseteq P'$.

Ein **Halbraum** ist eine Menge $S := \{s \in \mathbb{R}^n \mid a^T s \leq \lambda\}$ mit $a \in \mathbb{R}^n, \lambda \in \mathbb{R}$.

Ein **konvexes Polyeder** ist eine Menge $P \subseteq \mathbb{R}^n$, welche der Schnitt endlich vieler Halbräume ist.

Eine **Ecke** ist eine Extrempunkt eines konvexen Polyeders.

Ein **konvexes Polytop** ist ein beschränkter konvexer Polyeder.

Jede Nebenbedingung, also jede i-te Zeile der Gleichung $Ax \geq b$ bzw. $Ax \leq b$, eines LPs definiert einen Halbraum $a_i^T \geq b_i$ bzw $a_i^T \leq b_i$, dessen Grenze die Hyperebene $a_i^T = b_i$ ist.

Die **zulässigen Lösungen** eines LPs bilden daher ein konvexes Polyeder.

Die Zielfunktion $c^T x$ gibt die Richtung des **Zielvektors** c an.

Ein **beschränktes LP** ist ein LP, dass in Richtung seines Zielvektors beschränkt ist.

Satz 14 (**Optimale Ecken eines LP**). *Ein beschränktes LP besitzt eine Optimallösung, deren Wert einer Ecke des Lösungspolyeders entspricht. Eine Ecke ist genau dann optimal, wenn sie keine verbessernde Kante besitzt.*

6.2 Algorithmen zur Lösung von LPs

Die **Simplexmethode** tauscht Ecken des Lösungspolyeders entlang verbessender Kanten. Da dabei alle Ecken getauscht werden können ("schiefer Würfel") ist die worst-case-Laufzeit exponentiell.

Die **Ellipsoidmethode** ist der erste theoretisch polynomielle Algorithmus. Er unterliegt in der Praxis dennoch der Simplexmethode, welche schnelle "Warmstarts" erlaubt.

Die **Innere-Punkte-Methode** ist ebenfalls polynomiell, aber nur in einzelnen Fällen praktikabler als die Simplexmethode.

Ganzzahlige LPs sind im Allgemeinen $\mathcal{NP}$-schwer und können durch einen Verzicht auf die Ganzzahligkeit (Relaxion) und anschließendes heuristisches "runden" approximativ gelöst werden.

7 Approximationsalgorithmen

Sei nun $\mathcal{A}$ ein **Approximationsalgorithmus** für ein **Optimierungsproblem** Π, der zu jeder **Instanz** $I \in D_\Pi$ jeweils eine zulässige, aber nicht notwendigerweise optimale Lösung mit dem Wert $\mathcal{A}(I)$ berechnet.

Mit $OPT(I)$ bezeichnen wir den Wert einer optimalen Lösung für I.

$\mathcal{A}$ heißt **absoluter Approximationsalgorithmus** (AAA) falls ein $K \in \mathbb{N}_0$ existiert, so dass $\forall\ I \in D_\Pi$ gilt: $|A(I) - OPT(I) \leq K$.

Knapsack-Problem $\in \mathcal{NPC}$: Finde zu einer Menge M, Funktionen $\omega : M \to \mathbb{N}$, $c : M \to \mathbb{N}_0$ und $W \in \mathbb{N}_0$ ein $M' \subseteq M$ mit $\sum\limits_{a \in M'} \omega(a) \leq W$, das $\sum\limits_{a \in M'} c(a)$ maximiert.

Satz 15 (Kein AAA für Knapsack). *$\mathcal{P} \neq \mathcal{NP} \Rightarrow \nexists$ absoluter Approximationsalgorithmus für* Knapsack.

Beweis: Annahme $\exists$ AAA für Knapsack. Definiere zu einer bel. Instanz I mit M, ω, c, W eine Instanz I' mit M, ω, W und $c'(m) := (K+1) \cdot c(m)\ \forall\ m \in M$.
$\mathcal{A}(I')$ induziert eine Lösung M^* für I mit $|(K+1) \cdot c(M^*) - (K+1) \cdot OPT(I)| \overset{\mathcal{A} \text{ AAA}}{\leq} K$
$\Rightarrow |c(M^*) - OPT(I)| \leq \frac{K}{K+1} < 1 \overset{OPT(I) \in \mathbb{N}}{\Longrightarrow}$ Knapsack $\in P$ im Widerspruch zu $\mathcal{P} \neq \mathcal{NP}$.

7.1 Relative Approximationsalgorithmen

Ein **relativer Approximationsalgorithmus** ist ein polynomialer Algorithmus $\mathcal{A}$ für ein Optimierungsproblem Π für den gilt

$$\forall\ I \in D_\Pi \text{ ist } \mathcal{R}_\mathcal{A}(I) := \begin{cases} \frac{\mathcal{A}(I)}{OPT(I)} & \text{falls } \Pi \text{ Minimierungsproblem} \\ \frac{OPT(I)}{\mathcal{A}(I)} & \text{falls } \Pi \text{ Maximierungsproblem} \end{cases} \leq K \text{ mit } K \geq 1$$

Ein **ε-approximierender Algorithmus** ist ein relativer Approximationsalgorithmus für den gilt $\mathcal{R}_\mathcal{A} := \inf\{r \geq 1 \mid \forall I \in D_\Pi : \mathcal{R}_\mathcal{A}(I) \leq r\} \leq 1 + \varepsilon$

Die **asymptotische Gütegarantie** eines Approximationsalgorithmus $\mathcal{A}$ ist
$\mathcal{R}_\mathcal{A}^\infty := \inf\{r \geq 1 \mid \exists\ N > 0$, so dass $\forall\ I$ mit $OPT(I) \geq N$ gilt $R_\mathcal{A} \leq r\}$.

Allgemeines Knapsack-Problem $\in \mathcal{NPC}$: Gibt es zu einer Menge M mit $|M| = n, \omega_1, \ldots, \omega_n \in \mathbb{N}$ und $c_1, \ldots c_n, W, C \in \mathbb{N}_0$ geeignete $x_1, \ldots, x_n \in \mathbb{N}_0$, so dass $\sum\limits_{i=1}^{n} x_i \omega_i \leq W$ und $\sum\limits_{i=1}^{n} x_i c_i \geq C$?

