UNIVERSITÄT
LEIPZIG

Algorithmen und Datenstrukturen II

Vorlesung tr3s

Leipzig, 16.04.2024

Peter F. Stadler & Thomas Gatter & Ronny Lorenz

GREEDY-ALGORITHMEN

Greedy-Algorithmen I

Algorithmus

Annahme: Gewichtsfunktion w misst Güte einer Lösung (oder Teillösung).
Ziel: Konstruiere Lösung mit maximalen Gewicht (w).

1. **Starte** mit leerer Lösung
2. **In jedem Schritt:** Erweitere die bisher konstruierte Teillösung.
   Unter allen k möglichen Erweiterungen, die zu Teillösungen $L_1, \ldots, L_k$ führen, wähle L_i so, dass $w(L_i)$ maximal.

Also bedeutet “greedy” (gierig): *Nimm immer das größte Stück*.

Greedy-Algorithmen II

Oft wird die Greedy-Strategie im Sinne einer *Heuristik* verwendet.
(*Heuristik* bedeutet dabei: i.d.R. wird eine brauchbar gute Lösung erzeugt)

Es gibt aber auch Probleme, für die das Greedy-Prinzip optimal ist, d.h. tatsächlich immer zu einer optimalen Lösung führt.

Erkläre den Kruskal Algorithmus aus den 1. Foliensatz als Greedy Algorithmus. Was sind die Teillösungen, was sind die Erweiterungen?

Unabhängigkeitssystem I

Definition

Sei E eine endliche Menge und $\mathcal{M}$ eine Menge von Teilmengen von E, also $\mathcal{M} \subseteq \mathcal{P}(E)$. Dann heißt $(E, \mathcal{M})$ **Mengensystem**.

Ein Mengensystem $(E, \mathcal{M})$ heißt **Unabhängigkeitssystem**, wenn $\emptyset \in \mathcal{M}$ und zu jeder Menge in $\mathcal{M}$ auch alle Teilmengen enthalten sind, also gilt:

Aus $A \in \mathcal{M}$ und $B \subseteq A$ folgt $B \in \mathcal{M}$.

Unabhängigkeitssystem II

Ein Unabhängigkeitssystem ist somit **abgeschlossen** unter Bildung von Teilmengen. Man kann eine in $\mathcal{M}$ gefundene Menge “kleiner” machen und bleibt in $\mathcal{M}$.

Für jede Menge *A* in $\mathcal{M}$ gelten die Elemente von *A* als unabhängig voneinander.

Gewichtsfunktion und Optimierungsproblem I

Definition

Sei $(E, \mathcal{M})$ ein Mengensystem.

Sei w eine Funktion $w : E \to \mathbb{R}^+$, genannt **Gewichtsfunktion**

Erweitere Notation (Gewicht einer Menge $A \in \mathcal{M}$):

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

Gewichtsfunktion und Optimierungsproblem II

Optimierungsproblem

Finde $A^* \in \mathcal{M}$, das die Gewichtssumme maximiert. Für alle $A \in \mathcal{M}$ soll also gelten:

$$w(A^*) \geq w(A).$$

Kanonischer Greedy-Algorithmus

…für Unabhängigkeitssystem $(E, \mathcal{M})$ und Gewichtsfunktion $w : E \to \mathbb{R}^+$:

Algorithmus

$A \leftarrow \emptyset$;

for $e \in E$ *in Reihenfolge absteigender Gewichte* $w(e)$ **do**
  if $A \cup \{e\} \in \mathcal{M}$ **then**
    $A \leftarrow A \cup \{e\}$

return A

Liefert der Algorithmus “Greedy” immer die optimale Lösung? (***immer*** $\hat{=}$ für beliebige Mengensysteme und Gewichte?)

Beispiel: Greedy nicht immer optimal

Beispiel

- $E = \{a, b, c\}$
- $\mathcal{M} = \{\, \emptyset, \{a\}, \{b\}, \{c\}, \{b, c\} \,\}$
- $w(a) = 3,\ w(b) = w(c) = 2$

Dazu liefert der kanonische Greedy-Algorithmus die Lösung $\{a\}$ mit Gewichtssume $w(a) = 3$

Die optimale Lösung ist $\{b, c\}$ mit Gewichtssumme $w(b) + w(c) = 4$.

