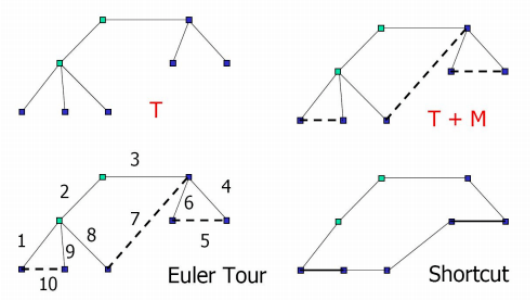
Kapitel 6

Heuristiken und Approximation



Inhalt

Lösungsansatz und Fragen

Ansatz:

- Wir verzichten auf die Forderung nach Optimalität.
- Es genügt uns, wenn eine Lösung fast optimal ist.
- Wir versuchen, auch Aussagen über die Güte einer Lösung herzuleiten.

Fragen:

- Wie können wir zu einem Problem einen Algorithmus finden, der für jede beliebige Instanz eine nahezu optimale Lösung liefert?
- Wie können wir "nahezu optimal" quantifizieren? Können wir eine Schranke für die Güte einer Lösung beweisen?
- Ist solch eine ermittelte Schranke scharf?
- Gibt es Grenzen für diese Güteschranken?

Methoden für den Entwurf von Approximationsalgorithmen

- Greedy-Algorithmen
- Dynamische Programmierung
- LP-Relaxationen
- Lokale Suche und Metaheuristiken

Optimierungsproblem aus Sicht der Approximation

Definition 6.1

Ein Optimierungsproblem Π besteht aus

- einer Menge $\mathcal{I}_\Pi$ von Instanzen oder Eingaben.
- Zu jeder Instanz $I \in \mathcal{I}_\Pi$ gehört eine Menge $\mathcal{S}_I$ von (zulässigen) Lösungen und
- eine Zielfunktion $f_I : \mathcal{S}_I \to \mathbb{R}$, die jeder (zulässigen) Lösung einen reellen Wert zuordnet.
- Zusätzlich ist vorgegeben, ob wir eine Lösung mit minimalem oder maximalen Wert $f_I(x)$ suchen.

Für eine Instanz $I \in \mathcal{I}_\Pi$ bezeichnet $\mathrm{OPT}(I)$ den Wert einer optimalen Lösung.

Geht die Instanz I aus dem Kontext hervor, schreiben wir auch kurz OPT statt $\mathrm{OPT}(I)$.

Beispiel 6.2

- Eine Instanz I von MST (minimum spanning tree problem, Minimalgerüst) wird durch einen ungerichteten Graphen $G = (V, E)$ und Kantengewichte $c : E \to \mathbb{N}$ beschrieben.
- Die Menge $\mathcal{S}_I$ der Lösungen ist die Menge aller Gerüste (aufspannender, zusammenhängender, kreisfreier Untergraph) für den Graphen G.
- Sei $T = (V, F) \in \mathcal{S}_I$ ein Gerüst von G. Dann lautet die Zielfunktion:

$$f_I(T) = \sum_{e \in F} c(e).$$

  Sie weist jedem Gerüst $T \in \mathcal{S}_I$ ein Gewicht zu.
- Wir möchten minimieren. Also:

$$\mathrm{OPT}(I) = \min_{T \in \mathcal{S}_I} f_I(T).$$

Approximationsalgorithmus

Definition 6.3

Ein Approximationsalgorithmus A für ein Optimierungsproblem Π ist ein Polynomialzeitalgorithmus, der zu jeder Instanz I eine Lösung aus $\mathcal{S}_I$ ausgibt.

Wir bezeichnen mit $A(I)$ die ausgegebene Lösung für die Instanz I und mit $w_A(I) = f_I(A(I))$ ihren Wert.

Bemerkung:

- Je näher $w_A(I)$ dem optimalen Wert $\mathrm{OPT}(I)$ ist, desto besser ist der Algorithmus.

Approximationsgüte

Definition 6.4

Ein Approximationsalgorithmus A für ein Minimierungs- bzw. Maximierungsproblem Π hat eine Approximationsgüte von $r \geq 1$ bzw. $r \leq 1$, wenn

$$w_A(I) \leq r \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad w_A(I) \geq r \cdot \mathrm{OPT}(I)$$

für alle Instanzen $I \in \mathcal{S}_I$ gilt. Wir sagen dann, dass A ein r-Approximationsalgorithmus ist.

Fortsetzung Definition.

Hängt der Approximationsfaktor von der Länge der Eingabe ab, dann ist

$$r : \mathbb{N} \to [1, \infty) \quad \text{bzw.} \quad r : \mathbb{N} \to [0, 1]$$

eine Funktion und es muss

$$w_A(I) \leq r(|I|) \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad w_A(I) \geq r(|I|) \cdot \mathrm{OPT}(I)$$

für alle Instanzen $I \in \mathcal{S}_I$ gelten. Hierbei bezeichnet $|I|$ die Länge der Eingabe.

Diskussion Approximationsgüte

- Haben wir für ein Minimierungsproblem einen 2-Approximationsalgorithmus, so bedeutet dies, dass der Algorithmus für jede Instanz eine Lösung berechnet, deren Wert höchstens doppelt so groß ist wie der optimale Wert.
- Haben wir einen $\frac{1}{2}$-Approximationsalgorithmus für ein Maximierungsproblem, so berechnet der Algorithmus für jede Instanz eine Lösung, deren Wert mindestens halb so groß ist wie der optimale Wert.

- Ein Polynomialzeitalgorithmus für ein Optimierungsproblem, der stets eine optimale Lösung berechnet, ist ein 1-Approximationsalgorithmus.
- Ist ein Optimierungsproblem NP-schwer, so kann, unter der Voraussetzung $P \neq NP$, kein 1-Approximationsalgorithmus existieren.
- Es wäre aber durchaus möglich, dass ein 1.01-Approximationsalgorithmus existiert, also ein Algorithmus mit polynomieller Laufzeit, der stets eine Lösung berechnet, deren Wert höchstens um 1% größer als der optimale Wert ist.

Beispiel Approximationsgüte: TSP

Beispiel 6.5

Aus der Graphentheorie kennen sie den folgenden 2-Approximationsalgorithmus für das metrische TSP (Traveling Salesman Problem).

Eine Instanz wird beschreiben durch eine Knotenmenge V und eine Metrik d auf V.

1. Sei $G = (V, E)$ der vollständiger Graph mit Knotenmenge V. Berechne ein Minimalgerüst T von G bezüglich der Metrik d.
2. Führe auf T eine Tiefensuche aus und vergebe dabei die Tiefensuchnummern in der Reihenfolge der Knotenbesuche.
3. Gebe die Knoten in der Reihenfolge der Tiefensuchnummern aus, zusätzlich den Startknoten am Ende nochmals. Dies beschreibt einen Hamiltonkreis C als Lösung.

