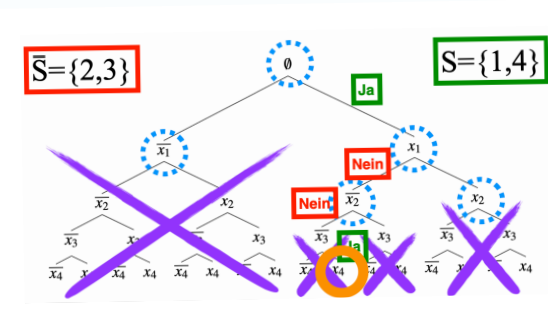




$$\sum_{i=1}^n x_i z_i \leq Z$$

$x_i \in [0, 1]$



3 Branch-and-Bound

Algorithmen und Datenstrukturen 2
Sommer 2024

Prof. Dr. Sándor Fekete

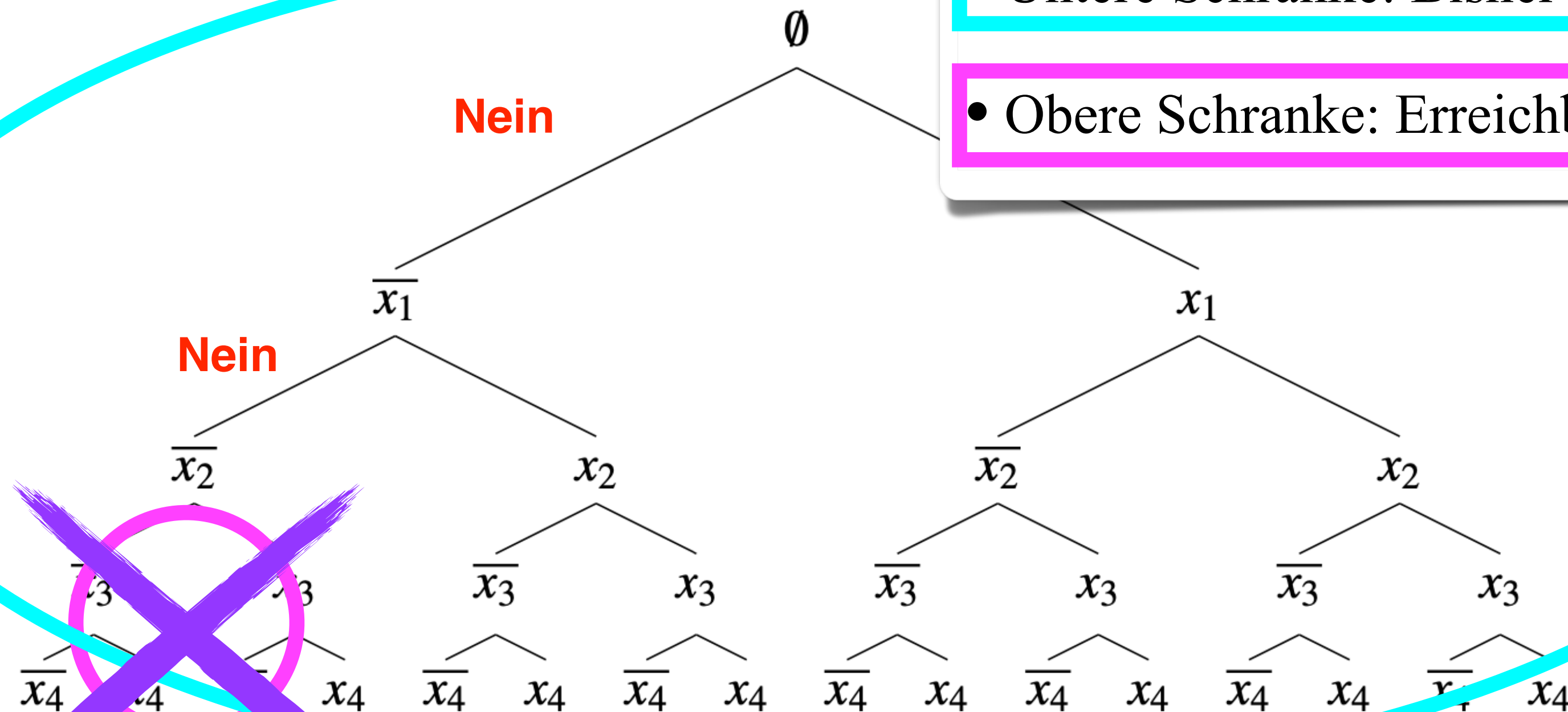
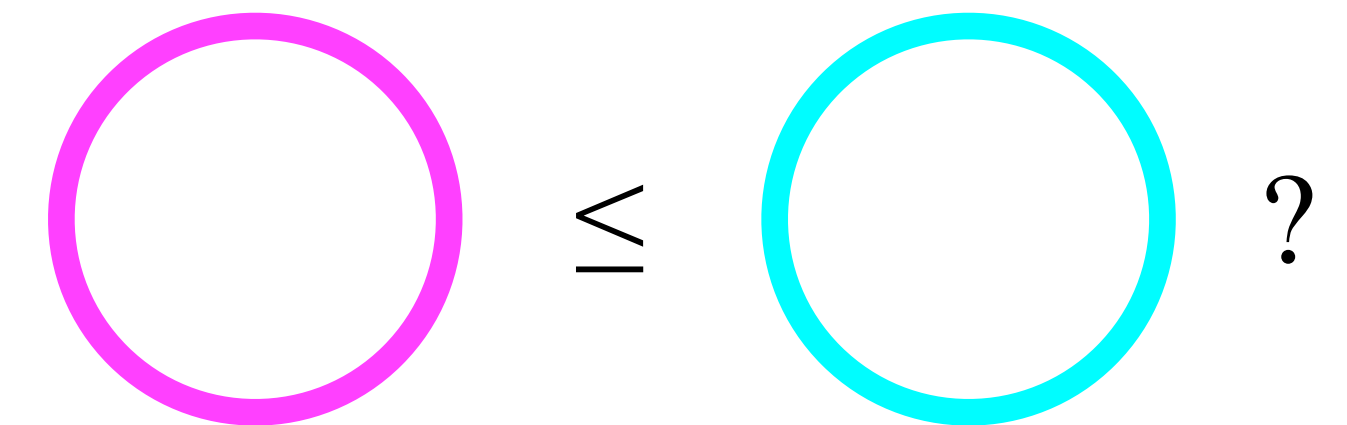
Ideen

Grundideen:

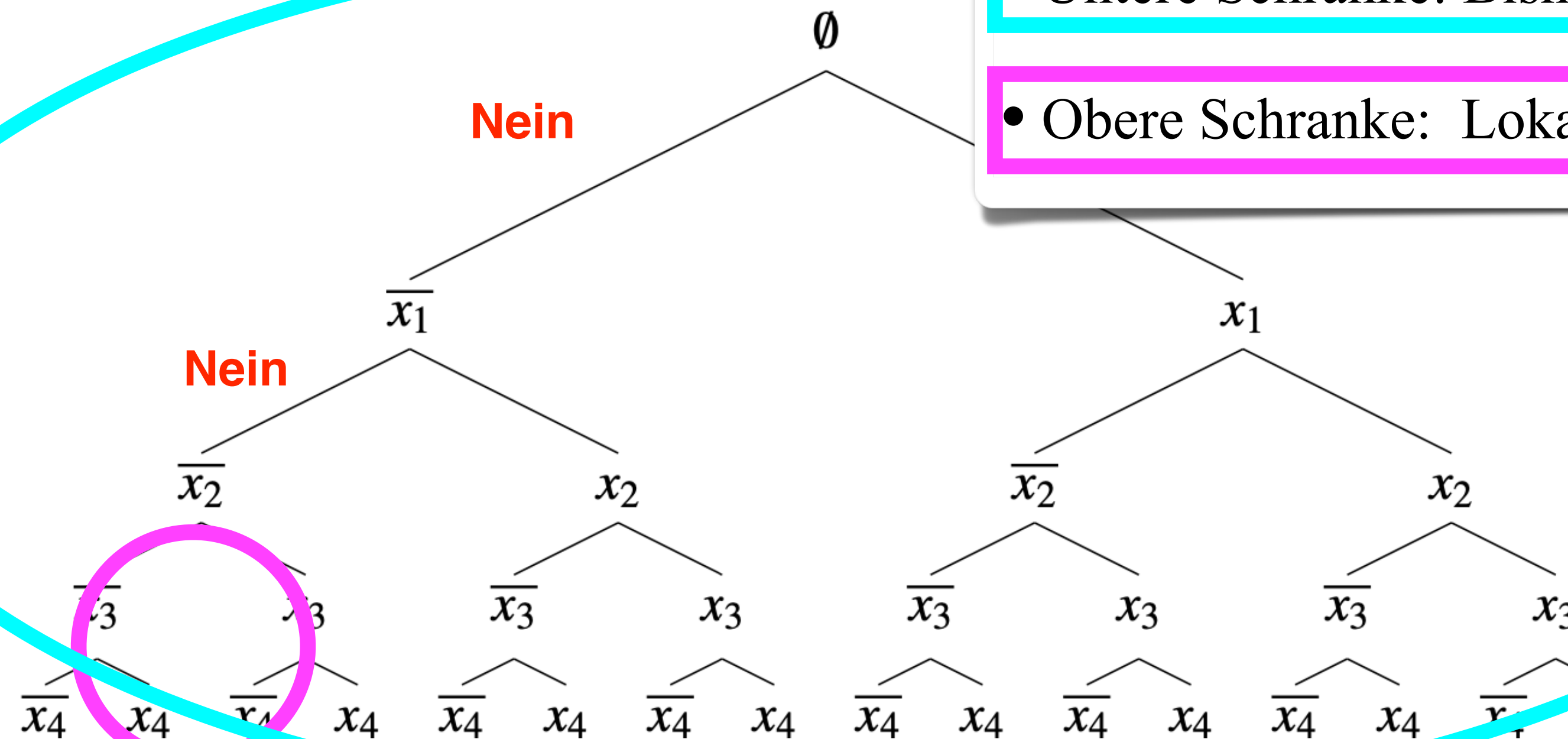
1. Enumeriere die möglichen Teilmengen in einem Enumerationsbaum
2. Behalte den Zielfunktionswert im Auge:
 - Untere Schranke: Erreichter Zielfunktionswert im ganzen Baum
 - Obere Schranke: Erreichbarer Zielfunktionswert im jeweiligen Baum
3. Beide Schranken sollten möglichst einfach und schnell zu berechnen sein
4. Wenn der erreichbare Wert in einem Teilbaum kleiner bleiben muss als der im ganzen Baum bereits erreichte, können wir den aktuellen Teilbaum abschneiden

4. Einsatz von Schranken - Grundidee

- Untere Schranke: Bisher erreichter Zielfunktionswert im ganzen Baum
- Obere Schranke: Erreichbarer Zielfunktionswert im jeweiligen Teilbaum



2. Geeignete Schranken - Knapsack



- Untere Schranke: Bisheriger Bestwert für Greedy (ganzzahlig)
- Obere Schranke: Lokaler erreichbarer Bestwert für Greedy (fraktional)

i	6	4	13	12	9	3	15	8	16	10	1	14	5	11	2	7
z_i	4	8	16	20	8	40	40	40	24	32	20	20	16	28	32	32
p_i	4	5	10	9	2	10	10	9	4	5	3	3	2	3	3	2

$$\sum_{i=1}^n p_i x_i = 44$$

i	6	4	13	12	9	3	15	8	16	10	1	14	5	11	2	7
z_i	4	8	16	20	8	40	40	40	24	32	20	20	16	28	32	32
p_i	4	5	10	9	2	10	10	9	4	5	3	3	2	3	3	2

$$\sum_{i=1}^n p_i x_i = 46 \quad X_{15} = 0,6$$

Relaxierung: Einfacher durch größere Lösungsmenge

3. Berechnung von Schranken

- Untere Schranke: Bisheriger Bestwert für Greedy (ganzzahlig)
- Obere Schranke: Lokaler erreichbarer Bestwert für Greedy (fraktional)

i	6	4	13	12	9	3	15	8	16	10	1	14	5	11	2	7
z _i	4	8	16	20	8	40	40	40	24	32	20	20	16	28	32	32
p _i	4	5	10	9	2	10	10	9	4	5	3	3	2	3	3	2

