Branching

Der Branching-Operator B

Unser Ziel: Löse das allgemeine Minimierungsproblem

$$\text{minimiere } f(x), \text{ so dass Lösung}(x).$$

- B zerlegt eine Menge von Lösungen in disjunkte Teilmengen.
- Die wiederholte Anwendung des Branching Operators erzeugt dann einen Branch & Bound Baum $\mathcal{B}$:
  - Die Wurzel von $\mathcal{B}$ entspricht der Menge aller Lösungen.
  - Ist v ein Knoten von $\mathcal{B}$ mit Lösungsmenge $\mathcal{L}(v)$, dann hat v die Kinder $v_1, \ldots, v_k$, deren Lösungsmengen $\mathcal{L}(v_i)$ die Lösungsmenge $\mathcal{L}(v)$ disjunkt zerlegen.
- Der Baum $\mathcal{B}$ ist viel zu größ: Sein Wachstum muss eingeschränkt werden.

Bounding

Der Bounding-Schritt

Für den Bounding-Schritt wird eine untere Schranke benötigt:

- Für jeden Knoten v von $\mathcal{B}$ nehmen wir an, dass eine untere Schranke `unten(v)` gegeben ist, so dass

$$\texttt{unten}(v) \leqslant f(y)$$

  für jede Lösung $y \in \mathcal{L}(v)$ gilt.
- Wann kann der Knoten v „abgeschnitten“ werden?

  Wenn $\texttt{unten}(v) \geqslant f(y_0)$ für eine aktuelle beste Lösung y_0 gilt!
- Neben den unteren Schranken benötigen wir also auch eine Heuristik, die eine gute Lösung y_0 berechnet.

Branch & Bound: Der Algorithmus

(1) Eine Lösung y_0 wird mit Hilfe einer **Heuristik** berechnet.

(2) Wiederhole, solange es **aktivierte** Blätter gibt:

(2a) Wähle das **erfolgversprechendste** aktivierte Blatt v von $\mathcal{B}$.

(2b) Der **Branching Schritt:** Wende den Branching Operator B auf v an, um die Kinder $v_1, \ldots, v_k$ zu erhalten. Inspiziere die k Teilmengen nacheinander:

- ⋆ Wenn es offensichtlich ist, dass v_i eine Lösung y_i enthält, die besser als y_0 ist, dann setze

$$\mathbf{y_0 = y_i}$$

und deaktiviere gegebenenfalls Blätter.

- ⋆ Ansonsten führe den **Bounding-Schritt** durch: Nur wenn $\texttt{unten}(v_i) < f(y_0)$, wird v_i **aktiviert**.

(3) Gib die Lösung y_0 als beste Lösung aus.

Was ist zu beachten?

- Damit der Bounding-Schritt erfolgreich ist,
  - muss die Anfangslösung y_0 möglichst nahe am Optimum liegen
  - und die untere Schranke unten(v) muss die Qualität der besten Lösung in v möglichst gut voraussagen.
- Die Wahl eines erfolgversprechendsten Blatts bestimmt die Suche in $\mathcal{B}$ und damit den Speicherplatzverbrauch.
  - **Tiefensuche** schont den Speicherplatzverbrauch. Allerdings wird die schnelle Entdeckung guter Lösungen leiden, da stets mit dem letzten, und nicht mit dem besten Knoten gearbeitet wird.
  - Der große Speicherverbrauch schließt **Breitensuche** als ein praktikables Suchverfahren für große Bäume aus.
  - In der „**best first search**“ wird der Knoten v mit der niedrigsten unteren Schranke gewählt. Man versucht also, schnell gute Lösungen zu erhalten.
  - Häufig werden Varianten der Tiefensuche und der best-first search kombiniert.

Branch & Bound für das Rucksackproblem

Ein Greedy Algorithmus für das Rucksackproblem

Eine möglichst wertvolle Auswahl von n Objekten mit Gewichten $g_1, \ldots, g_n \in \mathbb{R}$ und Werten $w_1, \ldots, w_n \in \mathbb{N}$ ist in einen Rucksack mit Gewichtsschranke $G \in \mathbb{R}$ zu packen.

Ein Greedy Algorithmus löst das fraktionale Rucksackproblem exakt. (Hier dürfen Anteile $0 \leqslant x_i \leqslant 1$ des $i$ten Objekts eingepackt werden.)

- Zuerst werden die Objekte absteigend, nach ihrem „Wert pro Kilo“ sortiert, also nach

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

- Wenn $\sum_{i=1}^{k} g_i \leqslant G < \sum_{i=1}^{k+1} g_i$:
  - Packe die Objekte $1, \ldots k$ ein und
  - fülle den Rucksack mit dem entsprechenden Anteil am Objekt $k+1$.

Branch & Bound für das Rucksackproblem

- Wir berechnen eine **Anfangslösung** mit Hilfe des dynamischen Programmieralgorithmus für das Rucksackproblem.
- Die **Knoten** des Branch & Bound Baums werden repräsentiert durch Paare (J, i):

  die Objekte aus $J \subseteq \{1, \ldots, i\}$ können, ohne die Kapazität G zu überschreiten, in den Rucksack gepackt werden.

- Der **Branching Operator** erzeugt die Kinder $(J, i+1)$ und $(J \cup \{i+1\}, i+1)$: Entweder wird das $i+1$.ste Objekt ausgelassen oder eingepackt.
- Wir bestimmen ein **erfolgversprechendstes Blatt** als die aktuell wertvollste Bepackung eines Blatts.

Eine obere Schranke für das Rucksack Problem

Das Rucksackproblem ist ein Maximierungsproblem und Branch & Bound benötigt eine **obere** statt einer **unteren** Schranke.

