ADS: Algorithmen und Datenstrukturen 2

Teil X

Peter F. Stadler & Konstantin Klemm

Bioinformatics Group, Dept. of Computer Science & Interdisciplinary Center for Bioinformatics, **University of Leipzig**

08. Juni 2011

Das 0/1-Rucksackproblem (Wdh.)

- Objekte $\{1, 2, \ldots, n\}$ mit Volumina $t_1, t_2, \ldots, t_n$ und Werten $p_1, p_2, \ldots, p_n$
- Rucksack hat Gesamtvolumen c.
- Menge zulässiger Lösungen

$$X = \{x \in \{0,1\}^n : \sum_{i=1}^{n} x_i t_i \leq c\}$$

- Optimierungsproblem: finde $y \in X$, so dass der Gesamtwert

$$f(y) = \sum_{i=1}^{n} y_i p_i$$

maximal wird.

Das 0/1-Rucksackproblem

- Mengensystem erlaubter Objektkombinationen ist generell *kein Matroid*. (Sieht man z.B. daran, daß nicht erweiterbare Objektmengen nicht gleiche Kardinalität haben müssen.)
- Daher liefert der kanonische Greedy-Algorithmus nicht immer die optimale Lösung.
- Dynamische Programmierung: Zeitaufwand $O(nc)$, also sehr zeitaufwendig bei großer Kapazität des Rucksacks c.

Fraktionales Rucksackproblem

- Wie 0/1-Rucksackproblem, aber $x_i \in [0, 1]$.
- Beliebige Bruchteile eines Objekts dürfen eingepackt werden.

Optimale Lösung wird gefunden mit folgendem Greedy-Ansatz:

- Berechne von jedem Objekt i den *Nutzen*, also das Verhältnis aus Gewinn und Volumen $p_i/t_i =: u_i$.
- Sortiere Objekte absteigend nach Nutzen: $u_1 \geq u_2 \geq \cdots \geq u_n$.
- Finde maximales i, so dass die Objekte $\{1, \ldots, i\}$ vollständig in den Rucksack passen, $\sum_{j=1}^{i} t_j \leq c$.
- Setze $x_j = 1$ für $j \leq i$, $x_j = 0$ sonst.
- Falls $i \neq n$, ersetze

$$x_{i+1} = \left(c - \sum_{j=1}^{i} t_j \right) / \ t_{i+1}$$

Beispiel

i	1	2	3	4	5
p_i	8	2	7	8	3
t_i	3	1	4	5	2
u_i	$\frac{8}{3}$	2	$\frac{7}{4}$	$\frac{8}{5}$	$\frac{3}{2}$

$c = 10$

- Greedy für fraktionalen Rucksack erreicht Wert 20.2
  $x_1 = x_2 = x_3 = 1$, $x_4 = \frac{2}{5}$, $x_5 = 0$.
- optimale Lösung für 0/1-Rucksack hat Wert 20
  Objekte $\{1, 2, 3, 5\}$, also $x_1 = x_2 = x_3 = x_5 = 1$, $x_4 = 0$
- Kanonischer Greedy für 0/1-Rucksack erreicht Wert 19,
  Objekte $\{1, 4, 5\}$, also $x_1 = x_4 = x_5 = 1$, $x_2 = x_3 = 0$.

Ganzzahlige und relaxierte Probleme

- Wichtige Unterscheidung zwischen ganzzahligen und reellzahligen ("kontinuierlichen") Optimierungsproblemen.
- Einschränkung auf ganze Zahlen macht viele Probleme qualitativ schwerer lösbar (Beispiel hier: 0/1-Rucksack).
- Das einem ursprünglich ganzzahligen Problem zugeordnete koninuerliche Problem heißt *Relaxierung* oder *relaxiertes Problem*.
- Relaxierung wird oft betrachet, weil sie eine untere / obere Schranke an die Werte der Kosten-/Gewinnfunktion liefert.

Branch and Bound

Bisher behandelte Strategien für Optimierungsprobleme:

- Dynamische Programmierung
- Greedy-Verfahren

Was tun, wenn diese nicht anwendbar sind?

- Naiver Ansatz ("Brute Force"): vollständige Aufzählung aller möglichen Lösungen.
- *Branch and Bound*: Identifiziere möglichst große Teilmengen des Lösungsraums, die keine optimale Lösung enthalten können, und überspringe diese beim Aufzählen.

Beispiel für Grundprinzip

Aufzählen von Lösungen

$x_1 \ldots x_5$	Ges.Wert	Ges.Vol.
00000	0	0
10000	8	3
01000	7	4
11000	15	7
001$**$	≤ 13	

...