Satz 16. *Der* Greedy-Knapsack*-Algorithmus $\mathcal{A}$ erfüllt $\mathcal{R}_\mathcal{A} = 2$*

Beweis: FOLGT NOCH

Algorithmus 32 GREEDY-KNAPSACK $\in O(n \log n)$

Eingabe: $\omega_1, \ldots, \omega_n \in \mathbb{N}, c_1, \ldots c_n, W \in \mathbb{N}_0$
Ausgabe: 1-approximative Lösung $x_1, \ldots, x_n$
Sortiere "Kostendichten" $p_i := \frac{c_i}{w_i}$ absteigend und indiziere neu
Für $i = 1, \ldots, n$
  $x_i \leftarrow \lfloor \frac{W}{w_i} \rfloor$
  $W \leftarrow W - x_i \cdot w_i$

BIN-PACKING-Problem $\in \mathcal{NPC}$: Zerlege eine endliche Menge $M = \{a_1, \ldots, a_n\}$ mit einer Gewichtsfunktion $s : M \to (0, 1]$ in möglichst wenige Teilmengen $B_1, \ldots, B_m$, so dass $\forall\ j = 1, \ldots, m$ gilt $\sum\limits_{a_i \in B_j} s(a_i) \leq 1$.

Algorithmus 33 NEXT FIT $\in O(n)$

Eingabe: Menge M, Gewichtsfunktion $s : M \to (0, 1]$
Ausgabe: 1-approximative Lösung m für BIN PACKING
$m \leftarrow 1$
Für alle $a_i \in \{a_1, \ldots a_n\}$
  Falls $s(a_i) > 1 - \sum_{a_j \in B_m} s(a_j)$
    $m \leftarrow m + 1$
  $B_m \leftarrow B_m \cup \{a_i\}$

Satz 17. *Der* NEXT FIT*-Algorithmus* NF *erfüllt* $\mathcal{R}_{NF} = 2$

Beweis:
Seien $B_1, \ldots, B_k$ die zu einer Instanz I von NF benutzten Mengen und jeweils $s(B_i) := \sum\limits_{a_j \in B_i} s(a_j)$.
$\forall\ i = 1, \ldots, k-1$ gilt $s(B_i) + s(B_{i+1} > 1 \Rightarrow \sum\limits_{i=1}^{k} s(B_i) > \frac{k}{2}$ falls k gerade, bzw. $\sum\limits_{i=1}^{k-1} s(B_i) > \frac{k-1}{2}$ falls k ungerade
$\Rightarrow \frac{k-1}{2} < OPT(I) \Rightarrow k = NF(I) < 2 \cdot OPT(I) + 1 \overset{NF(I) \in \mathbb{N}}{\Longrightarrow} NF(I) \leq 2 \cdot OPT(I)$

Algorithmus 34 FIRST FIT $\in O(n^2)$

Eingabe: Menge M, Gewichtsfunktion $s : M \to (0, 1]$
Ausgabe: Approximative Lösung m für BIN PACKING
$m \leftarrow 1$
Für alle $a_i \in \{a_1, \ldots a_n\}$
  $r \leftarrow \min\{t \leq m + 1 \mid s(a_i) \leq 1 - \sum_{a_j \in B_t} s(a_j)\}$
  Falls $r = m + 1$
    $m \leftarrow m + 1$
  $B_r \leftarrow B_r \cup \{a_i\}$

Satz 18. *Der* FIRST FIT*-Algorithmus* FF *erfüllt* $\mathcal{R}^{\infty}_{FF} = \frac{17}{10}$

Beweis: "Zu aufwändig" ;-)

7.2 Approximationsschemata

Ein **polynomiales Approximationsschema (PAS)** ist eine Familie von Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$, so dass $\mathcal{A}_\varepsilon$ ein ε-approximierender Algorithmus ist.

Ein **vollpolynomiales Approximationsschema (FPAS)** ist eine PAS, dessen Laufzeit polynomial in $\frac{1}{\varepsilon}$ ist.

Satz 19. *$\mathcal{P} \neq \mathcal{NP} \Rightarrow \nexists$ PAS für $\mathcal{NP}$-schwere Optimierungsprobleme das polynomial in $\log \frac{1}{\varepsilon}$ ist.*

Kein Beweis im Skript angegeben.

Multiprocessor Scheduling-Problem $\in \mathcal{NPC}$: Finde zu n Jobs $J_1, \ldots, J_n$ mit Bearbeitungsdauer $p_1, \ldots, p_n$ eine Zuordnung auf $m < n$ Maschinen, die keiner Maschine zwei Jobs gleichzeitig zuordnet und die Gesamtdauer Makespan $:= \max\limits_{1 \leq j \leq m} (\sum\limits_{J_i \text{ auf } M_j} p_i)$ minimiert.

Die Entscheidungsvariante von Bin-Packing ist äquivalent zu Multiprocessor Scheduling.

Algorithmus 35 List Scheduling $\in O(n)$

Eingabe: Jobs $J_1, \ldots, J_n$, m Maschinen
Ausgabe: Zuweisung der Jobs auf Maschinen
Lege die ersten m Jobs auf die m Maschinen
Sobald eine Maschine fertig ist, ordne ihr den nächsten Job zu

Satz 20. *Der* List Scheduling*-Algorithmus LS erfüllt $\mathcal{R}_{LS} = 2 - \frac{1}{m}$*

Beweis: Sei S_i die Startzeit und T_i die Abschlusszeit von J_i im List-Schedule.
Sei J_k der zuletzt beendete Job $\Rightarrow T_k = \text{Makespan}_{LS}$.
Vor S_k war keine Maschine untätig: $S_k \leq \frac{1}{m} \cdot \sum\limits_{j \neq k} p_j$

$T_k = S_k + p_k \leq \frac{1}{m} \cdot \sum\limits_{j \neq k} p_j + p_k = \frac{1}{m} \cdot \sum\limits_{j=1}^{m} p_j + (1 - \frac{1}{m}) \cdot p_k \leq T_{OPT} + (1 - \frac{1}{m}) \cdot T_{OPT} = (2 - \frac{1}{m}) \cdot T_{OPT}$

List Scheduling kann zu einem PAS abgewandelt werden:

Algorithmus 36 $\mathcal{A}_l$ für Multiprocessor Scheduling $\in O(m^l + n)$

Eingabe: Jobs $J_1, \ldots, J_n$, m Maschinen, Konstante $1 \leq l \leq n$
Ausgabe: Zuweisung der Jobs auf Maschinen
Ordne die l längsten Jobs den Maschinen nach einem optimalem Schedule zu
Ordne die restlichen $n - l$ Jobs mittels List Scheduling zu

Satz 21. *Für das* List-Scheduling*-PAS $\{\mathcal{A}_l \mid 1 \leq l \leq n\}$ gilt*

$$\mathcal{R}_{\mathcal{A}_l} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \lfloor \frac{l}{m} \rfloor}$$

