$$P(x,i) = \begin{cases} P(x,i-1), \\ max\{P(x,i- \end{cases}$$

i \ x	0	1	2	3	3	5
0	1	0	0	0	0	0
1	1	0	0	0	0	0
2	1	0	0	0	+13	
3						

2 Dynamic Programming

Algorithmen und Datenstrukturen 2
Sommer 2026

Prof. Dr. Sándor Fekete

Dynamic Programming für Anfänger

- Wieviel ist das?

  1+1+1+1+1+1+1+1+1+1+1+1+1 =13

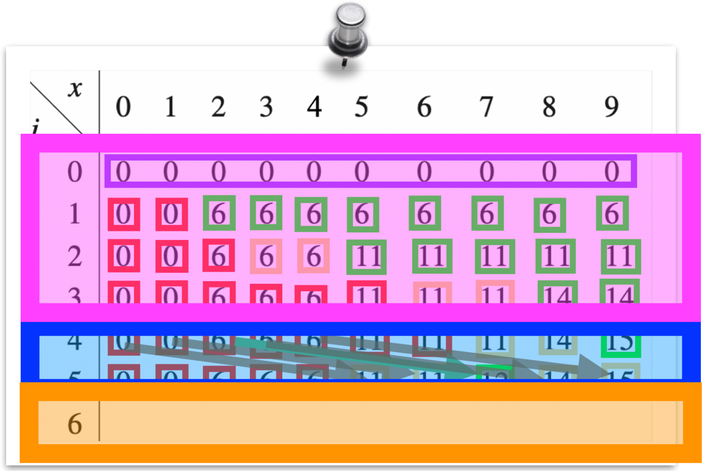
- Und das?

  1+1+1+1+1+1+1+1+1+1+1+1+1+1 = 13+1 = 14

- Dynamic Programming:
  - Nicht von vorn lösen
  - Zwischenergebnisse merken

Dynamic Programming: Grundstruktur

- Nicht von vorn lösen
- Zwischenergebnisse merken
- Man kann die Vorgeschichte beim Weiterrechnen von der Zukunft trennen und deshalb den Übergang durch Enumerieren lösen.



Objekt i

$$P(x,i) = \begin{cases} P(x,i-1), \\ max\{P(x,i- \end{cases}$$

i \ x	0	1	2	3	3	5
0	1	0	0	0	0	0
1	1	0	0	0	0	0
2	1	0	0	0	+13	
3						

2.1 Dynamic Programming für Subset Sum

Algorithmen und Datenstrukturen 2
Sommer 2026

Prof. Dr. Sándor Fekete

Prinzip

$$\mathcal{S}(x,i) = \begin{cases} 1, & \text{falls x mit } z_1, ..., z_i \text{ erzeugt werden kann} \\ 0, & sonst \end{cases}$$

$$\mathcal{S}(x,0) = 0, \text{ für alle } x \in \{1, ..., Z\}; \mathcal{S}(0,0) = 1$$

i \ x	0	1	2	3	3	5	6	7	…	13	14	15	16	17	18	19	20	…	240
0	1	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	…	0
1	1	0	0	0	0	0	0	1	…	0	0	0	+13	0	0	0	0	…	0
2	1	0	0	0	+13	0	1	…	1	0	0	0	0	0	0	1	…	0	
3																			

$$\mathcal{S}(x, i-1) = 1 \Rightarrow \mathcal{S}(x,i) = 1$$

$$\mathcal{S}(x - z_i, i-1) = 1 \Rightarrow \mathcal{S}(x,i) = 1$$

Algorithmus

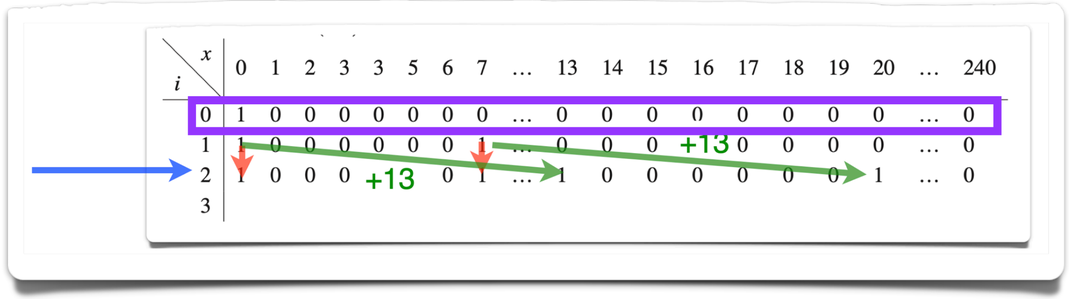
Algorithmus 2.2 . (Dynamic Programming für SUBSET SUM).

Eingabe: *Zahlen* $z_1, ..., z_n$ *mit* $\sum_{i=1}^{n} z_i = Z$

Ausgabe: *Boolesche Funktion* $\mathcal{S} : \{0, ..., Z\} \times \{0, ..., n\} \to \{0, 1\}$

$$\text{mit } \mathcal{S}(x,i) = \begin{cases} 1, & \text{falls } x \text{ mit } z_1, ..., z_i \text{ erzeugt werden kann} \\ 0, & \text{sonst} \end{cases}$$

```
1:  S(0,0) := 1
2:  for (x=1) to Z do
3:      S(x,0) := 0;
4:  for (i = 1) to n do
5:      for (x = 0) to z_i − 1 do
6:          S(x,i) := S(x,i−1);
7:      for (x = z_i) to Z do
8:          if ( S(x,i−1) = 1 OR (S((x−z_i),i−1) = 1) then
9:              S(x,i) := 1;
10:         else
11:             S(x,i) := 0;
12: return
```



Laufzeit

Algorithmus 2.2 . (Dynamic Programming für SUBSET SUM).

Eingabe: *Zahlen* $z_1, ..., z_n$ *mit* $\sum_{i=1}^{n} z_i = Z$

Ausgabe: *Boolesche Funktion* $\mathscr{S} : \{0, ..., Z\} \times \{0, ..., n\} \to \{0, 1\}$

$$\text{mit } \mathscr{S}(x,i) = \begin{cases} 1, & \text{falls } x \text{ mit } z_1, ..., z_i \text{ erzeugt werden kann} \\ 0, & \text{sonst} \end{cases}$$

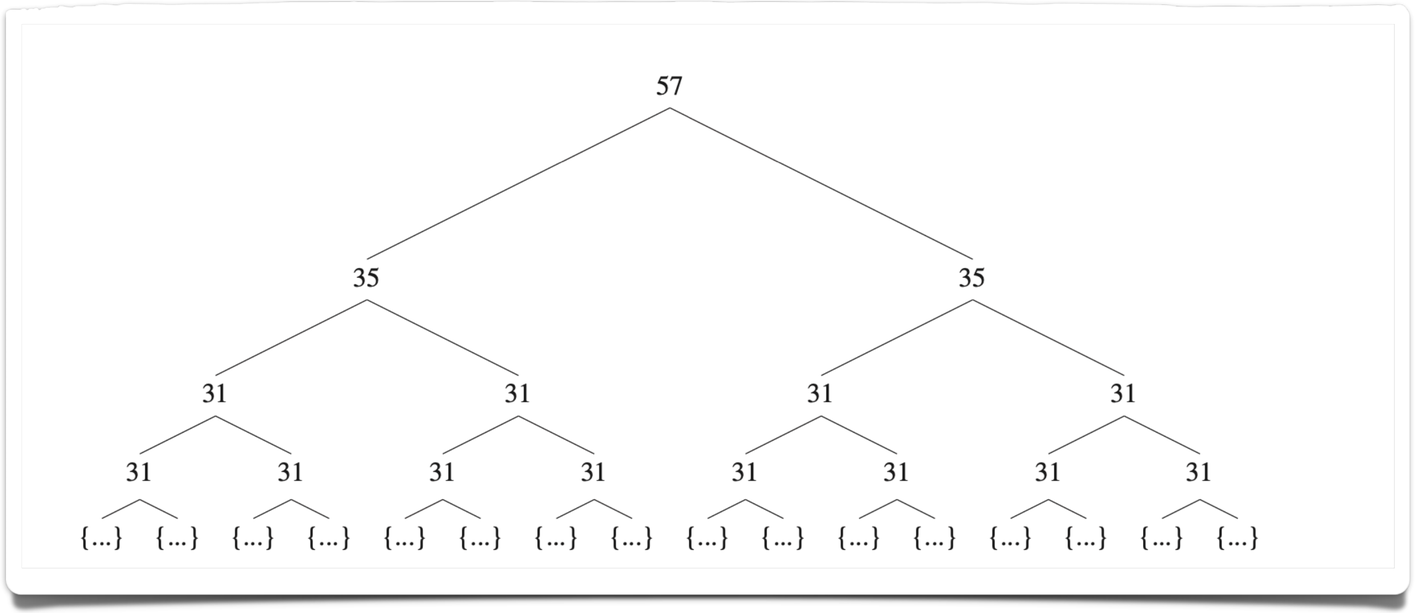
```
1:  S(0,0) := 1
2:  for (x=1) to Z do
3:      S(x,0) := 0;
4:  for (i = 1) to n do
5:      for (x = 0) to z_i − 1 do
6:          S(x,i) := S(x,i−1);
7:      for (x = z_i) to Z do
8:          if ((S(x,i−1) = 1) OR (S((x−z_i),i−1) = 1)) then
9:              S(x,i) := 1;
10:         else
11:             S(x,i) := 0;
12: return
```

$O(nZ)$

Effizienz?!

Dynamic Programming

$O(nZ)$



2^n

- Exponentiell viele Fälle!
- Wo sparen wir Arbeit?

Z klein im Vergleich zu 2^n : viele Duplikate gespart

Z groß im Vergleich zu 2^n : viel unnötige Arbeit

$$P(x,i) = \begin{cases} P(x,i-1), \\ max\{P(x,i- \end{cases}$$

i \ x	0	1	2	3	3	5
0	1	0	0	0	0	0
1	1	0	0	0	0	0
2	1	0	0	0	+13	
3						

2.2 Dynamic Programming für Knapsack

Algorithmen und Datenstrukturen 2
Sommer 2026

Prof. Dr. Sándor Fekete

Von Subset Sum zu Knapsack

$$\mathcal{S}(x,i) = \begin{cases} 1, & \text{falls x mit } z_1, ..., z_i \text{ erzeugt werden kann} \\ 0, & sonst \end{cases}$$

i \ x	0	1	2	3	3	5	6	7	…	13	14	15	16	17	18	19	20	…	240
0	1	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	…	0
1	1	0	0	0	0	0	0	1	…	0	0	0	0	0	0	0	0	…	0
2	1	0	0	0	0	0	0	1	…	1	0	0	0	0	0	0	1	…	0
3	1	0	0	0	0	0	0	1	…	1	0	0	0	1	0	0	1	…	0

$P(x,i)$

der höchste Nutzen

der sich mit den Objekten

$1, \ldots, i$ erzielen lässt

wenn die Kosten auf x begrenzt sind

Beispiel: Knapsack

Beispiel 2.4 (Knapsackproblem).

Sei $Z = 9$ und seien folgende sieben Objekte gegeben:

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

Dynamic Programming für Knapsack

$P(x, i)$

der höchste Nutzen

der sich mit den Objekten $1, \ldots, i$ erzielen lässt

wenn die Kosten auf x begrenzt sind

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

1. Objekt i ist zu teuer für Kostenschranke x, falls $z_i > x$

2. Objekt i ist nicht zu teuer, bringt aber keine Verbesserung

3. Objekt i ist nicht zu teuer und bringt eine Verbesserung

i \ x	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6
2	0	0	6	6	6	11	11	11	11	11
3	0	0	6	6	6	11	11	11	14	14
4	0	0	6	6	6	11	11	11	14	15
5	0	0	6	6	6	11	11	12	14	15
6	0	0	6	6	6	11	11	12	14	15
7	0	0	6	6	6	11	11	12	14	15

Systematisch!

„Bellman-Rekursion“

$$P(x,i) = \begin{cases} P(x,i-1), & \text{falls } z_i > x \\ max\{P(x,i-1), P(x-z_i,i-1)+p_i\}, & \text{sonst} \end{cases}$$

1. Objekt i ist zu teuer für Kostenschranke x, falls $z_i > x$

2. Objekt i ist nicht zu teuer, bringt aber keine Verbesserung

3. Objekt i ist nicht zu teuer und bringt eine Verbesserung

i \ x	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6
2	0	0	6	6	6	11	11	11	11	11
3	0	0	6	6	6	11	11	11	14	14
4	0	0	6	6	6	11	11	11	14	15
5	0	0	6	6	6	11	11	12	14	15
6	0	0	6	6	6	11	11	12	14	15
7	0	0	6	6	6	11	11	12	14	15

Historie

Richard Bellman (1920-1984)

DYNAMIC
PROGRAMMING

BY

RICHARD BELLMAN

1957

PRINCETON UNIVERSITY PRESS
PRINCETON, NEW JERSEY

Preface

The purpose of this work is to provide an introduction to the mathematical theory of multi-stage decision processes. Since these constitute a somewhat formidable set of terms we have coined the term "dynamic programming" to describe the subject matter. Actually, as we shall see, the distinction involves more than nomenclature. Rather, it involves a certain conceptual framework which furnishes us a new and versatile mathematical tool for the treatment of many novel and interesting problems both in this new discipline and in various parts of classical analysis. Before expanding upon this theme, let us present a brief discussion of what is meant by a multi-stage decision process.

