LC-Suche

$LC - Search(t)$:
if t is a solution **then return** t;
$L := \{t\}$; (Lebendige Knoten)
forever do
  $E := \min L$; (bezüglich $\hat{c}(x)$)
  $L := L - \{E\}$;
  for each child x of E **do**
    if x is a solution **then return** x;
    $L := L \cup \{x\}$
  od;
  if $L = \emptyset$ **then return** "no solution"
od

LC-Suche

Wie wählen wir $\hat{c}(x)$?

- Falls wir $\hat{c}(x) =$ Tiefe von x im Suchbaum wählen, erhalten wir wieder eine Breitensuche.
- Seien $c(x)$ die Entfernung zu einem Lösungsknoten im Unterbaum von x. Wählen wir $\hat{c}(x) = c(x)$, dann wird stets ein kürzester Pfad zu einer Lösung gefunden. Problem: $c(x)$ ist nicht bekannt.
- Sei $\hat{g}(x)$ eine Schätzung des Aufwands, eine Lösung im Unterbaum von x zu finden. Wir können $\hat{c}(x) = f(h(x)) + \hat{g}(x)$ wählen, wobei f eine monoton steigende Funktion und $h(x)$ der bisherige Aufwand war, x von der Wurzel zu erreichen.
- $h(x) = 0$ führt zu bisweilen DFS-ähnlichem Verhalten.

Branch-and-Bound für Optimierungsprobleme

Eine bounding function diente dazu, Zweige des Suchbaums abzuschneiden, in welchen garantiert keine Lösung zu finden ist.

Wenn wir *Optimierungsprobleme* betrachten, dann kann diese Technik weiter verwendet werden und wir können sie sogar verfeinern: Ist eine *Schranke* L bekannt, dann können wir sogar Zweige abschneiden, in welchen nur Lösungen $> L$ vorkommen können (bei einem Minimierungsproblem).

Branch-and-Bound für Optimierungsprobleme

Woher erhalten wir eine geeignete Schranke?

- durch eine Heuristik
- durch Approximationsalgorithmen
- triviale Schranken (z.B. $-\infty$ oder ∞)
- durch Lösungen, die durch den Algorithmus selbst gefunden werden
- durch Kombinationen obiger Möglichkeiten

Beispiel: Scheduling

Wir betrachten folgendes Problem:

Es müssen n Aufgaben bewältigt werden. Für jede dieser Aufgaben gibt es eine *deadline* d_i, eine *Strafe* p_i und eine *Bearbeitungsdauer* t_i.

Es dauert t_i Minuten, Aufgabe i zu lösen. Sie muß spätestens nach d_i Minuten gelöst sein, sonst wird eine Strafe von $\$p_i$ fällig.

Gesucht ist eine Reihenfolge, in der die Aufgaben bearbeitet werden sollen, um eine minimale Strafe zu bezahlen.

Beispiel

Es sind vier Aufgaben gegeben:

i	p_i	d_i	t_i
1	5	1	1
2	10	3	2
3	6	2	1
4	3	1	1

Der Lösungsraum besteht hier aus allen Plänen.

Eine optimale Reihenfolge ist leicht zu finden. Es genügt also, die Teilmengen von $\{1, 2, 3, 4\}$ zu betrachten, die wir ohne Strafe planen können.

Beispiel: Scheduling

Wir können mit einem leeren Plan beginnen, der natürlich eine legale (aber suboptimale) Lösung ist.

Wir verwenden die boolschen Variablen $x_1, \ldots, x_n$, um einen Plan darzustellen.

Wir können den Baum abschneiden, wenn der aktuelle Teilplan bereits teurer ist, als ein bekannter Plan.

Wir können auch eine obere Schranke aus einem Teilplan gewinnen: Wir addieren zu seiner Strafe noch die Strafen aller Aufgaben hinzu, die noch betrachtet werden müssen.

Ergebnisse

Wir betrachten 20 Aufgaben mit $d_i = 100i$. Die Strafen p_i werden zufällig zwischen 0 und 999 gewählt. Ebenso wählen wir t_i zufällig zwischen 0 und 299.

Soviele Knoten wurden im Experiment betrachtet:

- FIFO: 459009 Knoten
- LIFO: 459009 Knoten
- FIFO Branch-and-Bound: 9726 Knoten
- LIFO Branch-and-Bound: 348 Knoten
- LC Branch-and-Bound: 303 Knoten

Als $\hat{c}(x)$ wurde die Strafe des Knotens x verwendet.