$\sum_{i=1}^n p_i x_i = 44$

i	6	4	13	12	9	3	15	8	16	10	1	14	5	11	2	7
z _i	4	8	16	20	8	40	40	40	24	32	20	20	16	28	32	32
p _i	4	5	10	9	2	10	10	9	4	5	3	3	2	3	3	2

$\sum_{i=1}^n p_i x_i = 46$ $x_{15} = 0,6$

Heuristik: Schnell zu lösen, aber nicht notwendig optimal

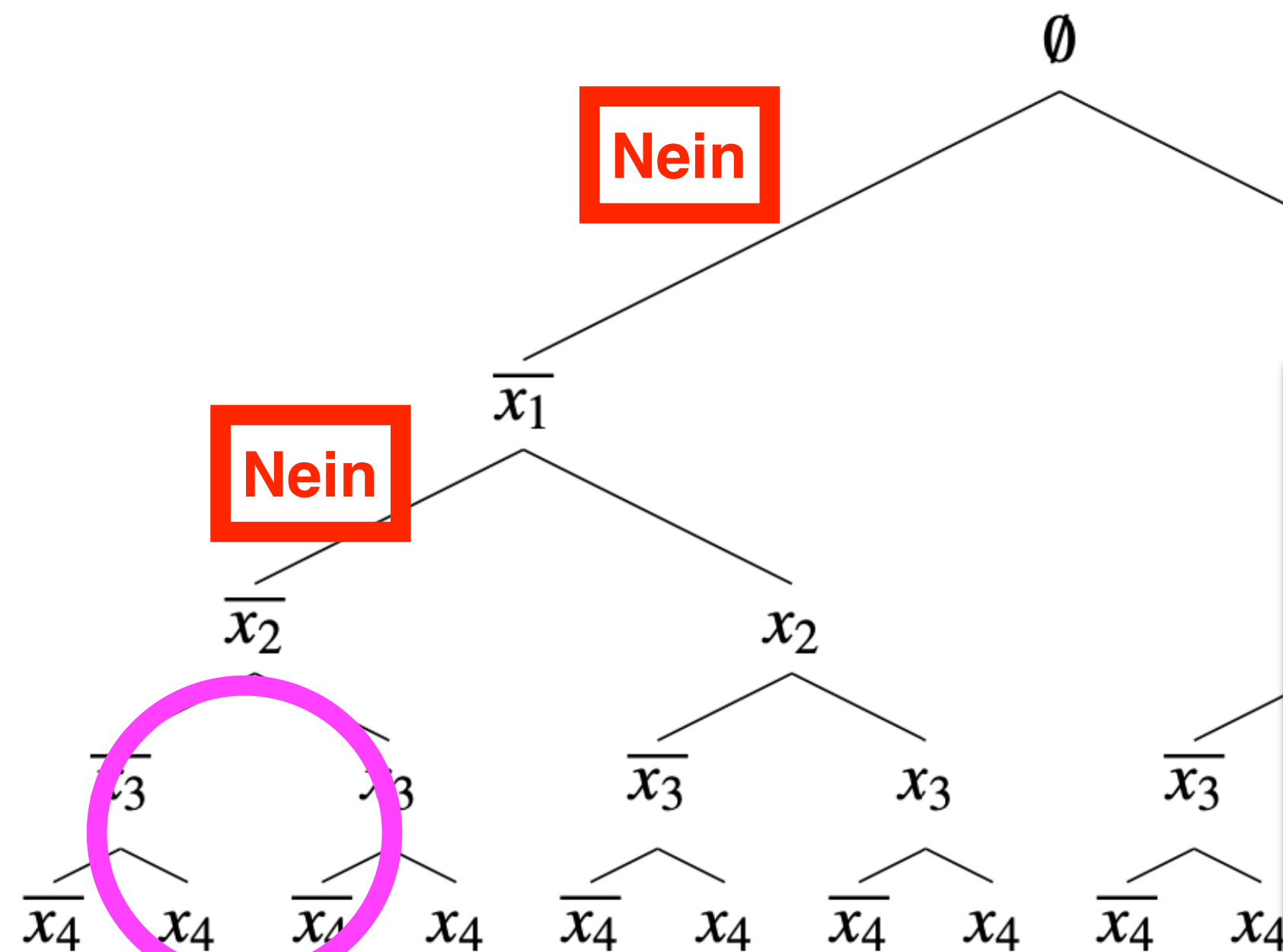
Relaxierung: Schnell und optimal zu lösen durch größere Lösungsmenge

3. Berechnung von Schranken II



S : negativ fixiert ($x_i = 0$)

Bisherige Entscheidungen fixieren Teilmenge!



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

3.4 Ausblicke

Historie

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 maximand² 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.

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.

Knapsack-Probleme

0-1-Knapsack

$$\max \sum_{i=1}^n x_i p_i$$

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

$$x_i \in \{0, 1\}$$

Schwieriger

Fraktionales Knapsack

$$\max \sum_{i=1}^n x_i p_i$$

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

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

Einfacher

Ganzzahlige und lineare Optimierungsprobleme

Ganzzahliges Optimierungsproblem

$$\max \sum_{i=1}^n x_i p_i$$

$$\begin{array}{ccccccc} a_{11}x_1 & + \dots & + a_{1n}x_n & \leq & b_1 \\ a_{21}x_1 & + \dots & + a_{2n}x_n & \leq & b_2 \\ \vdots & \vdots & \vdots & \vdots & \\ a_{m1}x_1 & + \dots & + a_{mn}x_n & \leq & b_m \end{array}$$

$$x_i \in \{0, 1\}$$

Beweisbar schwer zu lösen

Lineares Optimierungsproblem

$$\max \sum_{i=1}^n x_i p_i$$

$$\begin{array}{ccccccc} a_{11}x_1 & + \dots & + a_{1n}x_n & \leq & b_1 \\ a_{21}x_1 & + \dots & + a_{2n}x_n & \leq & b_2 \\ \vdots & \vdots & \vdots & \vdots & \\ a_{m1}x_1 & + \dots & + a_{mn}x_n & \leq & b_m \end{array}$$

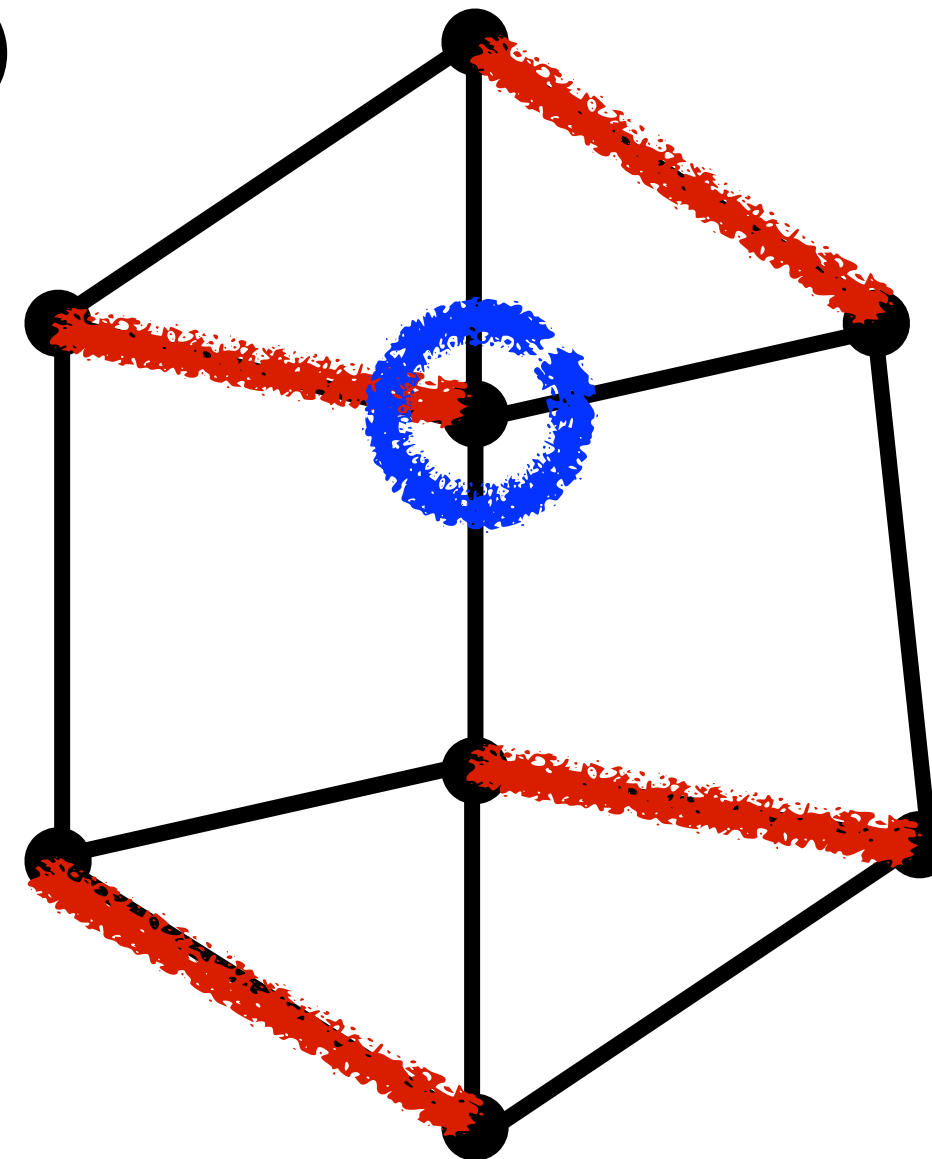
$$x_i \in [0, 1]$$

Beweisbar effizient lösbar

Ganzzahlige Optimierungsprobleme: Matching

Gegeben: Ein Graph $G=(V,E)$

 $x_e = 1$



$$\begin{aligned} \max \sum_{e \in E} x_e \\ \forall v \in V : \sum_{e \in \delta(v)} x_e \leq 1 \\ x_e \in \{0, 1\} \end{aligned}$$