The functional equations which arise in this way are of a novel type, completely different from any of the functional equations encountered in classical analysis. The particular one we shall employ for purposes of discussion in this chapter is

$$(1) \qquad f(x) = \operatorname*{Max}_{0 \le y \le x} [g(y) + h(x-y) + f(ay + b(x-y))].$$

where g and h are known functions and a and b are known constants, satisfying the condition $0 \le a, b < 1$.

Algorithmus

Algorithmus 2.5 *(Dynamic Programming für Knapsack)*

Eingabe: $z_1, ..., z_n, Z, p_1, ...p_n$

Ausgabe: *Funktion* $P : \{1, ..., Z\} \times \{1, ..., n\} \to \mathbb{R};\ (x, i) \mapsto P(x, i)$

$$mit\ P(x,i) = max \sum_{i=1}^{i} p_j y_j\ mit \sum_{j=1}^{i} z_j y_j \leq x,\ für\ y_j \in \{0,1\}$$

```
1: for (x = 0) to Z do
2:     P(x,0) := 0
3: for (i = 1) to n do
4:     for (x = 0) to (z_i − 1) do
5:         P(x,i) := P(x,i−1)
6:     for (x = z_i) to Z do
7:         if ((P(x − z_i, i−1) + p_i) > P(x,i−1)) then
8:             P(x,i) := P(x − z_i, i−1) + p_i
9:         else
0:             P(x,i) := P(x,i−1)
```

i \ x	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6
2	0	0	6	6	6	11	11	11	11	11
3	0	0	6	6	6	11	11	11	14	14
4	0	0	6	6	6	11	11	11	14	15
5	0	0	6	6	6	11	11	12	14	15
6	0	0	6	6	6	11	11	12	14	15
7	0	0	6	6	6	11	11	12	14	15

Korrektheit und Laufzeit

Satz 2.6 *Algorithmus 2.5 berechnet besten Lösungswert für das Rucksackproblem in einer Laufzeit $O(nZ)$.*

Algorithmus 2.5 *(Dynamic Programming für Knapsack)*

Eingabe: $z_1, ..., z_n, Z, p_1, ...p_n$

Ausgabe: Funktion $P : \{1, ..., Z\} \times \{1, ..., n\} \to \mathbb{R};\ (x, i) \mapsto P(x, i)$

$$\text{mit } P(x,i) = max \sum_{i=1}^{i} p_j y_j \text{ mit } \sum_{j=1}^{i} z_j y_j \leq x, \text{ für } y_j \in \{0,1\}$$

```
1: for (x = 0) to Z do
2:     P(x, 0) := 0
3: for (i = 1) to n do
4:     for (x = 0) to (z_i − 1) do
5:         P(x, i) := P(x, i − 1)
6:     for (x = z_i) to Z do
7:         if ((P(x − z_i, i − 1) + p_i) > P(x, i − 1)) then
8:             P(x, i) := P(x − z_i, i − 1) + p_i
9:         else
0:             P(x, i) := P(x, i − 1)
```

$O(nZ)$

i \ x	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6
2	0	0	6	6	6	11	11	11	11	11
3	0	0	6	6	6	11	11	11	14	14
4	0	0	6	6	6	11	11	11	14	15
5	0	0	6	6	6	11	11	12	14	15
6	0	0	6	6	6	11	11	12	14	15
7	0	0	6	6	6	11	11	12	14	15

Induktion!

1. Objekt i ist zu teuer für Kostenschranke x, falls $z_i > x$
2. Objekt i ist nicht zu teuer, bringt aber keine Verbesserung
3. Objekt i ist nicht zu teuer und bringt eine Verbesserung

Sparsamer?!

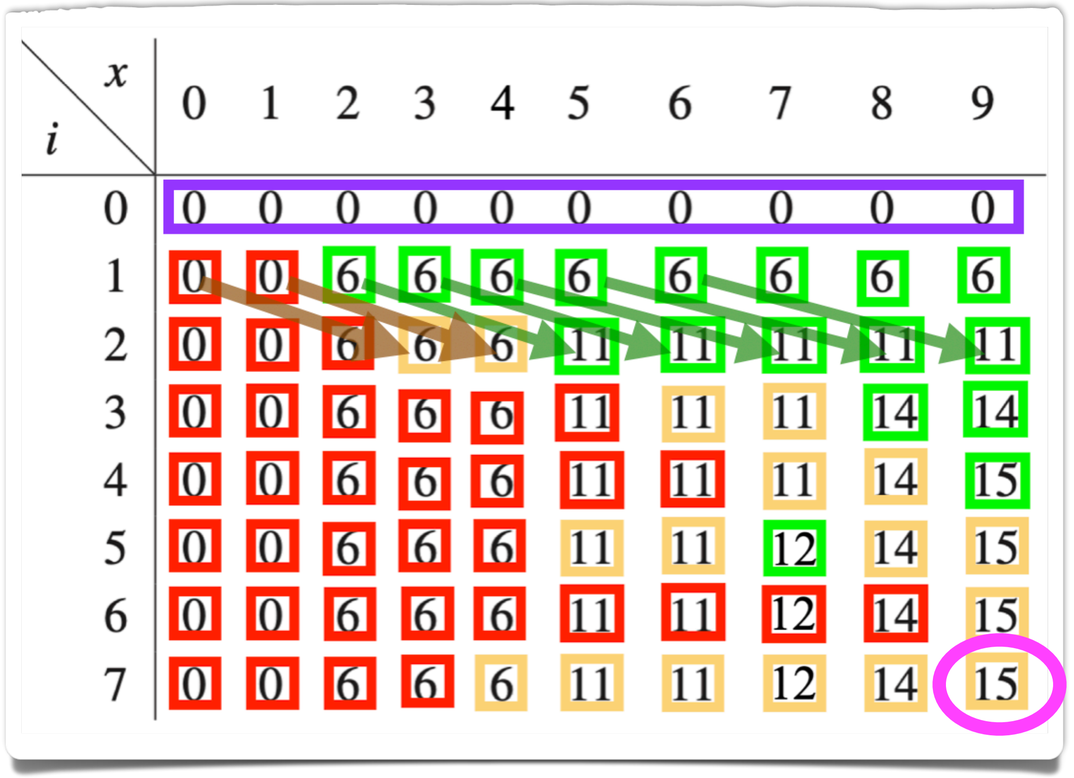
Limitierender Faktor bei Dynamic Programming?

- Laufzeit?
- Speicherplatz!

$O(nZ)$

Idee:

- Nicht ganze Tabelle speichern
- Nur die relevante Zeile

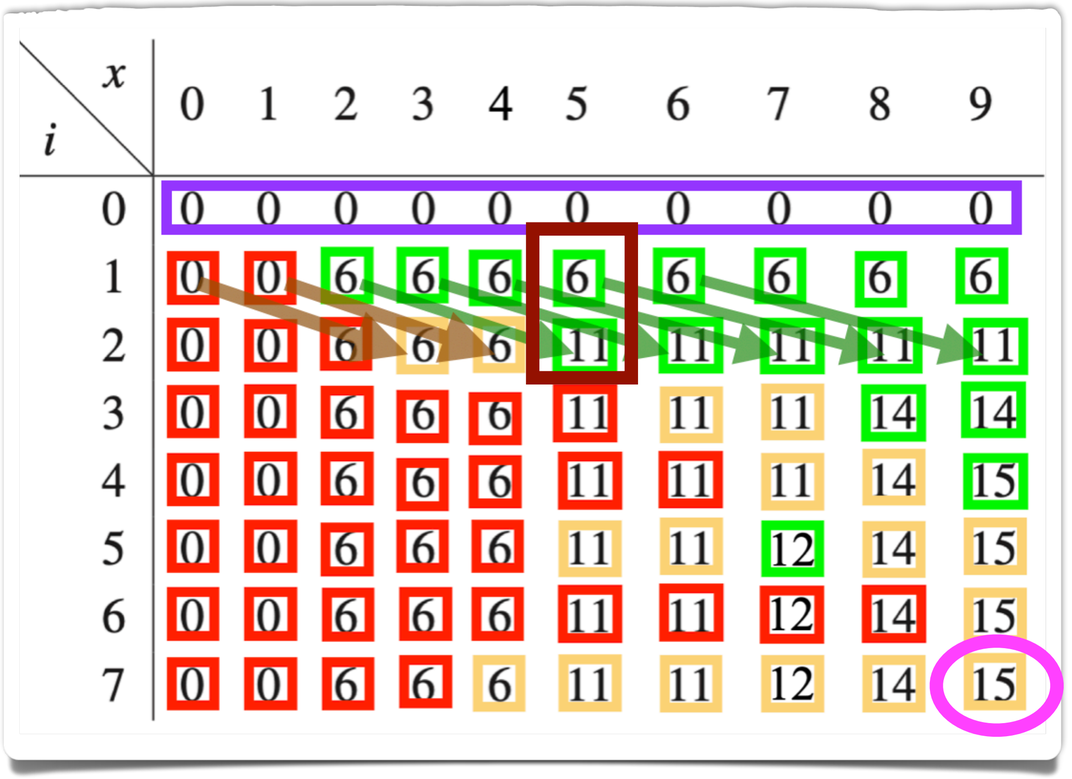


i \ x	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6
2	0	0	6	6	6	11	11	11	11	11
3	0	0	6	6	6	11	11	11	14	14
4	0	0	6	6	6	11	11	11	14	15
5	0	0	6	6	6	11	11	12	14	15
6	0	0	6	6	6	11	11	12	14	15
7	0	0	6	6	6	11	11	12	14	15

Sparsamer!

Ideen:

- Nur die jeweils letzte Zeile wird benötigt
- Updates von rechts nach links, damit nur erledigte Werte überschrieben werden



i \ x	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6
2	0	0	6	6	6	11	11	11	11	11
3	0	0	6	6	6	11	11	11	14	14
4	0	0	6	6	6	11	11	11	14	15
5	0	0	6	6	6	11	11	12	14	15
6	0	0	6	6	6	11	11	12	14	15
7	0	0	6	6	6	11	11	12	14	15

Algorithmus (sparsamer)

Algorithmus 2.7 Dynamic Programming für MAXIMUM KNAPSACK - sparsamer

Eingabe: $z_1, ..., z_n, Z, p_1, ...p_n$

Ausgabe: Funktion $P : \{1, ..., Z\} \to \mathbb{R};\ x \mapsto P(x)$

$$\text{mit } P(x) = \max \sum_{j=1}^{n} p_j y_j \text{ mit } \sum_{j=1}^{n} z_j y_j \leq x, \text{ für } y_j \in \{0, 1\}$$

```
1: for (x = 0) to Z do
2:     P(x) := 0
3: for (i = 1) to n do
4:     for (x = Z) downto z_i do
5:         if ((P(x − z_i) + p_i) > P(x)) then
6:             P(x) := P(x − z_i) + p_i
7: return p* := P(Z)
```

Korrektheit und Laufzeit

Satz 2.8 *Algorithmus 2.7 berechnet besten Lösungswert für das Rucksackproblem in einer Laufzeit O(nZ).*

Algorithmus 2.5 *(Dynamic Programming für Knapsack)*

Eingabe: $z_1, ..., z_n, Z, p_1, ...p_n$

Ausgabe: *Funktion* $P : \{1, ..., Z\} \times \{1, ..., n\} \to \mathbb{R};\ (x, i) \mapsto P(x, i)$

$$\text{mit } P(x,i) = max \sum_{i=1}^{i} p_j y_j \text{ mit } \sum_{j=1}^{i} z_j y_j \leq x, \text{ für } y_j \in \{0,1\}$$

```
1: for (x = 0) to Z do
2:     P(x,0) := 0
3: for (i = 1) to n do
4:     for (x = 0) to (z_i − 1) do
5:         P(x,i) := P(x,i−1)
6:     for (x = z_i) to Z do
7:         if ((P(x − z_i, i−1) + p_i) > P(x,i−1)) then
8:             P(x,i) := P(x − z_i, i−1) + p_i
9:         else
10:            P(x,i) := P(x,i−1)
```

