Branch and Bound

Branch-and-Bound ist eine Form des Backtrackings, bei der man (meist implizit) eine Baumstruktur erhält, die den momentanen Fortschritt des Ablaufs enthält.

Das zu bearbeitende Problem sei ein Minimierungsproblem, wobei ähnlich wie beim Backtracking n Komponenten mit Werten zu versehen sind, so dass ein gewisser Zielwert minimiert wird.

Als Beispiel betrachten wir das bekannte TSP-Problem:

Entscheidende Beobachtung ist hier, dass man eine untere Schranke für die Länge eines Weges durch alle betrachteten Städte leicht erhält, indem man zunächst alle Zeilen, dann alle Spalten der Entfernungsmatrix um ihren jeweiligen Minimaleintrag herabsetzt. Die Summe der abgezogenen Werte ist der sogenannte *Bound*.

Man kann leicht nachprüfen, dass jede Tour durch alle beteiligten Städte mindestens so lang ist, wie der Bound angibt.

Zum TSP-Problem

Beispiel des Branch-and-Bound Algorithmus für TSP:

$$\begin{bmatrix} \infty & 10 & 15 & 20 \\ 5 & \infty & 9 & 10 \\ 6 & 13 & \infty & 12 \\ 8 & 8 & 9 & \infty \end{bmatrix}$$

Die Zeilen werden um 10,5,6,8 verringert, dann die Spalten um 0,0,1,5. Als Bound ergibt sich daher 35.

Nun wählen wir den Eintrag 5 in der 2. Zeile, 1. Spalte, und verzweigen, je nachdem, ob wir den umgehen, d.h. Eintrag auf ∞ setzen, oder nutzen und daher Rest der 2. Zeile und der 1. Spalte, sowie den *Gegeneintrag*, also (1,2)-Eintrag ∞ setzen.

Ohne $(2,1)$-Eintrag:

$$\begin{bmatrix} \infty & 10 & 15 & 20 \\ \infty & \infty & 9 & 10 \\ 6 & 13 & \infty & 12 \\ 8 & 8 & 9 & \infty \end{bmatrix}$$

Bound $= 34$

Mit $(2,1)$-Eintrag:

$$\begin{bmatrix} \infty & \infty & 15 & 20 \\ 5 & \infty & \infty & \infty \\ \infty & 13 & \infty & 12 \\ \infty & 8 & 9 & \infty \end{bmatrix}$$

Bound $= 40$

Weiter geht es mit der linken Matrix. Im nächsten Schritt wird der Eintrag $(3,1)$ untersucht, usw. (bitte selbst weiterführen)

Als Ergebnis erhält man die Rundreise $1 \to 2 \to 4 \to 3 \to 1$ mit Kosten 35.

Greedy-Algorithmen

Die Greedy-Methode ist eine spezielle Form des Dynamischen Programmierens. Dabei merkt man sich immer nur *ein* Zwischenergebnis.

Dieses Verfahren ist anwendbar, wenn eine Lösung aus einer Reihe von Komponenten besteht, die man sukzessive, d.h. eines nach dem anderen, der schon bestehenden Teillösung hinzufügt.

Die Auswahl des nächsten hinzuzufügenden Elements ist hier der entscheidende Schritt. Sie geschieht in der Regel anhand einer *lokalen* Bewertungsfunktion.

Wir werden zwei Beispiele betrachten: Zusammenstellung eines Geldbetrags aus Münzen/Scheinen, sowie Kürzeste-Wege-Suche.

Einen Geldbetrag zusammenstellen

Wir behaupten, dass man mit einer Greedy-Strategie jeden beliebigen Geldbetrag in Euro- und Cent-Münzen, sowie -Scheinen mit einer minimalen Anzahl von Münzen und Scheinen kombinieren kann.

Als Stückelung eines Betrags bezeichnen wir die Angabe einer Anzahl zu jedem erhältlichen Geldschein und jeder Münze, so dass als Summe der Betrag herauskommt. Die Stückelung ist minimal, wenn sie mit minimal vielen Scheinen/Münzen auskommt.

```
INPUT: Geldbetrag
OUTPUT: Minimale Stückelung
Starte mit leerer Menge, setze Summe auf 0.
Solange Summe kleiner als Zielbetrag:
        Füge größtmöglichen Wert (mit einem Schein/Münze) hinzu.
        Erhöhe Summe um den hinzugefügten Wert.
```

Der Algorithmus liefert tatsächlich die optimale Lösung. (Beweis?)