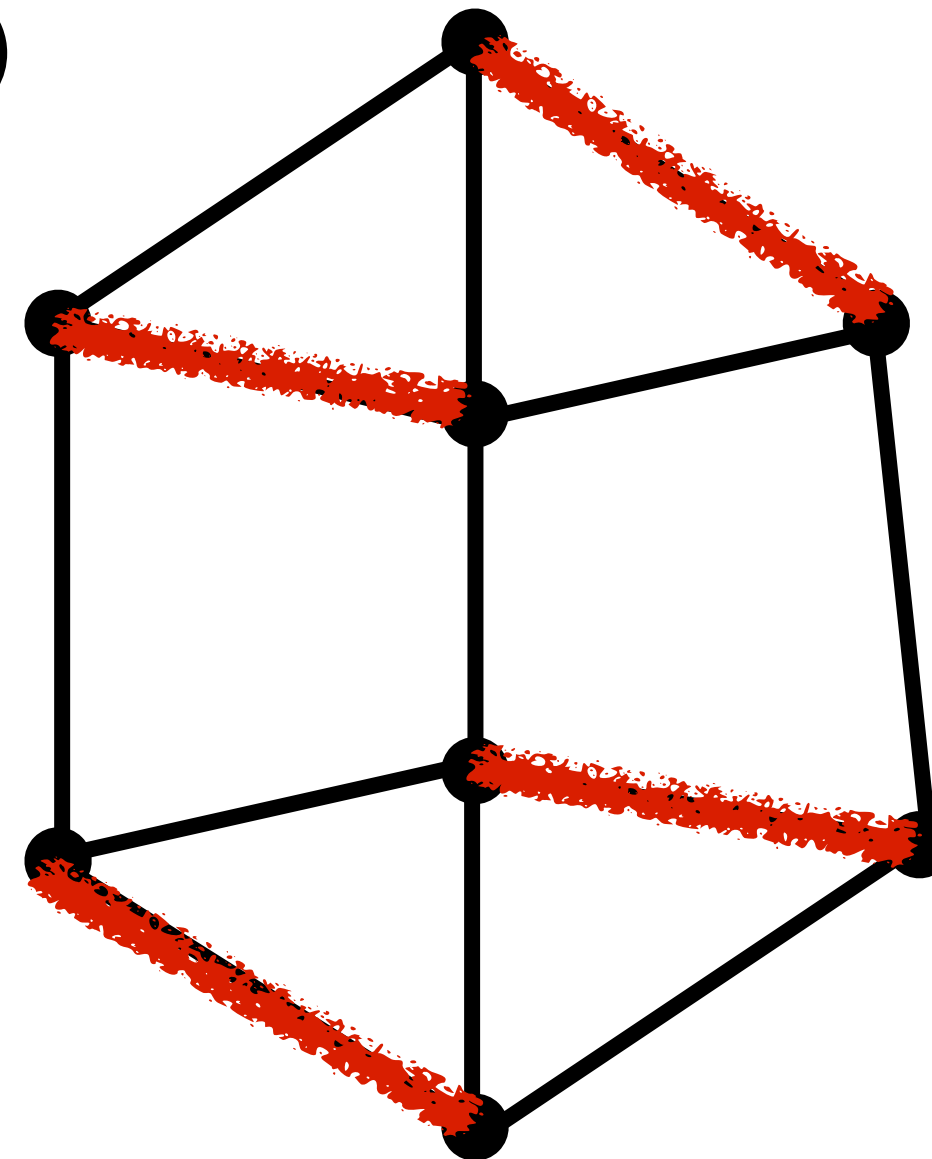
Gesucht: Ein größtmögliches **Matching** M in G :
eine möglichst große Menge disjunkter Kanten

Ganzzahlige Optimierungsprobleme: Matching

Gegeben: Ein Graph $G=(V,E)$

$$\text{---} x_e = 1$$

$$\sum_{e \in E} x_e = 4$$



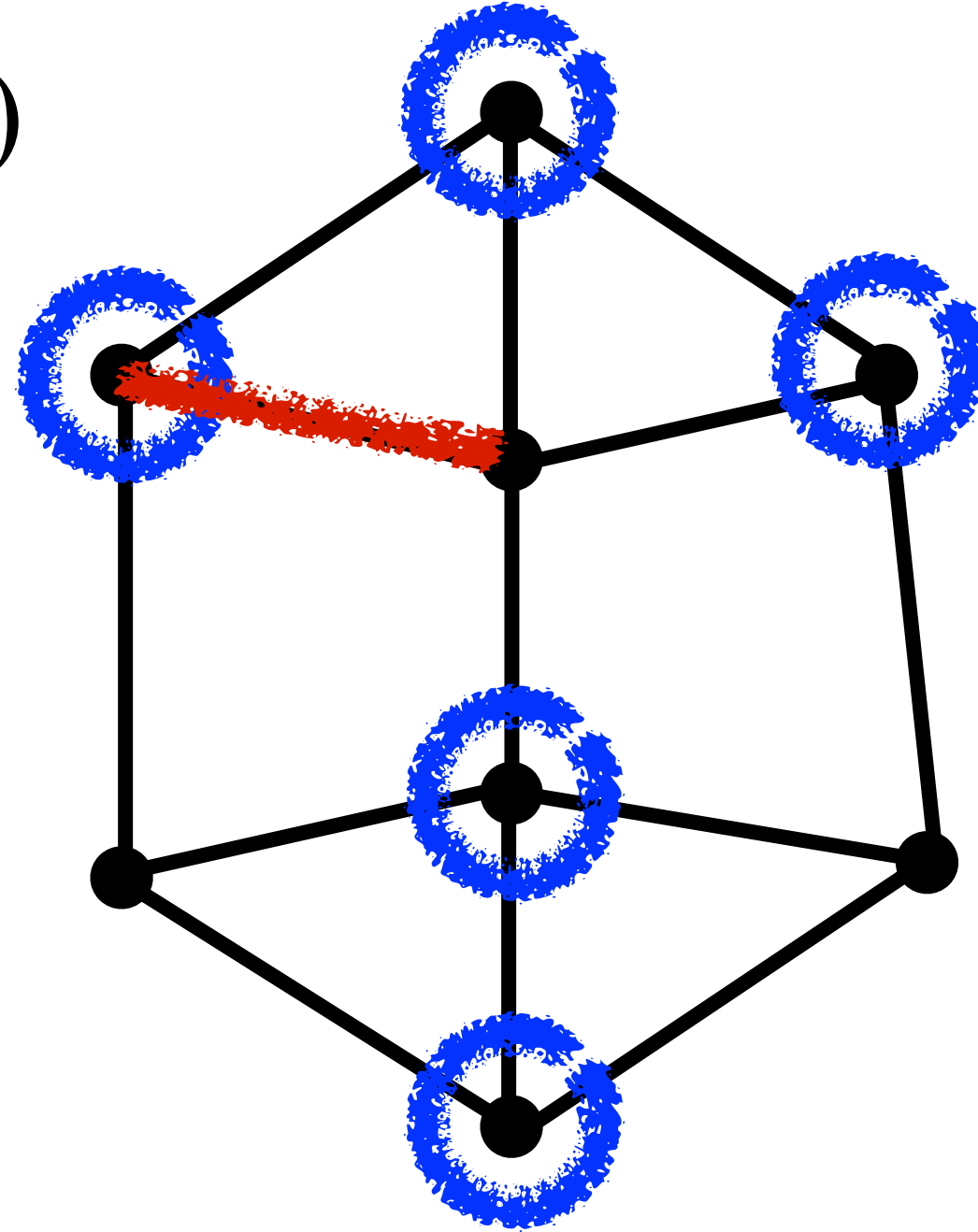
$$\begin{aligned} & \max \sum_{e \in E} x_e \\ & \forall v \in V : \sum_{e \in \delta(v)} x_e \leq 1 \\ & x_e \in \{0, 1\} \end{aligned}$$

Gesucht: Ein größtmögliches **Matching** M in G :
eine möglichst große Menge disjunkter Kanten

Ganzzahlige Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph $G=(V,E)$

$$\bigcirc y_v = 1$$



$$\begin{aligned} \min \sum_{v \in V} y_v \\ \forall e \in E : \sum_{v \in \delta(e)} y_v \geq 1 \\ y_v \in \{0, 1\} \end{aligned}$$

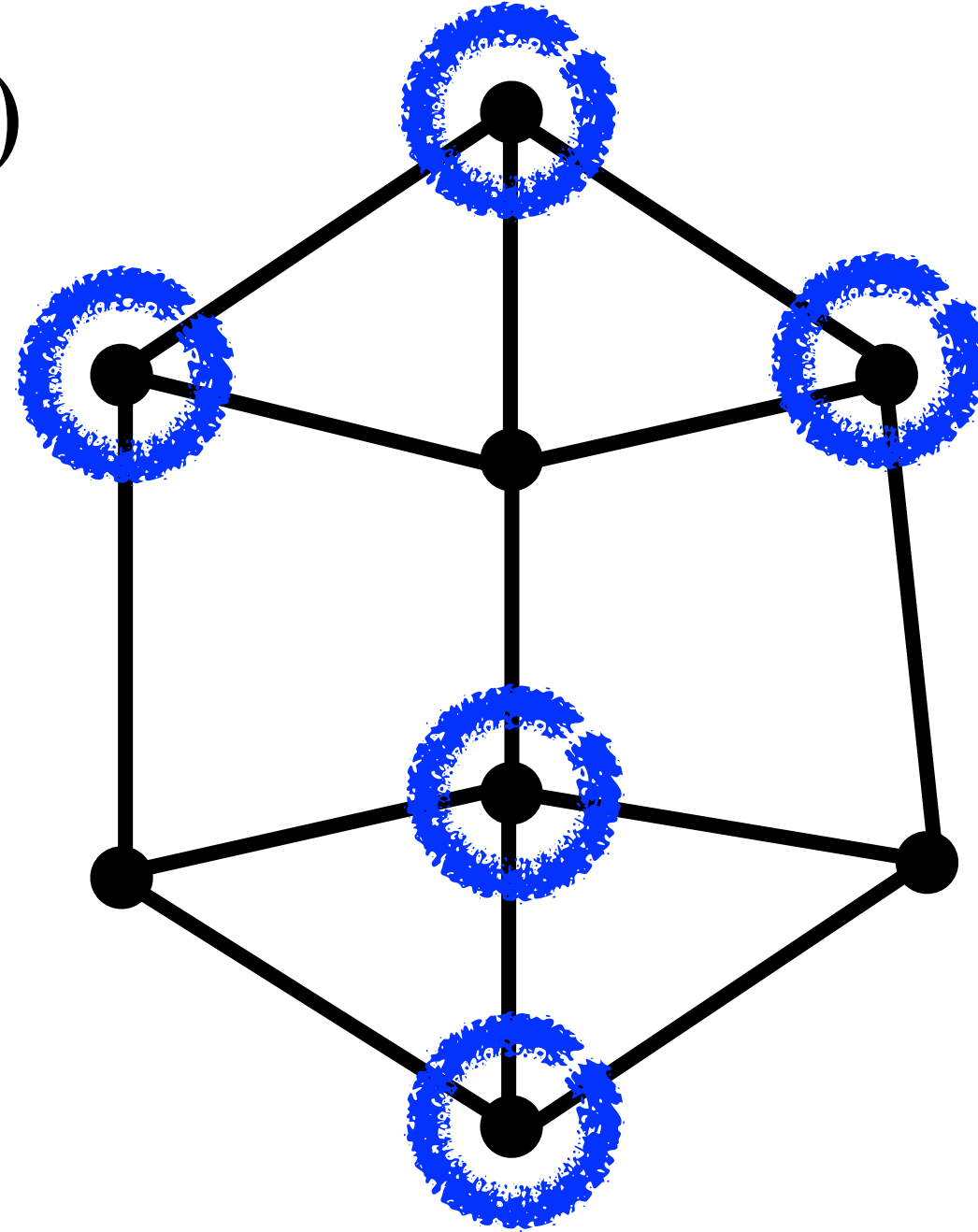
Gesucht: Ein kleinstmögliches **Vertex Cover** S in G :
eine möglichst kleine Menge von Knoten,
die alle Kanten überdecken

Ganzzahlige Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph $G=(V,E)$

$$\bigcirc y_v = 1$$

$$\sum_{v \in V} y_v = 5$$



$$\begin{aligned} \min \sum_{v \in V} y_v \\ \forall e \in E : \sum_{v \in \delta(e)} y_v \geq 1 \\ y_v \in \{0, 1\} \end{aligned}$$