$O(nZ)$

Algorithmus 2.7 *(Dynamic Programming für Knapsack - sparsamer)*

Eingabe: $z_1, ..., z_n, Z, p_1, ...p_n$

Ausgabe: *Funktion* $P : \{1, ..., Z\} \times \{1, ..., n\} \to \mathbb{R};\ (x, i) \mapsto P(x, i)$

$$\text{mit } P(x,i) = max \sum_{i=1}^{i} p_j y_j \text{ mit } \sum_{j=1}^{i} z_j y_j \leq x, \text{ für } y_j \in \{0,1\}$$

```
1: for (x = 0) to Z do
2:     P(x,0) := 0
3: for (i = 1) to n do
4:     for (x = Z) downto z_i do
5:         if ((P(x − z_i) + p_i) > P(x)) then
6:             P(x,i) := P(x − z_i) + p_i
7: return p* := P(Z)
```

$$P(x,i) = \begin{cases} P(x,i-1), \\ max\{P(x,i- \end{cases}$$

i \ x	0	1	2	3	3	5
0	1	0	0	0	0	0
1	1	0	0	0	0	0
2	1	0	0	0	+13	
3						

2.3 Andere Beispiele für Dynamic Programming

Algorithmen und Datenstrukturen 2
Sommer 2026

Prof. Dr. Sándor Fekete

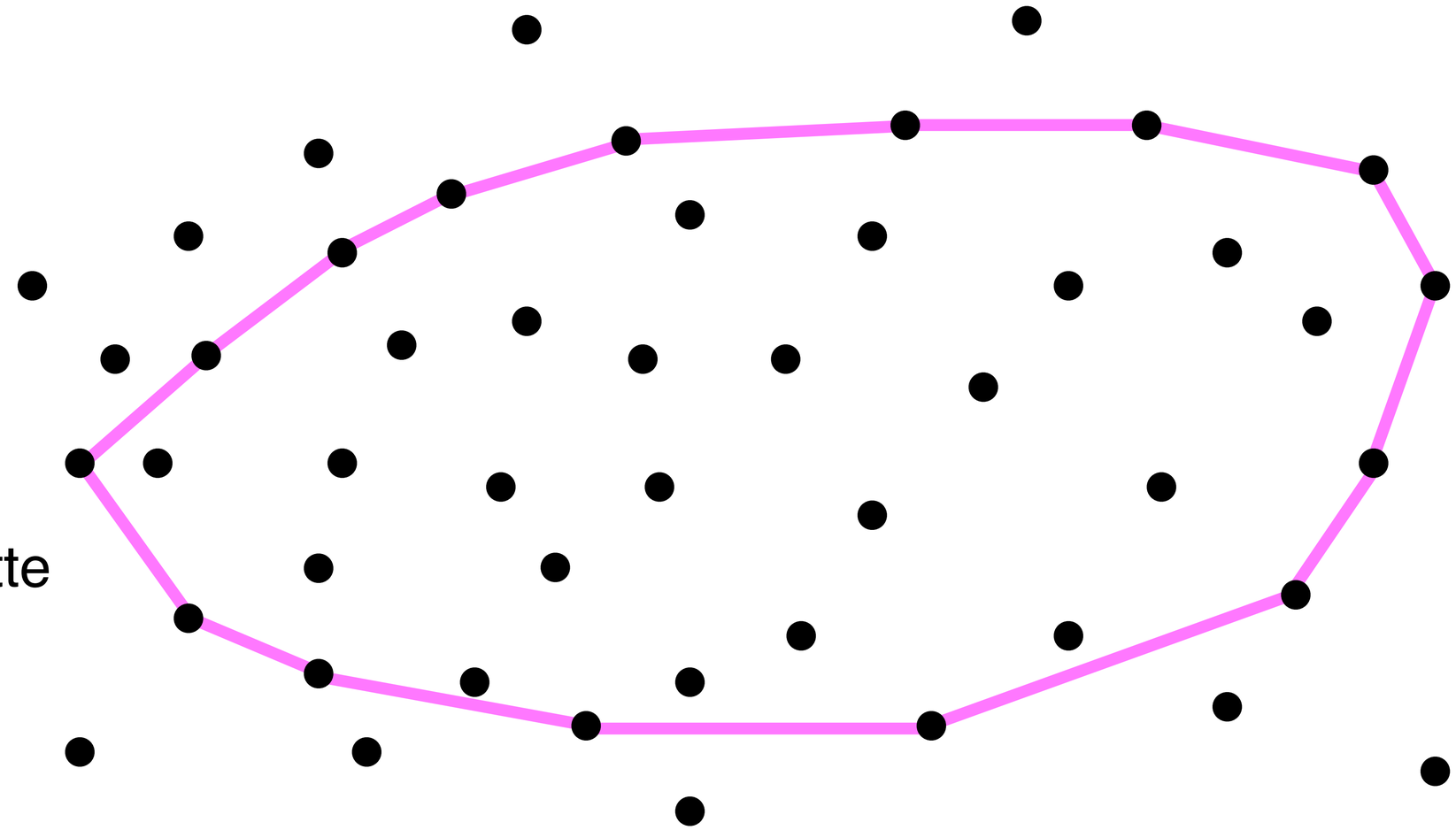
Längste konvexe Kette

Gegeben:

Punktmenge

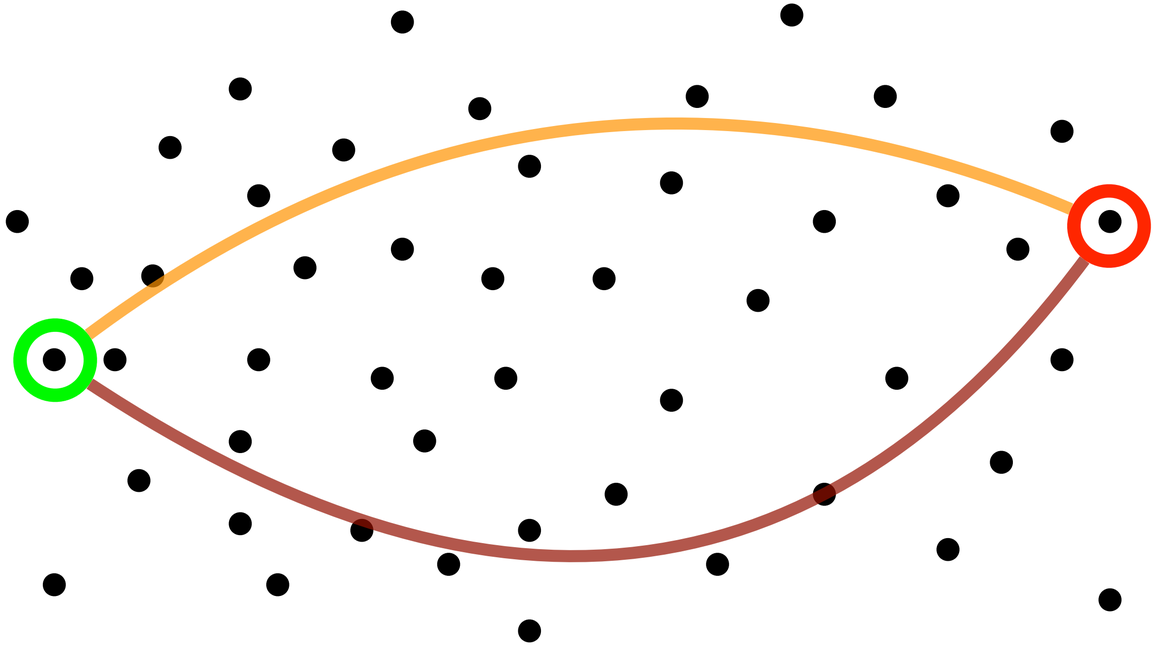
Gesucht:

Längste konvexe Kette



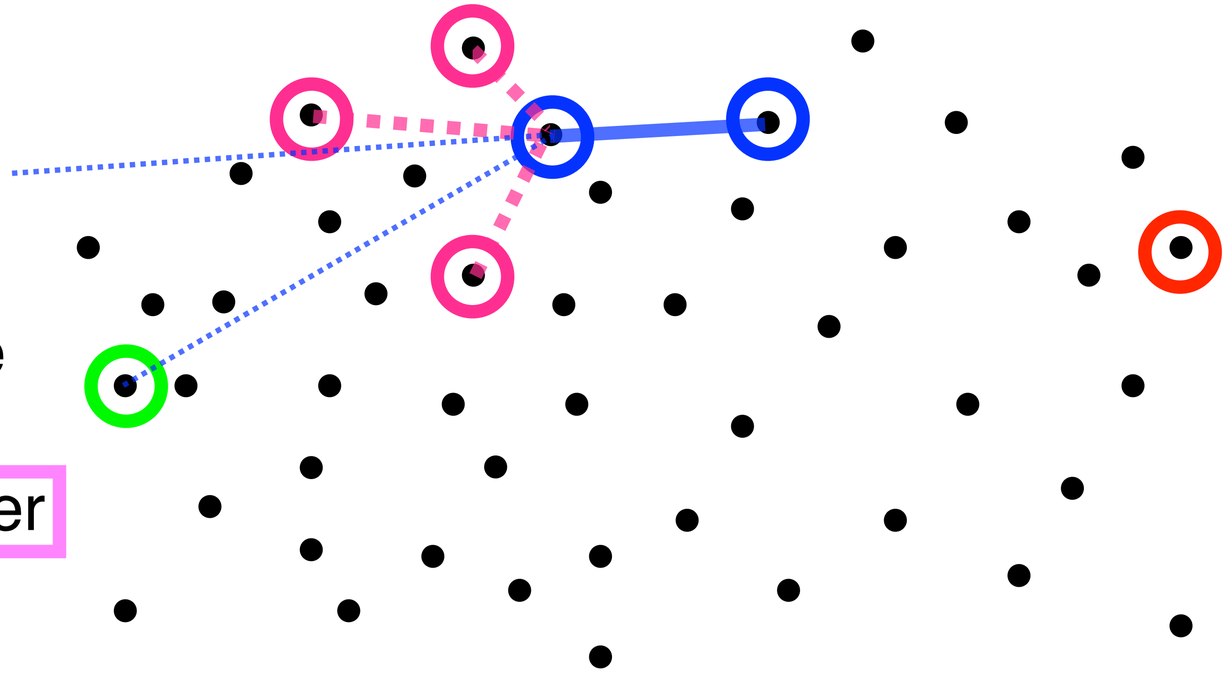
Idee

1. Enumeriere Startpunkt
   1.1 Enumeriere Endpunkt
      1.1.1. Betrachte obere und untere Kette



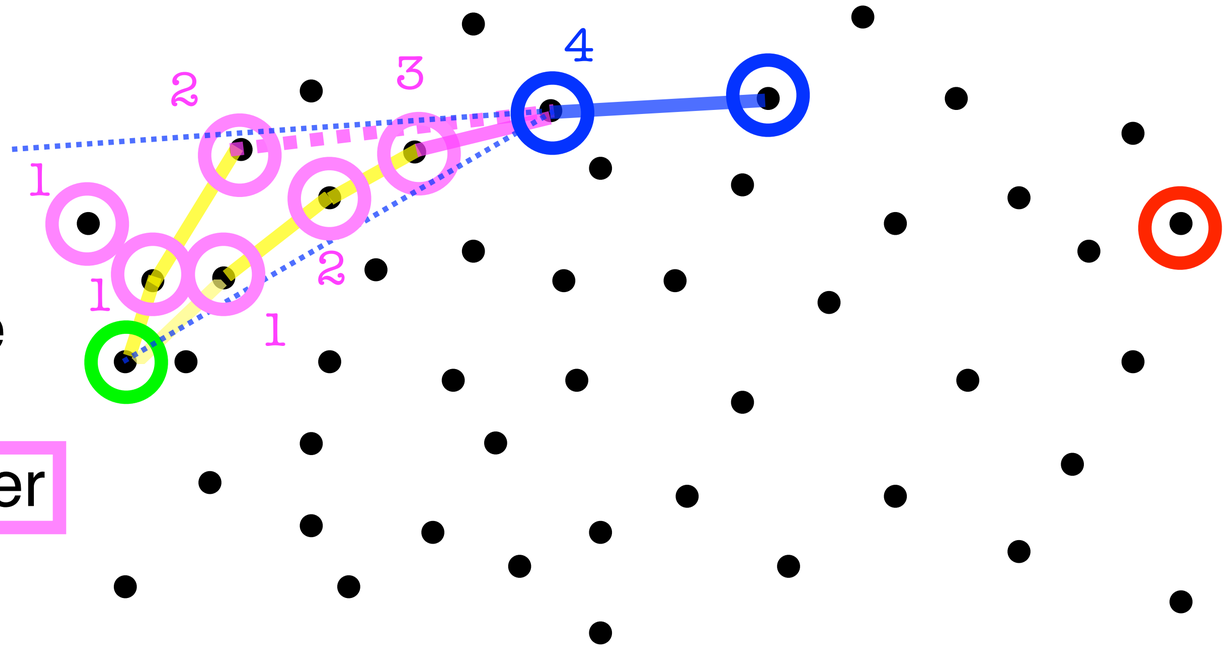
Idee

1. Enumeriere Startpunkt
   1.1 Enumeriere Endpunkt
      1.1.1. Betrachte obere und untere Kette
         1.1.1.1 Für jedes Punktepaar:
            1.1.1.1.1. Ermittle besten Vorgänger