Gesucht: Kriterium – wann ist Greedy optimal?

Matroid I

Definition (Matroid)

Ein Unabhängigkeitssystem $(E, \mathcal{M})$ heißt **Matroid**, wenn für alle $A, B \in \mathcal{M}$ die folgende **Austauscheigenschaft** gilt:

$$\text{Ist } |A| > |B|, \text{ dann gibt es } x \in A \setminus B, \text{ so dass } B \cup \{x\} \in \mathcal{M}.$$

Definition (Basis)

Sei $(E, \mathcal{M})$ ein Matroid und $B \in \mathcal{M}$ ein maximales Element, also für alle $x \in E \setminus B$: $B \cup \{x\} \notin \mathcal{M}$. B heisst **Basis** des Matroids $(E, \mathcal{M})$.

Matroid II

Beobachtung: Alle Basen eines Matroids sind gleich groß. Also: sind B und C Basen des Matroids $(E, \mathcal{M})$, dann gilt $|B| = |C|$.

Wikipedia:
Sehen Sie sich insbesondere den Zusammenhang mit Vektoren an! (Link)

Matroid garantiert Greedy-Optimalität

Theorem (Greedy-Optimalität)

Sei $(E, \mathcal{M})$ ein Unabhängigkeitssystem

Für *jede beliebige* Gewichtsfunktion $w : E \to \mathbb{R}^+$ berechnet der kanonische Greedy-Algorithmus eine *optimale* Lösung

$$\Updownarrow$$

$(E, \mathcal{M})$ ist ein Matroid.

Greedy-Optimalität - Beweis (vgl. Schöning) I

Beweis ($\Longrightarrow$)

Wir zeigen: *Wenn* $(E, \mathcal{M})$ *kein Matroid ist, gibt es eine Gewichtsfunktion für die Greedy nicht funktioniert.*

Nehmen wir an, die Austauscheigenschaft gelte nicht. Dann gibt es $A, B \in \mathcal{M}$ mit $|A| < |B|$, so dass für alle $b \in B - A$ gilt $A \cup \{b\} \notin \mathcal{M}$.
Wir definieren folgende Gewichtsfunktion $w : E \to \mathbb{R}^+$

$$w(x) = \begin{cases} r + 1, & x \in A \\ r, & x \in B - A \\ 0, & sonst \end{cases}$$

Greedy-Optimalität - Beweis (vgl. Schöning) II

Beweis ($\Longrightarrow$)

Sei $r = |B|$ und wegen $|A| < |B|$ gilt $|A| \leq r - 1$.
Der Greedy-Algorithmus wählt dann eine Menge T mit $A \subseteq T$ und $T \cap (B - A) = \emptyset$.
Also ist $w(T) = w(x \in A) \times |A| = (r + 1) \times |A| \leq (r + 1)(r - 1) = r^2 - 1$.

Wählt man stattdessen eine Lösung $T' \supseteq B$, so hat diese den Wert $w(T') \geq r \times |B| = r^2$.

Der Greedy-Algorithmus liefert also nicht die optimale Lösung!

Greedy-Optimalität - Beweis (vgl. Schöning) III

Beweis ($\Longleftarrow$)

Sei (E, U) ein Matroid mit Gewichtsfunktion w,
welche die Elemente von $E = \{e_1, \ldots, e_n\}$ absteigend anordnet,
$w(e_1) \geq \cdots \geq w(e_n)$ und $T = \{e_{i_1}, \ldots, e_{i_k}\}$ die vom Greedy-Algorithmus gefundene Lösung.

Nehmen wir an, $T' = \{e_{j_1}, \ldots, e_{j_k}\}$ sei eine bessere Lösung mit $w(T') > w(T)$. Dann muss es einen Index μ geben mit $w(e_{j_\mu}) > w(e_{i_\mu})$ und μ dem kleinsten Index.

Greedy-Optimalität - Beweis (vgl. Schöning) IV

Beweis ($\Longleftarrow$)

Es sei $A = \{e_{i_1}, \ldots, e_{i_{\mu-1}}\}$ und $B = \{e_{j_1}, \ldots, e_{j_\mu}\}$ und $|A| < |B|$.