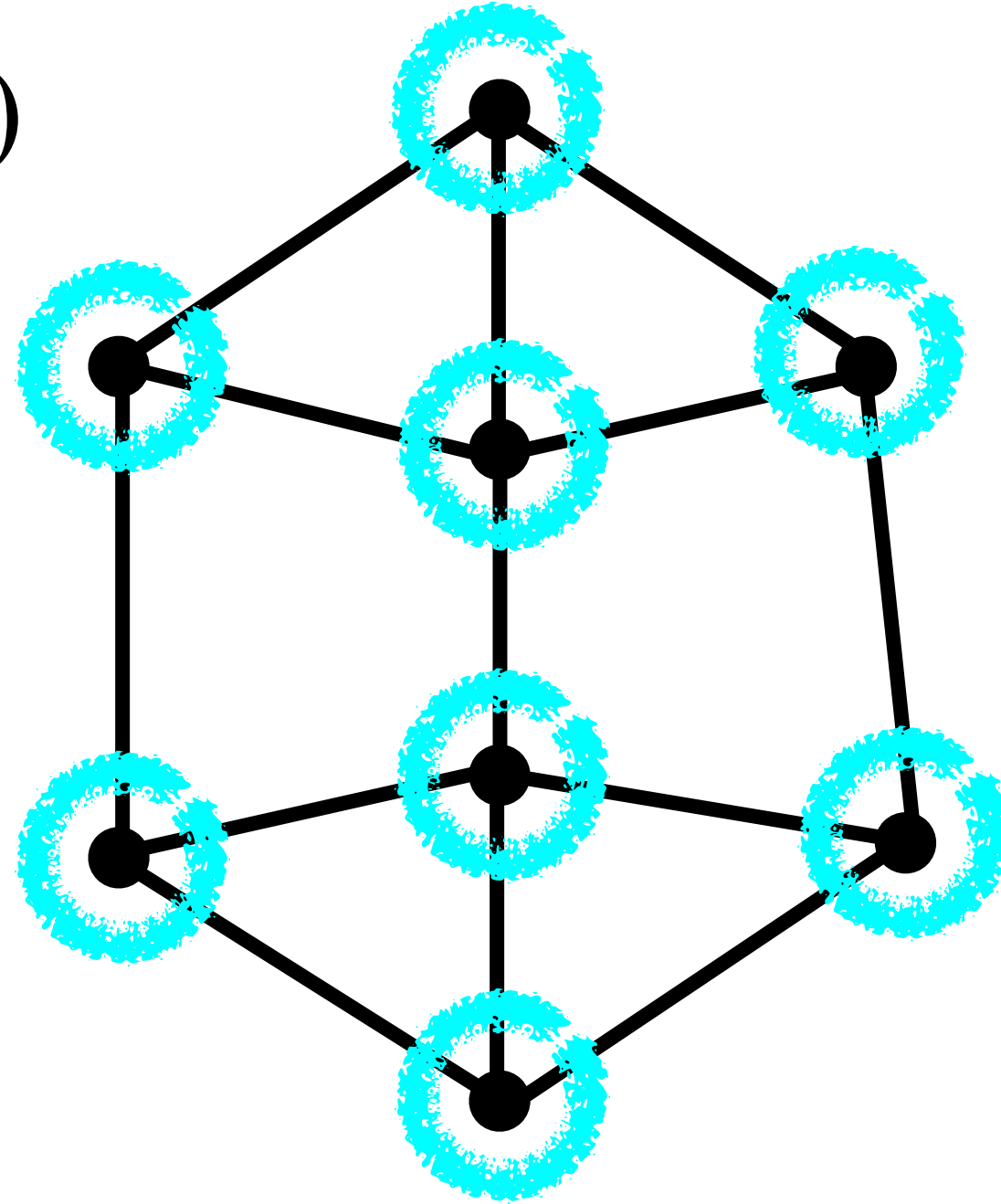
Gesucht: Ein kleinstmögliches **Vertex Cover** S in G :
eine möglichst kleine Menge von Knoten,
die alle Kanten überdecken

Lineare Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph $G=(V,E)$


$$y_v = \frac{1}{2}$$

$$\sum_{v \in V} y_v = 4$$



$$\forall e \in E :$$

$$\sum_{e \in \delta(v)} y_v \geq 1$$

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

Gesucht: Ein kleinstmögliches **Vertex Cover** S in G :
eine möglichst kleine Menge von Knoten,
die alle Kanten überdecken

Relaxierung

Lineare Optimierungsprobleme: Vertex Cover

Ganzzahlig?!

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 maximand² 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.

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.

Z.B. geeignetes Branch-and-Bound

3.5 Quiz!

BRUTE-FORCE
SOLUTION:
 $O(n!)$

DYNAMIC
PROGRAMMING
ALGORITHMS:
 $O(n^2 2^n)$

Algorithmus 4.3 Greedy:
Input: $z_1, \dots, z_n, Z, p_1, \dots$
Output: $S \subseteq \{1, \dots, n\}$
 $G'_0 = \max_{x \in S} G_0$
return

4 Approximation

Algorithmen und Datenstrukturen 2
Sommer 2024

Prof. Dr. Sándor Fekete

4.1 Motivation

Optimierungsprobleme

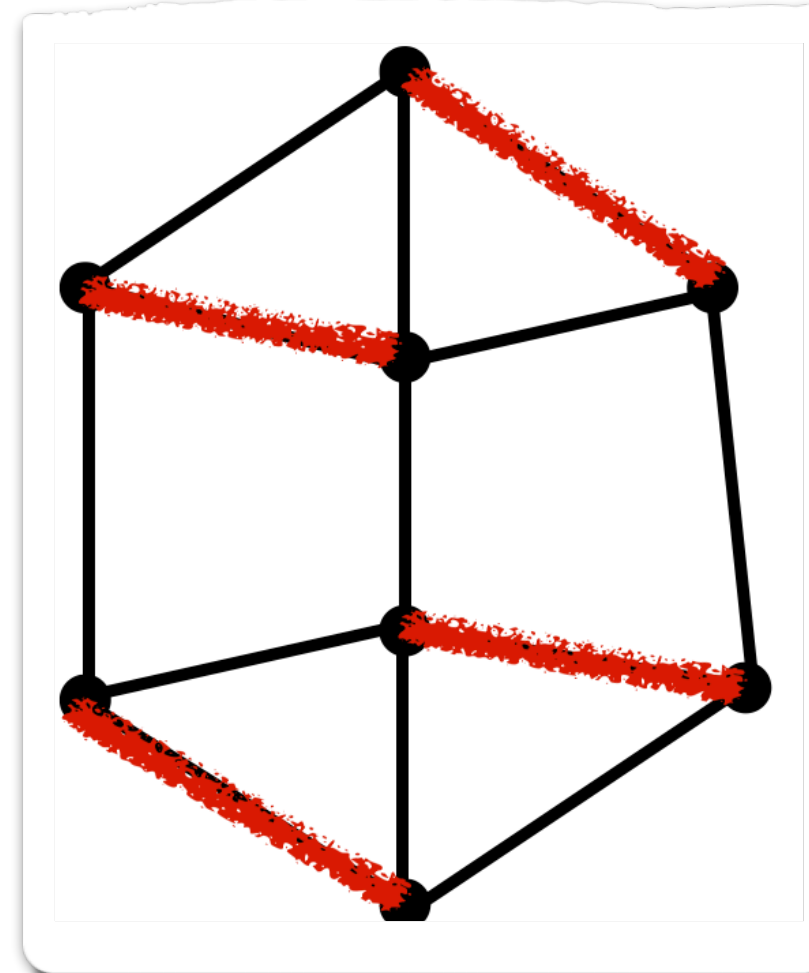
0-1 Knapsack

1	Nummer	2	3	4	5	6
20	Minuten	32	40	8	16	4
3	Punkte	3	10	5	2	4
			10			
			32			
			5			
7	8		11	12	16	13
32	40		28	20	24	16
2	9		3	9	4	10
		9	14			
		8	20			
		2	3	15		
				40		
				10		

Kann Knut die Klausur bestehen?