0/1 - Rucksack, $c = 10$

i	1	2	3	4	5
p_i	8	7	8	3	2
t_i	3	4	5	2	1

Keine Lösung x mit $x_1 = x_2 = 0$ und $x_3 = 1$ kann wertvoller als 8+3+2=13 sein. Diese Lösungen brauchen daher nicht aufgezählt zu werden, denn sie sind alle schlechter als der bisher gesehene Maximalwert 15.

Optimierungsproblem und Schranke

Gegeben

- Menge erlaubter Lösungen X
- Kostenfunktion $f : X \to \mathbb{R}$

Gesucht

- $y \in X$ so dass $f(y) \leq f(x)$ für alle $x \in X$
  $y =$ globales *Minimum* der Kostenfunktion.

Sei $g : \mathcal{P}(X) \to \mathbb{R}$. Die Funktion g heißt *untere Schranke* von f, wenn für alle $A \subseteq X$ gilt:

$$g(A) \leq \min_{x \in A} f(x)$$

und für alle $x \in X$

$$g(\{x\}) = f(x)$$

Branch and Bound, Grundalgorithmus

```
b ← +∞, INIT(S), PUSH(S,X) ;
while not EMPTY(S) do
    A=POP(S);
    if  g(A) < b then
        if  |A| == 1 then
            b ← g(A);
        end
        else
            if A ≠ ∅ then
                (B, C)=Split(A);
                PUSH(S,B) ;
                PUSH(S,C) ;
            end
        end
    end
end
```

Anwendung auf das Rucksack-Problem

Da Branch and Bound *minimiert* (per Konvention), verwenden wir die Kostenfunktion mit umgedrehtem Vorzeichen:

$$f(x) = -\sum_{i=1}^{n} x_i p_i$$

Um Branch and Bound anwenden zu können, müssen wir festlegen:

1. Die Funktion Split (Mengenaufteilung)
2. untere Schranke g auf Teilmengen von X.

Split-Operation

Wir betrachten nur solche Teilmengen von $A \subseteq X$, die sich durch eine Maske $m \in (0, 1, *)^n$ folgendermassen beschreiben lassen

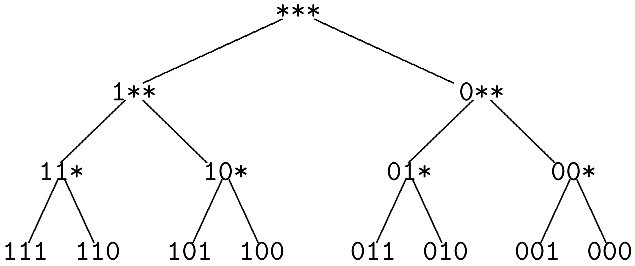
$$A = \{x \in X : \forall i : m_i \neq * \Rightarrow x_i = m_i\}$$

Die Maske schreibt also für eine Position i den Wert von x_i vor, wenn $m_i = 0$ oder $m_i = 1$. Ansonsten ist der Wert dort beliebig ($m_i = *$).

Split(A)=(B, C) mit $B = \{x \in A : x_k = 0\}$, $C = \{x \in A : x_k = 1\}$
$k = \min\{i : \exists x, y \in A : x_i \neq y_i\}$

Beispiel:
Split(010***) = (0100**, 0101**)

Verzweigung für $n = 3$



Bestimmung einer unteren Schranke g

Berechnung von g ist ein Kompromiss aus zwei Forderungen:

- Untere Schranke $g(A)$ soll möglichst nah am wahren Kostenminimum auf $A \subseteq X$ liegen
- Zeitaufwand soll möglichst gering sein, insbesondere deutlich geringer als die Berechnung durch Aufzählen von A.

Schranke für Teilmenge mit Maske $m = (m_1, m_2, \ldots, m_n)$

1. Berechne Kosten γ und verbrauchtes Volumen v auf der festgelegten Objektmenge, also auf $I = \{i : m_i \neq *\}$.

$$\gamma = -\sum_{i \in I} m_i p_i \ , \qquad v = \sum_{i \in I} m_i t_i$$

2. Finde mit Greedy die minimalen Kosten β für das fraktionale Rucksackproblem mit Greedy auf der nicht festgelegten Objektmenge für Kapazität $c' = c - v$.

Dann ist $\gamma + \beta$ untere Schranke an Kosten auf der Teilmenge.