Beweis: Analog zum vorherigen Beweis sei J_i der zuletzt beendete Job $\Rightarrow T_i = \text{Makespan}_{\mathcal{A}_l}$.
$\sum\limits_{j=1}^{n} p_j \geq m \cdot (T_i - p_i) + p_i = m \cdot T_i - (m-1) \cdot p_i \overset{l \text{ längsten}}{\geq} m \cdot T_i - (m-1) \cdot p_{l+1}$

$T_{OPT} \geq \frac{1}{m} \cdot \sum\limits_{j=1}^{n} p_j \;\Rightarrow\; T_{OPT} + \frac{m-1}{m} \cdot p_{l+1} \geq T_i \quad (1)$

Mind. eine Maschine muss $1 + \lfloor \frac{l}{m} \rfloor$ Jobs bearbeiten $\Rightarrow$ $T_{OPT} \geq p_{l+1} \cdot (1 + \lfloor \frac{l}{m} \rfloor)$ (2)

$$\Rightarrow \mathcal{R}_{\mathcal{A}_l} = \frac{T_i}{T_{OPT}} \overset{(1)}{\leq} 1 + \frac{1}{T_{OPT}} \cdot \frac{m-1}{m} \cdot p_{l+1} \overset{(2)}{\leq} 1 + \frac{m-1}{m} \cdot \frac{1}{1 + \lfloor \frac{l}{m} \rfloor} = 1 + \frac{1 - \frac{1}{m}}{1 + \lfloor \frac{l}{m} \rfloor}$$

7.3 Asymptotische PAS für Bin Packing

Ein **asymptotisches PAS (APAS)** ist eine Familie von Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$, so dass $\mathcal{A}_\varepsilon$ ein asymptotisch ε-approximierender Algorithmus ist, d.h. $\mathcal{R}^{\infty}_{\mathcal{A}_\varepsilon} \leq 1 + \varepsilon$

7.3.1 Ein APAS für Bin Packing

Wir konstruieren ein APAS für Bin Packing indem wir zunächst unsere Problemstellung einschränken, dann 2 linear gruppierte Teilinstanzen lösen und am Ende die, bei der Einschränkung unberücksichtigen, "kleinen Elemente" durch First Fit hinzufügen.

Restricted-Bin-Packing$[\delta, m]$-Problem $\in O(n + c(\delta, m))$: Zerlege eine endliche Menge $M = \{a_1, \ldots, a_n\}$ mit einer Gewichtsfunktion $s : M \to \{v_1, \ldots, v_m\}$ mit $1 \geq v_1 > \ldots > v_m \geq \delta > 0$ in möglichst wenige Teilmengen $B_1, \ldots, B_m$, so dass $\forall\, j = 1, \ldots, m$ gilt $\sum\limits_{a_i \in B_j} s(a_i) \leq 1$.

Sei nun n_j die Anzahl der Elemente der Größe v_j, dann ist ein **BIN** ein m-Tupel $(b_1, \ldots, b_m)$ mit $0 \leq b_j \leq n_j$.

Ein **BIN-TYP** ist ein BIN $T_t = (T_{t1}, \ldots, T_{tm})$ mit $\sum\limits_{j=1}^{m} T_{tj} \cdot v_j \leq 1$.

Lemma 19. *Für festes m und δ ist die Anzahl möglicher Unterschiedlicher BIN-TYPEN $q := q(\delta, m) \leq \binom{m+k}{k}$ mit $k = \lfloor \frac{1}{\delta} \rfloor$.*

Beweis: Für einen BIN-TYP $T_t = (T_{t1}, \ldots, T_{tm})$ gilt $\sum\limits_{j=1}^{m} T_{tj} \cdot v_j \leq 1$

$\forall\, j = 1, \ldots, m$ gilt $v_j \geq \delta \;\Rightarrow\; \sum\limits_{j=1}^{m} T_{tj} \leq \lfloor \frac{1}{\delta} \rfloor = k$

Die Wahl der m Stellen des Tupels aus $\mathbb{N}_0$, welche höchstens zu k aufsummieren, entspricht der Wahl von $m + 1$ Stellen aus $\mathbb{N}_0$, die genau zu k aufzsummieren. Das kann man auch als Aufteilen von k Einsen auf $m + 1$ Stellen durch m "Trennwände" auffassen, was insgesamt einer Belegung von $m + k$ "Plätzen" mit k Einsen entspricht $\Rightarrow \binom{m+k}{k}$ Möglichkeiten.

Wir können nun eine Lösung einer Instanz von Restricted-Bin-Packing$[\delta, m]$ durch einen q-dimensionalen Vektor über $\mathbb{N}_0$, der angibt wieviele BINs von jedem BIN-TYP verwendet werden, darstellen.

ILP zu Restricted-Bin-Packing$[\delta, m]$: Sei $A := \begin{pmatrix} T_1 \\ \vdots \\ T_q \end{pmatrix} \in \mathbb{N}_0^{q \times m}$ und $N := (n_1, \ldots, n_m)$.

Minimiere für $X \in \mathbb{N}_0^{1 \times q}$ die Anzahl der BINs $1^T \cdot X^T$ unter der Nebenbedingung $X \cdot A = N$.

Lemma 20. *Eine Teillösung einer Instanz I für* Bin Packing*, bei der für ein $0 < \delta \leq \frac{1}{2}$ alle Elemente größer δ in β BINs gepackt werden, kann zu einer Lösung von I mit höchstens $\beta' \leq max\{\beta, (1+2\delta) \cdot OPT(I) + 1\}$ BINs erweitert werden.*

Beweis: Alle bis auf höchstens ein BIN sind mindestens zu $1-\delta$ gefüllt $\Rightarrow \sum_{i=1}^{n} s_i \geq (1-\delta) \cdot (\beta' - 1)$

$OPT(I) \geq \sum_{i=1}^{n} s_i \;\Rightarrow\; \beta' \leq \frac{1}{1-\delta} \cdot OPT(I) + 1 \overset{0<\delta\leq\frac{1}{2}}{\leq} \frac{1+\delta-2\delta^2}{1-\delta} \cdot OPT(I) + 1 = (1+2\delta) \cdot OPT(I) + 1$

Übersprungen: Details des Lineares Gruppierens.