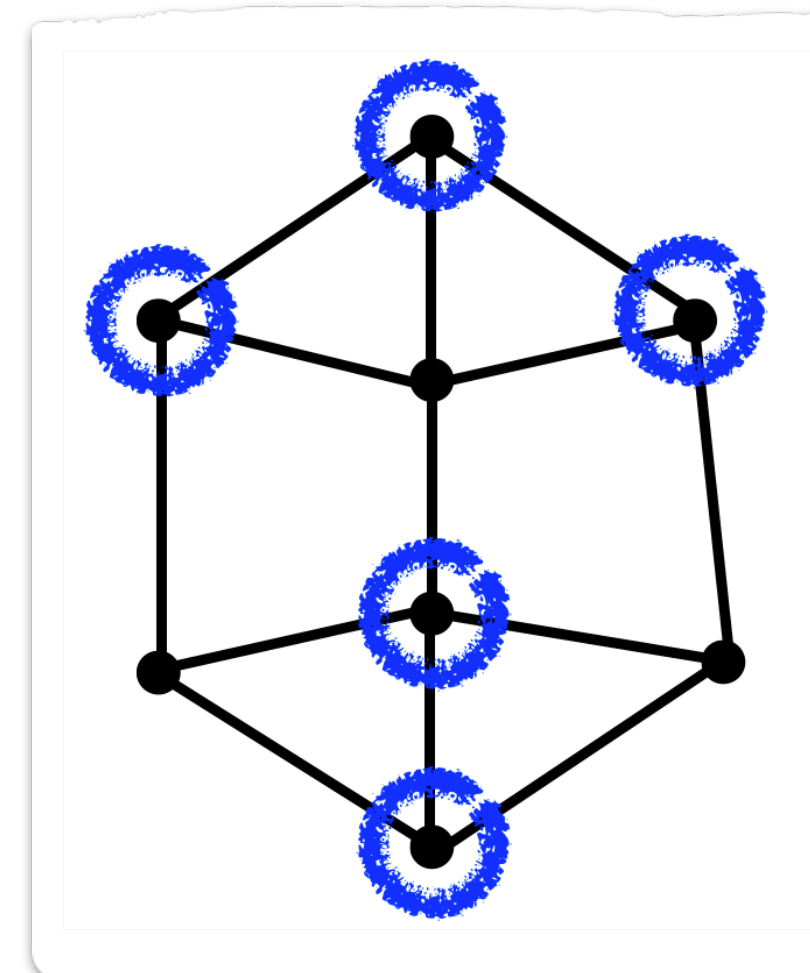
Max

Matching



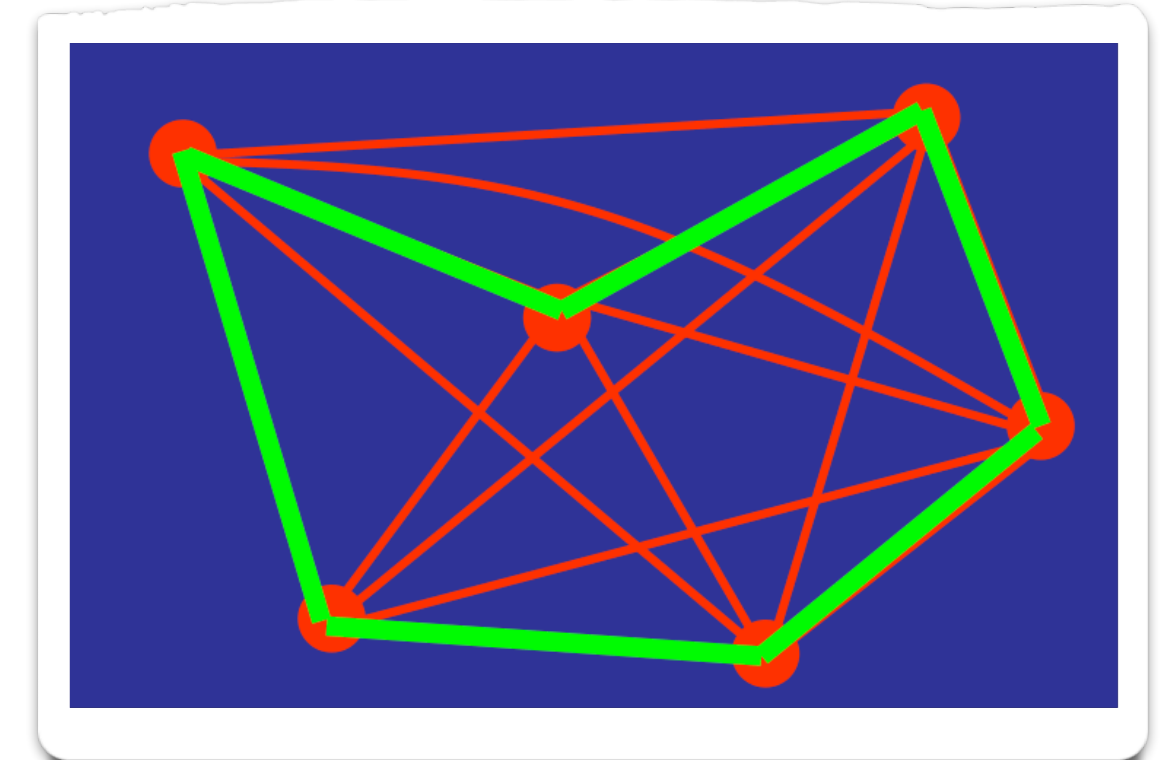
Max

Vertex Cover



Min

Traveling Salesman



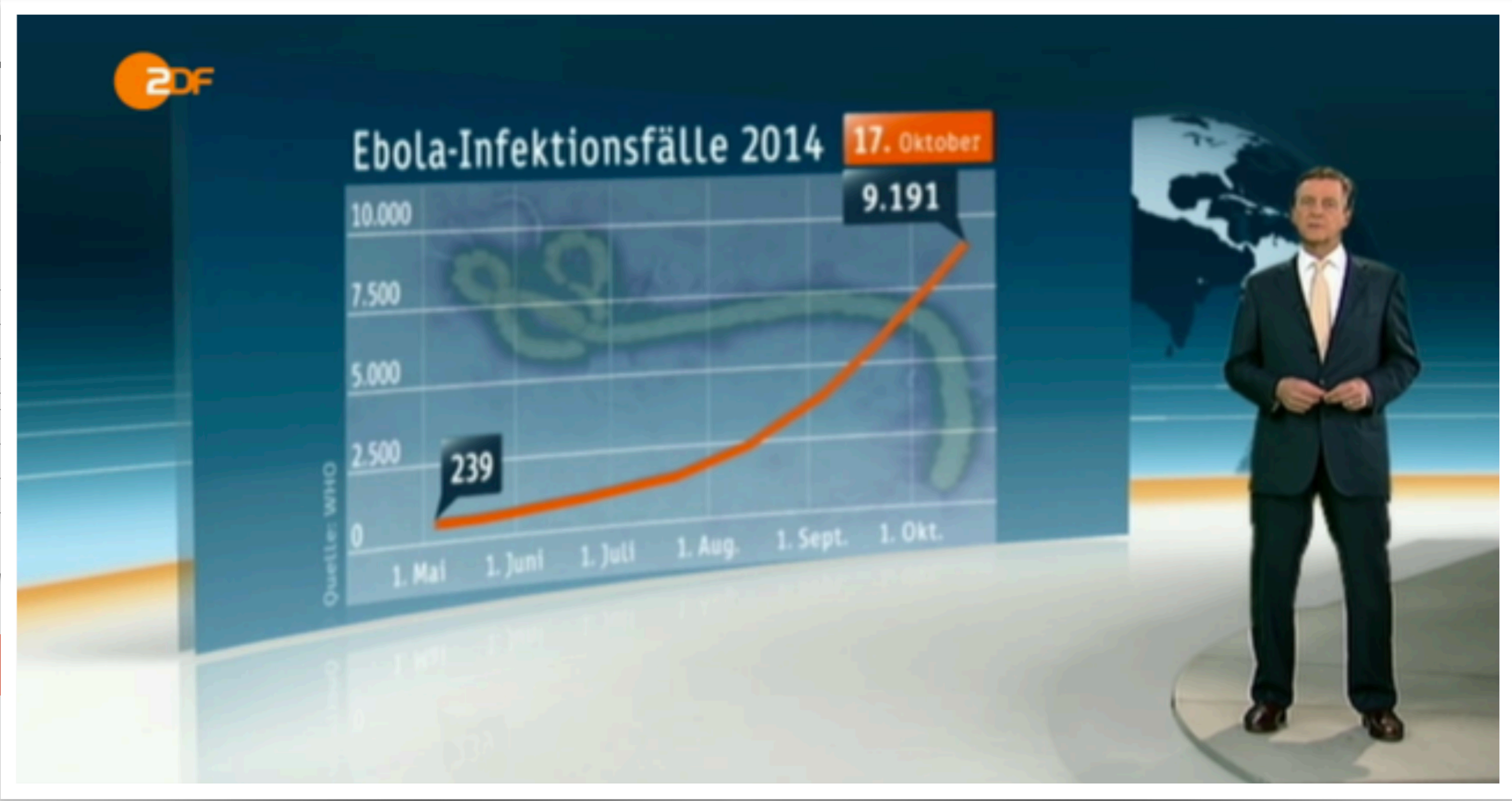
Min

Ziel: Finde eine bestmögliche Lösung unter **exponentiell** vielen möglichen!

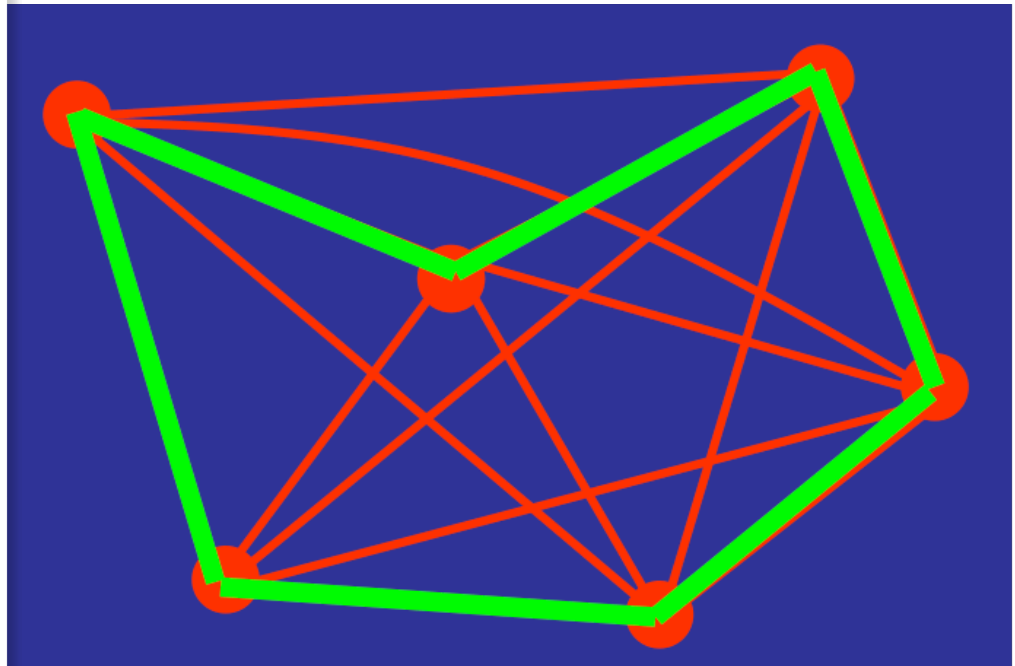
Optimierungsprobleme

0-1

1	Nummer	2	
20	Minuten	32	
3	Punkte	3	10
			32
			5
	8		
7	40	9	
32	9	8	
2		2	



Traveling Salesman



Min

Ziel: Finde eine bestmögliche Lösung unter **exponentiell** vielen möglichen!

Laufzeit?

0-1 Knapsack

$$O(nZ)$$

$$O(2^n f(n))$$

1	Nummer	2		3	4	5	6
20	Minuten	32		40	8	16	4
3	Punkte	3		10	5	2	4
			10				
			32		11	12	13
			5		28	20	16
7	8				3	9	10
32	40	9					
2	9			14		16	
		9		20		24	
		2		3	15	4	
					40		
					10		

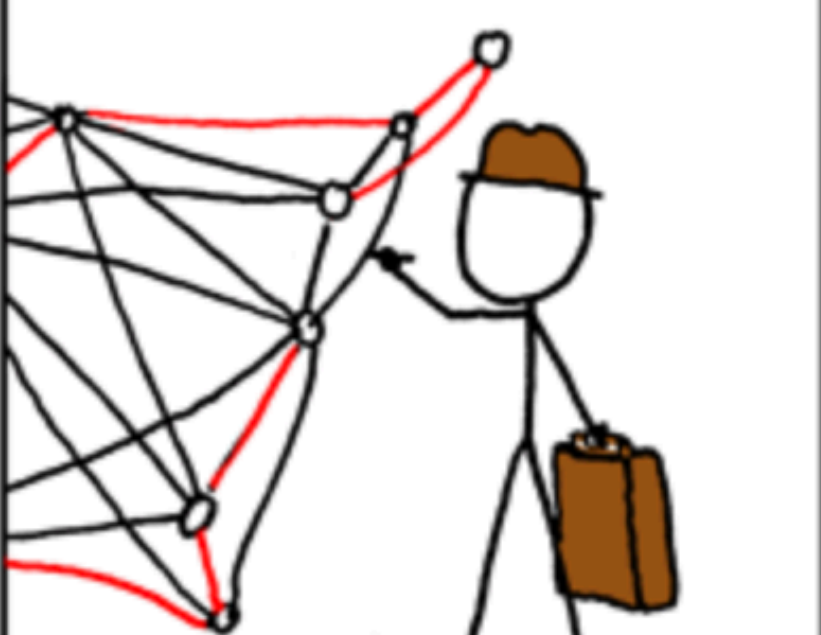
Kann Knut die Klausur bestehen?

Traveling Salesman

BRUTE-FORCE
SOLUTION:
 $O(n!)$



DYNAMIC
PROGRAMMING
ALGORITHMS:
 $O(n^2 2^n)$



Laufzeit?

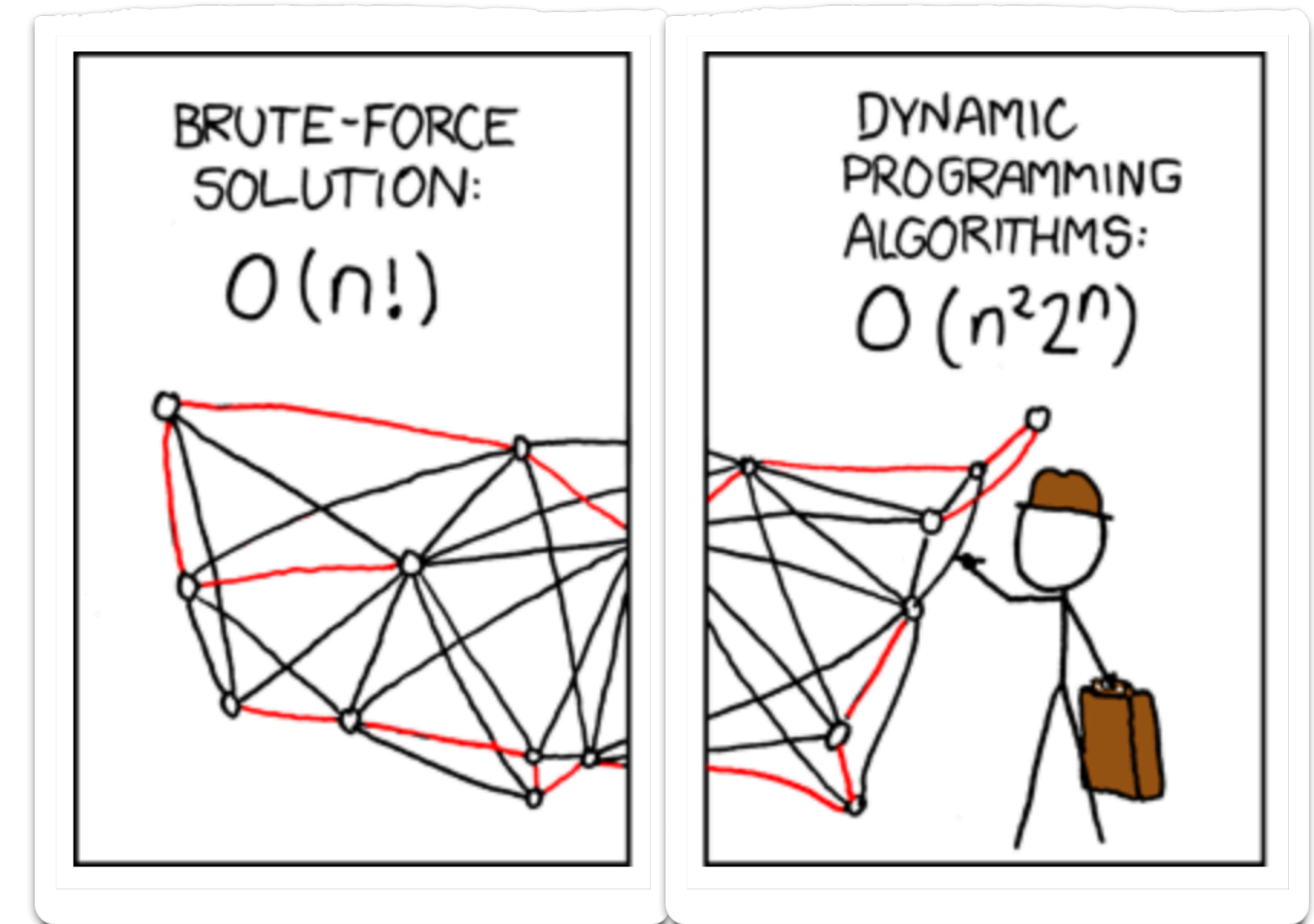
0-1 Knapsack

$$O(nZ)$$

$$O(2^n f(n))$$

1	Nummer	2	3	4	5	6
					16	4
Kann Knut die Klausur bestehen?						
32	9	8	14			
2		2	20	15		4
			3	40		
				10		
Kann Knut die Klausur bestehen?						

