BRANCH & BOUND

Ein Branch & Bound Algorithmus versucht eine (fast) optimale Lösung eines

Optimierungsproblems durch ein intelligentes Durchsuchen

des Lösungsraums zu finden.

der Branching Schritt

– ein *Branching-Operator* zerlegt eine Menge von Lösungen in (disjunkte) Teilmengen

– die wiederholte Anwendung vom Branching-Operator erzeugt einen *Branch & Bound Baum* $\mathcal{B}$:

- die Wurzel entspricht der Menge *aller* Lösungen
- seien $v_1, v_2, \ldots, v_k$ die Kinder vom Knoten v in $\mathcal{B}$, dann sind die Lösungsmengen $L(v_1), L(v_2), \ldots, L(v_k)$ eine (disjunkte) Zerlegung der Lösungsmenge $L(v)$

der Bounding Schritt

– für jeden Knoten $v \in \mathcal{B}$ gibt es eine untere Schranke $\mathrm{unten}(v)$, so dass für jede Lösung $y \in L(v)$ gilt

$$f(y) \geq \mathrm{unten}(v)$$

– v kann verworfen werden, wenn

$$\mathrm{unten}(v) \geq f(y_0)$$

für eine aktuelle beste Lösung y_0.

$\Longrightarrow$ das Blatt v wird *deaktiviert*

Branch & Bound Algorithmus

(für Minimierungsprobleme)

- anfänglich besteht der Baum $\mathcal{B}$ nur aus der *aktivierten* Wurzel
- eine Lösung y_0 wird mit Hilfe einer Heuristik berechnet
- WHILE es aktivierte Blätter gibt, DO
  - für das *erfolgversprechendste* aktivierte Blatt $v \in \mathcal{B}$ wende den Branching Operator auf v an, um die Kinder $v_1, v_2, \ldots v_k$ zu erhalten.
  - FOR $i = 1$ to k DO
    - berechne $\mathrm{unten}(v_i)$
    - wenn $\mathrm{unten}(v_i) \geq f(y_0)$, wird v_i deaktiviert
    - wenn eine $y_i \in L(v_i)$ gefunden wird mit $f(y_i) < f(y_0)$ setze $y_0 := y_i$ und deaktiviere ggf. Blätter

Bemerkungen

1. Damit schlechte Lösungsmengen frühzeitig deaktiviert werden:

– die Anfangslösung y_0 muss das Optimum gut approximieren

– die untere Schranke $\mathrm{unten}(v)$ muss nah am Optimum von $L(v)$ liegen

2. Wie wird ein erfolgversprechendstes Blatt v gewählt?

Best-fit search: wähle einen Knoten mit der niedrigsten unteren Schranke

Tiefensuche: schont den Speicherplatzverbrauch

Oft wird Tiefensuche mit Best-fit-search kombiniert.

Mögliche Branching Operatoren für VERTEX COVER

sei $G(V, E)$ ein Eingabegraph, wobei $V = \{1, 2, 3, \ldots, n\}$

- wenn $v \in \mathcal{B}$ auf Ebene i dann
  sei $i + 1 \in C$ im linken Kind von v
  und $i + 1 \notin C$ im rechten Kind

- wähle eine Kante $\{j, k\}$;
  im Linken Kind $j \in C$, im rechten Kind $k \in C$
  (siehe auch Abschnitt 1.2 im Skript)

- wähle einen neuen Knoten i;
  im linken Kind sei $i \in C$; im rechten Kind seien alle Nachbarn von i in C

Das RUCKSACK Problem

Eingabe: eine Gewichtsschranke G, und n Objekte
mit Gewichten $g_1, g_2, \ldots, g_n$
und Werten $w_1, w_2, \ldots, w_n$

(wir nehmen $g_i \leq G$ an)

Ausgabe: Bepacke einen Rucksack mit Objekten maximaler Gesamtwert
so dass es die Gewichtsschranke G nicht übersteigt

