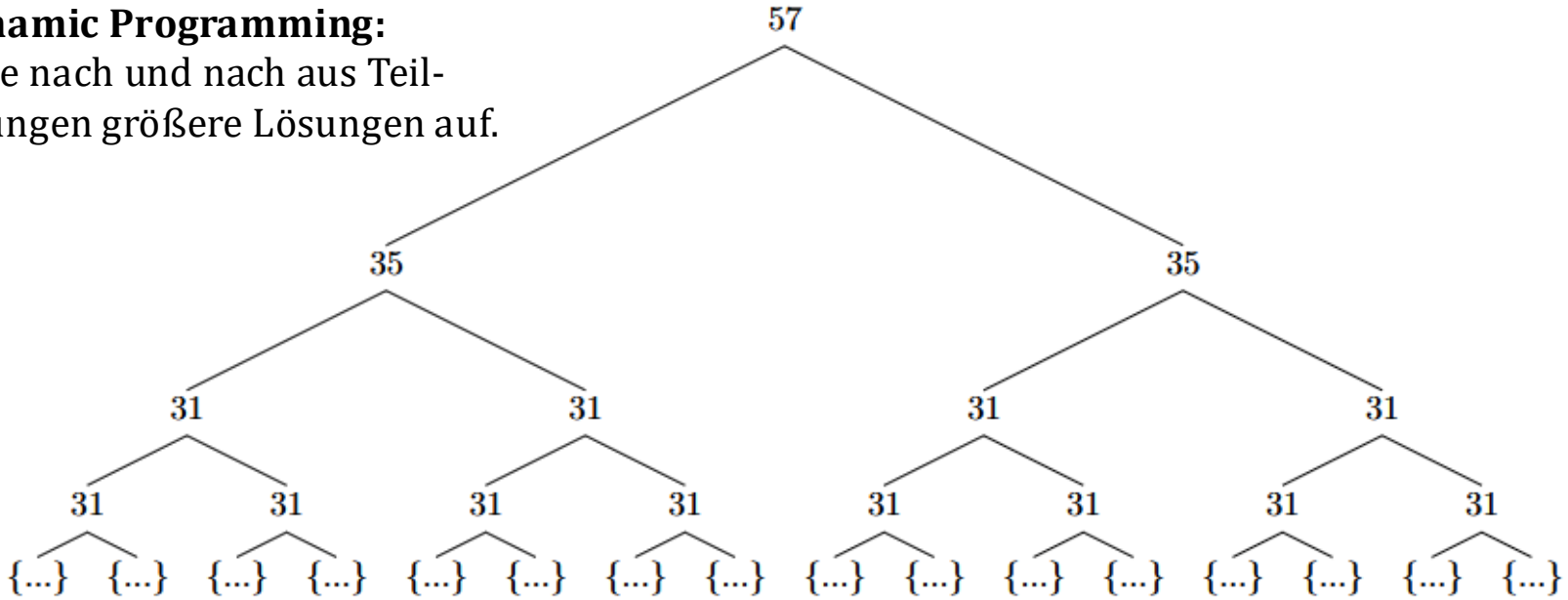
Technische Universität Braunschweig

Algorithmen und Datenstrukturen 2

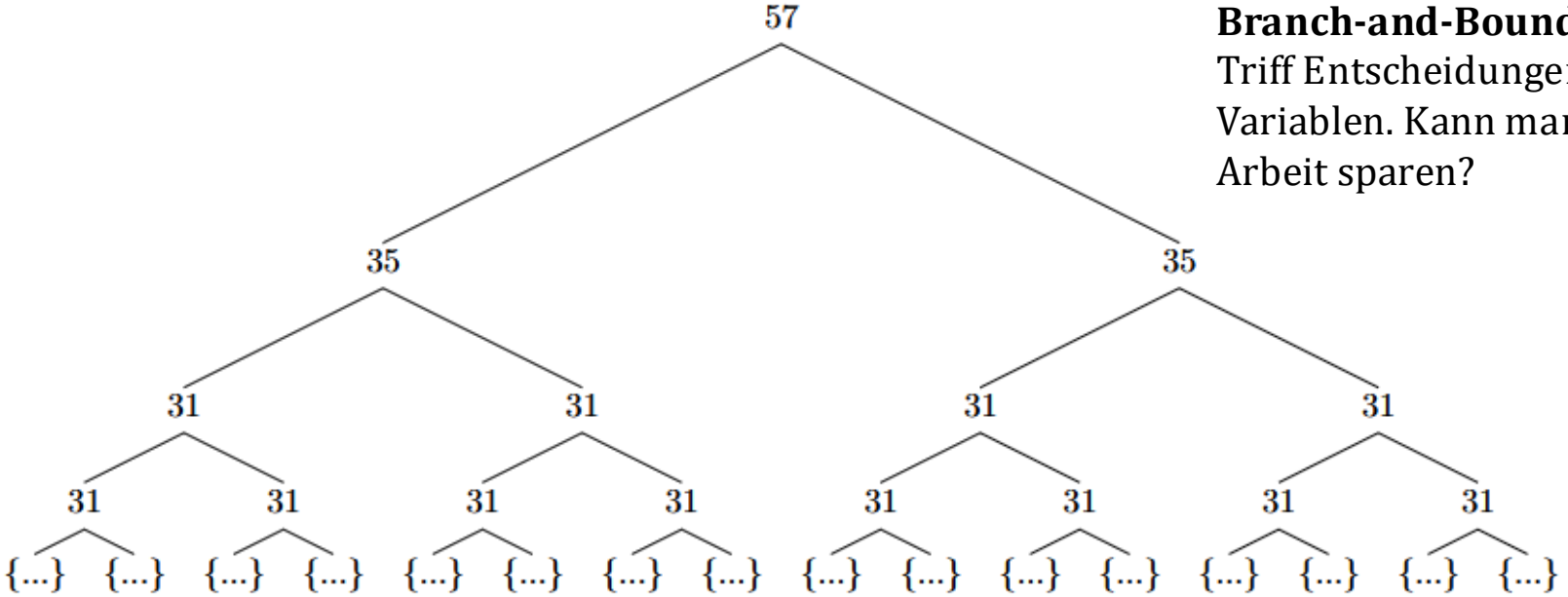
Arne Schmidt

Enumerationsbaum

Dynamic Programming:
Baue nach und nach aus Teil-lösungen größere Lösungen auf.



Enumerationsbaum



Branch-and-Bound:
Triff Entscheidungen für Variablen. Kann man hier Arbeit sparen?

Kapitel 3 – Branch-And-Bound

Entstehung von Branch-and-Bound

Alisa Land

Alison Harcourt
(geb. Doig)

ECONOMETRICA

VOLUME 28 July, 1960 NUMBER 3

AN AUTOMATIC METHOD OF SOLVING DISCRETE PROGRAMMING PROBLEMS

BY A. H. LAND AND A. G. DOIG

THERE IS A growing literature [**1**, **3**, **5**, **6**] about optimization problems which could be formulated as linear programming problems with additional constraints that some or all of the variables may take only integral values. This form of linear programming arises whenever there are indivisibilities. It is not meaningful, for instance, to schedule 3–7/10 flights between two cities, or to undertake only 1/4 of the necessary setting up operation for running a job through a machine shop. Yet it is basic to linear programming that the variables are free to take on any positive value,[1] and this sort of answer is very likely to turn up.

Die Idee

Triff nach und nach Entscheidungen, ob eine Variable höchstens b oder mindestens b+1 sein muss.

Halte dabei fest, welche Lösungen wir schon kennen, und was wir mit den Entscheidungen noch erreichen können.