Traveling Salesman



Ziel: Vielleicht nicht die *bestmögliche* Lösung, sondern eine *gute*, aber dafür schneller...?!

4.2 Greedy und Approximation

Beispiel 4.1: Knapsack



Greedy: 2



$Z=N$
 $n=2$

$z_1 = 1$
 $p_1 = 2$

$z_2 = N$
 $p_2 = N$

Algorithmus 4.2 GREEDY₀

Eingabe: $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe: $S \subseteq \{1, \dots, n\}$ und G_0

mit

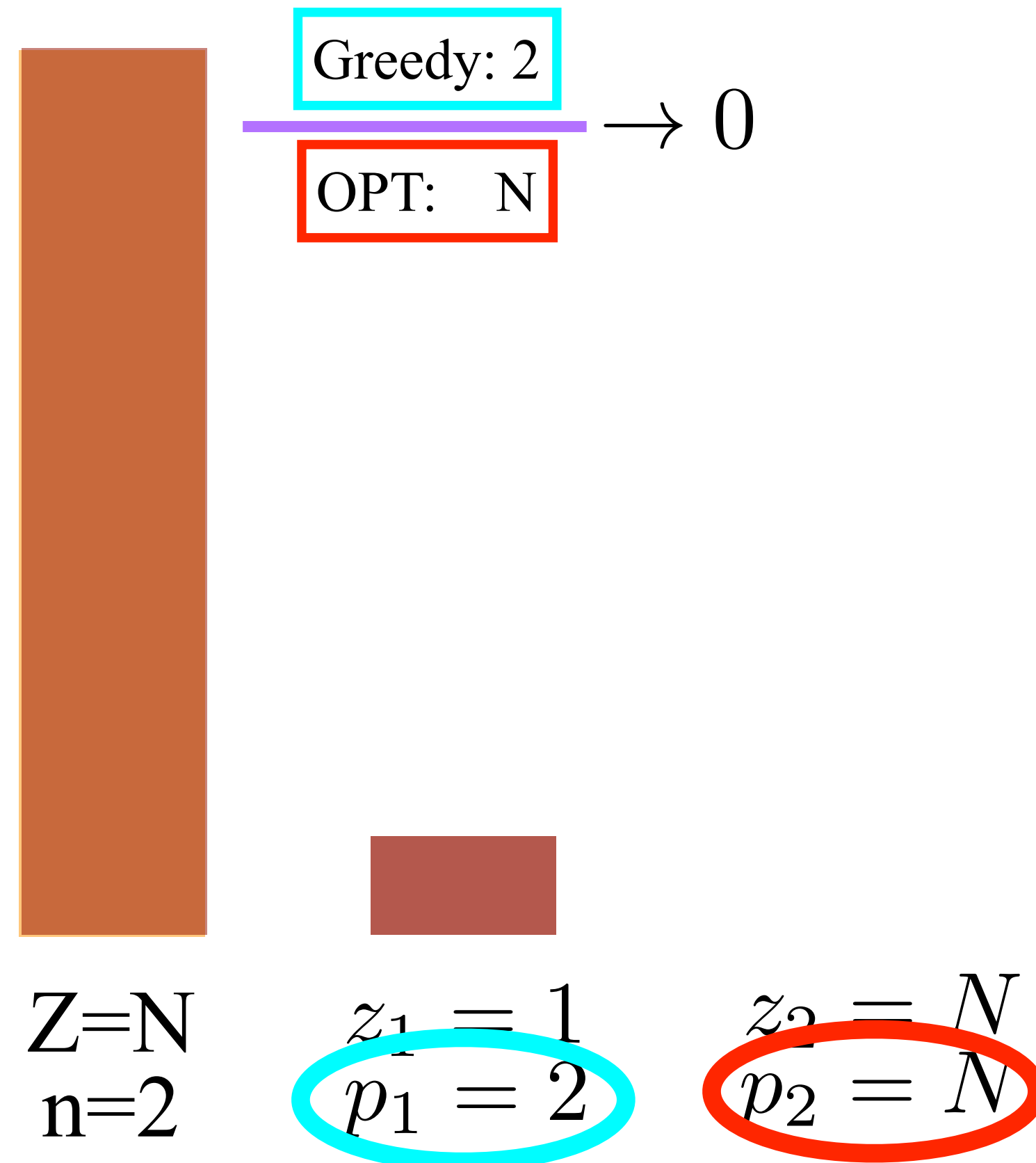
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \textit{Maximal}$$

- 1: Sortiere $\{1, \dots, n\}$ nach $\frac{z_i}{p_i}$ aufsteigend;
Dies ergibt die Permutation $\pi(1), \dots, \pi(n)$.
Setze $j = 1$.
- 2: **while** $(j \leq n)$ **do**
- 3: **if** $\left(\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z\right)$ **then**
- 4: $x_{\pi(j)} := 1$
- 5: $j := j + 1$
- 6: **return**

Beispiel 4.1: Knapsack



Algorithmus 4.2 GREEDY₀

Eingabe: $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe: $S \subseteq \{1, \dots, n\}$ und G_0
mit

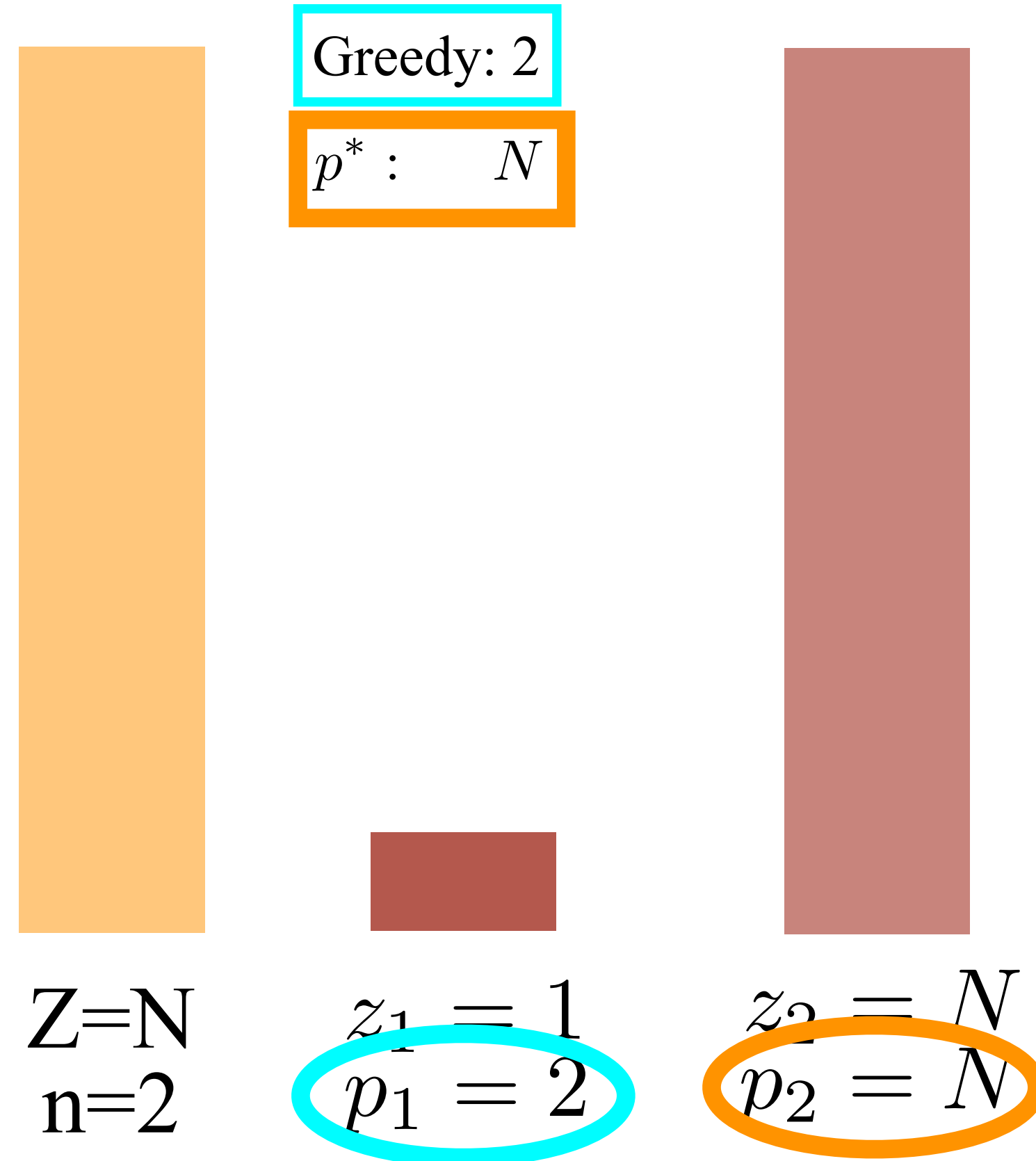
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

- 1: Sortiere $\{1, \dots, n\}$ nach $\frac{z_i}{p_i}$ aufsteigend;
Dies ergibt die Permutation $\pi(1), \dots, \pi(n)$.
Setze $j = 1$.
- 2: **while** $(j \leq n)$ **do**
- 3: **if** $\left(\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z\right)$ **then**
- 4: $x_{\pi(j)} := 1$
- 5: $j := j + 1$
- 6: **return**

Verbesserung?!



Algorithmus 4.3 GREEDY₀'

Eingabe: $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe: $S \subseteq \{1, \dots, n\}$ und G_0

mit

$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

- 1: Sortiere $\{1, \dots, n\}$ nach $\frac{z_i}{p_i}$ aufsteigend;
Dies ergibt die Permutation $\pi(1), \dots, \pi(n)$.
Setze $j = 1$.
- 2: **while** $(j \leq n)$ **do**
- 3: **if** $\left(\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z\right)$ **then**
- 4: $x_{\pi(j)} := 1$
- 5: $j := j + 1$
- 6: $G'_0 := \max \{G_0, \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}\}$
- 7: **return**

Wie gut ist das allgemein?