Beispiel: Branch and Bound für 0/1-Rucksack

i	1	2	3	4	5
p_i	8	2	7	8	3
t_i	3	1	4	5	2
u_i	$\frac{8}{3}$	2	$\frac{7}{4}$	$\frac{8}{5}$	$\frac{3}{2}$

b	$g(\text{TOP(S)})$	Stack S
$+\infty$	-20.2	∗∗∗∗∗
$+\infty$	-20.2	1∗∗∗∗, 0∗∗∗∗
$+\infty$	-20.2	11∗∗∗, 10∗∗∗, 0∗∗∗∗
$+\infty$	-20.2	111∗∗, 110∗∗, 10∗∗∗, 0∗∗∗∗
$+\infty$	-20.0	1110∗, 110∗∗, 10∗∗∗, 0∗∗∗∗
-20.0	-20.0	11101, 11100, 110∗∗, ...
-20.0	-17.0	11100, 110∗∗, ...
-20.0	-19.5	110∗∗, 10∗∗∗, 0∗∗∗∗
-20.0	-19.8	10∗∗∗, 0∗∗∗∗
-20.0	-17.0	0∗∗∗∗

$b \leftarrow +\infty$, INIT(S), PUSH(S,X) ;
while *not EMPTY(S)* **do**
  A=POP(S);
  if $g(A) < b$ **then**
    if $|A| == 1$ **then**
      $b \leftarrow g(A)$;
    end
    else
      if $A \neq \emptyset$ **then**
        (B, C)=Split(A);
        PUSH(S,B)
        ;
        PUSH(S,C)
        ;
      end
    end
  end
end

Resultat: $\min_{x \in X} f(x) = b_{\text{final}} = -20.0$

Branch and Bound für TSP

- Operiere auf Kantenmengen (statt auf Permutationen der Städte). Zulässige Kantenmengen: maximal je eine eingehende und eine ausgehende Kante pro Stadt, keine Zyklen kürzer als n (Anzahl Städte).
- Split-Operation für Kantenmengen wie für Objektmengen bei Rucksack.
- Untere Schranke: Kosten der gewählten Kanten + Kosten der billigsten eingehenden und ausgehenden Kanten der unverbundenen Städte.

Beispiel für untere Schranke bei TSP:

	1	2	3	4	5
1	-	5	13	8	17
2	7	-	9	4	14
3	12	10	-	6	7
4	8	4	9	-	11
5	15	14	8	12	-

Betrachte Teilmenge von Lösungen, die Kanten $(1, 2)$ und $(2, 3)$ enthalten.
Kosten der vorhandenen Kanten:
$\gamma = d_{12} + d_{23}$
minimale Kosten für ausgehende Kanten:
$\beta_{\text{aus}} = d_{34} + d_{41} + d_{54}$
minimale Kosten für eingehende Kanten:
$\beta_{\text{ein}} = d_{41} + d_{34} + d_{35}$

Wert der unteren Schranke
$\gamma + \max\{\beta_{\text{aus}}, \beta_{\text{ein}}\}$

Branch and Bound für TSP

- Operiere auf Kantenmengen (statt auf Permutationen der Städte). Zulässige Kantenmengen: maximal je eine eingehende und eine ausgehende Kante pro Stadt, keine Zyklen kürzer als n (Anzahl Städte).
- Split-Operation für Kantenmengen wie für Objektmengen bei Rucksack.
- Untere Schranke: Kosten der gewählten Kanten + Kosten der billigsten eingehenden und ausgehenden Kanten der unverbundenen Städte.

Beispiel für untere Schranke bei TSP:

	1	2	3	4	5
1	-	-	-	-	-
2	-	-	-	-	-
3	12	-	-	6	7
4	8	-	-	-	11
5	15	-	-	12	-

Betrachte Teilmenge von Lösungen, die Kanten $(1, 2)$ und $(2, 3)$ enthalten.
Kosten der vorhandenen Kanten:
$\gamma = d_{12} + d_{23}$
minimale Kosten für ausgehende Kanten:
$\beta_{\text{aus}} = d_{34} + d_{41} + d_{54}$
minimale Kosten für eingehende Kanten:
$\beta_{\text{ein}} = d_{41} + d_{34} + d_{35}$

Wert der unteren Schranke
$\gamma + \max\{\beta_{\text{aus}}, \beta_{\text{ein}}\}$

Zusammenfassung: Deterministische Optimierung

- Problemstellung: Lösungsmenge X, Bewertungsfunktion $f : X \to \mathbb{R}$. Finde globales Extremum (Minimum oder Maximum) von f.
- Dynamische Programmierung: Lösung wird durch Erweitern von Lösungen kleinerer Teilprobleme konstruiert. Hierzu müssen i.A. mehrere Lösungen für jede Problemgröße berechnet werden.
- Greedy: Iteratives Erweitern der Lösung, so dass in jedem einzelnen Schritt maximale Kostenreduzierung bzw. Gewinnerhöhung stattfindet.
- Branch and Bound: Durchsuchen der Lösungsmenge unter Vernachlässigung von Teilmengen, die keine Verbesserung gegenüber bereits gefundenen Lösungen enthalten können.

Nächste Vorlesung: stochastische Optimierungsalgorithmen