Idee

1. Enumeriere Startpunkt
   1.1 Enumeriere Endpunkt
      1.1.1.Betrachte obere und untere Kette
         1.1.1.1 Für jedes Punktepaar:
            1.1.1.1.1.Ermittle besten Vorgänger



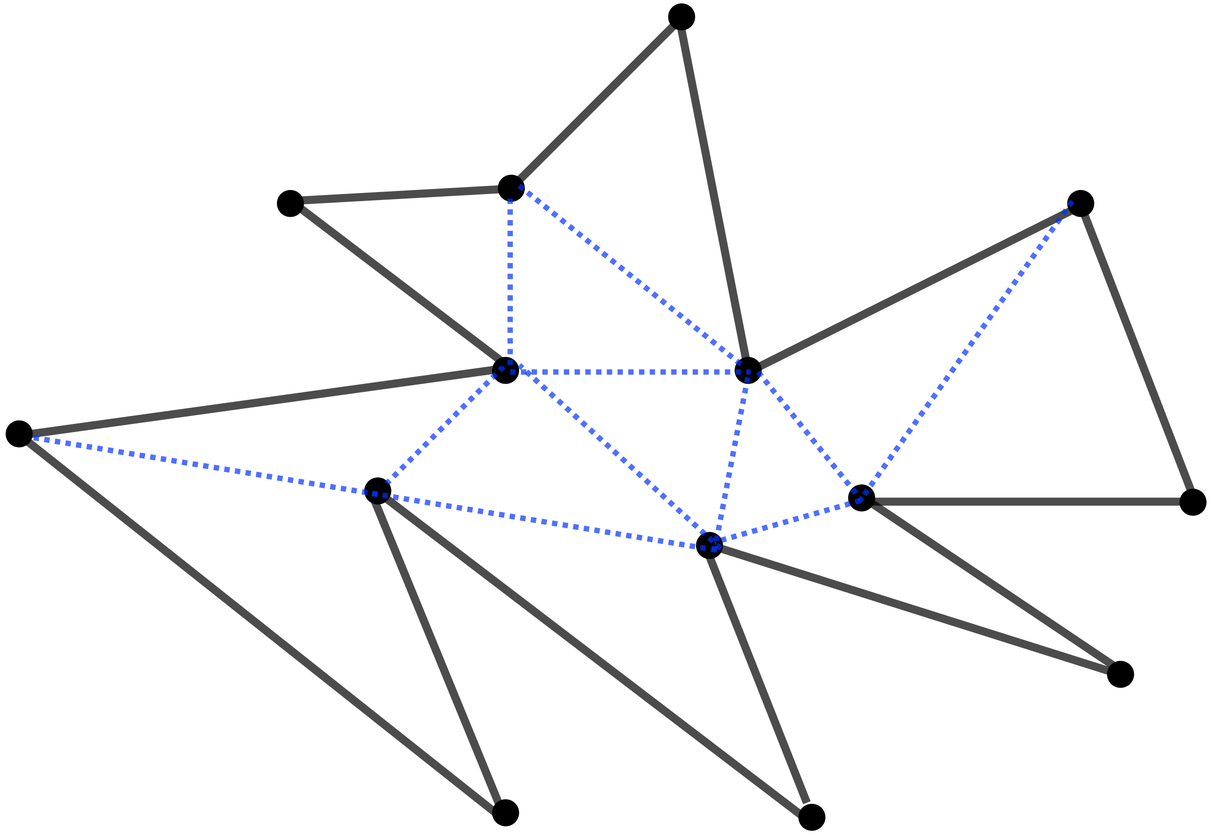
Optimale Triangulation

Gegeben:

„Einfaches“ Polygon
(mit zusammenhängendem Rand)

Gesucht:

Optimale Unterteilung in Dreiecke
durch „Diagonalen“



Optimale Triangulation

Gegeben:

„Einfaches“ Polygon
(mit zusammenhängendem Rand)

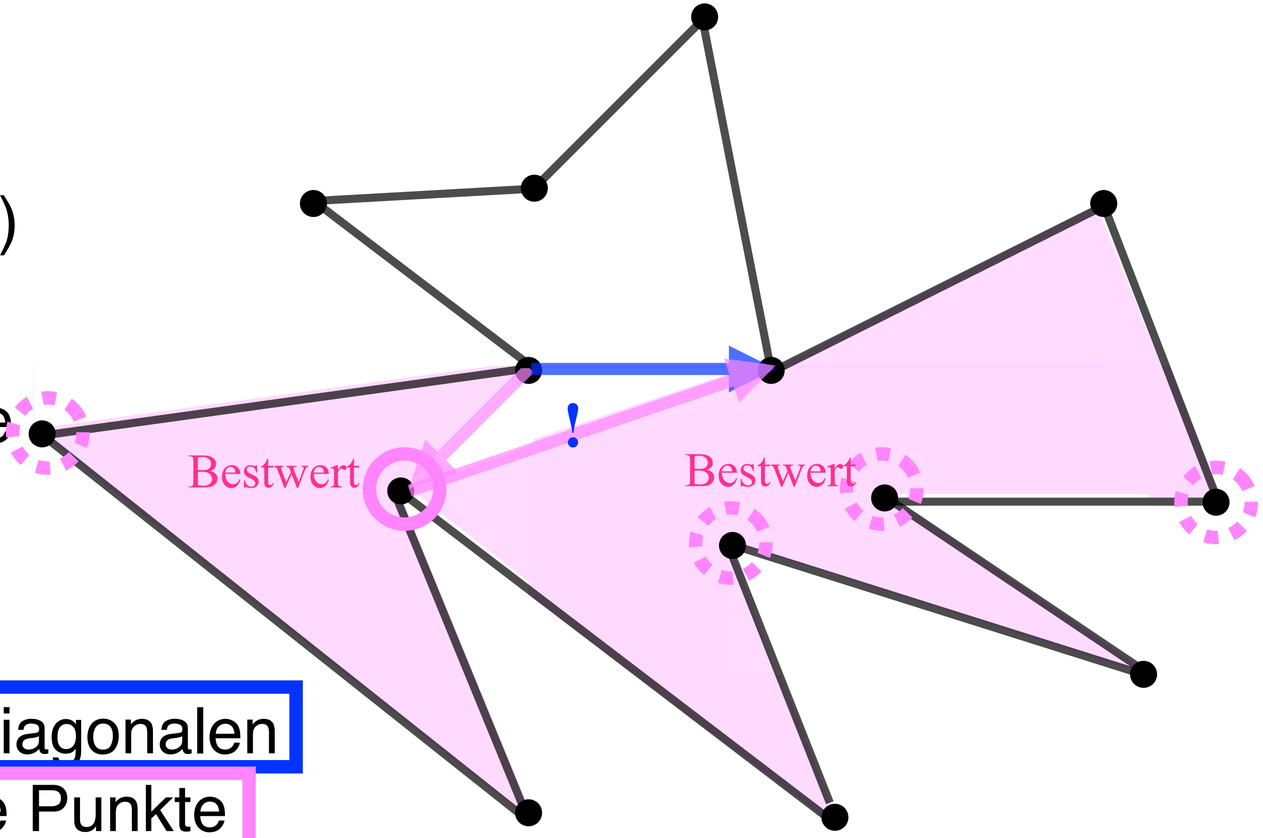
Gesucht:

Optimale Unterteilung in Dreiecke
durch „Diagonalen“

Idee:

1.Berechne für alle (gerichtete) Diagonalen

1.1.Optimiere über möglich dritte Punkte



Andere geometrische Probleme

CCCG 2014, Halifax, Nova Scotia, August 11–13, 2014

On the Chromatic Art Gallery Problem

Sándor P. Fekete* Stephan Friedrichs* Michael Hemmer* Joseph B. M. Mitchell†
Christiane Schmidt*

Abstract

For a polygonal region P with n vertices, a *guard cover* S is a set of points in P, such that any point in P can be seen from a point in S. In a *colored guard cover*, every element in a guard cover is assigned a color, such that no two guards with the same color have overlapping visibility regions. The Chromatic Art Gallery Problem (CAGP) asks for the minimum number of colors for which a colored guard cover exists.

We discuss the CAGP for the case of only two colors. We show that it is already *NP*-hard to decide whether two colors suffice for covering a polygon with holes, even when arbitrary guard positions are allowed. For simple polygons with a discrete set of possible guard locations, we give a polynomial-time algorithm for deciding whether a two-colorable guard set exists. This algo-

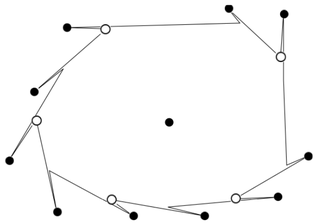


Figure 1: An example polygon with $n = 20$ vertices. A minimum-cardinality guard cover with $n/4$ guards (shown in white) requires $n/4$ colors, while a minimum-color guard cover (shown in black) has $n/2 + 1$ guards and requires only 3 colors.

Theorem 4.1 *For a simple polygon P with n vertices and ℓ discrete guard locations L, it can be decided in polynomial time whether there is a 2-colorable guard set.*

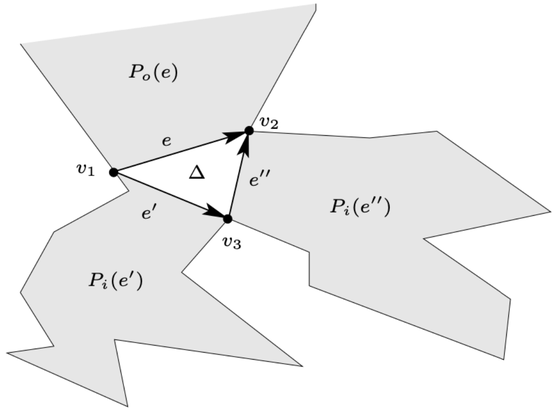
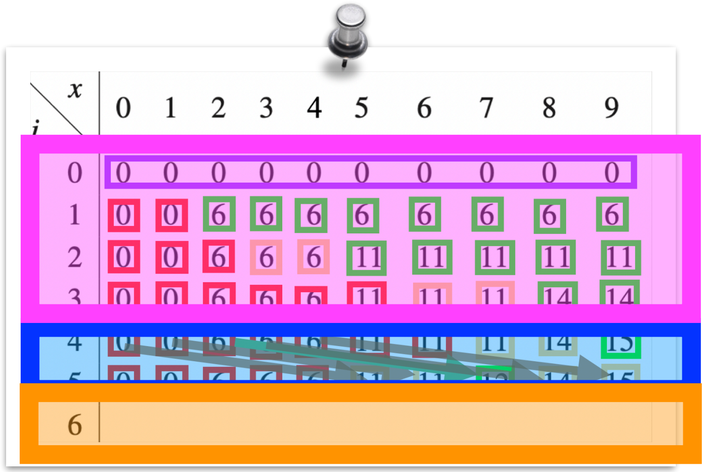


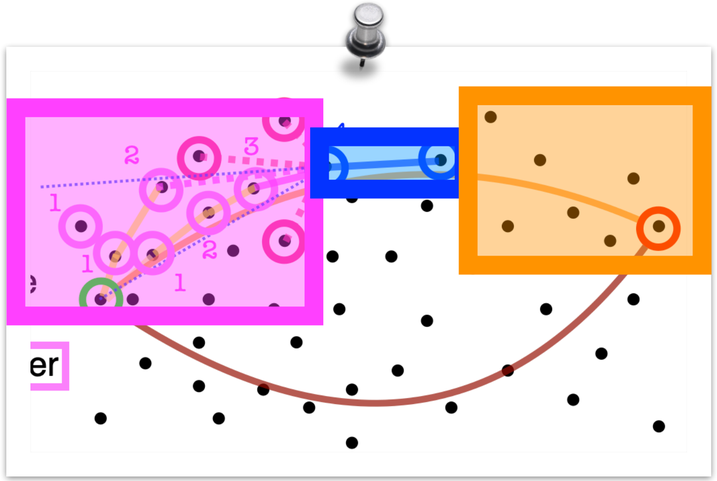
Figure 7: Solving a subproblem (e, Γ) by combining solutions for subproblems (e', Γ') and (e'', Γ'').