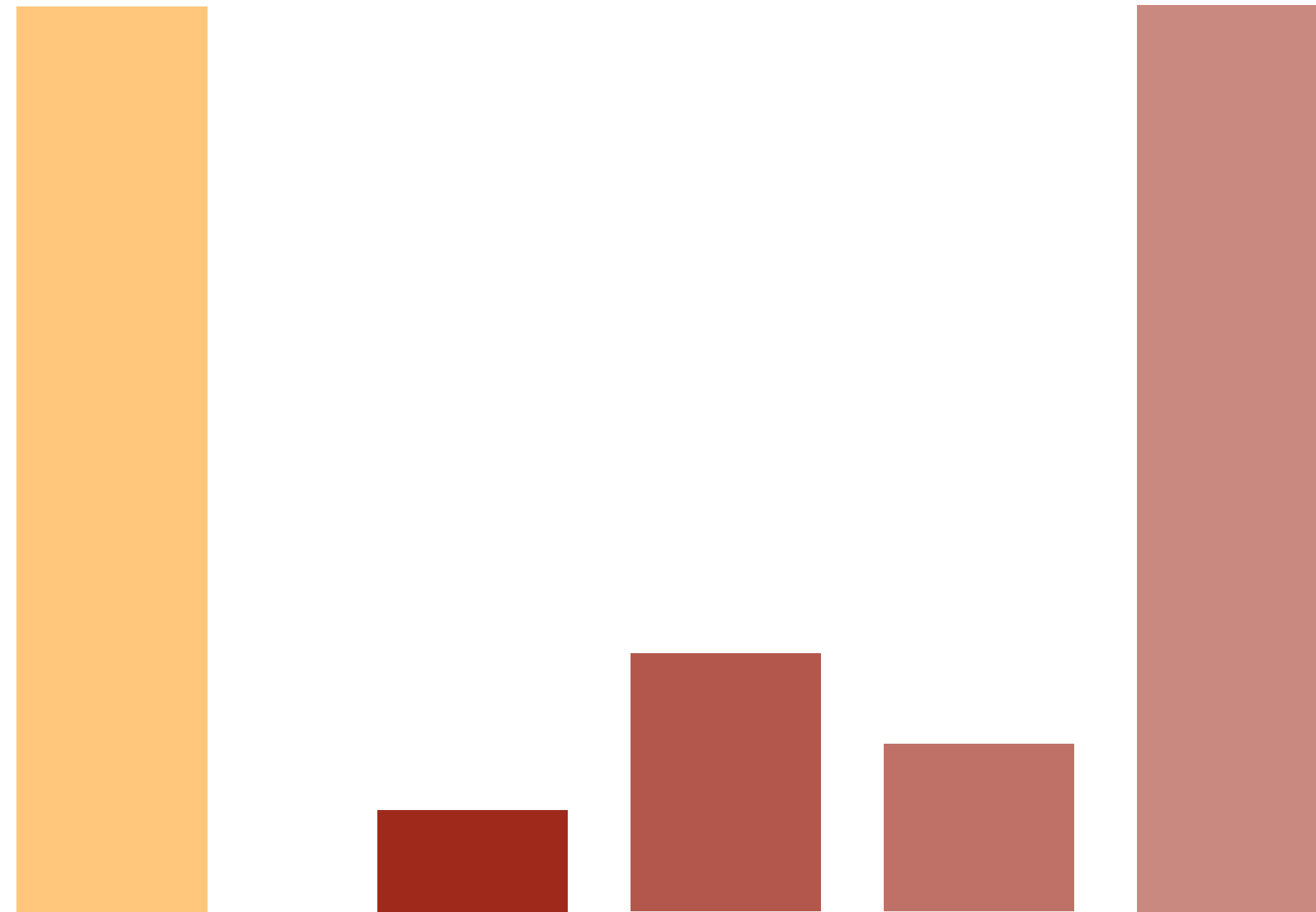
Algorithmus 4.3 Greedy-Approximation

Eingabe: $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe: $S \subseteq \{1, \dots, n\}$ mit $\sum_{i \in S} z_i \leq Z$ und Wert $G'_0 := \sum_{i \in S} p_i$

1: $G'_0 = \max \{ G_0 \mid \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\} \}$

2: **return**



Wie gut ist das allgemein?

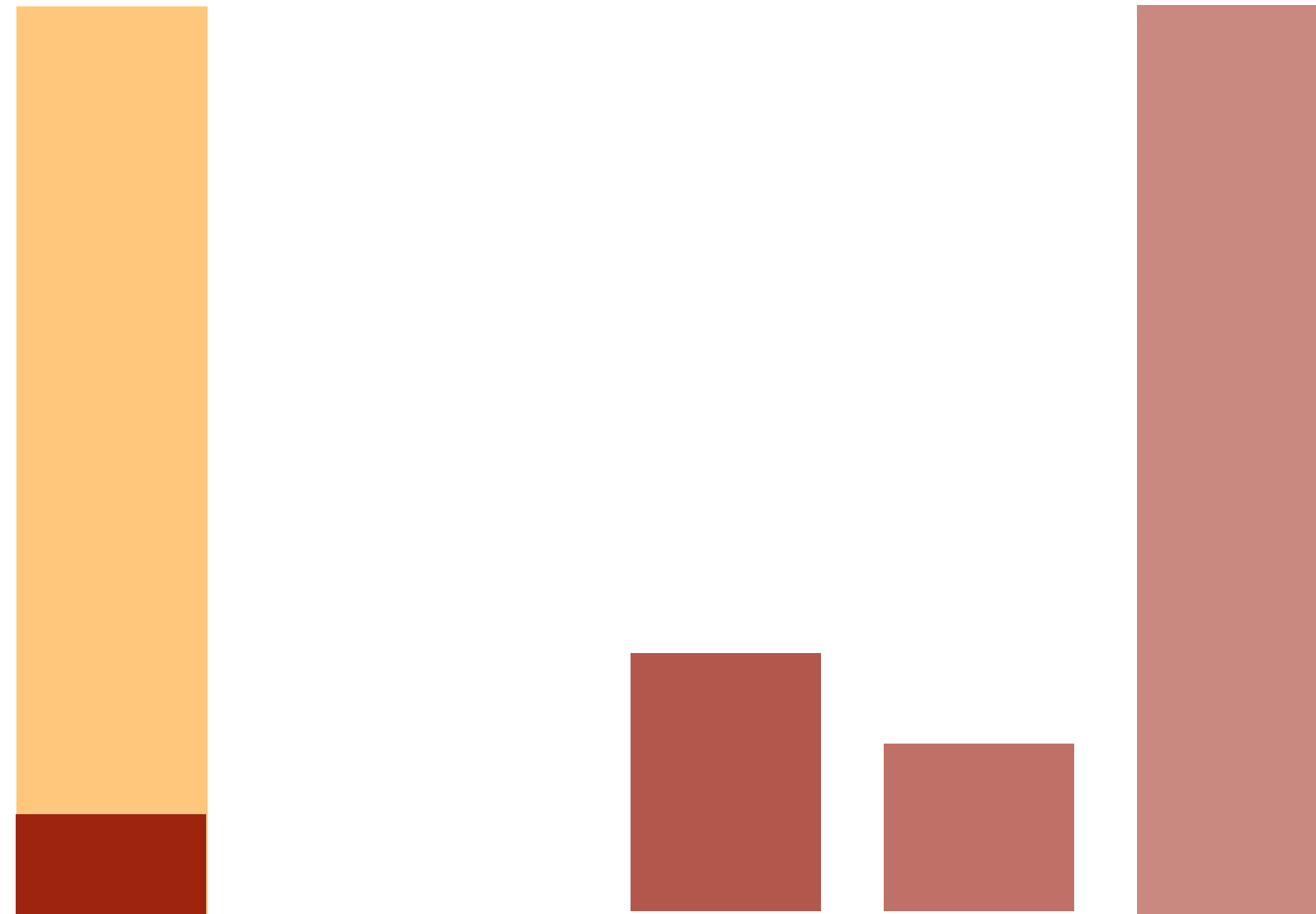
Algorithmus 4.3 Greedy-Approximation

Eingabe: $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe: $S \subseteq \{1, \dots, n\}$ mit $\sum_{i \in S} z_i \leq Z$ und Wert $G'_0 := \sum_{i \in S} p_i$

1: $G'_0 = \max \{ G_0 \mid \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\} \}$

2: **return**



Wie gut ist das allgemein?

Algorithmus 4.3 Greedy-Approximation

Eingabe: $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe: $S \subseteq \{1, \dots, n\}$ mit $\sum_{i \in S} z_i \leq Z$ und Wert $G'_0 := \sum_{i \in S} p_i$

1: $G'_0 = \max \{ G_0 \mid G_0 = \max \{ p_i \mid z_i < Z, i \in \{1, \dots, n\} \} \}$

2: **return**



Wie gut ist das allgemein?

Algorithmus 4.3 Greedy-Approximation

Eingabe: $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe: $S \subseteq \{1, \dots, n\}$ mit $\sum_{i \in S} z_i \leq Z$ und Wert $G'_0 := \sum_{i \in S} p_i$

1: $G'_0 = \max \{ G_0 \mid \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\} \}$

2: **return**

Wie gut ist das allgemein?

Algorithmus 4.3 Greedy-Approximation

Eingabe: $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe: $S \subseteq \{1, \dots, n\}$ mit $\sum_{i \in S} z_i \leq Z$ und Wert $G'_0 := \sum_{i \in S} p_i$

- 1: $G'_0 = \max \{ G_0, \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\} \}$
- 2: **return**

Wie gut ist das allgemein?

Algorithmus 4.3 Greedy-Approximation

Eingabe: $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe: $S \subseteq \{1, \dots, n\}$ mit $\sum_{i \in S} z_i \leq Z$ und Wert $G'_0 := \sum_{i \in S} p_i$

1: $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

Satz 4.4. Algorithmus 4.3 berechnet eine Lösung mit G'_0 ;
im Vergleich mit dem Optimalwert OPT gilt: $G'_0 \geq \frac{1}{2}OPT$.

$$\begin{array}{c}
 G'_0 \geq G_0 \qquad G'_0 \geq p^* \\
 OPT \leq \text{FractionalKP} \leq G_0 + x_i p_i \leq G_0 + p^* \leq 2G'_0
 \end{array}$$

4.3 Approximationsalgorithmen

Approximation

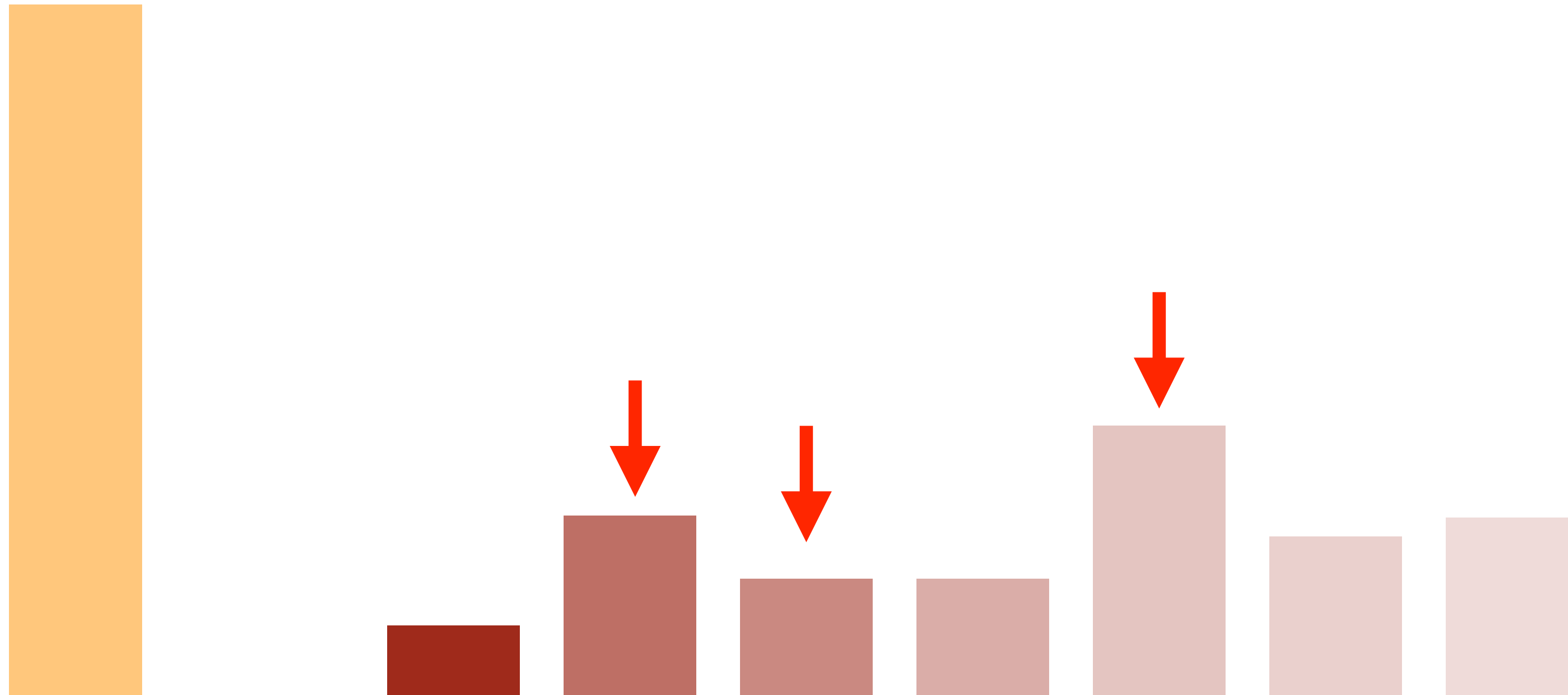
Definition 4.5 (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz I von MAX
 - a) ALG in *polynomieller Zeit* in der Größe von I eine zulässige Lösung mit Wert ALG liefert
 - b) für den Vergleich mit dem zugehörigen Optimalwert $\text{OPT}(I)$ gilt **$\text{ALG}(I) \geq c \cdot \text{OPT}(I)$** (dabei ist **$c \leq 1$**)
2. Entsprechend für **Minimierungsproblem** wiederum:
 - a) ALG liefert in *polynomieller Zeit* in der Größe von I eine zulässige Lösung mit Wert $\text{ALG}(I)$
 - b) es gilt für den Vergleich mit dem zugehörigen Optimalwert $\text{OPT}(I)$: **$\text{ALG}(I) \leq c \cdot \text{OPT}(I)$** (dabei ist **$c \geq 1$**)