Beispiel Approximationsgüte: Rucksackproblem

Beispiel 6.6

- Der Algorithmus GreedyKP kann beliebig schlechte Lösungen liefern.
- Durch eine einfache Anpassung erreichen wir einen $\frac{1}{2}$-Approximationsalgorithmus.
- siehe Folie 232 ff.

Keine Approximationsgüte für das allgemeine TSP möglich

Satz 6.7

Wenn es einen $1+\epsilon$-Approximationsalgorithmus für das TSP gibt, gilt $\mathcal{P} = \mathcal{NP}$.

Beweis.

Es sei $G = (V, E)$ ein Graph. Wir wollen entscheiden, ob G hamiltonsch ist.

Wir betrachten dazu den vollständigen Graphen K mit Knotenmenge V und Kantengewichten

$$c_e = \begin{cases} 1 & \text{für } e \in E \\ 2 + \epsilon|V| & \text{sonst.} \end{cases}$$

G ist genau denn hamiltonsch ist, wenn der vollständige Graph K eine TSP-Tour mit einer Länge von $|V| =: n$ enthält.

Fortsetzung Beweis.

Es sei A ein $1+\epsilon$-Approximationsalgorithmus für das TSP.

Wenn G hamiltonsch ist, muss A eine Tour mit einer Länge $w_A \leq (1+\epsilon)n$ liefern.

Wenn die Tour, die A ausgibt, eine Kante $\notin E$ enthält, würde die Tourlänge

$$w_A \geq (n-1) + (2+\epsilon n) = (1+\epsilon)n + 1 > (1+\epsilon)n$$

betragen. Solch eine Tour dürfte A also nicht ausgeben.

Der Approximationsalgorithmus müsste also eine optimale Tour ausgeben.

Damit hätten wir einen polynomialen Algorithmus für HC.

Greedy

- Paradigma für Optimierungsprobleme
- greedy = gierig, gefräßig
- Man versucht ein Optimierungsproblem durch lokale Auswahl eines jeweils besten Elements unter Beachtung der Nebenbedingungen zu lösen.
- Allgemein liefert dies keine optimale Lösung sondern nur irgendeine Lösung.

Teilmengensystem

Wir betrachten Greedy-Algorithmen formal auf der Basis von Teilmengensystemen.

Definition 6.8

Es sei E eine endliche Menge und $\mathcal{T}$ eine nichtleere Menge von Teilmengen von E.

Das Mengensystem $(E, \mathcal{T})$ heißt Teilmengensystem, wenn für alle $A, B \subseteq E$ gilt:

$$A \in \mathcal{T} \text{ und } B \subseteq A \Longrightarrow B \in \mathcal{T}$$

d. h., jede der Teilmengen in $\mathcal{T}$ ist bezüglich $\subseteq$ abgeschlossen.

Beispiele für Teilmengensysteme

Beispiel 6.9

1. Es sei $E = \{e_1, \ldots, e_n\}$ eine beliebige endliche Menge und $k \leq n$. Dann ist $(E, \mathcal{T})$ mit

$$\mathcal{T} = \{A \subseteq E \,|\, |A| \leq k\}$$

ein Teilmengensystem.

2. Es sei $G = (V, E)$ ein zusammenhängender Graph. Dann ist $(E, \mathcal{T})$ mit

$$\mathcal{T} = \{F \subseteq E \,|\, (V, F) \text{ ist ein kreisfreier Untergraph von } G\}$$

ein Teilmengensystem.

Fortsetzung Beispiel.

3. Es sei $G = (V, E)$ ein vollständiger Graph. Dann ist $(E, \mathcal{T})$ mit

$$\mathcal{T} = \{F \subseteq E \mid F \text{ ist Kantenteilmenge eines Hamiltonkreis}\}$$

ein Teilmengensystem.

4. Es sei $E = \{e_1, \ldots, e_n\}$ eine beliebige endliche Menge und $c : E \to \mathbb{R}_+$ eine Gewichtung der Elemente von E. Weiterhin sei $K \in \mathbb{R}_+$. Dann ist $(E, \mathcal{T})$ mit

$$\mathcal{T} = \left\{ F \subseteq E \,\middle|\, \sum_{e \in F} c(e) \leq K \right\}$$

ein Teilmengensystem.

Teilmengensystem und Optimierungsproblem

Definition 6.10

Das zu einem Teilmengensystem $(E, \mathcal{T})$ gehörige Optimierungsproblem besteht darin, für eine beliebige Gewichtsfunktion $w : E \to \mathbb{R}$ eine in $\mathcal{T}$ maximale Menge T zu finden, deren Gesamtgewicht

$$w(T) := \sum_{e \in T} w(e)$$

maximal (bzw. minimal) ist.

Bemerkung: Eine Menge $T \in \mathcal{T}$ ist maximal, wenn es in $\mathcal{T}$ keine echte Obermenge von T gibt.

Beispiele: Teilmengensystem und Optimierungsproblem

Beispiel 6.11

1. MST entspricht dem Optimierungsproblem zum Teilmengensystem (2) aus Beispiel 6.9. Hierbei entspricht $w(e)$ dem Gewicht einer Kante.
2. TSP entspricht dem Optimierungsproblem zum Teilmengensystem (3) aus Beispiel 6.9. Hierbei entspricht $w(e)$ der Länge einer Kante.

Greedy-Algorithmus

Algorithmus 6.12 (Maximierung)

Gegeben: Teilmengensystem $(E, \mathcal{T})$ *mit* $E = \{e_1, \ldots, e_n\}$

Ordne alle Elemente $e \in E$ *absteigend nach Gewicht:*
$w(e_1) \geq w(e_2) \geq \cdots \geq w(e_n)$
$T = \emptyset$
for $i := 1$ **to** n **do**
    if $T \cup \{e_i\} \in \mathcal{T}$ **then**
        $T := T \cup \{e_i\}$
    end
end
gebe T *aus*

Bemerkung: Für eine Minimierung werden die Elemente von E aufsteigend sortiert, also $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_n)$.

Anwendung von Greedy: Algorithmus von Kruskal

Optimierungsproblem: MST, siehe Beispiel 6.2.

Algorithmus 6.13 (Kruskal)

$F := \emptyset$; $ZHK := \emptyset$; $i := 0$;
$L :=$ *Liste der Kanten aufsteigend sortiert nach ihrer Länge;*
for all $v \in V$ **do** $ZHK := ZHK \cup \{\{v\}\}$; **end**
while $|ZHK| > 1$ **do**
    $i := i + 1$;
    Es sei $\{v, w\}$ *das i-te Element von L;*
    if *v und w liegen in verschiedenen ZHKs* K_1 *und* K_2 **then**
        $ZHK := (ZHK \setminus \{K_1, K_2\}) \cup \{\{K_1 \cup K_2\}\}$;
        $F := F \cup \{\{v, w\}\}$;
    end
end
Ausgabe von F;

Diskussion Greedy-Algorithmus

Der Greedy-Algorithmus liefert nicht immer eine Menge T mit maximalem Gewicht.

Beispiel 6.14

1. Sei $E = \{a, b, c\}$ und $\mathcal{T} = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}\}$ mit $w(a) = w(b) = 2, w(c) = 3$.