Wegen der **Austauscheigenschaft** muss es ein Element $e_{j_\sigma} \in B - A$ geben, so dass $A \cup \{e_{j_\sigma}\} \in \mathcal{M}$. Es gilt aber $w(e_{j_\sigma}) \geqslant w(e_{j_\mu}) > w(e_{i_\mu})$, also hätte der Greedy-Algorithmus das Element e_{j_σ} bereits vor e_{i_μ} auswählen müssen. *Widerspruch.*

□

ausführlicher Beweis: *(Link)*

Nutzen - Algorithmendesign

Gegeben eine **Optimierungsaufgabe**,

Verfahren

- strukturiere die Anwendung in ein passend gewähltes Teilmengensystem $(E, \mathcal{M})$
  - Lösungen des Optimierungsproblems müssen Lösungen des zu $(E, \mathcal{M})$ gehörenden Maximierungs- bzw. Minimierungsproblems sein
- überprüfe, ob $(E, \mathcal{M})$ die Matroid-Eigenschaft besitzt
- falls ja, wende den kanonischen Greedy-Algorithmus an
  - das erfordert eine Prozedur, mit der überprüft werden kann, ob eine gegebene Teilmenge $A \in \mathcal{M}$ eine Lösung bzw. Teillösung des Optimierungsproblems ist

Auftragsplanung I

Gegeben:

- Menge E von n Aufträgen mit jeweils gleichem Zeitbedarf (ein Tag)
- Zu jedem Auftrag $x \in E$: Gewinn $w(x) > 0$ (Gewichtsfunktion).
- Anzahl Tage $d(x)$, bis zu dem der Auftrag x abgeschlossen sein muss

Auftragsplanung II

Gesucht Menge $A \subseteq E$ von Aufträgen, so dass

I. der Gesamtgewinn $\sum_{x \in A} w(x)$ maximal ist, und
II. A sich so sortieren lässt, dass jeder Auftrag vor seinem Abschlusstermin erledigt wird. Die Menge $\mathcal{M}$ zulässiger Auftragsmengen ist also gegeben durch Mengen $A = \{x_1, x_2, \ldots, x_k\}$, so dass $d(x_i) \geq i$ für $i = 1, 2, 3, \ldots$. Der Index i gibt an, an welchem Tag der Auftrag x_i erledigt wird. Jeder Auftrag entspricht dem Wert $w(x_i)$.

Durch welche Aufträge kann eine gegeben Auftragsliste erweitert werden?

Auftragsplanung: Beispiel

Beispiel

Auftragsmenge $E = \{a, b, c, d, e\}$

Auftrag x	Wert w(x)	Termin d(x)
a	13	2
b	7	1
c	9	1
d	3	2
e	5	1

Auftragsplanung: Kanonischer Greedy-Algorithmus

Problem

Ordne Aufträge in E absteigend nach Gewinn

Jetzt der kanonische Greedy-Algorithmus wie folgt:

Algorithmus

$A \leftarrow \emptyset$;
for $e \in E$ *in Reihenfolge absteigenden Gewinns* $w(e)$ **do**
  if $A \cup \{e\} \in \mathcal{M}$ **then**
    $A \leftarrow A \cup \{e\}$
return A

Auftragsplanung: Lösungsraum A

Lösung	Deadline	Gewinn
$\emptyset$	∞	0
$\{a\}$	2	13
$\{c\}$	1	9
$\{b\}$	1	7
$\{e\}$	1	5
$\{d\}$	2	3
$\{c, a\}$	1,2	22
$\{b, a\}$	1,2	20
$\{e, a\}$	1,2	18
$\{d, a\}$	2,2	16
$\{d, c\}$	1,2	12
$\{d, b\}$	1,2	10
$\{e, d\}$	1,2	8

- jede Zeile ist ein Element des Lösungsraumes $\mathcal{M}$
- Greedy geht *nicht* durch die Zeilen, sondern
- geht linear durch $E = \{a, b, c, d, e\}$
- Wähle bestes 1. Element (a), dann (wenn noch möglich) bestes 2. Element (c), …
- Da wir maximal zwei Tage Zeit haben, können nur zwei Elemente gewählt werden
- $\{d, a\}$ ist legal, da beide Elemente am Tag 2 fertig sein müssen (a oder d einfach an Tag 1 erledigen, aber ist nicht optimal)

Auftragsplanung

Aufträge sortiert nach w, notiert in der Form $(x, [d(x), w(x)])$

$(a, [2, 13])$, $(c, [1, 9])$, $(b, [1, 7])$, $(e, [1, 5])$, $(d, [2, 3])$.