Algorithmus 37 APAS für Bin Packing $\in O(c_\varepsilon + n \log n)$

Eingabe: Instanz I mit n Elementen der Größen $s_1 \geq \ldots \geq s_n$ und ε
Ausgabe: Approximationslösung für Bin Packing
$\delta \leftarrow \frac{\varepsilon}{2}$
$J \leftarrow$ Instanz von $RBP[\delta, n']$ mit allen Elementen aus I größer δ
$k \leftarrow \lceil \frac{\varepsilon^2}{2} \cdot n' \rceil$
$m \leftarrow \lfloor \frac{n'}{k} \rfloor$
$J_{LO} \leftarrow \bigcup_{j=2}^{m} \{k \text{ mal } (j-1)k+1\text{-größte Element}\} \cup \{ \text{ |Rest| mal } (m-1)k+1\text{-größte Element}\}$
$J_{HI} \leftarrow J_{LO} \cup \{k \text{ mal das größte Element}\}$
Löse J_{LO} durch das ILP optimal
Füge die k größten Elemente in maximal k zusätzliche BINs
Verwende diese BINs für J_{HI} als Lösung auf J
Erweitere diese Lösung mittels First Fit zu einer Lösung von I

7.3.2 Ein AFPAS für Bin Packing

TODO !!!

8 Randomisierte Algorithmen

Ein **Las Vegas Algorithmus** liefert immer das korrekte Ergebnis, bei zufälliger Laufzeit.

Ein **Monte Carlo Algorithmus mit beidseitigem Fehler** kann sowohl fälschlicherweise Ja, als auch fälschlicherweise Nein antworten.

Ein **Monte Carlo Algorithmus mit einseitigem Fehler** antwortet entweder nie fälschlicherweise Ja, oder er antwortet nie fälschlicherweise Nein.

$\mathcal{RP}$ ist die Klasse der Entscheidungsprobleme Π, für die es einen polynomialen Algorithmus A gibt, so dass $\forall\ I \in D_\Pi$ gilt: $\Pr[A(I) = \text{“}Ja''] \begin{cases} \geq \frac{1}{2} & I \in Y_\Pi \\ = 0 & I \notin Y_\Pi \end{cases}$

$\mathcal{PP}$ ist die Klasse der Entscheidungsprobleme Π, für die es einen polynomialen Algorithmus A gibt, so dass $\forall\ I \in D_\Pi$ gilt: $\Pr[A(I) = \text{“}Ja''] \begin{cases} > \frac{1}{2} & I \in Y_\Pi \\ < \frac{1}{2} & I \notin Y_\Pi \end{cases}$

$\mathcal{BPP}$ ist die Klasse der Entscheidungsprobleme Π, für die es einen polynomialen Algorithmus A gibt, so dass $\forall\ I \in D_\Pi$ gilt: $\Pr[A(I) = \text{“}Ja''] \begin{cases} \geq \frac{3}{4} & I \in Y_\Pi \\ \leq \frac{1}{4} & I \notin Y_\Pi \end{cases}$

8.1 Grundlagen der Wahrscheinlichkeitstheorie I

Können an anderen Stellen nachgelesen werden.

8.2 Randomisierte MinCut-Algorithmen

8.2.1 Ein einfacher Monte Carlo Algorithmus für MinCut

Fasse einen Graphen $G = (V, E)$ mit Kantengewichten $c : E \to \mathbb{N}$ als Multigraph mit jeweils $c(u, v)$ Kanten zwischen u und v auf.

Algorithmus 38 RANDOM MINCUT $\in O(|V|^2)$

Eingabe: Multigraph $G = (V, E)$
Ausgabe: Schnitt in Form zweier Superknoten
Solange $|V| > 2$
  $\{u, v\} \leftarrow$ zufällige Kante aus E
  $G \leftarrow G$ mit verschmolzenem u, v

Satz 22. *Die Wahrscheinlichkeit, dass* RANDOM MINCUT *einen minimalen Schnitt findet ist größer als* $\frac{2}{|V|^s}$ *falls die Kanten gleichverteilt gewählt werden.*

Beweis: Sei k die Größe eines minimalen Schnittes und $n := |V| \Rightarrow$ Jeder Knoten hat mindestens Grad $k \Rightarrow |E| \geq k \cdot \frac{n}{2}$
$A_i :=$ im i-ten Schritt wird keine Kante des minimalen Schnittes gewählt

$\Pr[A_1] \geq 1 - \frac{k}{\frac{k \cdot n}{2}} = 1 - \frac{2}{n}$
$\Pr[\bigcap_{i=1}^{n-2} A_i] \geq 1 - \frac{k}{\frac{k \cdot (n-i+1)}{2}} = \prod_{i=1}^{n-2} (1 - \frac{2}{n-i+1}) = \frac{2}{n \cdot (n-1)}$

Wendet man RANDOM MINCUT $\frac{n^2}{2}$ mal unabhängig voneinander an ($O(|V|^4)$), dann ist die Wahrscheinlichkeit, dass ein bestimmter Schnitt nicht gefunden wurde höchstens $(1 - \frac{2}{n^2})^{\frac{n^2}{2}} < \frac{1}{e}$.

8.2.2 Ein effizienterer randomisierter MinCut-Algorithmus

Algorithmus 39 FAST RANDOM MINCUT $\in O(|V|^2 \log |V|)$

Eingabe: Multigraph $G = (V, E)$
Ausgabe: Schnitt
Falls $|V| \leq 6$
  Berechne MINCUT deterministisch
Sonst
  $l \leftarrow \lceil \frac{n}{\sqrt{2}} \rceil$
  $G_1 \leftarrow$ RANDOM MINCUT bis l Knoten übrig
  $G_2 \leftarrow$ RANDOM MINCUT bis l Knoten übrig
  $C_1 \leftarrow$ FAST RANDOM MINCUT(G_1) (rekursiv)
  $C_2 \leftarrow$ FAST RANDOM MINCUT(G_2) (rekursiv)
  Gebe den kleineren der beiden Schnitte C_1, C_2 aus

Satz 23. *Die Wahrscheinlichkeit, dass* FAST RANDOM MINCUT *einen minimalen Schnitt findet, ist in* $\Omega(\frac{1}{log\ |V|})$.

Der Beweis ist laut Skript "lang und schwierig" und es wird lediglich eine Beweis-Idee angegeben.

8.3 Grundlagen der Wahrscheinlichkeitstheorie II

Werden wieder nur erwähnt um die Nummerierung konsistent zum Skript zu halten.

8.4 Das Maximum Satisfiability Problem

MAXIMUM SATISFIABILITY PROBLEM: Finde zu m Klauseln über n Variablen V eine Wahrheitsbelegung die eine maximale Anzahl von Klauseln erfüllt.

Bereits MAX-2-SAT ist $\mathcal{NP}$-schwer, aber nicht in $\mathcal{NP}$.