(Maximierungsproblem: jeder Knoten von $\mathcal{B}$ braucht eine *obere* Schranke $\mathrm{oben}(v)$ für die Bepackungen in $L(v)$)

Das fraktionale RUCKSACK Problem

man darf auch Bruchteile von Objekten in den Rucksack packen:
ein Anteil $x_i \in [0,1]$ des i-ten Objekts wird für jedes i eingepackt
(maximiere $\sum_i w_i \cdot x_i$ so dass $\sum_i g_i \cdot x_i \leq G$)

Greedy Algorithmus:

- sortiere die Objekte absteigend nach Wert-pro-Kilo:

$$\frac{w_1}{g_1} \geq \frac{w_2}{g_2} \geq \cdots \geq \frac{w_n}{g_n}$$

- wenn

$$\sum_{i=1}^{k} g_i \leq G < \sum_{i=1}^{k+1} g_i$$

packe die Objekte $\{1, 2, \ldots k\}$ ein und noch einen
Bruchteil x_{k+1} so dass $\sum_{i=1}^{k} g_i + x_{k+1} g_{k+1} = G$

Beobachtungen

Beobachtung 1: Der greedy Algorithmus ist optimal für das *fraktionale* RUCKSACK.

(Für das ganzzahlige RUCKSACK ∞-approximativ.)

Beobachtung 2: Eine beste (ganzzahlige) Bepackung ist höchstens so gut wie eine beste fraktionale Bepackung:

$$W_{\max} \leq W_{\max}^{\text{frac}}$$

($W_{\max}^{\text{frac}}$ ist obere Schranke und einfach zu berechnen!)

Branch & Bound für RUCKSACK

- ein Knoten des B&B Baums ist $v = (J, i)$ wobei $J \subseteq \{1, 2, \ldots, i\}$;
  $L(v)$ besteht aus Bepackungen die von den ersten i Objekten genau J enthalten;
- der Branching Operator erzeugt die Kinder $(J \cup \{i+1\}, i+1)$ und $(J, i+1)$
- um eine gute Lösung y in $L(v)$ zu finden wenden wir das FPTAS auf $\{i+1, i+2, \ldots, n\}$ und $G - \sum_{k \in J} g_k$ an;
- die obere Schranke in $L(v)$ wird mit Greedy auf $\{i+1, \ldots, n\}$ bestimmt

$$\text{oben}(J, i) = \sum_{k \in J} w_k + W^{\text{frac,OPT}}_{\{i+1,\ldots,n\}}$$

- als erfolgsversprechendstes Blatt (J, i) wird eine bislang wertvollste Bepackung gewählt.

Der BB Algorithmus

1. $\mathcal{B}$ besteht aus der aktivierten Wurzel $v_0 = (\emptyset, 0)$;

2. berechne eine Bepackung $I_0 \subseteq \{1, 2, \ldots, n\}$ mit Dyn. Prog. (PTAS);

3. WHILE es aktivierte Blätter gibt, DO
   i. sei $v = (J, i)$ mit $\sum_{k \in J} w_k$ maximal
   ii. sei $J_1 = J \cup \{i + 1\}$ und $v_1 = (J_1, i + 1)$
   iii. $\mathrm{oben}(v_1) := \sum_{k \in J_1} w_k + W^{\mathrm{frac,OPT}}_{\{i+2,\ldots,n\}}$

   IF $\mathrm{oben}(v_1) \leq W(I_0)$ deaktiviere v_1

   ELSE sei I' Ausgabe von PTAS auf $\{i + 2, \ldots, n\}$ und $I_1 = J_1 \cup I'$

   IF $W(I_1) > W(I_0)$, setze $I_0 := I_1$; deaktiviere ggf. Blätter
   iv. wiederhole ii. und iii. für $J_2 = J$ und $v_2 = (J_2, i + 1)$

4. gib I_0 aus (alle Blätter wurden deaktiviert und I_0 ist optimal)