Greedy-Aufbau von A

$A = \emptyset$
$A = \{(a, [2, 13])\}$ ... ist $A \in \mathcal{M}$? ... ja!
$A = \{(a, [2, 13]), (c, [1, 9])\}$ ... ist $A \in \mathcal{M}$? ... ja!
$A = \{(a, [2, 13]), (c, [1, 9]), (b, [1, 7])\}$ ... ist $A \in \mathcal{M}$? ... NEIN!
$A = \{(a, [2, 13]), (c, [1, 9])\}$ behalten
$A = \{(a, [2, 13]), (c, [1, 9]), (e, [1, 5])\}$ ... ist $A \in \mathcal{M}$? ... NEIN!
$A = \{(a, [2, 13]), (c, [1, 9])\}$ behalten
$A = \{(a, [2, 13]), (c, [1, 9]), (d, [2, 3])\}$ ... ist $A \in \mathcal{M}$? ... NEIN!
$A = \{(a, [2, 13]), (c, [1, 9])\}$ behalten
fertig

Auftragsplanung

Wann ist eine Lösung A zulässig ?

Sortiere die Aufträge nach Fälligkeitsdatum.
Dann muss für alle Aufträge $x_i \in A$ gelten: $d(x_i) \geq i$, das Fälligkeitsdatum für den i-ten Auftrag ist frühestens Tag i.

Zeigen Sie, dass die Matroid-Eigenschaften erfüllt sind!

…und noch mal der Kruskal-Algorithmus

Kruskal (1956) für minimalen Spannbaum

1. Erzeuge Liste Q mit der Kantenmenge E aufsteigend sortiert nach Gewicht (kleinstes zuerst). Initialisiere F als leere Menge.
2. Entferne vorderste Kante $e = \{u, v\} \in E$ (mit kleinstem Gewicht) aus Q
3. Falls (V, F) keinen Pfad zwischen u und v enthält:

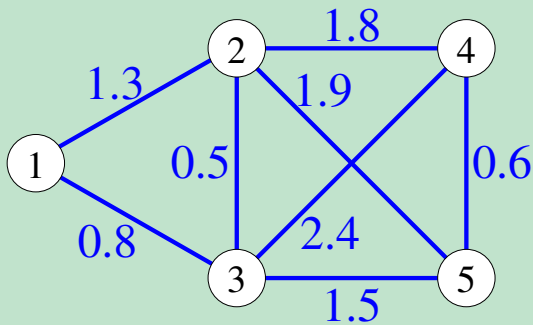
$$F := F \cup \{e\}$$

4. Falls Q nicht leer, zurück zu 2.
5. Ausgabe F.

Greedy-Algorithmus zur *Minimierung* der Summe der Kantengewichte w(e)