   Dann berechnet der Greedy-Algorithmus $T = \{c\}$ mit $w(T) = 3$, obwohl $T' = \{a, b\}$ mit $w(T') = 4$ optimal ist.
2. Für TSP (Teilmengensystem (3) aus Beispiel 6.9) liefert der Greedy-Algorithmus i. A. keine optimale Lösung (siehe Beispiel 6.20).

Frage: Für welche Teilmengensysteme liefert der Greedy-Algorithmus eine optimale Lösung?

Matroid

Definition 6.15

Ein Teilmengensystem $(E, \mathcal{T})$ heißt Matroid, wenn für alle $A, B \in \mathcal{T}$ das folgende Austauschaxiom gilt:

$$|A| = |B| + 1 \quad \Longrightarrow \quad \exists a \in A \setminus B : B \cup \{a\} \in \mathcal{T}.$$

Die Mengen in $\mathcal{T}$ werden unabhängige Mengen genannt. Jede maximale unabhängige Menge heißt Basis.

Kardinalität der Basen

Lemma 6.16

Also Basen eines Matroids haben die gleiche Kardinalität.

Beweis.

Ann.: Es existieren zwei Basen A, B mit $|A| \neq |B|$.

- O.B.d.A. gelte $|A| > |B|$.
- Wegen der Teilmengensystemeigenschaft folgt, dass eine Menge $A' \subseteq A$ existiert mit $|A'| = |B| + 1$.
- Mit dem Austauschaxiom folgt, dass $a \in A'$ existiert mit $B \cup \{a\} \in \mathcal{T}$.
- Wegen $B \subseteq B \cup \{a\}$ ist dies ein Widerspruch zur Basiseigenschaft von B.

Optimalität des Greedy-Algorithmus für Matroide

Satz 6.17

Der Greedy-Algorithmus löst das zu einem Teilmengensystem $(E, \mathcal{T})$ gehörige Optimierungsproblem für jede Gewichtsfunktion $w : E \to \mathbb{R}$ genau dann, wenn $(E, \mathcal{T})$ ein Matroid ist.

Beweis

"$\Leftarrow$":

- Nach Lemma 6.16 haben alle Basen von $(E, \mathcal{T})$ die gleiche Kardinalität.
- Es sei k diese Kardinalität und es sei $T = \{e_1, \ldots, e_k\}$ die vom Greedy-Algorithmus berechnete Menge $T \in \mathcal{T}$.
  Es gelte $w(e_1) \geq \cdots \geq w(e_k)$.
- Es sei $U = \{g_1, \ldots, g_k\} \in \mathcal{T}$ eine andere Basis aus $\mathcal{T}$.
  Es gelte $w(g_1) \geq \cdots \geq w(g_k)$.

Fortsetzung Beweis.

- Annahme: Es existiert ein i mit $w(g_i) > w(e_i)$.
- Es sei i der kleinste solche Index, also $w(g_1) \leq w(e_1), \ldots, w(g_{i-1}) \leq w(e_{i-1})$.
- Sei $A = \{g_1, \ldots, g_i\}$ und $B = \{e_1, \ldots, e_{i-1}\}$. Mit dem Austauschaxiom folgt, dass $g_j \in A \setminus B$ existiert, so dass $B \cup \{g_j\} \in \mathcal{T}$ gilt.
- Wegen $w(g_j) \geq w(g_i) > w(e_i)$ hätte der Greedy-Algorithmus vor e_i schon g_j genommen. Widerspruch!
- Also gilt $w(g_i) \leq w(e_i)$ für $1 \leq i \leq k$ und somit $w(U) \leq w(T)$.

Fortsetzung Beweis.

"$\Rightarrow$": Das Austauschaxiom gelte nicht.

- Dann existieren $A, B \in \mathcal{T}$ mit $|A| = |B| + 1$ und $B \cup \{a\} \notin \mathcal{T}$ für alle $A \setminus B$.
- Es sei $r := |A|$ und $w : E \to \mathbb{R}$ sei gegeben durch:

$$w(e) := \begin{cases} r+1 & \text{für } x \in B, \\ r & \text{für } x \in A \setminus B, \\ 0 & \text{sonst.} \end{cases}$$

- Der Greedy-Algorithmus wählt dann eine Menge T mit $T \supseteq B$ und $T \cap (A \setminus B) = \emptyset$ und Gewicht $w(T) = |B| \cdot (r+1) = r^2 - 1$.
- Wählt man stattdessen ein $U \in \mathcal{T}$ mit $U \supseteq A$, dann ergibt sich $w(U) \geq |A| \cdot r = r^2 > w(T)$.

Graphisches Matroid

Folgerung 6.18

Das Teilmengensystem von Beispiel 6.9 (2) ist ein Matroid.

Beweis.

- Der Kruskal-Algorithmus entspricht dem Greedy-Algorithmus für dieses Teilmengensystem und er liefert für jede Kantengewichtung eine optimale Lösung (siehe Graphentheorie).
- Mit Satz 6.17 folgt, dass das Teilmengensystem ein Matroid ist.

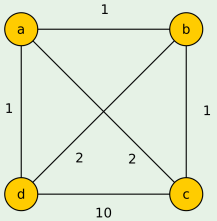
Definition 6.19

Das Teilmengensystem von Beispiel 6.9 (2) heißt graphisches Matroid.

Weitere Teilmengensysteme

Beispiel 6.20

- Das Teilmengensystem von Beispiel 6.9 (1) ist ein Matroid. Dies lässt sich einfach direkt nachweisen.
- Das Teilmengensystem von Beispiel 6.9 (3) ist kein Matroid.
  Begründung: Der Greedy-Algorithmus liefert für den folgenden Graphen den Hamiltonkreis (a, b, c, d, a) mit Gesamtlänge 13.



Optimal wäre (a, c, b, d, a) mit Gesamtlänge 6.

Anwendung Scheduling

Beispiel 6.21

Gegeben Sei eine Menge $J = \{1, 2, \ldots, n\}$ von Jobs.
Jeder Job hat

- eine Deadline $d(i)$ und
- eine Strafe $p(i)$, die bei nicht rechtzeitiger Fertigstellung gezahlt werden muss.

Pro Tag kann nur ein Job abgearbeitet werden.
Wie müssen wir die Jobs einplanen, so dass die Gesamtstrafe minimiert wird?

Job i	1	2	3	4	5	6
$d(i)$	1	1	2	3	3	6
$p(i)$	10	9	6	7	4	2

Fortsetzung Beispiel.

$(J, \mathcal{T})$ mit

$$\begin{aligned} \mathcal{T} &:= \{P \subseteq J | \forall j \in P : j \text{ ist pünktlich planbar}\} \\ &= \{P \subseteq J | \forall n \in \mathbb{N} : |\{j \in P | d(j) \leq n\}| \leq n\} \end{aligned}$$

ist ein Matroid. Also nehmen wir $p(i)$ als Gewichtsfunktion und wenden den Greedy-Algorithmus an.
Terminplan:

Job i	1	2	3	4	5	6
$d(i)$	1	1	2	3	3	6
$p(i)$	10	9	6	7	4	2
$t(i)$	1	5	2	3	6	4

Strafe: 13. Die Jobs 2 und 5 werden nicht rechtzeitig fertig.