8.4.1 Der Algorithmus Random Sat

Satz 24. *Der Erwartungswert der Lösung von* RANDOM SAT *zu einer Instanz mit m Klauseln, die jeweils mindestens k Literale enthalten, ist mindestens* $(1 - \frac{1}{2^k}) \cdot m$

Beweis: Je Klausel wird mit einer Wahrscheinlichkeit von $\frac{1}{2^k}$ die eine nicht erfüllende der 2^k Belegung gewählt.

Algorithmus 40 RANDOM SAT $\in O(n)$

Eingabe: m Klauseln über n Variablen V
Ausgabe: Wahrheitsbelegung $\omega \to \{wahr, falsch\}$
Für alle $x \in V$
  $\omega(x) := wahr$ mit Wahrscheinlichkeit $\frac{1}{2}$

8.5 Das MaxCut-Problem

MAXCUT-Problem $\in \mathcal{NPC}$: Finde zu einem Graphen $G = (V, E)$ mit Gewichtsfunktion $c : E \to \mathbb{N}$ einen Schnitt $(S, V \setminus S)$ von G der $c(S, V \setminus S)$ maximiert.

8.5.1 Ein Randomisierter Algorithmus für MaxCut basierend auf semidefiniter Programmierung

Definiere zu einer Instanz I von MAXCUT ein **ganzzahliges quadratisches Programm** $IQP(I)$ mit Hilfe eine Gewichtsmatrix $C = (c_{ij})$ mit $c_{ij} := \begin{cases} c(\{i,j\}) & \text{falls } \{i,j\} \in E \\ 0 & \text{sonst} \end{cases}$ für $i, j = 1, \ldots |V|$.

$IQP(I)$: Maximiere $\frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{j-1} c_{ij} \cdot (1 - x_i \cdot x_j)$ unter den Nebenbedingungen $x_i, x_j \in \{-1, 1\}$ für $1 \leq i, j \leq n$.

Mit $x_i = 1$ falls $i \in S$ und $x_i = -1$ falls $i \in V \setminus S$ induziert eine optimale Belegung der x_i einen maximalen Schnitt.

8.5.2 Relaxierung von IQP(I)

$QP^2(I)$: Maximiere $\frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{j-1} c_{ij} \cdot (1 - x^i \cdot x^j)$ unter den Nebenbedingungen $x^i, x^j \in \mathbb{R}^2$ mit $\|x^i\| = \|x^j\| = 1$ für $1 \leq i, j \leq n$.

Jede Lösung $x_1, \ldots, x_n$ von $IQP(I)$ induziert eine Lösung $\begin{pmatrix} x_1 \\ 0 \end{pmatrix}, \ldots, \begin{pmatrix} x_n \\ 0 \end{pmatrix}$ von $QP^2(I)$ $\Rightarrow$ $QP^2(I)$ ist Relaxierung von $IQP(I)$

Algorithmus 41 RANDOM MAXCUT

Eingabe: Graph $G = (V, E)$, $c : E \to \mathbb{N}$
Ausgabe: Schnitt $(S, V \setminus S)$
Berechne eine optimale Lösung $(x^1, \ldots, x^n)$ von QP^2
$r \leftarrow$ zufälliger 2-dimensionaler, normierter Vektor
$S \leftarrow \{i \in V \mid x^i \cdot r \geq 0\}$ // x^i oberhalb der Senkrechten l zu r durch 0

Satz 25. *Der Erwartungswert der Kapazität des von* RANDOM MAXCUT *berechneten Schnittes ist* $\frac{1}{\pi} \sum_{j=1}^{n} \sum_{i=1}^{j-1} c_{ij} \cdot \arccos(x^i \cdot x^j)$*, falls r gleichverteilt gewählt wird.*

Beweis: $\Pr[\text{sgn}(x^i \cdot r) \neq \text{sgn}(x^j \cdot r)] = \Pr[x^i \text{ und } x^j \text{ werden von } l \text{ getrennt}] =$
$= \Pr[s \text{ oder } t \text{ liegen auf dem kürzeren Kreisbogen der Länge } \arccos(x^i \cdot x^j) \text{ zwischen } x^i \text{ und } x^j] =$
$= \frac{\arccos(x^i \cdot x^j)}{2\pi} + \frac{\arccos(x^i \cdot x^j)}{2\pi} = \frac{\arccos(x^i \cdot x^j)}{\pi}$

Satz 26. *Zu einer Instanz I von* MaxCut *berechnet* Random MaxCut *einen Schnitt mit Kapazität $C_{RMC}(I)$ für den gilt:* $\frac{\mathrm{E}(C_{RMC}(I))}{OPT(I} \geq 0,8785$.

Der Beweis ist sehr "technisch" und unvollständig.

Da derzeit nicht bekannt ist ob QP^2 in polynomialer Laufzeit gelöst werden kann, modifiziert man Random MaxCut in dem man ein QP^n mit $x^j, x^j \in \mathbb{R}^n$ verwendet, dass mit Hilfe positiv semidefiniter Matrizen ($x^T \cdot M \cdot x \geq 0$) in polynomialer Zeit gelöst werden kann.

Für jedes $\varepsilon > 0$ gilt für diesen Semi-Definit-Cut genannten Algorithmus $\mathcal{A}_\varepsilon$, dass $\frac{OPT(I)}{\mathcal{A}_\varepsilon(I)} \leq 1+\varepsilon$.

9 Parallele Algorithmen

9.1 Das PRAM Modell

Das **PRAM Modell** unterscheidet sich vom RAM Modell durch eine unbegrenzte Anzahl Prozessoren, die sowohl auf ihren lokalen, unbegrenzten, als auch auf einen globalen, unbegrenzten Speicher zugreifen können.

Beim gleichzeitigen Zugriff auf den globalen Speicher unterscheidet man 4 Modelle: **C**oncurrent **R**ead, **C**oncurrent **W**rite, **E**xclusive **R**ead und **E**xclusive **W**rite. Wir beschränken uns auf CREW-PRAM.