Beispiel-Lauf: Kruskal-Algorithmus

Beispiel

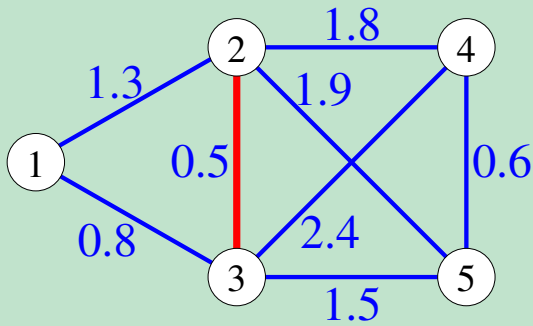


$Q = [\{2,3\}, \{4,5\}, \{1,3\}, \{1,2\}, \{3,5\}, \{2,4\}, \{2,5\}, \{3,4\}]$

$F = \{\}$

Beispiel-Lauf: Kruskal-Algorithmus

Beispiel

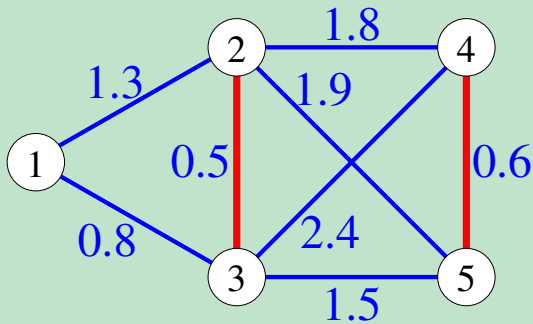


$Q = [\{4,5\}, \{1,3\}, \{1,2\}, \{3,5\}, \{2,4\}, \{2,5\}, \{3,4\}]$

$F = \{\{2,3\}\}$

Beispiel-Lauf: Kruskal-Algorithmus

Beispiel

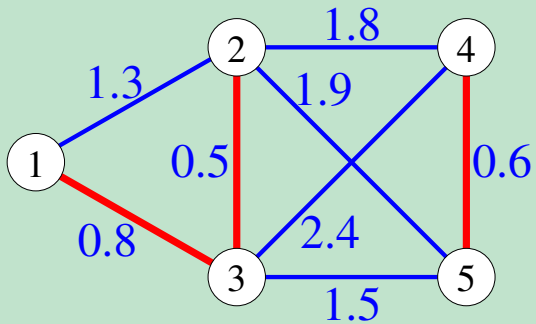


$Q = [\{1,3\},\{1,2\},\{3,5\},\{2,4\},\{2,5\},\{3,4\}]$

$F = \{\{2,3\},\{4,5\}\}$

Beispiel-Lauf: Kruskal-Algorithmus

Beispiel

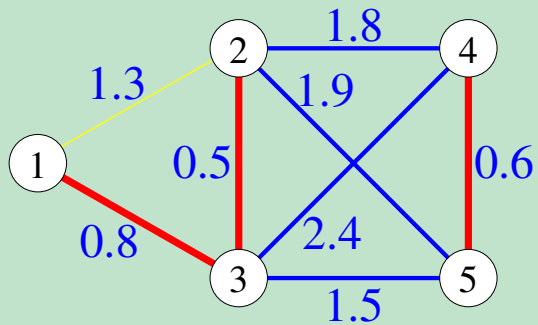


$Q = [\{1,2\},\{3,5\},\{2,4\},\{2,5\},\{3,4\}]$

$F = \{\{2,3\},\{4,5\},\{1,3\}\}$

Beispiel-Lauf: Kruskal-Algorithmus

Beispiel

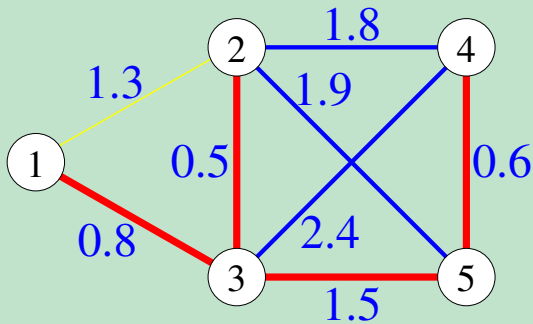


$Q = [\{3,5\},\{2,4\},\{2,5\},\{3,4\}]$

$F = \{\{2,3\},\{4,5\},\{1,3\}\}$

Beispiel-Lauf: Kruskal-Algorithmus

Beispiel

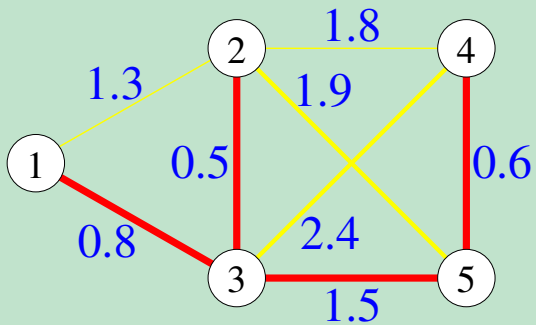


$Q = [\{2,4\},\{2,5\},\{3,4\}]$

$F = \{\{2,3\},\{4,5\},\{1,3\},\{3,5\}\}$

Beispiel-Lauf: Kruskal-Algorithmus

Beispiel



$Q = []$

$F = \{\{2,3\},\{4,5\},\{1,3\},\{3,5\}\}$

Kruskal: Korrektheit I

- E = Kantenmenge
- Gewichtsfunktion $w : E \to \mathbb{R}$ = Kantengewichte
- $\mathcal{M}$ enthält $F \subseteq E$ genau dann wenn (V, F) azyklisch (Wald) ist.