Matching

Definition 6.22

Es sei $G = (V, E)$ ein Graph.

Eine Kantenteilmenge $M \subseteq E$ heißt Matching, wenn es keinen Knoten $v \in V$ gibt, der zu mehr als einer Kante aus M inzident ist.

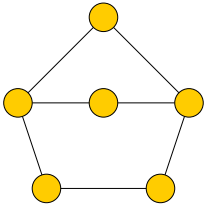
Ein Matching $M \subseteq E$ heißt (inklusions-)maximal, wenn für alle $e \in E \setminus M$ gilt, dass $M \cup \{e\}$ kein Matching ist.

Greedy-Algorithmus für Vertex Cover

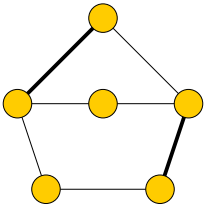
Algorithmus 6.23 (Matching-VC)

1. *Berechne wie folgt ein maximales Matching $M \subseteq E$:*
   1. *Setze $M = \emptyset$.*
   2. *Gehe die Kantenmenge E einmal durch. Füge dabei $e = \{v, w\} \in E$ zu M hinzu, wenn weder v noch w inzident zu einer Kante aus M ist.*
2. *Sei $V(M)$ die Menge aller Knoten, die zu einer Kante in M inzident sind. Gib $V(M)$ aus.*

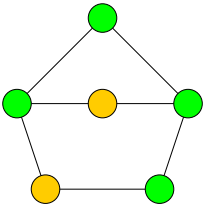
Veranschaulichung Matching-VC



$G = (V, E)$



maximales Matching



zugehörige Knotenüberdeckung

Eigenschaften von Matching-VC

Satz 6.24

Für einen Graphen $G = (V, E)$ hat der Algorithmus Matching-VC die folgenden Eigenschaften:

1. *Seine Laufzeit beträgt $O(|V| + |E|)$.*
2. *Er berechnet stets eine Knotenüberdeckung G.*
3. *Die Approximationsgüte beträgt 2.*

Matching-VC ist also ein 2-Approximationsalgorithmus mit linearer Laufzeit.

Beweis der Eigenschaften

Beweis.

1.
   - Wir numerieren die Knoten mit Zahlen $0, \ldots, |V| - 1$.
   - Wir speichern in einem Feld für die Knoten (Knotennummer als Index), welche Knoten zu einer Kante aus M inzident sind.
   - Wir iterieren über die Kanten und testen dabei in jeder Iteration, ob wir die aktuelle Kante $e = \{v, w\}$ zu M hinzufügen können. Laufzeit für Test und Hinzufügen ist $O(1)$, insgesamt also $O(|E|)$.
   - Wir laufen über das Knotenfeld und geben die Knoten, die als inzident markiert sind, aus. Laufzeit: $O(|V|)$.
2. Annahme: $V(M)$ ist keine Knotenüberdeckung.
   - Dann gibt es eine Kante $e = \{v, w\}$ mit $v, w \notin V(M)$.
   - Damit wäre $M \cup \{e\}$ ein Matching.
   - Somit hätte Matching-VC e zu M hinzufügen müssen. Widerspruch!

Fortsetzung Beweis.

3.
   - Sei V^* eine minimale Knotenüberdeckung.
   - V^* deckt jede Kante aus M ab. Für jede Kante aus M muss also mindestens einer der beiden Knoten in V^* sein.
   - Kein Knoten $v \in V^*$ kann mehr als eine Kante von M abdecken (sonst kein Matching).
   - Damit folgt: $\mathrm{OPT}(G) = |V^*| \geq |M|$.
   - Für unsere konstruierte Lösung $V(M)$ gilt $|V(M)| = 2|M|$.

Insgesamt folgt damit:

$$\frac{|V(M)|}{\mathrm{OPT}(G)} \leq \frac{2|M|}{|M|} = 2.$$

Beweis der Approximationsgüte

Zum Beweis einer Approximationsgüte bei einem Minimierungsproblem müssen wir zwei Dinge tun:

1. Herleitung einer oberen Schranke für die Lösung des Approximationsalgorithmus.
   - Bei Matching-VC war dies $2|M|$.
2. Herleitung einer unteren Schranke für den Wert der optimalen Lösung.
   - Bei Matching-VC war dies $|M|$.

Der Quotient dieser Schranken ist ein Abschätzung für die Approximationsgüte.

Zum Beweis einer Approximationsgüte bei einem Maximierungsproblem:

1. Herleitung einer unteren Schranke für die Lösung des Approximationsalgorithmus.
2. Herleitung einer oberen Schranke für den Wert der optimalen Lösung.

Die Schwierigkeit besteht oft darin, eine Schranke für den Wert einer optimalen Lösung zu finden.

Set Cover

Definition 6.25

Das Problem Set Cover (SC) wird wie folgt definiert:

Eingabe: Grundmenge S mit n Elementen,
m Teilmengen $S_1, \ldots, S_m \subseteq S$ mit $\cup_{i=1}^{m} S_i = S$,
Kosten $c_1, \ldots, c_m \in \mathbb{N}$.

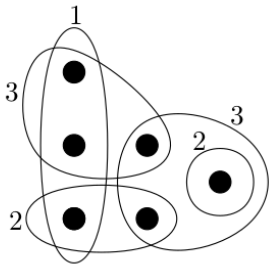
Lösungen: alle Teilmengen $A \subseteq \{1, \ldots, m\}$ mit:

$$\cup_{i \in A} S_i = S$$

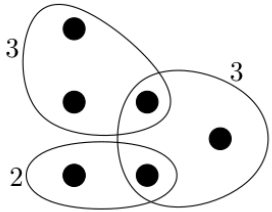
Eine Menge A mit dieser Eigenschaft nennen wir Set Cover von S.

Zielfunktion: minimiere $c(A) = \sum_{i \in A} c_i$

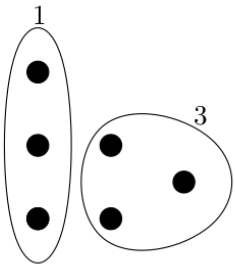
Veranschaulichung Set Cover



(a) Instanz



(b) Set Cover mit Wert 8



(c) Set Cover mit Wert 4

Fakt 6.26

Das Problem Set Cover ist NP-schwer.