9.2 Komplexität paralleler Algorithmen

Die **Laufzeit** eines parallelen Algorithmus $\mathcal{A}$ ist
$T_{\mathcal{A}}(n) := \max_{|I|=n}\{\#\text{ Berechnungsschritte von } \mathcal{A} \text{ bei Eingabe } I\}$

Die **Prozessorenzahl** eines parallelen Algorithmus $\mathcal{A}$ ist
$P_{\mathcal{A}}(n) := \max_{|I|=n}\{\#\text{ Prozessoren, die während des Ablauf von } \mathcal{A} \text{ bei Eingabe } I \text{ gleichzeitig aktiv sind}\}$

Der **Speed Up** eines parallelen Algorithmus $\mathcal{A}$ ist speed-up$(\mathcal{A}) := \frac{\text{worst-case Laufzeit des besten sequentiellen Algorithmus}}{\text{worst-case Laufzeit von } \mathcal{A}}$

Die **Kosten** eines parallelen Algorithmus $\mathcal{A}$ sind $C_{\mathcal{A}}(n) := T_{\mathcal{A}}(n) \cdot P_{\mathcal{A}}(n)$

Ein paralleler Algorithmus $\mathcal{A}$ ist **kostenoptimal** falls die schärfste untere Schranke für die Laufzeit eines sequentiellen Algorithmus gleich dem asymptotischen Wachstum von $C_{\mathcal{A}}(n)$ ist.

Die **Effizienz** eines parallelen Algorithmus $\mathcal{A}$ ist $E_{\mathcal{A}}(n) := \frac{worst-caseLaufzeitdesbestensequentiellenAlgorithmus}{C_{\mathcal{A}}(n)}$

9.3 Komplexitätsklassen

Nick's Class $\mathcal{NC}$ ist die Klasse der Probleme, die durch einen parallelen Algorithmus in polylogarithmischer Laufzeit mit polynomialer Prozessorenzahl gelöst werden können.

Steve's Class $\mathcal{SC}$ ist die Klasse der Probleme, die durch einen sequentiellen Algorithmus in polynomialer Laufzeit mit polylogarithmischem Speicherbedarf gelöst werden können.

Sowohl die Frage $\mathcal{P} = \mathcal{NC}$? als auch $\mathcal{NC} = \mathcal{SC}$? sind ungeklärt.

9.4 Parallele Basisalgorithmen

9.4.1 Broadcast

Übertrage im EREW-PRAM Modell einen gegebenen Wert an alle N Prozessoren mit Hilfe des Algorithmus Broadcast:

Algorithmus 42 BROADCAST $\in O(logN)$

Eingabe: Wert m
Seiteneffekte: Array A der Länge N im globalen Speicher
P_1 kopiert m in den eigenen Speicher
P_1 schreibt m in $A[1]$
Für $i = 0, \ldots, \lceil \log N \rceil - 1$
  Für alle $j = 2^i + 1, \ldots, 2^{i+1}$ **führe parallel aus**
    P_j kopiert m aus $A[j - 2^i]$ in den eigenen Speicher
    P_j schreibt m in $A[j]$

$P_{\text{BROADCAST}}(N) = \frac{N}{2}$ und $C_{\text{BROADCAST}}(N) \in O(N \log N)$.

9.4.2 Berechnung von Summen

Algorithmus 43 SUMME$(a_1, \ldots, a_n) \in O(logn)$

Eingabe: Werte $a_1, \ldots, a_n$, o.B.d.A sei $n = 2^m, m \in \mathbb{N}$
Ausgabe: $\sum_{i=1}^{n} a_i$
Für $i = 1, \ldots, m$
  Für alle $j = 1, \ldots, \frac{n}{2^i}$ **führe parallel aus**
    P_j berechnet $a_j \leftarrow a_{2j-1} + a_{2j}$
Gebe a_1 aus.

$P_{\text{SUMME}}(n) = \frac{n}{2}$ und $C_{\text{SUMME}}(n) \in O(n \log n) \Rightarrow$ SUMME nicht kostenoptimal.

Wenn SUMME durch Rescheduling so abgewandelt wird, dass nur noch $\lceil \frac{n}{\log n} \rceil$ Prozessoren verwendet werden ist SUMME kostenoptimal: $T_{\text{SUMME}} \leq c \cdot \sum_{i=1}^{\log n} \frac{\log n}{2^i} = c \cdot \log n \cdot \underbrace{\sum_{i=1}^{\log n} \frac{1}{2^i}}_{\leq 1} \in O(\log n)$

SUMME$(a_1, \ldots, a_n)$ kann leicht zu jeder n-fachen binären Operation abgewandelt werden.

ODER$(x_1, \ldots x_n)$ kann auf einer CRCW-PRAM sogar in $O(1)$ bestimmt werden, wenn mehrere Prozessoren genau dann dieselbe globale Speicherzelle beschreiben dürfen wenn sie denselben Wert schreiben.

Algorithmus 44 CRCW-ODER$(x_1, \ldots, x_n) \in O(1)$

Eingabe: Werte $x_1, \ldots, x_n \in \{0, 1\}$
Ausgabe: $x_1 \vee \ldots \vee x_n$
Für alle $i = 1, \ldots, n$ **führe parallel aus**
  P_n liest x_i
  Falls $x_i = 1$
    P_i schreibt 1 in den globalen Speicher
Gebe Wert aus dem globalen Speicher aus.

9.4.3 Berechnung von Präfixsummen

Wir berechnen alle Präfixsummen $A_k := \sum_{i=1}^{k} a_i$ mit $1 \leq k \leq n$ aus n Eingabewerten $a_n, \ldots, a_{2n-a}$ (technische Benennung) aus Differenzen der Zwischenergebnisse von SUMME$(a_n, \ldots, a_{2n-a})$.

Algorithmus 45 PRÄFIXSUMMEN$(a_n, \ldots, a_{2n-a}) \in O(logn)$

Eingabe: Werte $a_n, \ldots, a_{2n-a}$, o.B.d.A sei $n = 2^m, m \in \mathbb{N}$
Ausgabe: Präfixsummen $\sum_{i=1}^{k} a_i$ für $1 \leq k \leq n$
Für $i = m-1, \ldots, 0$
  Für alle $j = 2^i, \ldots, 2^{i+1}-1$ **führe parallel aus**
    P_j berechnet $a_j \leftarrow a_{2j} + a_{2j+1}$
$b_1 \leftarrow a_1$
Für $i = 1, \ldots m$
  Für alle $j = 2^i, \ldots, 2^{i+1}-1$ **führe parallel aus**
    Falls j ungerade
      $b_j \leftarrow b_{\frac{j-1}{2}}$
    Sonst
      $b_j \leftarrow b_{\frac{j}{2}} - a_{j+1}$

Durch Rescheduling kann die Prozessorenzahl wieder auf $\lceil \frac{n}{\log n} \rceil$ reduziert und PRÄFIXSUMMEN dadurch kostenoptimiert werden.

9.4.4 Die Prozedur List Ranking oder Short Cutting

Teile jedem Element eines Wurzelbaums, das nur seinen direkten Vorgänger kennt, die Wurzel mit.