Wende Satz über Greedy-Optimalität an:

Kruskal findet optimale Lösung für beliebige Kantengewichte

$$\Updownarrow$$

$(E, \mathcal{M})$ ist ein Matroid.

Der Kruskal *minimiert*, während der kanonische Greedy-Algorithmus *maximiert*. Der Algorithmus ist direkt anwendbar nach Transformation der Kantengewichte $w'(e) = c - w(e)$. Die Konstante c ist so groß zu wählen, dass alle $w'(e)$ positiv werden.

Kruskal: Korrektheit II

Behauptung: $(E, \mathcal{M})$ ist ein Matroid.

Beweis

$(E, \mathcal{M})$ ist *Unabhängigkeitssystem*, denn Teilgraphen von azyklischen Graphen sind azyklisch.
Austauscheigenschaft: Seien $A, B \in \mathcal{M}$ Teilgraphen mit $|A| > |B|$, also (V, A) azyklisch und (V, B) azyklisch. (V, A) ist disjunkte Vereinigung von $|V| - |A|$ Bäumen, (V, B) ist disjunkte Vereinigung von $|V| - |B|$ Bäumen. Somit besteht (V, B) aus mehr Bäumen als (V, A), und es gibt Knoten $u, v \in V$, die in demselben Baum von (V, A) liegen, aber nicht innerhalb eines Baums von (V, B). Insbesondere können u und v so gewählt werden, dass die Kante $\{u, v\} := e$ in A liegt aber nicht in B. Ausserdem ist $(V, B \cup \{e\})$ azyklisch nach Wahl von u und v aus verschiedenen Bäumen von (V, B). Also $e \in A \setminus B$ und $B \cup \{e\} \in \mathcal{M}$. □

$(E, \mathcal{M})$ *heißt graphisches Matroid.*

GREEDY-ARTIGE ALGORITHMEN

Greedy-artige Algorithmen

Die Idee, Lösungen von Optimierungsproblemen *auf "greedy"-Art* ist allgemeiner anwendbar als der strikt definierte "kanonische" Greedy Algorithmus aus dem vorherigen Abschnitt.

Wir betrachten kurz zwei solche Beispiele.

- Konstruktion des Huffmann-Baums
- Das (ungewichtete) *Activity Selection* Problem

Greedy Konstruktion des Huffmann-Baums

Die Buchstaben eines Alphabets $E = a_1, a_2, \ldots, a_n$ seien so durchnummeriert, dass $p(a_1) \leq p(a_2) \leq \cdots \leq p(a_n)$

Die Häufigkeit $p(a_i)$ dient als Gewicht: $w(a_i) = p(a_i)$.

Algorithmus

1. Erstelle für jeden Buchstaben einen einzelnen Knoten und notiere am Knoten die Häufigkeit.
2. Wiederhole die folgenden Schritte so lange, bis nur noch ein Baum übrig ist:
   - Wähle die m (typischerweise 2 für binäre Huffmann-Codes) Teilbäume mit der geringsten Häufigkeit in der Wurzel, bei mehreren Möglichkeiten die Teilbäume mit der geringsten Tiefe.
   - Fasse diese Bäume zu einem neuen (Teil-)Baum zusammen.
   - Notiere die Summe der Häufigkeiten in der Wurzel.

Was wird hier eigentlich optimiert?

Optimalität des Huffmann-Codebaums

... es soll die *erwartete* Länge einer Kodierung minimiert werden.