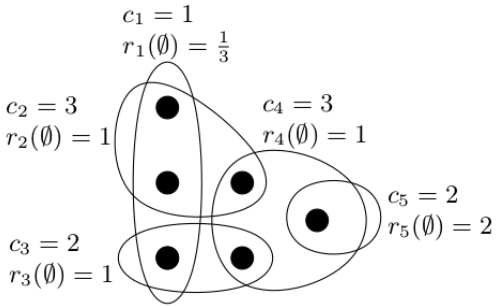
Greedy-Algorithmus für Set Cover

Algorithmus 6.27 (Greedy-SC)

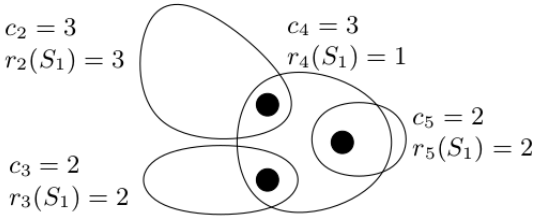
$A := \emptyset$; $C := \emptyset$;
while $C \neq S$ **do**
    Wähle i bzw. S_i mit minimalen relativen Kosten $r_i(C) = \frac{c_i}{|S_i \setminus C|}$.
    Setze $\text{price}(x) := \frac{c_i}{|S_i \setminus C|}$ *für alle* $x \in S_i \setminus C$.
    $A := A \cup \{i\}$; $C := C \cup S_i$;
end
return A

Bemerkung: Die mittlere Anweisung in der Schleife dient nur der nachfolgenden Analyse.

Veranschaulichung Greedy-SC



(a) S_1 hat minimale relative Kosten



(b) S_4 hat minimale relative Kosten

Eigenschaften von Greedy-SC

Definition 6.28

Mit $H_n = \sum_{i=1}^{n} \frac{1}{i}$ bezeichnen wir die *n*-te harmonische Zahl.

Satz 6.29

Der Algorithmus Greedy-SC ist ein H_n-Approximationsalgorithmus für das Problem Set Cover.

Beweis.

- vergleiche $c(A)$ mit Kosten einer optimalen Lösung
- Wir verteilen $c(A)$ auf die Elemente der Grundmenge S.
- Hierzu dient price : $S \to \mathbb{R}$. Es gilt

$$c(A) = \sum_{x \in S} \text{price}(x).$$

- Was können wir über die Kosten der optimalen Lösung aussagen?
- Wir ordnen die Elemente von S in der Reihenfolge, in der sie zu C hinzugefügt werden.
- Sei $x_1, \ldots, x_n$ die Reihenfolge, die sich ergibt.

Lemma 6.30

Für jedes $k \in \{1, \ldots, n\}$ *gilt* $\text{price}(x_k) \leq \text{OPT}/(n - k + 1)$.

Beweis von Lemma 6.30.

- Wir betrachten ein beliebiges $k \in \{1, \ldots, n\}$. Sei $i \in \{1, \ldots, m\}$ der Index von S_i, durch deren Hinzunahme x_k abgedeckt wird. Sei A die Auswahl unmittelbar bevor i zu A hinzugefügt wird.
- Nicht abgedeckte Elemente $\overline{C} = S \setminus C$ mit $C = \bigcup_{j \in A} S_j$.
- $|\overline{C}| \geq n - k + 1$, da höchstens die Elemente $x_1, \ldots, x_{k-1}$ abgedeckt sind.
- Wir betrachten nun die Set-Cover-Instanz I' eingeschränkt auf die Elemente von $\overline{C}$, also bereits abgedeckten Elemente werden aus S und jeder Menge S_j entfernt.
- Sei dann A^* die optimale Lösung für I' mit $\mathrm{OPT}' = c(A^*)$. Damit gilt:

$$\mathrm{OPT}' = \sum_{j \in A^*} c_j = \sum_{j \in A^*} \left(|S_j \setminus C| \cdot \frac{c_j}{|S_j \setminus C|} \right) = \sum_{j \in A^*} \left(\sum_{x_j \in S_j \setminus C} r_j(C) \right).$$

Beweis von Lemma 6.30.

Aus der Wahl von S_i als nächste Menge folgt, dass es in der aktuellen Situation keine Menge gibt, deren relative Kosten kleiner als $r_i(C)$ sind. Damit folgt:

$$\begin{aligned}\mathrm{OPT} \geq \mathrm{OPT}' &= \sum_{j \in A^*} \left(\sum_{x_j \in S_j \setminus C} r_j(C) \right) \\ &\geq \sum_{j \in A^*} \left(\sum_{x_j \in S_j \setminus C} r_i(C) \right) \geq |S \setminus C| \cdot r_i(C) \\ &\geq (n - k + 1) \cdot r_i(C).\end{aligned}$$

Wegen $r_i(C) = \mathrm{price}(x_k)$ ist damit das Lemma beweisen.

Fortsetzung Beweis von Satz 6.29.

Aus Lemma 6.30 folgt nun der Beweis von Satz 6.29:

$$\begin{aligned} c(A) &= \sum_{x \in S} \text{price}(S) = \sum_{k=1}^{n} \text{price}(x_k) \\ &\leq \sum_{k=1}^{n} \frac{\text{OPT}}{n-k+1} = \text{OPT} \cdot \sum_{k=1}^{n} \frac{1}{k} \\ &= \text{OPT} \cdot H_n. \end{aligned}$$

Schranken für harmonische Zahlen

Lemma 6.31

Für alle $n \in \mathbb{N}$ gilt:

$$\log(n+1) \leq H_n \leq 1 + \log(n).$$

Der Beweis ist Übungsaufgabe.

Folgerung 6.32

Der Algorithmus Greedy-SC hat einen Approximationsfaktor von $O(\log(n))$.

Rucksackproblem und dynamische Programmierung

Algorithmus 6.33

```
// Backward Computation
V_{n+1,c} := 0 für c = 0, ..., C
for i := n downto 1 do
    for s := 0 to C do
        V_{i,s} := V_{i+1,s}
        a_{i,s} := 0
        if s ≥ w_i ∧ V_{i,s} < p_i + V_{i+1,s-w_i} then
            V_{i,s} := p_i + V_{i+1,s-w_i}
            a_{i,s} := 1
        end
    end
end
```

Fortsetzung Algorithmus.

// Forward Computation
$z := V_{1,C}$
$s := C$
for $i := 1$ **to** n **do**
    $x_i := a_{i,s}$
    if $x_i = 1$ **then**
        $s := s - w_i$
    end
end
return $\mathbf{x}, z$

Satz 6.34

Algorithmus 6.33 berechnet in Zeit $O(Cn)$ eine optimale Lösung für eine Instanz des Rucksackproblems.

Diskussion Algorithmus

- Dynamische Programmierung: Zustände, Aktionen, Zustandsübergänge
- bei diskreten Zustands- und Aktionenmengen immer als Wegeproblem in einem Graphen modellierbar
- Laufzeit gut einschätzbar, im Gegensatz zu Branch-and-Bound
- Laufzeit $O(Cn)$ ist nicht linear sondern exponentiell!
- Begründung: C ist ein einzelner Wert, die Laufzeit wächst exponentiell in der Kodierungslänge von C.
- praktischer Aspekt: für beschränktes C ein linearer Algorithmus

Pseudopolynomialität

Definition 6.35

Es sei Π ein Optimierungsproblem, in dessen Instanzen ganze Zahlen enthalten sind.