Letztlich verantwortlich hierfür: die Werte 1,2,5,10,20,50,100,200,500,... Cent.

Wie ist das bei *Sickeln*? Die gibt es zu Beträgen 1,6,9,15,22 und 80.

Kürzeste Wege

Der bekannteste Greedy-Algorithmus ist die Kürzeste-Wege-Suche nach Dijkstra. Ausgangsdaten sind ein kantengewichteter Graph $G = (V, E, \gamma)$, sowie ein Startknoten $s \in V$.

Der Algorithmus arbeitet mit drei Mengen von Knoten:

B ist die Menge der *fertig bearbeiteten* Knoten, initial $B = \{s\}$.

R ist die Menge der *Randknoten*, $R = \{v \in V \mid v \notin B \wedge \exists u \in B : (u, v) \in E\}$.

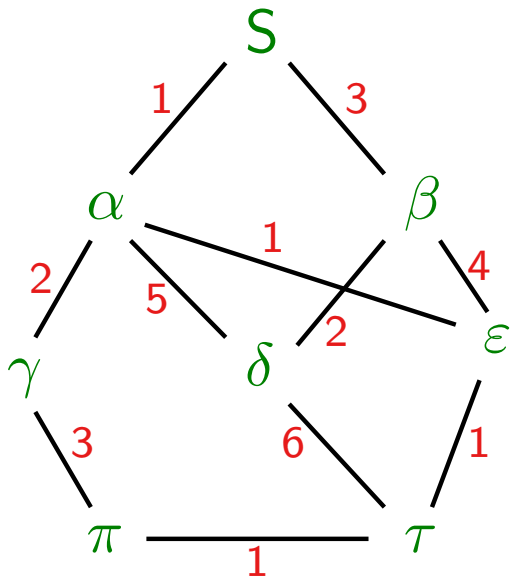
U ist die Menge der *unbekannten* Knoten, d.h. $U = V \setminus (B \cup R)$.

Nun wird sukzessive ein Knoten aus R gewählt, von dem man einen kürzesten Weg zu s kennt – das ist der Knoten aus R mit dem kleinsten *Schätzwert* für den Abstand zu s. Wie der Schätzwert gebildet wird und warum der kleinste dieser Schätzwerte tatsächlich einen korrekten Abstandswert angibt, ist natürlich erst noch zu beweisen (später...).

Nun wird der Knoten aus R ermittelt, der zu B hinzukommt. Danach müssen natürlich alle drei Mengen (B, R, U) in der Datenhaltung angepasst werden. Schritt für Schritt wird so B immer größer, bis $B = V$ gilt, d.h. alle kürzesten Wege zu s bekannt.

Dijkstra-Algorithmus: Beispiel

Wir führen den Algorithmus an folgendem Beispiel durch:



Bitte selbst zu Ende machen. (Ergebnis s.u.)

Zu Beginn der Ausführung haben wir:

$B = \{S\}$, $R = \{\alpha, \beta\}$, $U = \{\gamma, \delta, \varepsilon, \pi, \tau\}$ und

$d(S) = 0$, $D(\alpha) = 1$, $D(\beta) = 3$.

Nach dem ersten Durchgang wird daraus:

$B = \{S, \alpha\}$, $R = \{\beta, \gamma, \delta, \varepsilon\}$, $U = \{\pi, \tau\}$,

$d(S) = 0$, $d(\alpha) = 1$,

$D(\beta) = 3$, $D(\gamma) = 3$, $D(\delta) = 6$, $D(\varepsilon) = 2$.

Nach dem zweiten Durchgang wird daraus:

$B = \{S, \alpha, \varepsilon\}$, $R = \{\beta, \gamma, \delta, \tau\}$, $U = \{\pi\}$,

$d(S) = 0$, $d(\alpha) = 1$, $d(\varepsilon) = 2$

$D(\beta) = 3$, $D(\gamma) = 3$, $D(\delta) = 6$, $D(\tau) = 3$.

Ergebnis: $d(S) = 0, d(\alpha) = 1, d(\varepsilon) = 2, d(\beta) = 3, d(\gamma) = 3, d(\tau) = 3, d(\pi) = 4, d(\delta) = 5.$