Dynamic Programming: Grundstruktur

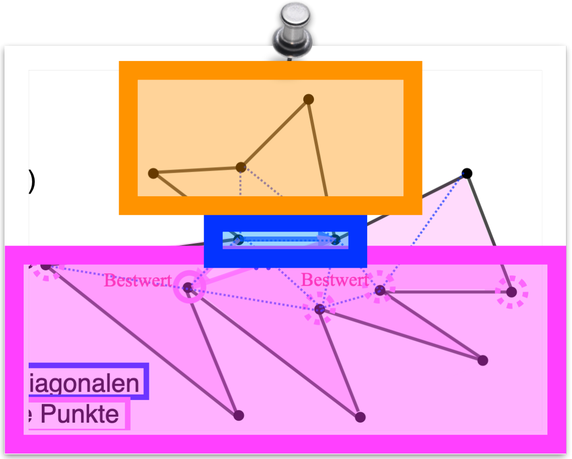
- Nicht von vorn lösen
- Zwischenergebnisse merken
- Man kann die Vorgeschichte beim Weiterrechnen von der Zukunft trennen und deshalb den Ubergang durch Enumerieren lösen.



Objekt i



Kante i



Kante i

$$\sum_{i=1}^{n} x_i z_i \leq Z$$

$$x_i \in [0,1]$$

3 Branch-and-Bound

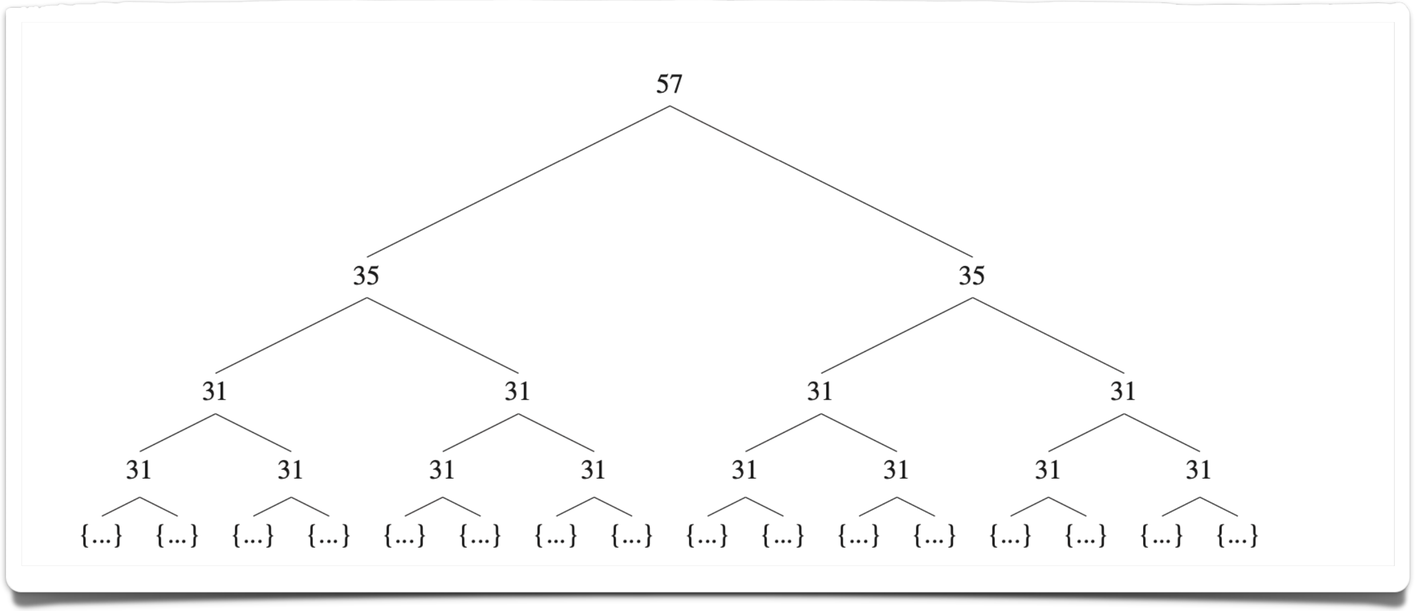
Algorithmen und Datenstrukturen 2
Sommer 2026

Prof. Dr. Sándor Fekete

3.1 Motivation

Enumerationsprinzip

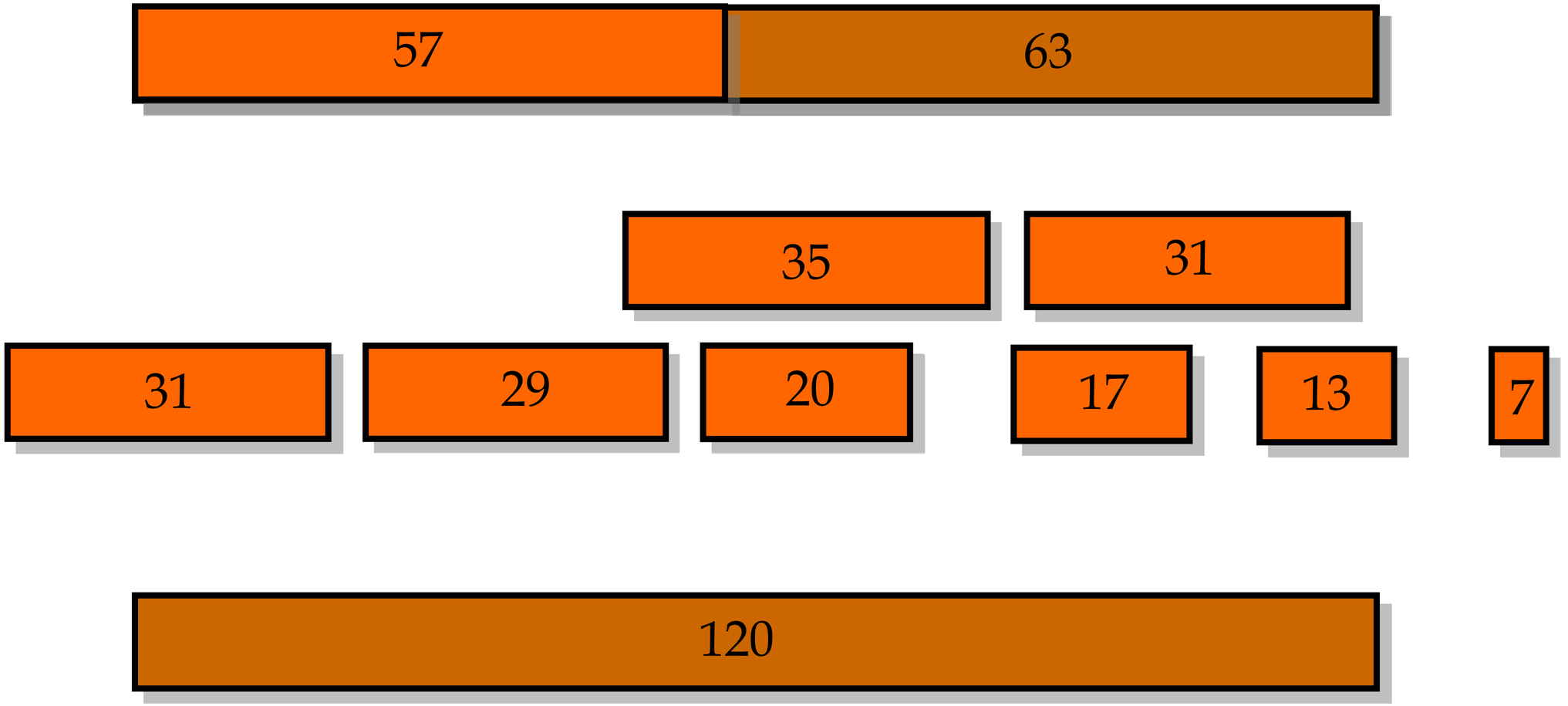
Dynamic Programming



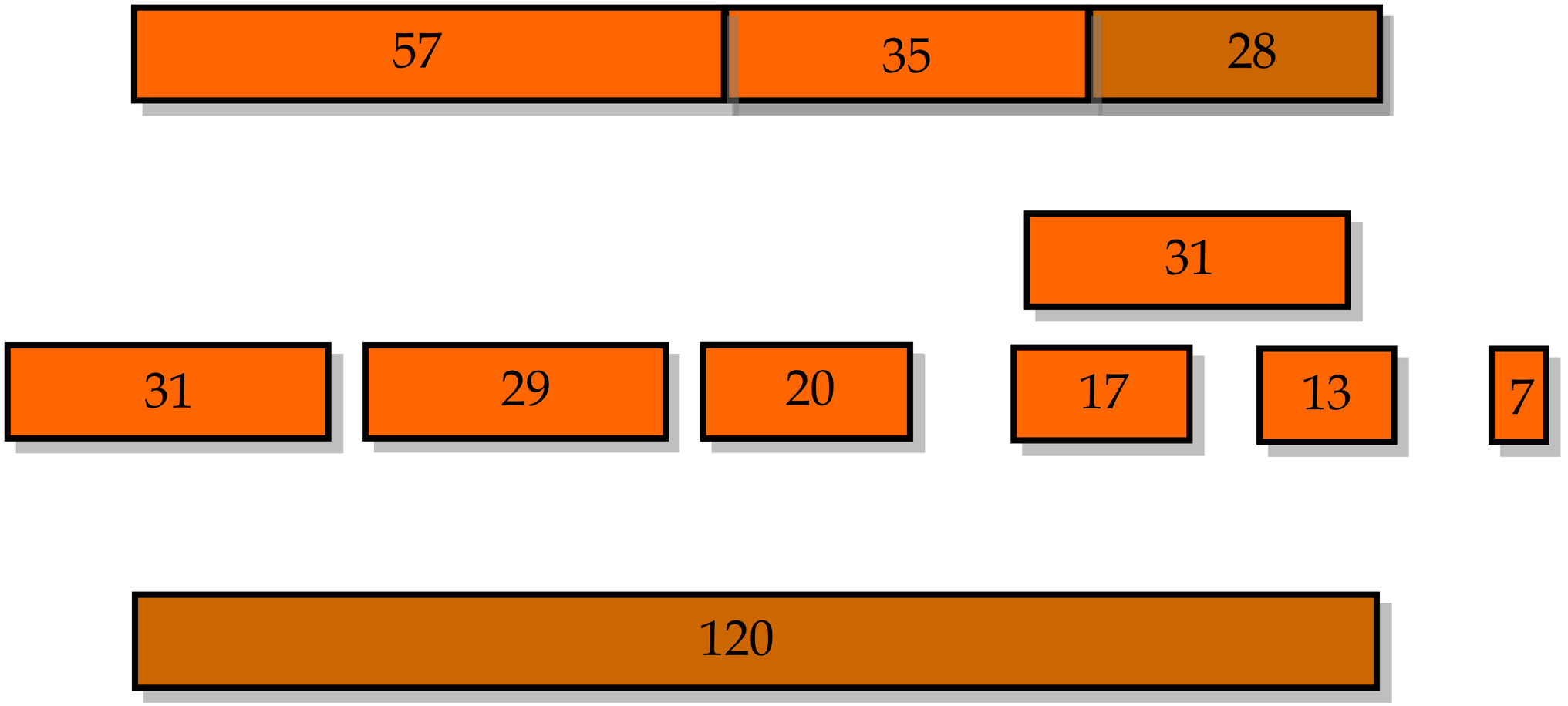
Branch and Bound

- Exponentiell viele Fälle!
- Wie geht das systematisch?
- Wo kann man Arbeit sparen?