Ein Algorithmus A zur Lösung von Π heißt pseudopolynomieller Algorithmus, wenn seine Laufzeit durch ein Polynom in der Eingabelänge und dem größten Absolutwert der Zahlen der Eingabe nach oben beschränkt ist.

Bemerkungen:

- Algorithmus 6.33 ist ein pseudopolynomieller Algorithmus für das Rucksackproblem.
- Wir können damit Instanzen, für die C polynomiell in der Anzahl der Objekte beschränkt ist, effizient lösen.
- Gilt bspw. $C = O(n^2)$, haben wir einen $O(n^3)$ Algorithmus.

Approximationsschema

Definition 6.36

Ein Approximationsschema A für ein Optimierungsproblem Π ist ein Algorithmus,

- der zu jeder Eingabe der Form (I, ϵ) mit $I \in \mathcal{I}_\Pi$ und $\epsilon > 0$ eine Lösung $A(I, \epsilon) \in \mathcal{S}_I$ berechnet.
- Dabei muss für den Wert $w_A(I, \epsilon) = f_I(A(I, \epsilon))$ dieser Lösung für jede Eingabe (I, ϵ) bei einem Minimierungs- oder Maximierungsproblem Π die Ungleichung

$$w_A(I, \epsilon) \leq (1 + \epsilon) \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad w_A(I, \epsilon) \geq (1 - \epsilon) \cdot \mathrm{OPT}(I)$$

gelten.

- Wie konstruiert man ein Approximationsschema?
- Wie hängt dabei die Laufzeit von ϵ ab?

PTAS und FPTAS

Definition 6.37

Ein Approximationsschema A heißt polynomielles Approximationsschema (PTAS), wenn die Laufzeit von A für jede feste Wahl von $\epsilon > 0$ durch ein Polynom in $|I|$ nach oben beschränkt ist.

Wir nennen ein Approximationsschema A voll-polynomielles Approximationsschema (FPTAS), wenn die Laufzeit von A durch ein bivariates Polynom in $|I|$ und $1/\epsilon$ nach oben beschränkt ist.

Diskussion PTAS und FPTAS

- Jedes FPTAS ist auch ein PTAS. Umgekehrt gilt dies jedoch nicht.
- Ein PTAS könnte z. B. eine Laufzeit von $O(|I|^{1/\epsilon})$ haben.
- Diese ist für jedes feste $\epsilon > 0$ polynomiell in $|I|$.
- Diese Laufzeit ist bei einem FPTAS jedoch nicht erlaubt, weil sie nicht polynomiell in $1/\epsilon$ beschränkt ist.
- Eine erlaubte Laufzeit eines FPTAS ist z. B. $O(|I|^3/\epsilon^2)$.

- Ein PTAS impliziert, dass es für das entsprechende Problem für jede Konstante $\epsilon > 0$ einen $(1+\epsilon)$- bzw. $(1-\epsilon)$-Approximationsalgorithmus gibt.
- Der Grad des Polynoms hängt aber im Allgemeinen vom gewählten ϵ ab.
- Ist die Laufzeit zum Beispiel $O(|I|^{1/\epsilon})$, so ergibt sich für $\epsilon = 0.01$ eine Laufzeit von $O(|I|^{100})$.
- Eine solche Laufzeit ist nur von theoretischem Interesse.
- Bei einem FPTAS ist es hingegen oft so, dass man auch für kleine ϵ eine passable Laufzeit erhält.

Alternativer DP-Ansatz für Rucksackproblem

- Für die Konstruktion eines FPTAS benötigen wir zunächst einen anderen DP-Ansatz.
- Wir betrachten hier (o.B.d.A.) nur den optimalen Zielfunktionswert.
- Teilproblem: Finde unter allen Teilmengen der Objekte $1, \ldots, i$ mit Gesamtnutzen $\geq p$ eine mit dem kleinsten Gewicht.
- Es sei $W(i, p)$ das Gewicht solch einer Teilmenge. Existiert keine solche Teilmenge, so setzen wir $W(i, p) = \infty$.
- Wir lösen das Rucksackproblem, indem wir eine Tabelle konstruieren, die die $W(i, p)$-Werte enthält.
- Obere Schranke für den erreichbaren Nutzen: nP mit $P := \max_{i \in \{1,\ldots,n\}} p_i$.

- Seien die Werte $W(i-1,p)$ für ein i und alle $p \in \{0,\ldots,nP\}$ bekannt. Wie können wir $W(i,p)$ aus den bekannten Werten berechnen?
- Gesucht ist eine Teilmenge $I \subseteq \{1,\ldots,i\}$ mit $\sum_{i\in I} p_i \geq p$ und kleinstmöglichem Gewicht. Wir unterscheiden nun zwei Fälle.
- $i \notin I$
  - Dann gilt $I \subseteq \{1,\ldots,i-1\}$ mit $\sum_{i\in I} p_i \geq p$.
  - Unter allen diesen Teilmengen besitzt I minimales Gewicht.
  - Damit gilt $W(i,p) = W(i-1,p)$.
- $i \in I$
  - Dann ist $I \setminus \{i\}$ eine Teilmenge von $\{1,\ldots,i-1\}$ mit Nutzen von mindestens $p - p_i$.
  - Unter allen Teilmengen, die diese Bedingung erfüllen, hat $I \setminus \{i\}$ minimales Gewicht.
  - Es gilt also $w(I \setminus \{i\}) = W(i-1,p-p_i)$ und somit $W(i,p) = W(i-1,p-p_i) + w_i$.

Algorithmus 6.38

1. $W(i,p) := 0$ *für* $i \in \{1,\ldots,n\}$ *und* $p \leq 0$.
2. $P := \max_{i\in\{1,\ldots,n\}} p_i$
3. $W(1,p) := w_1$ *für* $p = 1,\ldots,p_1$
4. $W(1,p) := \infty$ *für* $p = p_1+1,\ldots,nP$
5. $W(i,p) = \min\{W(i-1,p), W(i-1,p-p_i)+w_i\}$ *für* $i = 2,\ldots,n$ *und* $p = 1,\ldots,nP$
6. **return** $\max p \in \{1,\ldots,nP\}$ *mit* $W(n,p) \leq C$.

Lemma 6.39

Algorithmus 6.38 bestimmt in Zeit $O(n^2P)$ *den maximalen Nutzen für eine Instanz des Rucksackproblems.*

Idee: FPTAS

- Algorithmus 6.38 ist effizient für kleine Nutzenwerte (polynomiell in der Eingabelänge beschränkt).
- Idee: große Nutzenwerten skalieren, d. h. alle durch dieselbe Zahl K teilen.
- Der maximal erreichbare Nutzen wird dann ebenfalls um den Faktor K kleiner.
- optimale Lösung ändert sich nicht
- Trick: Schneide die Nachkommastellen der skalierten Nutzenwerte ab!
- Wähle dabei K so, dass Algorithmus 6.38 polynomielle Laufzeit hat.
- Abhängig von K ergibt sich dann auch eine Approximationsgüte.