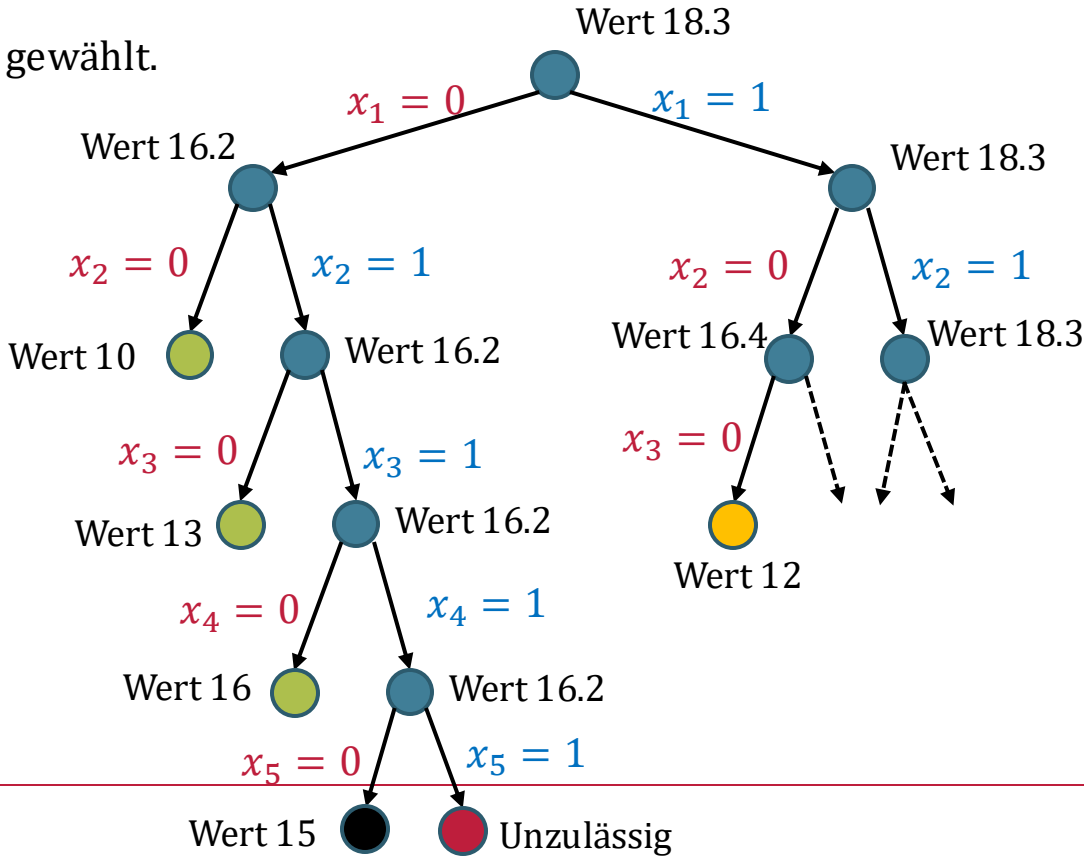
Bei binären Problemen (Variablen sind 0 oder 1) entscheiden wir also: Die Variable ist definitiv in der Lösung, oder gar nicht!

Beispiel: Maximum Knapsack

Sei Z = 20 und Objekte folgendermaßen gewählt.

Objekt i	1	2	3	4	5
Gewicht z_i	7	9	6	3	5
Wert p_i	7	8	5	2	3

Über Greedy für Fractional Knapsack erhalten wir eine obere Schranke!