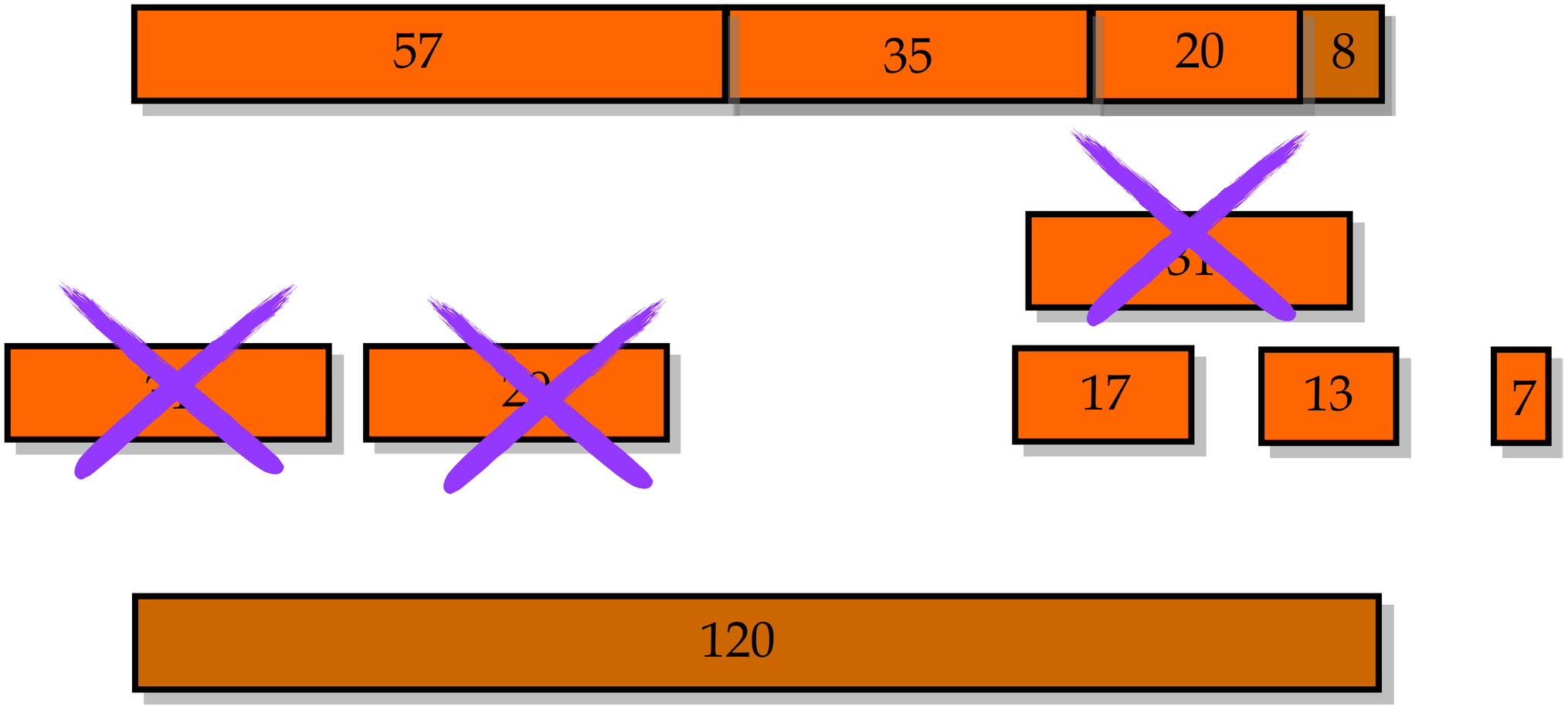
Zur Erinnerung: Subset Sum



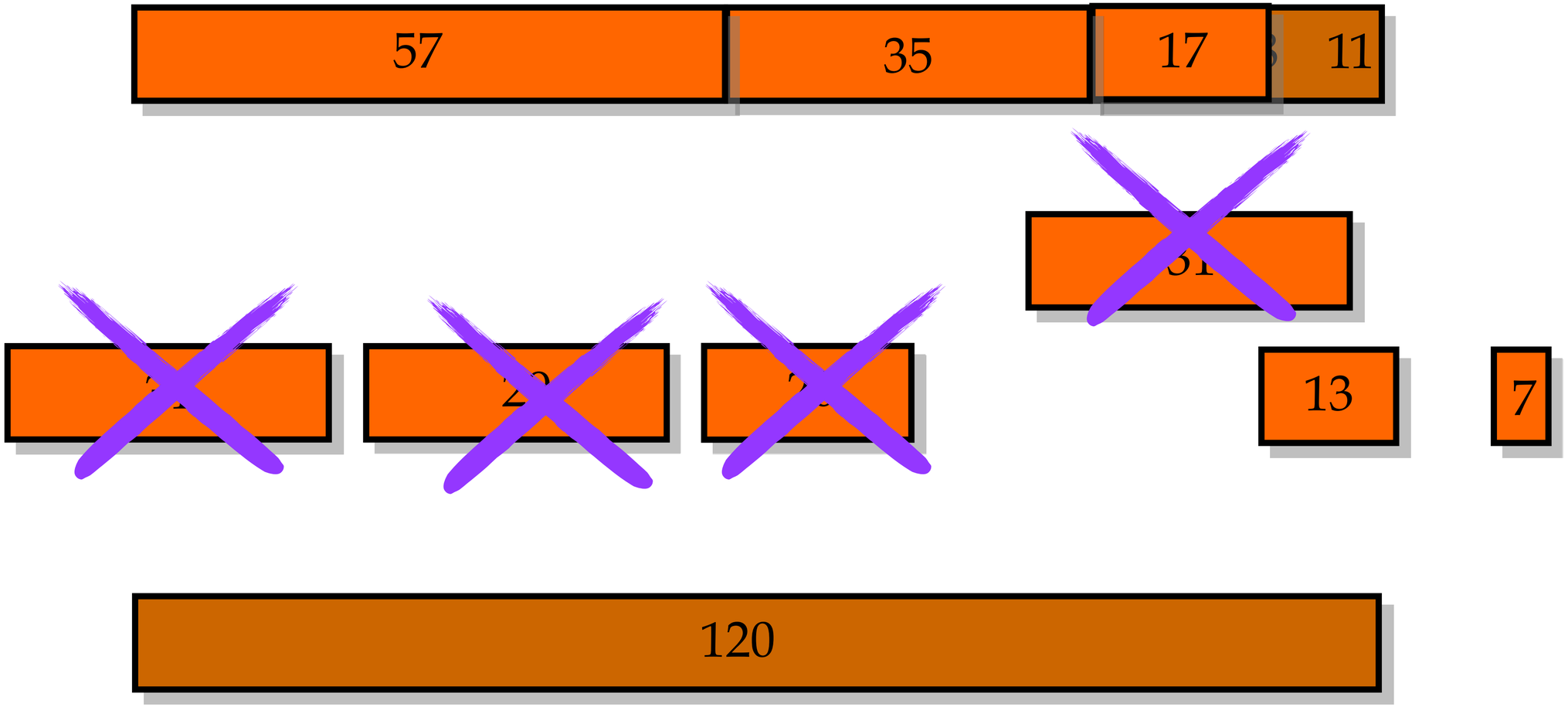
Zur Erinnerung: Subset Sum



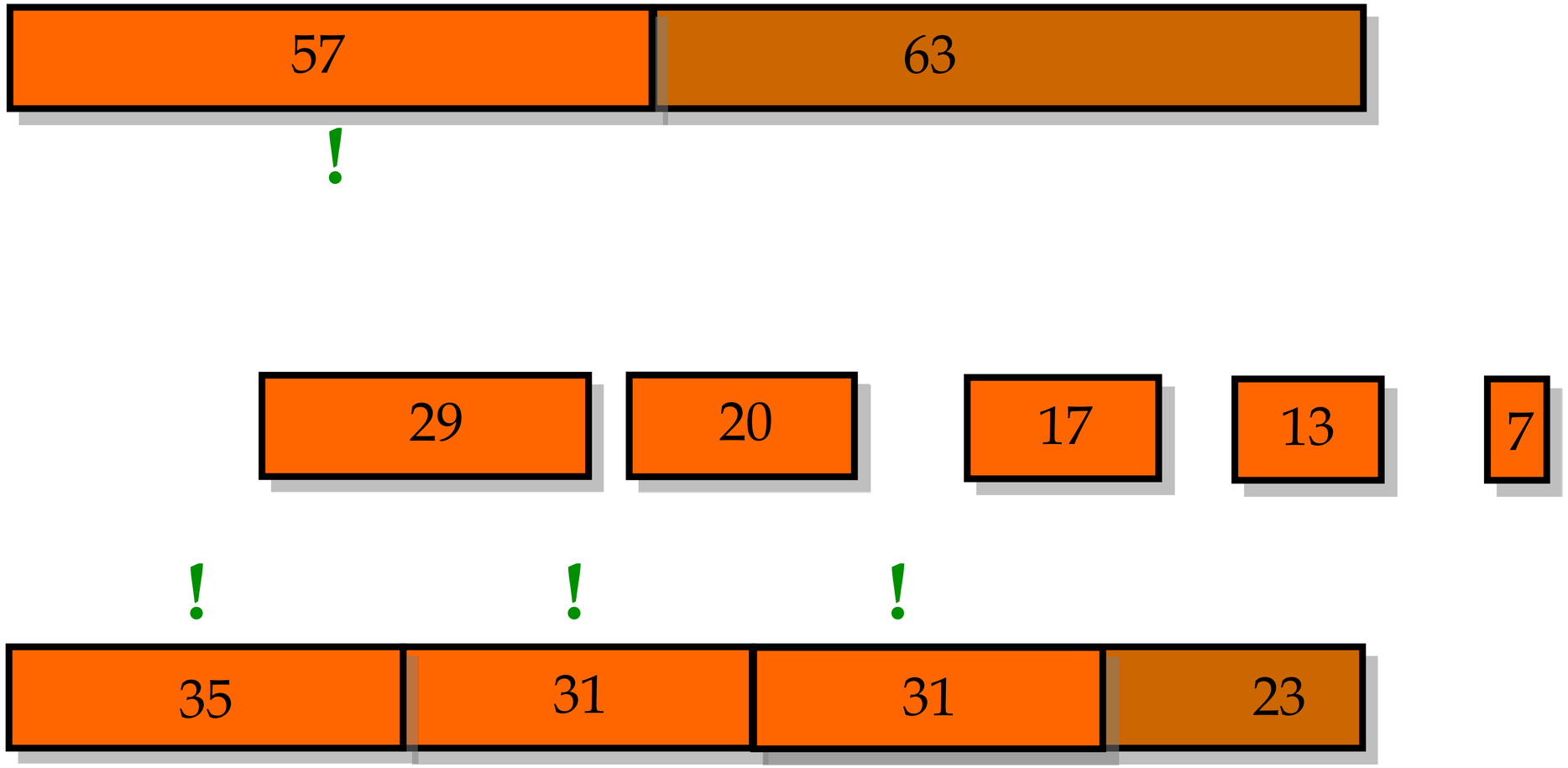
Zur Erinnerung: Subset Sum



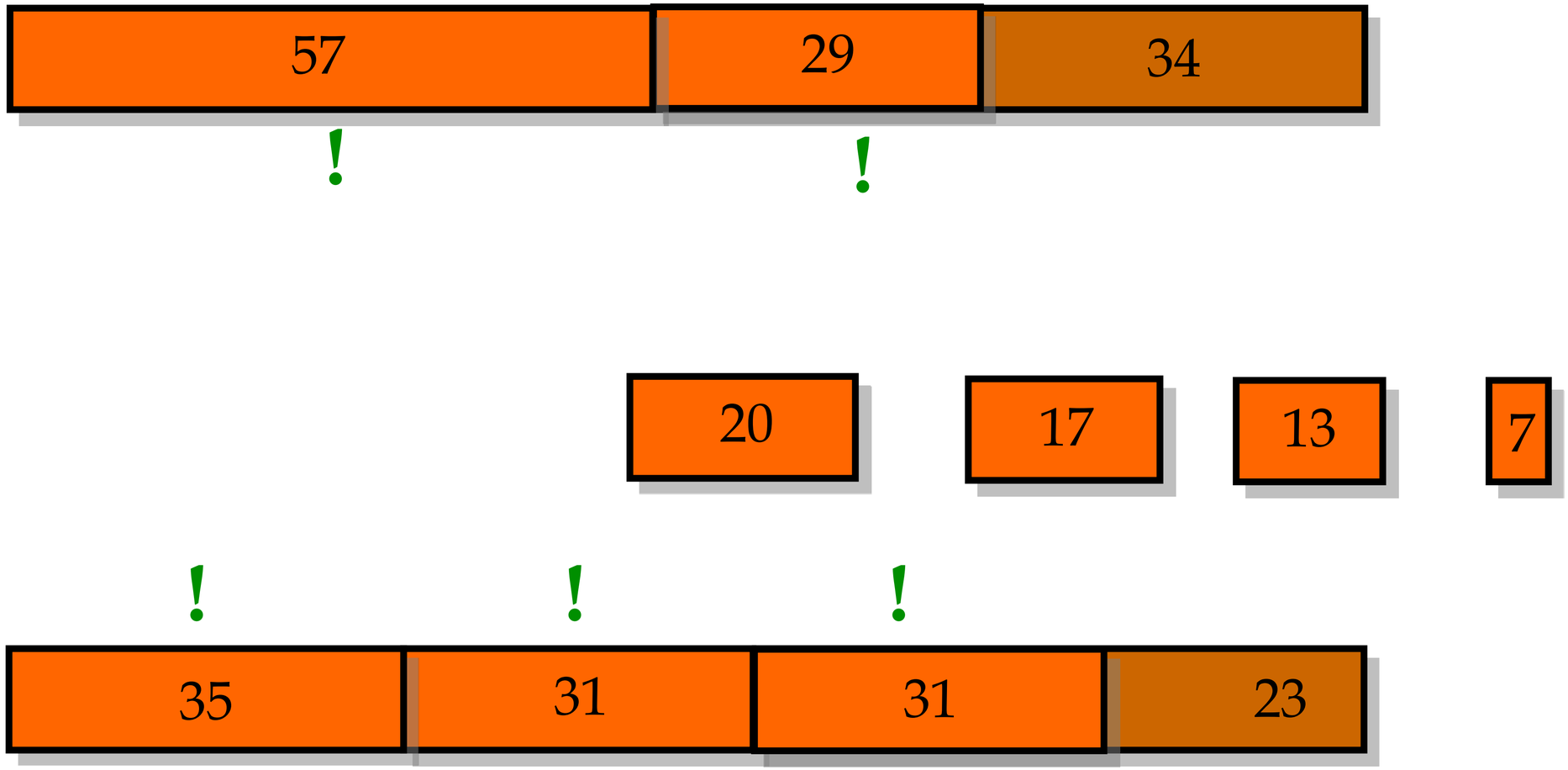
Zur Erinnerung: Subset Sum



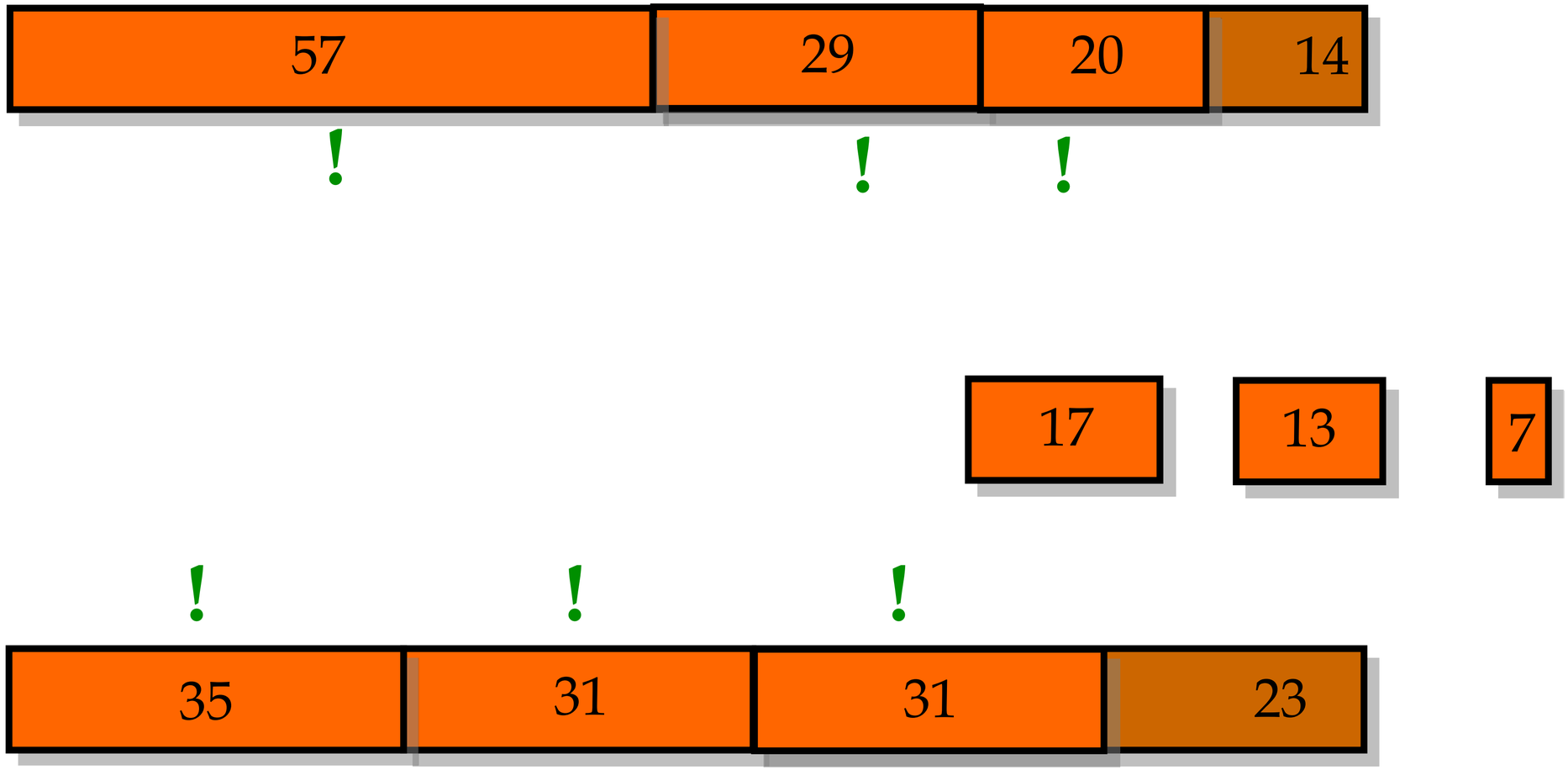
Zur Erinnerung: Subset Sum