- Für die partielle Lösung (J, i) berechnen wir die Restkapazität $G' = G - \sum_{j \in J} g_j$ und
- wenden den (fraktionalen) Greedy Algorithmus auf die Objekte $i + 1, \ldots, n$ mit der neuen Gewichtsschranke G' an.
  - Da der Greedy Algorithmus eine optimale Lösung des fraktionalen Rucksackproblems berechnet und
  - da der optimale Wert des fraktionalen Problems mindestens so groß wie der optimale Wert des ganzzahligen Problems ist,
  - haben wir eine obere Schranke gefunden.

Branch & Bound für das metrische TSP

Das metrische TSP

Bestimme eine kürzeste Rundreise für n Städte, wobei die Distanzen zwischen den Städten die Dreiecksungleichungen erfüllen.

- Wir haben bereits ein Fülle von Heuristiken kennengelernt.
- Das Gewicht eines minimalen Spannbaums ist eine untere Schranke, die aber leider nicht gut genug ist.

1-Bäume

- Wähle eine beliebige Stadt s und berechne einen minimalen Spannbaum $B_{\min}$ für alle Städte bis auf Stadt s.
- Füge s zu $B_{\min}$ hinzu: Verbinde s mit den **beiden** nächstliegenden Städten.
- Wir haben den 1-Baum $B^*_{\min}$ erhalten.

  Warum 1-Baum? $B^*_{\min}$ hat einen einzigen Kreis.
- Das Gewicht $l(B^*_{\min})$ des 1-Baums ist unsere neue untere Schranke.

Warum ist $l(B^*_{\min})$ eine untere Schranke?

Eine verbesserte untere Schranke (1/2)

Geht es noch besser? Ersetze die Distanzen $d_{r,s}$ zwischen r und s durch $d^*_{r,s} = d_{r,s} + \lambda_r + \lambda_s$ mit beliebigen λ_r.

- Wie verändert sich die Länge $L(R)$ einer Rundreise R?
  Die neue Länge ist
$$L^*(R) = L(R) + 2 \cdot \sum_{r=1}^{n} \lambda_r.$$
- Wie verändert sich die Länge $l(B)$ eines 1-Baums B?
  - $\mathrm{grad}_B(r)$ sei die Anzahl der Nachbarn von r in B.
  - Die neue Länge ist
$$l^*(B) = l(B) + \sum_{r=1}^{n} \lambda_r \cdot \mathrm{grad}_B(r).$$
- Es ist $L^*(R) \geqslant \min_B l^*(B)$. Also folgt
$$L(R) + 2 \cdot \sum_{r=1}^{n} \lambda_r \geqslant \min_B l(B) + \sum_{r=1}^{n} \lambda_r \cdot \mathrm{grad}_B(r).$$

Eine verbesserte untere Schranke (2/2)

Aus $L(R) + 2 \cdot \sum_{r=1}^{n} \lambda_r \geqslant \min_B l(B) + \sum_{r=1}^{n} \lambda_r \cdot \text{grad}_B(r)$ folgt

$$L(R) \geqslant f(\lambda_1, \ldots, \lambda_n) := \min_B l(B) + \sum_{r=1}^{n} \lambda_r \cdot (\text{grad}_B(r) - 2).$$

- Maximiere $f(\lambda_1, \ldots, \lambda_n)$ und die untere Schranke wird so groß wie möglich.

  $\max_{\lambda \in \mathbb{R}^n} f(\lambda)$ wird die **Held-Karp Schranke** genannt.
- Wie schwierig ist die Maximierung?
  - f ist ein Minimum linearer Funktionen und f ist damit konkav.
  - f hat also nur globale Optima.
- Wir werden später die Berechnung der Held-Karp Schranke weiter untersuchen.

Branching und Bounding für M-TSP

- Der **Bounding-Schritt**:
  - Berechne die Held-Karp Schranke $\max_{\lambda \in \mathbb{R}^n} f(\lambda)$.
  - Bestimme einen für die Distanzen $d^*_{r,s} = d_{r,s} + \lambda_r + \lambda_s$ minimalen 1-Baum B_{opt}.
  - B_{opt} ist zusammenhängend mit genau einem Kreis. B_{opt} ist eine Rundreise oder es gibt eine Stadt s mit mindestens **drei** Nachbarn.
- Die Kanten $\{r, s\}, \{t, s\}$ seien die längsten, mit s inzidenten Kanten in B_{opt}.
- Der **Branching-Schritt** produziert drei Kinder-Probleme.
  (1) Die Kante $\{r, s\}$ wird verboten.
  (2) $\{r, s\}$ muss durchlaufen werden, aber $\{t, s\}$ ist verboten.
  (3) Sowohl $\{r, s\}$ wie auch $\{t, s\}$ müssen durchlaufen werden. Alle anderen mit s inzidenten Kanten werden verboten.

  Das Ursprungsproblem wird disjunkt zerlegt und B_{opt} wird in jedem Teilproblem ausgeschaltet.

Branch & Bound für M-TSP: Der Algorithmus

- Die Teilprobleme werden durch Paare (S, T) beschrieben:
  - Die Kanten in $S \subseteq \{ \{u, v\} \mid u \neq v\}$ sind erzwungen,
  - die Kanten in $T \subseteq \{ \{u, v\} \mid u \neq v\}$ sind verboten.
- Der Bounding-Schritt bestimmt einen minimalen 1-Baum B_{opt} mit maximalem Gewicht für (S, T).
- Der Branching-Schritt schaltet, falls notwendig, B_{opt} aus.
- Der Branch & Bound Baum wird mit Tiefensuche erzeugt:

  Die zuletzt erzeugten Teilprobleme werden weiterverarbeitet, sind also die vielversprechendsten Teilprobleme.