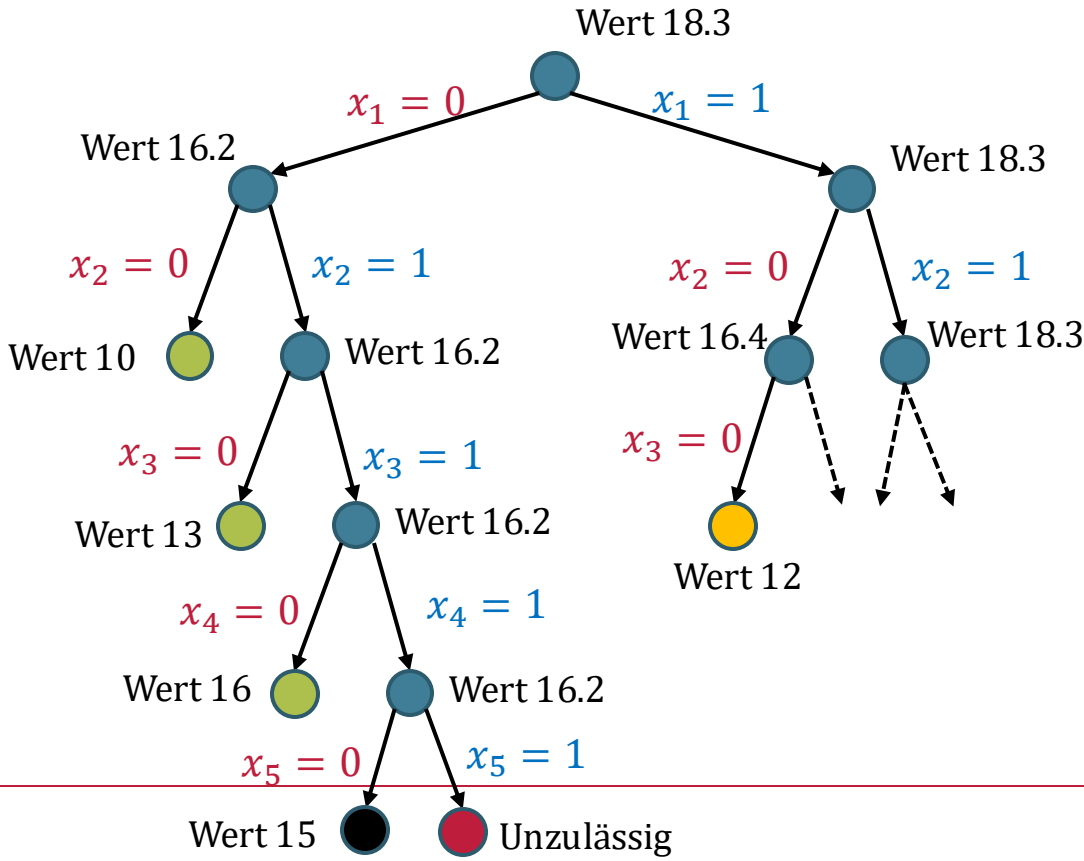
Beispiel: Maximum Knapsack

Zusammenfassend:

- **(Grüner Knoten):** Bessere Ganzzahlige Lösung gefunden!
- **(Gelber Knoten)** Es gibt hier keine bessere Lösung als bisher!
- **(Roter Knoten)** Unzulässige Entscheidungen!
- **(Schwarzer Knoten)** Wir können keine Entscheidung mehr treffen!

Dabei wichtig:

- Die obere Schranke ist nur für den Teilbaum gültig.
- Die Untere (zulässige Lösung) für den gesamten Enumerationsbaum!



Überlegungen

Greedy für Fractional Knapsack liefert eine obere Schranke.

Können wir nicht den Greedy-Algorithmus für Maximum-Knapsack nutzen, um eine untere Lösung zu bestimmen?

An jedem Knoten des Baums kann sowohl eine untere als auch eine obere Schranke berechnet werden!

Weiteres Beispiel

Z = 9, sowie:

i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

Sei im folgenden:

- P die untere Schranke (Maximum aus bisher bester Lösung und Wert von Greedy0).
- U die obere Schranke (Wert der Greedy Lösung für Fractional Knapsack).

Schauen wir uns den Branch-and-Bound-Baum an der Tafel an!

Algorithmus

Algorithmus 3.1 Branch-And-Bound als Unterroutine

Eingabe: $z_1, ..., z_n, Z, p_1, ...p_n$ (global: Kosten/Nutzenwerte, Kostenschranke)

P (global: bester bekannter Lösungswert)

ℓ (nächster Index, über den verzweigt werden soll)

$x_j = b_j$ für $j = 1, ..., \ell - 1$ mit $b_j \in \{0, 1\}$ (bislang fixierte Binärvariable)

Ausgabe: $\max \left\{ \sum_{j=1}^{\ell-1} b_j p_j + \sum_{j=\ell}^{n} x_j p_j \mid \sum_{j=1}^{\ell-1} b_j z_j + \sum_{j=\ell}^{n} x_j z_j \leq Z, x_j \in \{0,1\} \right\}$

Also: Lösung des Knapsackproblems mit den ersten $\ell - 1$ Variablen fixiert

```
1:  procedure BRANCH-AND-BOUND(ℓ)
2:      if (Σ_{j=1}^{ℓ-1} b_j z_j > Z) then return          ▷ unzulässig
3:      Compute L := LB(b_1, ..., b_{ℓ-1})
4:      if L > P then P := L                               ▷ Lösungswert verbessert
5:      if (ℓ > n) then return                             ▷ Blatt im Baum erreicht
6:      U := UB(b_1, ..., b_{ℓ-1})                         ▷ (Obere Schranke berechnen)
7:      if (U > P) then
8:          b_ℓ := 0; BRANCH-AND-BOUND(ℓ + 1);
9:          b_ℓ := 1; BRANCH-AND-BOUND(ℓ + 1);
10:     return
```

Nächste Woche

Wichtig zu beachten:
Die Variablen wurden hier dem Index entsprechend gebrancht.