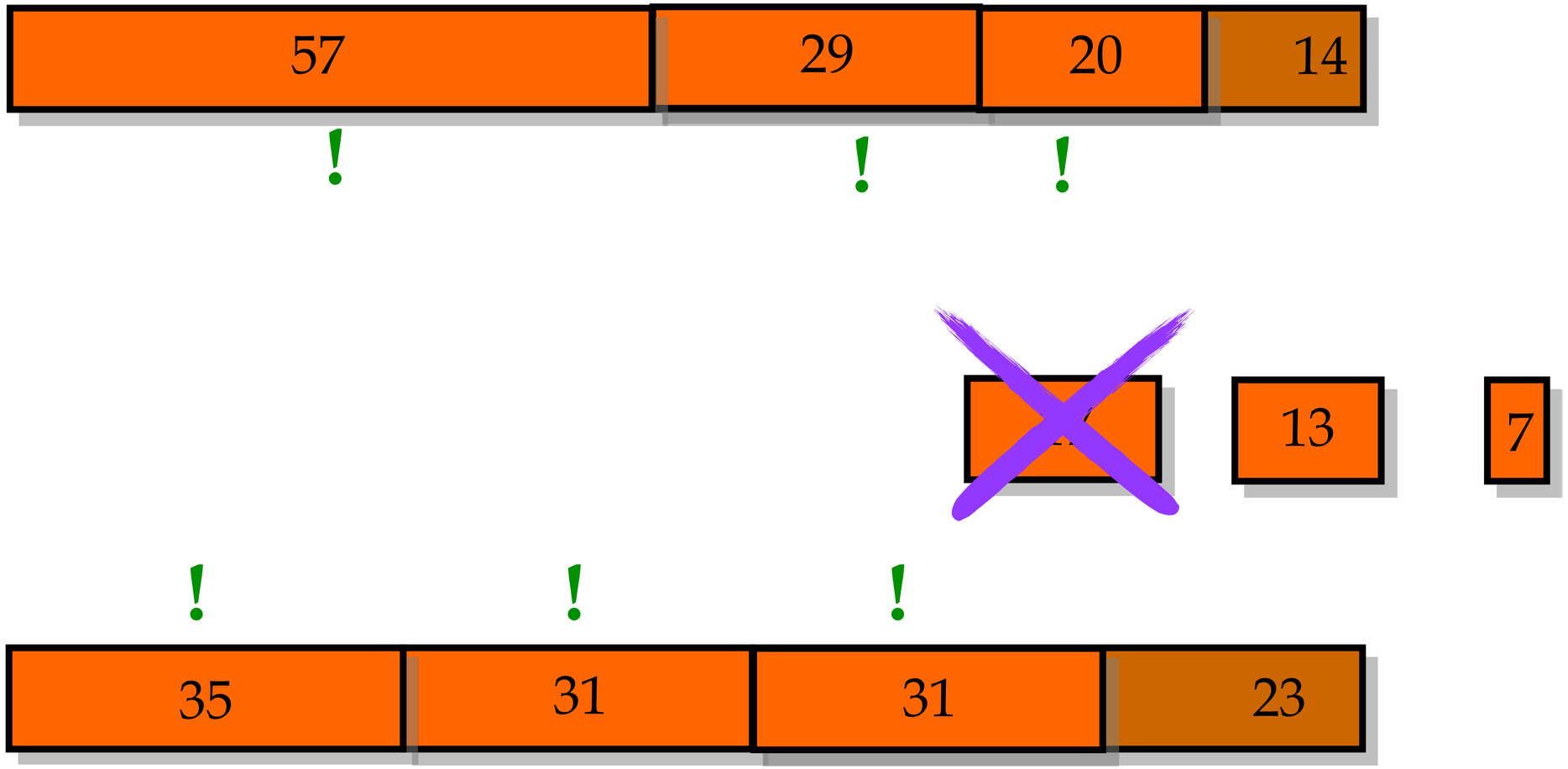
Zur Erinnerung: Subset Sum



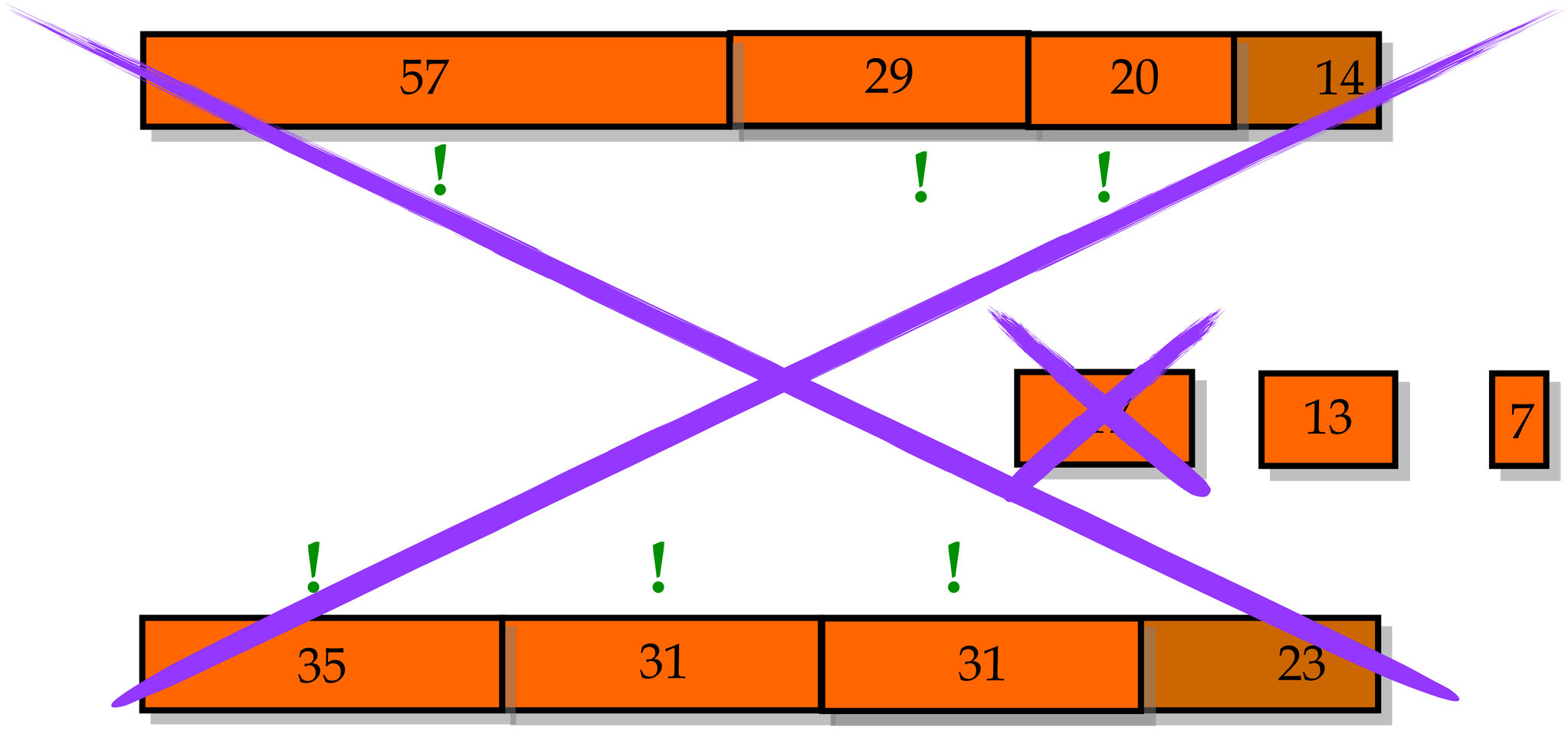
Zur Erinnerung: Subset Sum



Zur Erinnerung: Subset Sum



Zur Erinnerung: Subset Sum



Dynamic Programming

Satz 2.6 *Algorithmus 2.5 berechnet besten Lösungswert für das Rucksackproblem in einer Laufzeit $O(nZ)$.*