Algorithmus 46 LIST RANKING$(n, h) \in O(logn)$

Eingabe: Wurzelbaum der Höhe h mit n Elementen
Ausgabe: $\forall\ i = 1, \ldots, n$ VOR$[i]$
Für $i = 1, \ldots, \lceil \log h \rceil$
  Für alle $j = 1, \ldots, n$ **führe parallel aus**
    P_j berechnet VOR$[j] \leftarrow$ VOR[VOR$[j]$]

9.4.5 Binäroperationen einer partitionierten Menge

BINÄRE PARTITIONS-OPERATION-Problem: Berechne die p Ergebnisse einer binären Operation auf p Gruppen (variabler Größe) von insgesamt n Werten mit K Prozessoren.

1. Fall: $K \geq n$: Berechne Gruppenergebnisse analaog PRÄFIXSUMMEN in $O(logn)$.
2. Fall: $K < n$: Teile in K Abschnitte der Größe $\lceil \frac{n}{K} \rceil$ auf und berechne mit jedem Prozessor sequentiell die Teilergebnisse eines Abschnittes.
Je Abschnitt können noch höchstens 2 Operationen mit Teilergebnissen der benachbarten Abschnitte (in den beiden Grenzfällen höchstens 1 Operation) nötig sein um die Ergebnisse von abschnittsüberschreitenden Gruppen zu berechnen.
Ordne jeder Gruppe G_i, deren Ergebnis noch aus n_i Teilergebnissen berechnet werden muss, $\lfloor \frac{n_i}{2} \rfloor$

Prozessoren zu, welche in log K die Gruppenergebnisse berechnen. Insgesamt ergibt sich eine Laufzeit von $\lceil \frac{n}{K} \rceil - 1 + \log K$.

9.5 Ein paralleler Algorithmus für die Berechnung der Zusammenhangskomponenten

Algorithmus 47 ZUSAMMENHANG$(G) \in O(log^2|V|)$

Eingabe: $G = (V, E)$
Ausgabe: $\forall\ i \in V$ ist $K[i]$ der kleinste Knoten der Zusammenhangskomponente von i
Seiteneffekte:
$K[]$ enthält zu jedem Knoten die Nummer seiner Zusammenhangskomp.
$N[]$ enthält zu jedem Knoten die Nummer der kleinsten benachbarten Zusammenhangskomp.
Für alle $i = 1, \ldots, |V|$ **führe parallel aus**
  $K[i] \leftarrow i$
Für $l = 1, \ldots, \lceil \log |V| \rceil$
  Für alle $i = 1, \ldots, |V|$ **führe parallel aus**
    Falls $\exists\ \{i, j\} \in E$ mit $K[i] \neq K[j]$
      $N[i] \leftarrow \min\{K[j] \mid \{i, j\} \in E, K[i] \neq K[j]\}$ // finde kleinste verbundene Komp.
    Sonst
      $N[i] \leftarrow K[i]$
  Für alle $i = 1, \ldots, |V|$ **führe parallel aus**
    // finde kleinste benachbarte Komp. von Knoten der gleichen Komp.
    $N[i] \leftarrow \min\{N[j] \mid K[i] = K[j], N[j] \neq K[j]\}$
  Für alle $i = 1, \ldots, |V|$ **führe parallel aus**
    $N[i] \leftarrow \min\{N[i], K[i]\}$ // benachbarte Komponente = gleiche o. benachbarte Komp.
  Für alle $i = 1, \ldots, |V|$ **führe parallel aus**
    $K[i] \leftarrow N[i]$ // gleiche Komp. = benachbarte Komp.
  Für $m = 1, \ldots, \lceil \log n \rceil$
    Für alle $i = 1, \ldots, |V|$ **führe parallel aus**
      $K[i] \leftarrow K[K[i]]$ // Komp. = Nachbarkomp.

Durch Rescheduling kann die Prozessorenzahl von ZUSAMMENHANG zwar von $|V|^2$ auf $\lceil \frac{|V|^2}{\log |V|} \rceil$ gesenkt werden, was mit $O(|V|^2 \log |V|)$ dennoch nicht kostenoptimal ist.

9.6 Modifikation zur Bestimmung eines MST

Führe die Schritte innerhalb der l-Schleife bis zur Erweiterung von T nur für die Repräsentanten der Komponenten aus. Berechne nach jedem Durchlauf der l-Schleife die Adjanzenzmatrix mit auf die Repräsentanten übertragenene Kanten neu. Dadurch kommt MST mit $\lceil \frac{n^2}{\log^2 n} \rceil$ Prozessoren aus.

9.7 Parallelisierung von Select zur Bestimmung des k-ten Elements aus n Elementen

9.8 Ein paralleler Algorithmus für das Scheduling-Problem für Jobs mit festen Start- und Endzeiten

Algorithmus 48 MSTG(G)

Eingabe: $G = (V, E)$, $c : V \times V \to \mathbb{R} \cup \{\infty\}$
Ausgabe: MST von G in Form von Kanten T
Seiteneffekte:
$S[]$ enthält zu jedem Knoten die Nummer des Knoten der gleichen Komp., der eine min. Kante zu einer anderen Komp. hat
$K[]$ enthält zu jedem Knoten die Nummer des Knoten der gleichen Komp. mit einer min. Kante
$N[]$ enthält zu jedem Knoten die Nummer des Knoten einer anderen Komp., der durch eine min. Kante benachbart ist

Für alle $i = 1, \ldots, |V|$ **führe parallel aus**
  $K[i] \leftarrow i$
Für $l = 1, \ldots, \lceil \log |V| \rceil$
  Für alle $i = 1, \ldots, |V|$ **führe parallel aus**
    // finde Knoten einer anderen Komp., der durch eine min. Kante benachbart ist
    $N[i] \leftarrow k$ mit $c_{ik} = \min\limits_{1 \le j \le |V|} \{c_{ij} \mid K[i] \neq K[j]$
  Für alle $i = 1, \ldots, |V|$ **führe parallel aus**
    // finde Knoten der gleichen Komp., der eine min. Kante zu einer anderen Komp. hat
    $S[i] \leftarrow t$ mit $c_{tN[t]} = \min\limits_{1 \le j \le |V|} \{c_{jN[j]} \mid K[i] = K[j]$
    $N[i] \leftarrow N[t]$
  Für alle $i = 1, \ldots, |V|$ **führe parallel aus**
    Falls $N[N[i]] = S[i]$ und $K[i] < K[N[i]]$
      $S[i] \leftarrow 0$ // i kann über $\{S[i], N[i]\}$ günstiger als über $\{i, N[i]\}$ angebunden werden
  Für alle $i = 1, \ldots, |V|$ **führe parallel aus**
    Falls $S[i] \neq 0$ und $K[i] = i$ und $c_{N[i],S[i] \neq \infty}$
      $T \leftarrow T \cup \{N[i], S[i]\}$ // Füge min. Kante zu einer anderen Komp. zum MST hinzu
    Falls $S[i] \neq 0$
      $K[i] = K[N[i]]$ // Knoten mit min. Kante = benachbarter Knoten mit min. Kante
  Für $m = 1, \ldots, \lceil \log n \rceil$
    Für alle $i = 1, \ldots, |V|$ **führe parallel aus**
      $K[i] \leftarrow K[K[i]]$ // Komp. = Nachbarkomp.

