Ausgabe für FIFO Branch-and-Bound:

```
0 1 0 0 0 0 1 1 1 0 1 1 1 1 0 1 0 0 1 0 -> 2914
0 1 0 0 0 0 1 1 1 0 1 1 1 1 0 1 0 0 1 1 -> 2903
0 1 0 0 1 0 0 1 1 0 1 1 1 1 0 1 0 1 1 0 -> 2886
0 1 0 0 1 0 0 1 1 0 1 1 1 1 1 1 0 0 1 1 -> 2877
0 1 0 0 1 0 0 1 1 1 1 1 1 1 0 1 0 0 1 0 -> 2783
0 1 0 0 1 0 0 1 1 1 1 1 1 1 0 1 0 0 1 1 -> 2772
0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 -> 2593
0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 1 -> 2582
Nodes: 9726
```

Ausgabe für LIFO Branch-and-Bound:

```
0 1 0 1 1 1 0 1 0 1 1 1 0 1 0 1 1 0 0 1 -> 3717
0 1 0 1 1 1 0 1 0 1 1 1 0 1 0 1 1 0 0 0 -> 3728
0 1 0 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1 0 1 -> 3670
0 1 0 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1 0 0 -> 3681
0 1 0 1 1 1 0 1 0 1 1 1 0 1 0 1 0 0 1 1 -> 3346
0 1 0 1 1 1 0 1 0 1 1 1 0 1 0 1 0 0 1 0 -> 3357
0 1 0 1 1 1 0 1 0 1 0 0 1 1 0 1 0 0 1 1 -> 3342
0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 1 -> 3303
0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 -> 3314
0 1 0 1 1 1 0 1 0 0 0 1 1 1 0 1 0 1 0 1 -> 3271
0 1 0 1 1 1 0 1 0 0 0 1 1 1 0 1 0 1 0 0 -> 3282
0 1 0 1 1 1 0 1 0 0 0 1 1 1 0 1 0 0 1 1 -> 2947
0 1 0 1 1 1 0 1 0 0 0 1 1 1 0 1 0 0 1 0 -> 2958
0 1 0 1 1 0 1 0 1 0 0 1 1 1 0 1 0 0 1 1 -> 2842
0 1 0 1 1 0 1 0 1 0 0 1 1 1 0 1 0 0 1 0 -> 2853
0 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1 0 0 1 1 -> 2769
0 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1 0 0 1 0 -> 2780
0 1 0 0 1 1 1 0 1 0 1 1 1 1 0 1 0 0 1 1 -> 2655
0 1 0 0 1 1 1 0 1 0 1 1 1 1 0 1 0 0 1 0 -> 2666
0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 1 -> 2582
0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 -> 2593
Nodes: 348
```

Ausgabe für LC Branch-and-Bound:

```
0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 1 -> 2582
0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 -> 2593
Nodes: 303
```

Hier wurden bereits im Inneren des Suchbaums so gute Schranken berechnet, daß im Endeffekt nur zwei Blätter besucht werden mußten.

Selbst für $N = 40$ bleibt der Suchbaum angenehm klein:

```
0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 0 1 1 0 \
1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 1 -> 3540
0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 0 1 1 0 \
1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 -> 3583
Nodes: 19969
```

Eine bessere Schranke

Der LC-Suchalgorithmus berechnete eine Schranke für die zu zahlende Strafe, indem er die Strafen der noch nicht betrachteten Aufgaben zur bisher zu zahlenden Strafe addierte.

Dies ist eine sehr konservative Abschätzung.

Wir können diese Abschätzung leicht Verbessern:

Seien $x_1, \ldots, x_{k-1}$ schon festgelegt. Wir gehen $i = k, \ldots, n$ durch und dabei setzen $x_i = 1$, falls dadurch die Deadline für die Aufgabe i nicht verletzt wird.

Dies führt zu einer Menge von Aufgabe, die ohne Strafe durchgeführt werden können.

```
#include < stdio.h >
#include "schedule.h"
int main(void)
{
  int i, z = 0, upper = 100000; struct node root = { 0, 0, 0};
  create_jobs(); insert(root);
  while(!isempty()) {
    struct node E = extract(), E1, E2;
    int k = E.k, u = E.p, ut = E.t; z++;
    for(i = k; i < N; i++)
      if(ut + jobs[i].t ≤ jobs[i].d) ut += jobs[i].t;
      else u += jobs[i].p;
    if(u < upper) upper = u;
    if(k ≡ N) {
      for(i = 0; i < k; i++) printf("%d ", E.x[i]);
      printf("-> %d\n", E.p);
      continue;}
    E1 = E; E1.k = k + 1; E1.x[k] = 0; E1.p += jobs[k].p;
    E2 = E; E2.k = k + 1; E2.t += jobs[k].t; E2.x[k] = 1;
    if(E1.p ≤ upper) insert(E1);
    if(E2.p ≤ upper && E2.t ≤ jobs[k].d) insert(E2);
  }
  printf("Nodes: %d\n", z); return 0;
}
```

$N = 40$, LC-Suche ohne Heuristik:

```
0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 0 1 1 0 \
1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 1 -> 3540
0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 0 1 1 0 \
1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 -> 3583
Nodes: 19969
```

$N = 40$, LC-Suche mit Heuristik:

```
0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 0 1 1 0 \
1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 1 -> 3540
Nodes: 12867
```

Mit Heuristik: 5 mal weniger Speicherverbrauch

(Bei LIFO allerdings noch viel, viel, viel weniger.)