Algorithmus 2.5 *(Dynamic Programming für Knapsack)*

Eingabe: $z_1, ..., z_n, Z, p_1, ...p_n$

Ausgabe: Funktion $P : \{1, ..., Z\} \times \{1, ..., n\} \to \mathbb{R};\ (x, i) \mapsto P(x, i)$

$$\text{mit } P(x,i) = max \sum_{i=1}^{i} p_j y_j \text{ mit } \sum_{j=1}^{i} z_j y_j \leq x, \text{ für } y_j \in \{0,1\}$$

```
1: for (x = 0) to Z do
2:     P(x,0) := 0
3: for (i = 1) to n do
4:     for (x = 0) to (z_i − 1) do
5:         P(x,i) := P(x,i−1)
6:     for (x = z_i) to Z do
7:         if ((P(x − z_i, i−1) + p_i) > P(x,i−1)) then
8:             P(x,i) := P(x − z_i, i−1) + p_i
9:         else
0:             P(x,i) := P(x,i−1)
```

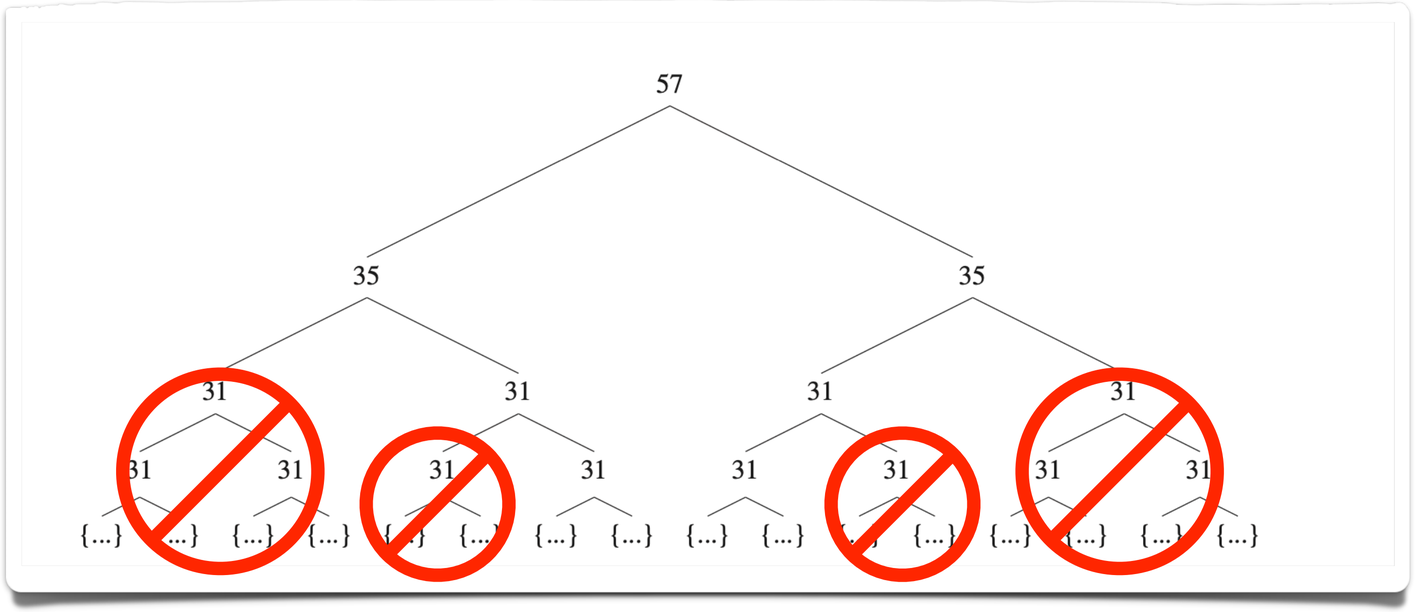
$O(nZ)$

i \ x	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6
2	0	0	6	6	6	11	11	11	11	11
3	0	0	6	6	6	11	11	11	14	14
4	0	0	6	6	6	11	11	11	14	15
5	0	0	6	6	6	11	11	12	14	15
6	0	0	6	6	6	11	11	12	14	15
7	0	0	6	6	6	11	11	12	14	15

Induktion!

1. Objekt i ist zu teuer für Kostenschranke x, falls $z_i > x$
2. Objekt i ist nicht zu teuer, bringt aber keine Verbesserung
3. Objekt i ist nicht zu teuer und bringt eine Verbesserung

Branch-and-Bound



Branch and Bound

- Beschneide Enumerationsbaum
- Verfolge Mindest- und Maximalwerte

Historie

1960

ECONOMETRICA

VOLUME 28 July, 1960 NUMBER 3

AN AUTOMATIC METHOD OF SOLVING DISCRETE PROGRAMMING PROBLEMS

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

In the classical linear programming problem the behaviour of continuous, nonnegative variables subject to a system of linear inequalities is investigated. One possible generalization of this problem is to relax the continuity condition on the variables. This paper presents a simple numerical algorithm for the solution of programming problems in which some or all of the variables can take only discrete values. The algorithm requires no special techniques beyond those used in ordinary linear programming, and lends itself to automatic computing. Its use is illustrated on two numerical examples.

1. INTRODUCTION

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.

In some cases, notably those which can be expressed as transport problems, the linear programming solution will itself yield discrete values of the variables. In other cases the percentage change in the maximand[2] from common sense rounding of the variables is sufficiently small to be neglected. But there remain many problems where the discrete variable constraints are significant and costly.

Until recently there was no general automatic routine for solving such problems, as opposed to procedures for proving the optimality of conjectured solutions, and the work reported here is intended to fill the gap. About the time of its completion an alternative method was proposed by Gomory [**5**] and subsequently extended by Beale [**1**]. Gomory's method

[1] Or more generally, any value within a bounded interval.

[2] We shall speak throughout of maximisation, but of course an exactly analogous argument applies to minimisation.

Historie

1. INTRODUCTION

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.

Historie

1960

ECONOMETRICA

VOLUME 28 July, 1960 NUMBER 3

AN AUTOMATIC METHOD OF SOLVING DISCRETE PROGRAMMING PROBLEMS

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

In the classical linear programming problem the behaviour of continuous, nonnegative variables subject to a system of linear inequalities is investigated. One possible generalization of this problem is to relax the continuity condition on the variables. This paper presents a simple numerical algorithm for the solution of programming problems in which some or all of the variables can take only discrete values. The algorithm requires no special techniques beyond those used in ordinary linear programming, and lends itself to automatic computing. Its use is illustrated on two numerical examples.

1. INTRODUCTION

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.

In some cases, notably those which can be expressed as transport problems, the linear programming solution will itself yield discrete values of the variables. In other cases the percentage change in the maximand[2] from common sense rounding of the variables is sufficiently small to be neglected. But there remain many problems where the discrete variable constraints are significant and costly.

Until recently there was no general automatic routine for solving such problems, as opposed to procedures for proving the optimality of conjectured solutions, and the work reported here is intended to fill the gap. About the time of its completion an alternative method was proposed by Gomory [**5**] and subsequently extended by Beale [**1**]. Gomory's method

[1] Or more generally, any value within a bounded interval.

[2] We shall speak throughout of maximisation, but of course an exactly analogous argument applies to minimisation.

Historie

1960

ECONOMETRICA

VOLUME 28 July, 1960 NUMBER 3

AN AUTOMATIC METHOD OF SOLVING DISCRETE PROGRAMMING PROBLEMS

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

In the classical linear programming problem the behaviour of continuous, nonnegative variables subject to a system of linear inequalities is investigated. One possible generalization of this problem is to relax the continuity condition on the variables. This paper presents a simple numerical algorithm for the solution of programming problems in which some or all of the variables can take only discrete values. The algorithm requires no special techniques beyond those used in ordinary linear programming, and lends itself to automatic computing. Its use is illustrated on two numerical examples.

1. INTRODUCTION

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.

In some cases, notably those which can be expressed as transport problems, the linear programming solution will itself yield discrete values of the variables. In other cases the percentage change in the maximand[2] from common sense rounding of the variables is sufficiently small to be neglected. But there remain many problems where the discrete variable constraints are significant and costly.

Until recently there was no general automatic routine for solving such problems, as opposed to procedures for proving the optimality of conjectured solutions, and the work reported here is intended to fill the gap. About the time of its completion an alternative method was proposed by Gomory [**5**] and subsequently extended by Beale [**1**]. Gomory's method

[1] Or more generally, any value within a bounded interval.

[2] We shall speak throughout of maximisation, but of course an exactly analogous argument applies to minimisation.

497

Historie

Ailsa H. Land

1960

ECONOMETRICA

VOLUME 28 July, 1960 NUMBER 3

AN AUTOMATIC METHOD OF SOLVING DISCRETE PROGRAMMING PROBLEMS

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

In the classical linear programming problem the behaviour of continuous, nonnegative variables subject to a system of linear inequalities is investigated. One possible generalization of this problem is to relax the continuity condition on the variables. This paper presents a simple numerical algorithm for the solution of programming problems in which some or all of the variables can take only discrete values. The algorithm requires no special techniques beyond those used in ordinary linear programming, and lends itself to automatic computing. Its use is illustrated on two numerical examples.

1. INTRODUCTION

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.

In some cases, notably those which can be expressed as transport problems, the linear programming solution will itself yield discrete values of the variables. In other cases the percentage change in the maximand[2] from common sense rounding of the variables is sufficiently small to be neglected. But there remain many problems where the discrete variable constraints are significant and costly.

Until recently there was no general automatic routine for solving such problems, as opposed to procedures for proving the optimality of conjectured solutions, and the work reported here is intended to fill the gap. About the time of its completion an alternative method was proposed by Gomory [**5**] and subsequently extended by Beale [**1**]. Gomory's method

[1] Or more generally, any value within a bounded interval.

[2] We shall speak throughout of maximisation, but of course an exactly analogous argument applies to minimisation.

Historie

Ailsa H. Land

Alison Doig

1960

ECONOMETRICA

VOLUME 28 July, 1960 NUMBER 3

AN AUTOMATIC METHOD OF SOLVING DISCRETE PROGRAMMING PROBLEMS

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

In the classical linear programming problem the behaviour of continuous, nonnegative variables subject to a system of linear inequalities is investigated. One possible generalization of this problem is to relax the continuity condition on the variables. This paper presents a simple numerical algorithm for the solution of programming problems in which some or all of the variables can take only discrete values. The algorithm requires no special techniques beyond those used in ordinary linear programming, and lends itself to automatic computing. Its use is illustrated on two numerical examples.

1. INTRODUCTION

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.

In some cases, notably those which can be expressed as transport problems, the linear programming solution will itself yield discrete values of the variables. In other cases the percentage change in the maximand[2] from common sense rounding of the variables is sufficiently small to be neglected. But there remain many problems where the discrete variable constraints are significant and costly.

Until recently there was no general automatic routine for solving such problems, as opposed to procedures for proving the optimality of conjectured solutions, and the work reported here is intended to fill the gap. About the time of its completion an alternative method was proposed by Gomory [**5**] and subsequently extended by Beale [**1**]. Gomory's method

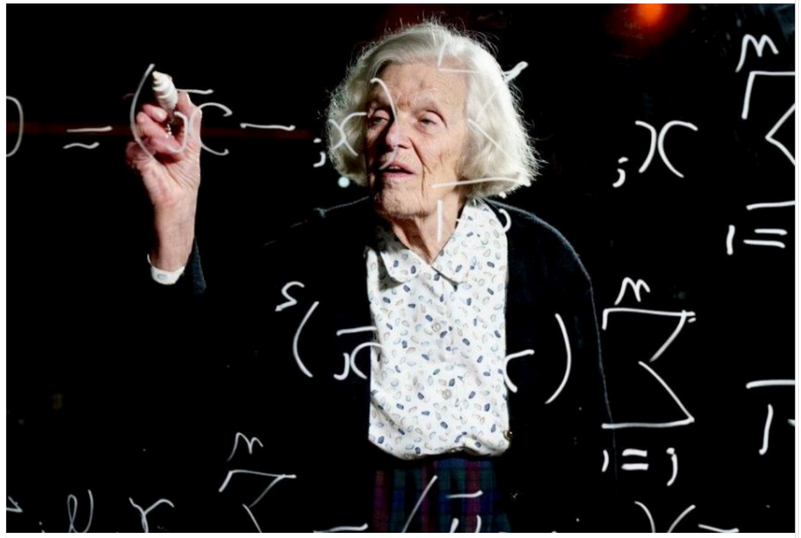
[1] Or more generally, any value within a bounded interval.

[2] We shall speak throughout of maximisation, but of course an exactly analogous argument applies to minimisation.

'I've always loved numbers': Meet the ground-breaking grandmother of Australian mathematics

7.30 / By Lauren Day

Posted 8 Oct 2018, updated 10 Oct 2018



Alison Grant Harcourt AO

In June 2019, Harcourt was made an Officer of the Order of Australia

Vielen Dank!

s.fekete@tu-bs.de