- Länge des Codes für ein Zeichen a = Tiefe $d(x)$ des entsprechenden Blattes im Code-Baum T
- Erwartete Länge einer Kodierung:

$$B(T) = \sum_{a \in \mathcal{A}} p(a)d(a)$$

Ein Huffmann-Baum selbst ist kein Matroid. In der Tat scheint bisher keine Matroid-Darstellung für dieses Problem bekannt zu sein.

Greedy Konstruktion des Huffmann-Baums

Um die Korrektheit zu zeigen, beweist man zunächst:

Lemma

Es gibt einen optimalen Code-Baum T in dem die beiden Zeichen mit dem kleinsten Gewicht Geschwister auf der niedrigsten Ebene von T sind.

Im zweiten Schritt zeigt man dann durch Induktion, daß der Huffmann Baum tatsächlich optimal ist.

Für Interessierte findet sich eine sehr gute, ausführliche Darstellung in Vorlesungsunterlagen von *Dave Mount* and der University of Maryland: *(Link)*
Der Beweis ist *kein* Prüfungstoff

Activity Selection: Problemstellung

Problem

- Gegeben sei eine Menge A von “Aktivitäten” k, z.B. Lehrveranstaltungen mit vorgegebenen Anfangs- und Endzeiten a_k, e_k.
- Eine Menge B von Aktivitäten ist kompatibel wenn zwei Aktivitäten in B zeitlich nicht überlappen.
- Gesucht ist Menge von kompatiblen Aktivitäten mit maximaler Kardinalität, i.e., es sollen so viele LVs wie möglich besucht werden.

Activity Selection: Lösung

Greedy Algorithmus

- Ordne die Aktivitäten *A* nach ihrer **End**-Zeit.
- Wähle schrittweise die Aktivität mit der frühesten End-Zeit aus, die mit den bisher gewählten kompatibel ist.

Bilden die Mengen von Aktivitäten einen Matroid ?

Activity Selection: Korrektheit

Nein! **Gegenbeispiel:**
$A = \{(1,2),(3,4),(1,5)\}$. Offenbar sind $B = \{(1,2),(3,4)\}$ und $B' = \{(1,5)\}$ kompatible, also zulässige Mengen, aber die Austauscheigenschaft ist verletzt.

Wieso darf das sein?

... wir erlauben nur eine ganz bestimmte Gewichtsfunktion, nämlich $w(k) = 1$ for all Aktivitäten $k \in A$. Der Greedy Ansatz funktioniert in der Tat auch nicht mehr, wenn die Aktivitäten gewichtet werden.

... bleibt die Frage, warum funktioniert das aber für den ungewichteten Fall? Nachdem wir Matroide nicht als Argument verwenden können, brauchen wir einen direkten Beweis ...

Activity Selection: Korrektheitsbeweis I

Beweis I

Sei a_1 eine Aktivität mit frühester Endzeit. Wir zeigen: Es gibt eine optimale Lösung B^* mit $a_1 \in B^*$.
Sei B eine optimale Lösung. Wenn $a_1 \in B$, dann ist nichts zu tun. Ansonsten sei a_k die zeitlich erste Aktivität in B. Da a_1 nicht später als a_k endet, kann a_k durch a_1 ersetzt werden: $B^* = B \setminus a_k \cup \{a_1\}$ erfüllt $|B^*| = |B|$, ist also optimal.
Sei nun B^* eine optimale Lösung mit $a_1 \in B^*$. Angenommen $B' = B^* \setminus \{a_1\}$ wäre keine optimale Lösung auf dem Teilproblem das nur mehr die mit a_1 kompatiblen Aktivitäten enthält. Dann betrachte eine optimale Lösung B'' dieses Teilproblems. Da alle Aktivitäten in B'' mit a_1 kompatibel sind, ist $B^{**} = B'' \cup \{a_1\}$ eine zulässige Menge. Nun gilt $|B^{**}| = |B''| + 1 > |B'| + 1 = |B^*|$.

Activity Selection: Korrektheitsbeweis II

Beweis II

Das widerspricht der Annahmen, dass B^* eine optimale Lösung war. Daher muss $B^* \setminus \{a_1\}$ eine optimale Lösung des Teilproblems sein.
Also erhalten wir tatsächlich eine optimale Lösung indem wir schrittweise eine kompatible Aktivität mit frühester End-Zeit wählen. □