FPTAS für Rucksackproblem

Algorithmus 6.40

Eingabe sei $(\mathcal{I}, \epsilon)$ *mit* $\mathcal{I} = (p_1, \ldots, p_n, w_1, \ldots, w_n, C)$.

1. $P := \max_{i \in \{1,\ldots,n\}} p_i$
2. $K := \frac{\epsilon P}{n}$
3. $p'_i := \left\lfloor \frac{p_i}{K} \right\rfloor$ *für* $i = 1, \ldots, n$
4. *Berechne mit Algorithmus 6.38 eine optimale Lösung für die Instanz* $\mathcal{I} = (p'_1, \ldots, p'_n, w_1, \ldots, w_n, C)$.

Satz 6.41

Algorithmus 6.40 ist ein FPTAS für das Rucksackproblem mit einer Laufzeit von $O(n^3/\epsilon)$.

Beweis für FPTAS

Beweis.

Laufzeit:

- Wird durch den Aufruf von Algorithmus 6.38 dominiert.
- Sei $P' = \max_{i \in \{1,\ldots,n\}} p'_i$.
- Es gilt:

$$P' = \max_{i \in \{1,\ldots,n\}} \left\lfloor \frac{p_i}{K} \right\rfloor = \left\lfloor \frac{P}{K} \right\rfloor = \left\lfloor \frac{n}{\epsilon} \right\rfloor \leq \frac{n}{\epsilon}.$$

- Somit beträgt die Laufzeit $O(n^2 P') = O(n^3/\epsilon)$.

Approximation:

- Sei $I' \subseteq \{1, \ldots, n\}$ eine optimale Lösung für das skalierte Problem mit Nutzenwerten $p'_1, \ldots, p'_n$.
- Sei $I \subseteq \{1, \ldots, n\}$ eine optimale Lösung für das originäre Problem mit Nutzenwerten $p_1, \ldots, p_n$.

Fortsetzung Beweis.

- I' ist auch optimal für die Nutzenwerte $p_1^*, \ldots, p_n^*$ mit $p_i^* = K \cdot p_i'$.
- Für $J \subseteq \{1, \ldots, n\}$ sei

$$p(J) = \sum_{i \in J} p_i \quad \text{und} \quad p^*(J) = \sum_{i \in J} p_i^*.$$

- Dann ist $p(I')$ der Wert der Lösung von Algorithmus 6.40 und $p(I)$ der Wert OPT einer optimalen Lösung.
- Die Nutzenwerte p_i und p_i^* weichen nicht stark voneinander ab. Es gilt

$$p_i^* = K \left\lfloor \frac{p_i}{K} \right\rfloor \geq K \left(\frac{p_i}{K} - 1 \right) = p_i - K$$

und

$$p_i^* = K \left\lfloor \frac{p_i}{K} \right\rfloor \leq p_i.$$

Fortsetzung Beweis.

- Dementsprechend gilt für jede Teilmenge $J \subseteq \{1, \ldots, n\}$

$$p(J) - nK \leq p^*(J) \leq p(J).$$

- I' ist eine optimale Lösung für die Nutzenwerte $p_1^*, \ldots, p_n^*$ (siehe oben).
- Es gilt also $p^*(I') \geq p^*(I)$ und somit

$$p(I') \geq p^*(I') \geq p^*(I) \geq p(I) - nK.$$

- Da jedes Objekt alleine in den Rucksack passt, gilt $p(I) \geq P$ und damit

$$\frac{p(I')}{\mathrm{OPT}} = \frac{p(I')}{p(I)} \geq \frac{p(I) - nK}{p(I)} = 1 - \frac{\epsilon P}{p(I)} \geq 1 - \epsilon.$$

Diskussion

- Beim FPTAS Abrundung der Nutzenwerte so, dass der pseudopolynomielle Algorithmus die abgerundete Instanz in polynomieller Zeit lösen kann.
- gute Approximation der optimalen Lösung bezüglich der eigentlichen Nutzenwerte
- Technik zum Entwurf eines FPTAS lässt sich für andere Probleme anwenden, für die es einen pseudopolynomiellen Algorithmus gibt.
- Geht das auch mit Runden der Gewichtswerte?
- eher nicht, optimale Lösung dann u. U. nicht mehr zulässig, evtl. sehr schlechte Approximation
- Faustregel: guter Ansatz, wenn pseudopolynomial bzgl. Koeffizienten in der Zielfunktion.
- Daher auch der Ansatz mit Algorithmus 6.38 statt 6.33.

Max-SAT

Definition 6.42

Das Optimierungsproblem MAX-SAT lautet:

- Instanzen: Gegeben ist eine aussagenlogische Formel

$$\Phi = C_1 \wedge C_2 \wedge \ldots C_m$$

in KNF

  - mit den Klauseln $C_1, \ldots, C_m$,
  - in den Variablen $x_1, \ldots, x_n$,
  - sowie Gewichten $w_1, \ldots, w_m \in \mathbb{R}_+$.

- Lösungen: alle Belegungen der Variablen $x_1, \ldots, x_n$.
- Zielfunktion: maximiere das Gesamtgewicht der durch die Beleguung erfüllten Klauseln.

Randomisierter Approximationsalgorithmus

Definition 6.43

- Ein randomisierter Approximationsalgorithmus A für ein Minimierungs- oder Maximierungsproblem Π erreicht einen Approximationsfaktor von $r \geq 1$ bzw. $r \leq 1$, wenn

$$E(w_A(I)) \leq r \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad E(w_A(I)) \geq r \cdot \mathrm{OPT}(I)$$

  für alle Instanzen $\mathcal{I}_\Pi$ gilt. Wir sagen dann, dass A ein randomisierter r-Approximationsalgorithmus ist.

- In vielen Fällen hängt der Approximationsfaktor von der Eingabelänge ab. Dann ist $r : \mathbb{N} \to [0, \infty)$ eine Funktion und es muss

$$E(w_A(I)) \leq r(|I|) \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad E(w_A(I)) \geq r(|I|) \cdot \mathrm{OPT}(I)$$

  für alle Instanzen I gelten.

Randomisierter Approximationsalgorithmus für Max-SAT

Algorithmus 6.44 (RandMaxSat)

Erzeuge eine Belegung $B : \{x_1, \ldots, x_n\} \to \{0, 1\}$ *wie folgt:*

Setze jede Variable x_i *unabhängig von den anderen Variablen mit einer Wahrscheinlichkeit von* $\frac{1}{2}$ *auf* 0 *bzw.* 1.

Satz 6.45

Algorithmus 6.44 ist ein randomisierter $\frac{1}{2}$*-Approximationsalgorithmus für Max-SAT.*

Beweis.