10 Parametrisierte Algorithmen

Independet Set-Problem: Finde zu einem Graphen $G = (V, E)$ und $k \in N$ eine Menge $V' \subseteq V$ mit $|V'| \geq k$, so dass $\forall\ v, w \in V'$ gilt $\{v, w\} \notin E$.

Vertex Cover-Problem: Finde zu einem Graphen $G = (V, E)$ und $k \in N$ eine Menge $V' \subseteq V$ mit $|V'| \leq k$, so dass $\forall\ \{v, w\} \in E$ gilt $u \in V'$ oder $v \in V'$.

Dominating Set-Problem: Finde zu einem Graphen $G = (V, E)$ und $k \in N$ eine Menge $V' \subseteq V$ mit $|V'| \leq k$, so dass $\forall\ v \in V$ gilt entweder ist $v \in V'$ oder ein Nachbar von v ist in V' enthalten.

Alle drei Probleme sind $\mathcal{NP}$-schwer und können durch Aufzählen aller $\binom{|V|}{k}$ Teilmengen $V' \subseteq V$ mit $|V'| = k$ in $O(|V|^k \cdot (|V| + |E|))$ entschieden werden.

Ein parametrisiertes Problem Π heißt **fixed parameter tractable**, wenn es einen Algorithmus gibt, der Π in $O(\mathcal{C}(k) \cdot p(n))$ löst, wobei $\mathcal{C}$ eine berechenbare Funktion die nur von dem Parameter k abhängig ist, p ein Polynom und n die Eingabegröße ist.

FPT ist die Klasse aller Probleme, die fixed parameter tractable sind.

Vertex Cover $\in$ FPT. Es wird vermutet, dass Independet Set und Dominating Set nicht in FPT sind.

10.1 Parametrisierte Komplexitätstheorie

Exkurs.

10.2 Grundtechniken zur Entwicklung parametrisierter Algorithmen

Kernbildung: Reduziere eine Instanz (I, k) in $O(p(|I|))$ auf eine äquivalente Instanz I' deren Größe $|I'|$ nur von k abhängt.

Tiefenbeschränkte Suchbäume: Führe eine erschöpfende Suche in einem geeigneten Suchbaum beschränkter Tiefe aus.

Für eine Instanz $((V, E), k)$ von Vertex Cover gilt:

- $\forall\ v \in V$ ist entweder v oder alle Nachbarn $N(v)$ in V'
- $\{v \in V \mid |N(v)| > k\} \subseteq V'$
- Falls Maximalgrad$(G) \leq k$ und $|E| > k^2$ hat G kein Vertex Cover mit k oder weniger Knoten.

Falls die Laufzeit einer Suche in einem tiefenbeschränkten Suchbaum durch $T(k) \leq T(k - t_1) + \ldots + T(k - t_s) + c$ abgeschätzt werden kann heißt $(t_1, \ldots, t_s)$ **Verzweigungsvektor**.

Algorithmus 49 VERTEX-COVER$(G, k) \in O(nk + 2^k \cdot k^2)$

Eingabe: Graph $G = (V, E)$, Parameter $k \in \mathbb{N}$
$H \leftarrow \{v \in V \mid |N(v)| > k\}$
Falls $|H| > k$
  Gebe "G hat kein Vertex Cover der Größe $\leq k$" aus.
Sonst
  $k' \leftarrow k - |H|$
  $G' \leftarrow G - H$ // Entferne alle $v \in H$ und damit verbundene Kanten
  Falls $|E'| > k \cdot k'$
    Gebe "G hat kein Vertex Cover der Größe $\leq k$" aus.
  Sonst
    Entferne isolierte Knoten aus G'
    Berechne VERTEX-COVER(G', k')

Konstruiere einen tiefenbeschränkten Suchbaum für VERTEX-COVER durch wiederholte Anwendung der anwendbaren Regel mit kleinster Nummer von folgenden Regeln:

1. $\exists\ v \in V$ mit $|N(v)| = 1\ \Rightarrow\ C \leftarrow C \cup N(v)$
2. $\exists\ v \in V$ mit $|N(v)| \geq 5\ \Rightarrow\ C \leftarrow C \cup N(v)$ o. $C \leftarrow C \cup \{v\}$
3. $\exists\ v \in V$ mit $N(v) = \{a, b\}$
   Falls a, b adjazent $\Rightarrow\ C \leftarrow C \cup \{a, b\}$
   Falls a, b nicht adjazent und $N(a) = \{v, w\} = N(b)\ \Rightarrow\ C \leftarrow C \cup \{v, w\}$
   Ansonsten $C \leftarrow C \cup N(v)$ o. $C \leftarrow C \cup N(a) \cup N(b)$
4. $\exists\ v \in V$ mit $N(v) = \{a, b, c\}$
   Falls a, b adjazent $\Rightarrow\ C \leftarrow C \cup N(v)$ o. $C \leftarrow C \cup N(c)$
   Falls a, b nicht adjazent und $N(a) = \{v, w\} = N(b)\ \Rightarrow\ C \leftarrow C \cup \{v, w\}$ o. $C \leftarrow C \cup N(v)$
   Falls a, b, c nicht adjazent $\Rightarrow\ C \leftarrow C \cup N(v)$ o. $C \leftarrow C \cup N(a)$ o. $C \leftarrow C \cup \{a\} \cup N(b) \cup N(c)$
   Falls a, b, c nicht adjazent und $|N(a)| = |N(b)| = |N(c)| = 3$ mit $N(a) = \{v, d, e\}$
   $\Rightarrow\ C \leftarrow C \cup N(v)$ o. $C \leftarrow C \cup N(a)$ o. $C \leftarrow C \cup N(b) \cup N(c) \cup N(d) \cup N(e)$
5. $\forall\ v \in V$ gilt $|N(v)| = 4\ \Rightarrow\ C \leftarrow C \cup \{v\}$ o. $C \leftarrow N(v)$ für ein beliebiges $v \in V$