- Wir definieren für jede Klausel C_j eine Zufallsvariable

$$Y_j = \begin{cases} w_j & \text{falls } B \text{ Klausel } C_j \text{ erfüllt,} \\ 0 & \text{sonst.} \end{cases}$$

- Dann gilt für das Gesamtgewicht Y der erfüllten Klauseln

$$E(Y) = E\left(\sum_{j=1}^{m} Y_j\right) = \sum_{j=1}^{m} E(Y_j) = \sum_{j=1}^{m} w_j \cdot P(B \text{ erfüllt } C_j).$$

- Wann wird C_j nicht erfüllt?
  - Wenn alle positiven Literale in C_j auf 0 und
  - alle negativen Literale auf 1 gesetzt sind.

  Bezeichne l_j die Anzahl an Literalen in C_j.

Fortsetzung Beweis.

- Damit folgt

$$P(B \text{ erfüllt } C_j \text{ nicht}) = \left(\frac{1}{2}\right)^{l_j} \leq \frac{1}{2}.$$

- Das Gesamtgewicht aller Klauseln ist eine obere Schranke für OPT. Damit folgt

$$E(Y) = \sum_{j=1}^{m} w_j \cdot P(B \text{ erfüllt } C_j) \geq \sum_{j=1}^{m} w_j \cdot \frac{1}{2} \geq \frac{1}{2} \cdot \text{OPT}.$$

Bemerkungen:

- Aus Beweis folgt: Der Approximationsfaktor wird besser, je länger die Klauseln sind.
- Für MAX-3-SAT erhalten wir einen Approximationsfaktor von $\frac{7}{8}$.

Max-SAT als ILP

Maximiere

$$\sum_{j=1}^{m} w_j z_j$$

unter den Nebenbedingungen

$$\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j \quad \text{für } j = 1, \ldots, m$$

$$y_i \in \{0, 1\} \quad \text{für } i = 1, \ldots, n$$

$$z_j \in \{0, 1\} \quad \text{für } j = 1, \ldots, m$$

Randomisiertes Runden für Max-SAT

Algorithmus 6.46

Erzeuge eine Belegung $B : \{x_1, \ldots, x_n\} \to \{0, 1\}$ *wie folgt:*

1. *Berechne eine Lösung für die LP-Relaxation (also* $y_j, z_j \in [0, 1]$*) von Max-SAT. Es sei* $(\mathbf{y}, \mathbf{z})$ *eine optimale Lösung der LP-Relaxation.*
2. *Setze jede Variable* x_i *unabhängig von den anderen Variablen mit einer Wahrscheinlichkeit von* y_i *auf* 1 *bzw. mit einer Wahrscheinlichkeit von* $1 - y_i$ *auf* 0*.*

Satz 6.47

Algorithmus 6.46 ist ein randomisierter $\left(1 - \frac{1}{e}\right)$*-Approximationsalgorithmus für Max-SAT.*

Beweis.

- Sei P_j die Menge der positiven Literale von C_j und N_j die Menge der negativen Literale.
- Damit gilt

$$P(B \text{ erfüllt } C_j \text{ nicht}) = \prod_{i \in P_j} (1 - y_i) \cdot \prod_{i \in N_j} y_i.$$

- Wir nutzen die Ungleichung vom arithmetischen und geometrischen Mittel

$$\sqrt[n]{\prod_{i=1}^{n} a_i} \leq \frac{1}{n} \sum_{i=1}^{n} a_i \quad \text{bzw.} \quad \prod_{i=1}^{n} a_i \leq \left(\frac{1}{n} \sum_{i=1}^{n} a_i \right)^n.$$

Fortsetzung Beweis.

- Angewendet mit $|C_j| = |P_j| + |N_j|$ ergibt sich

$$\begin{aligned}
\prod_{i \in P_j} (1 - y_i) \cdot \prod_{i \in N_j} y_i &\leq \left(\frac{1}{|C_j|} \left(\sum_{i \in P_j} (1 - y_i) + \sum_{i \in N_j} y_i \right) \right)^{|C_j|} \\
&= \left(\frac{1}{|C_j|} \left(|C_j| + \sum_{i \in P_j} (-y_i) + \sum_{i \in N_j} (y_i - 1) \right) \right)^{|C_j|} \\
&= \left(1 - \frac{1}{|C_j|} \left(\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \right) \right)^{|C_j|} .
\end{aligned}$$

Fortsetzung Beweis.

- Das LP erfüllt die Nebenbedingung $\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j$. Damit folgt

$$P(C_j \text{ nicht erfüllt}) \leq \left(1 - \frac{z_j}{|C_j|}\right)^{|C_j|}$$

bzw.

$$P(C_j \text{ erfüllt}) \geq 1 - \left(1 - \frac{z_j}{|C_j|}\right)^{|C_j|}.$$

- Die Funktion $f(x) = 1 - \left(1 - \frac{x}{|C_j|}\right)^{|C_j|}$ ist konkav. Damit liegt die Gerade g, die durch die Punkte $(0, f(0))$ und $(1, f(1))$ geht, nirgendwo überhalb von f.
- Wegen $f(0) = 0$ gilt

$$g(x) = f(1) \cdot x = \left(1 - \left(1 - \frac{1}{|C_j|}\right)^{|C_j|}\right) \cdot x.$$

Fortsetzung Beweis.

- Damit folgt:

$$P(C_j \text{ erfüllt}) \geq f(z_j) \geq g(z_j) = \left(1 - \left(1 - \frac{1}{|C_j|}\right)^{|C_j|}\right) \cdot z_j.$$

- Sei Y wieder das Gesamtgewicht der erfüllten Klauseln.

$$\begin{aligned} E(Y) &= \sum_{j=1}^{m} w_j \cdot P(C_j \text{ erfüllt}) \\ &\geq \sum_{j=1}^{m} w_j \left(1 - \left(1 - \frac{1}{|C_j|}\right)^{|C_j|}\right) \cdot z_j \\ &\geq \min_{k \geq 1} \left(1 - \left(1 - \frac{1}{k}\right)^{k}\right) \cdot \sum_{j=1}^{m} w_j z_j \end{aligned}$$

Fortsetzung Beweis.

- $$\begin{aligned} &= \min_{k \geq 1} \left(1 - \left(1 - \frac{1}{k}\right)^k\right) z_{LP} \\ &\geq \min_{k \geq 1} \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \mathrm{OPT}. \end{aligned}$$

- Die Funktion $1 - \left(1 - \frac{1}{k}\right)^k$ ist streng monoton fallend und geht von oben gegen $1 - \frac{1}{\mathrm{e}} \approx 0.632$. Damit erhalten wir

$$E(Y) \geq \left(1 - \frac{1}{\mathrm{e}}\right) \cdot \mathrm{OPT}.$$

Zusammenfassung

- Greedy-Algorithmen als Ansatz für Approximationsalgorithmen
- Auf Matroiden liefern Greedy-Algorithmen optimale Lösungen.
- Beweis der Approximationsgüte: Schranken für optimale Lösung, approximative Lösung und diese in Beziehung setzen.
- Dynamische Programmierung und Pseudopolynomialität
- FPTAS durch geeignetes Runden und Nutzung eines pseudopolynomialen Algorithmus
- Randomisierte Approximation: Runden auf Basis der Lösung einer LP-Relaxation