4.4 Bessere Approximation

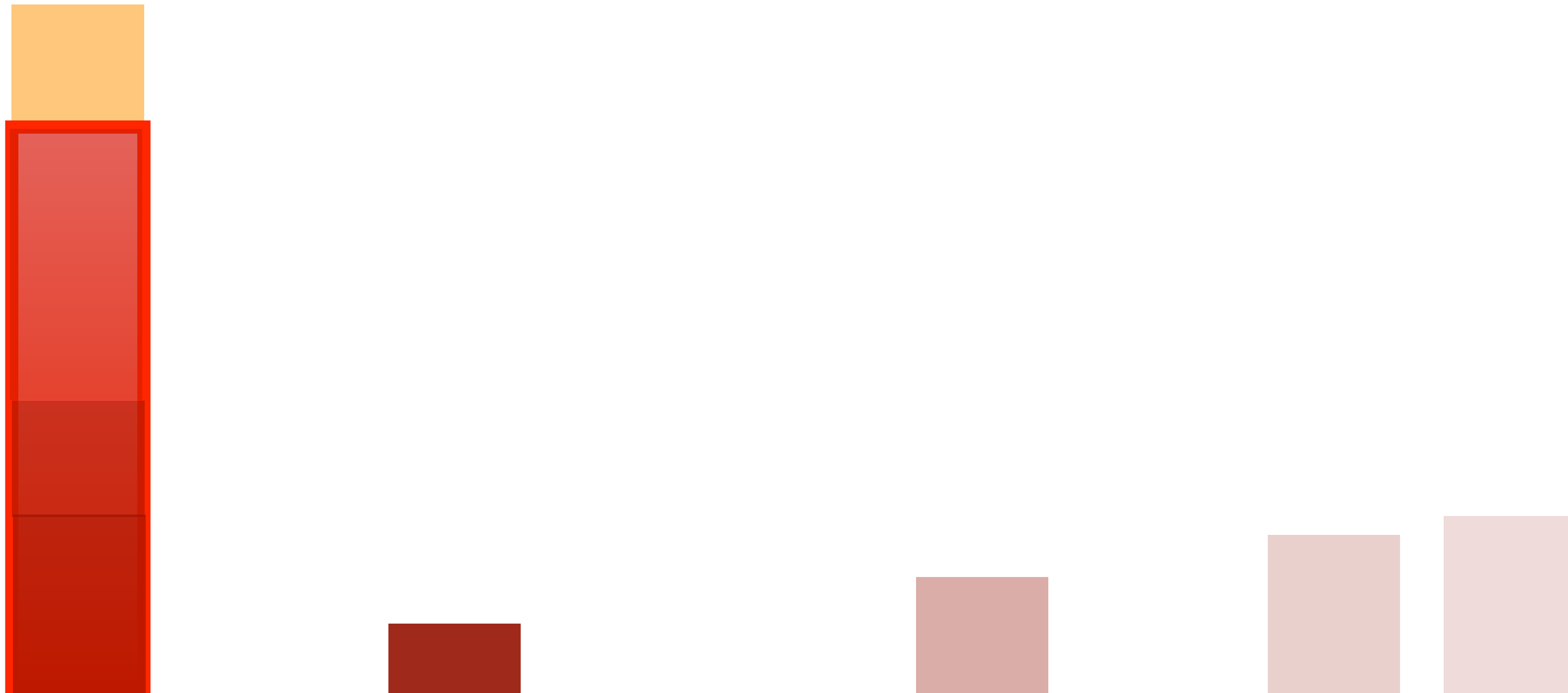
Ideen:

- Verwende die k wertvollsten Objekte aus einer Optimallösung.



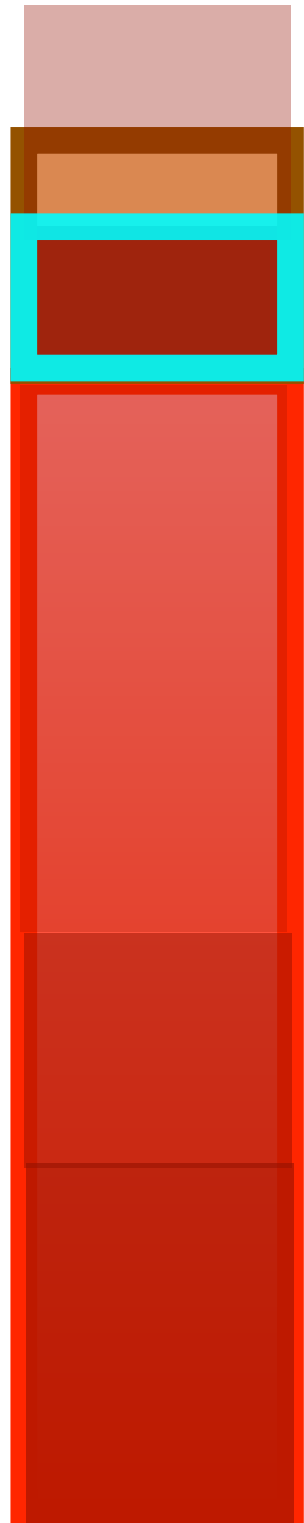
Ideen:

- Verwende die k wertvollsten Objekte aus einer Optimallösung.



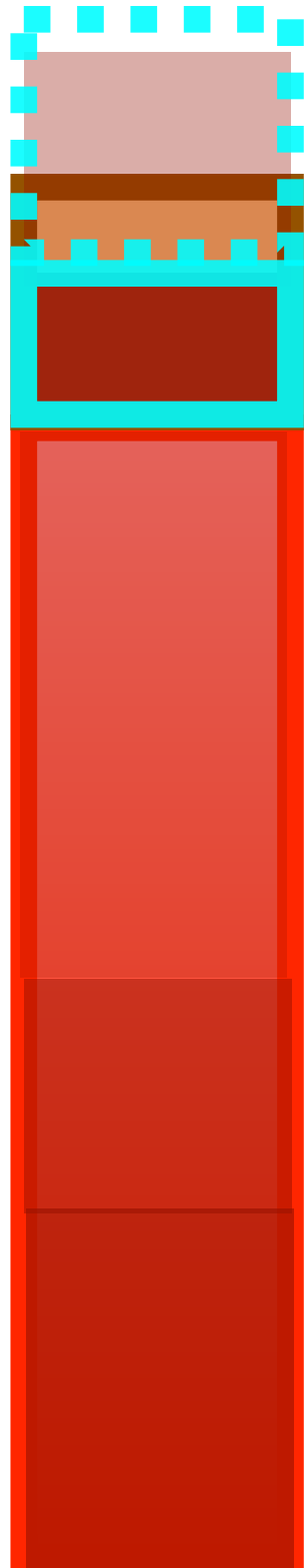
Ideen:

- Verwende die k wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.



Ideen:

- Verwende die k wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.
- Der Fehler ist durch *ein* Objekt beschränkt.

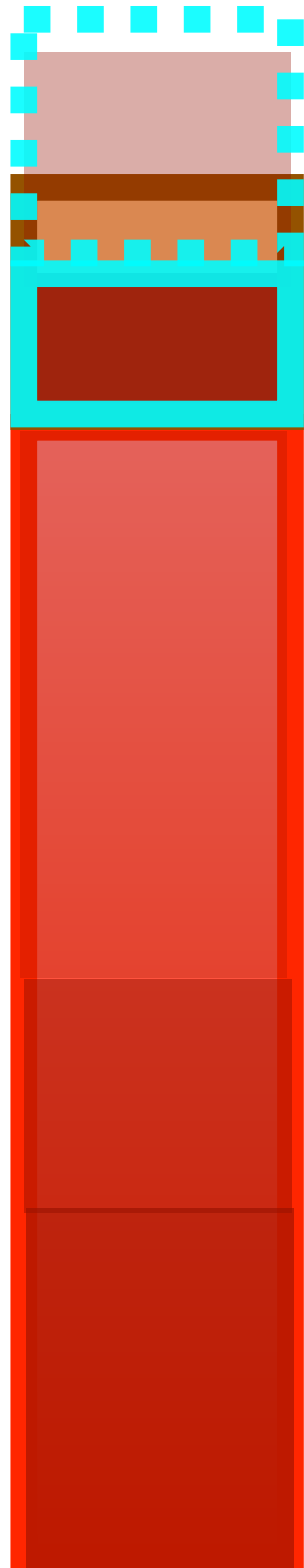


Ideen:

- Verwende die k wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.
- Der Fehler ist durch *ein* Objekt beschränkt.

Wie findet man die k wertvollsten Objekte?

- Enumeriere alle Teilmengen der Größe k ! $\rightarrow O(n^k)$ Teilmengen



Bessere Approximation von Knapsack

Algorithmus 4.6 GREEDY_k

Eingabe: $z_1, \dots, z_n, Z, p_1, \dots, p_n$ [Parameter k fixiert]

Ausgabe: $S \subseteq \{1, \dots, n\}$ mit $\sum_{i \in S} z_i \leq Z$ und Wert $G_k := \sum_{i \in S} p_i$

1: $G_k := 0, S := \emptyset$

2: **for all** $\bar{S} \subseteq \{1, \dots, n\}$ mit $|\bar{S}| \leq k$ **do**

3: **if** $(\sum_{i \in \bar{S}} z_i \leq Z)$ **then**

4: $G_k := \max\{G_k, \sum_{i \in \bar{S}} p_i + \text{GREEDY}_0(\{z_i | i \notin \bar{S}\}, Z - \sum_{i \in \bar{S}} z_i, \{p_i | i \notin \bar{S}\})\};$

5: Update S ;

6: **return** G_k, S

Gütegarantie

Satz 4.7. GREEDY_k ist ein $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

Definition 4.5 (Approximationsalgorithmus).

1. Für ein Maximierungsproblem MAX ist ein Algorithmus ALG ein c -Approximationsalgorithmus für MAX, wenn für jede Instanz I von MAX



a) ALG in polynomieller Zeit in der Größe von I eine zulässige Lösung mit Wert ALG liefert

b) für den Vergleich mit dem zugehörigen Optimalwert $\text{OPT}(I)$ gilt: $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$ (dabei ist $c \leq 1$)

Nicht 2^n , sondern n^2

(a) Die Laufzeit ist nicht mehr als $O(n^{k+2})$ - also polynomiell.

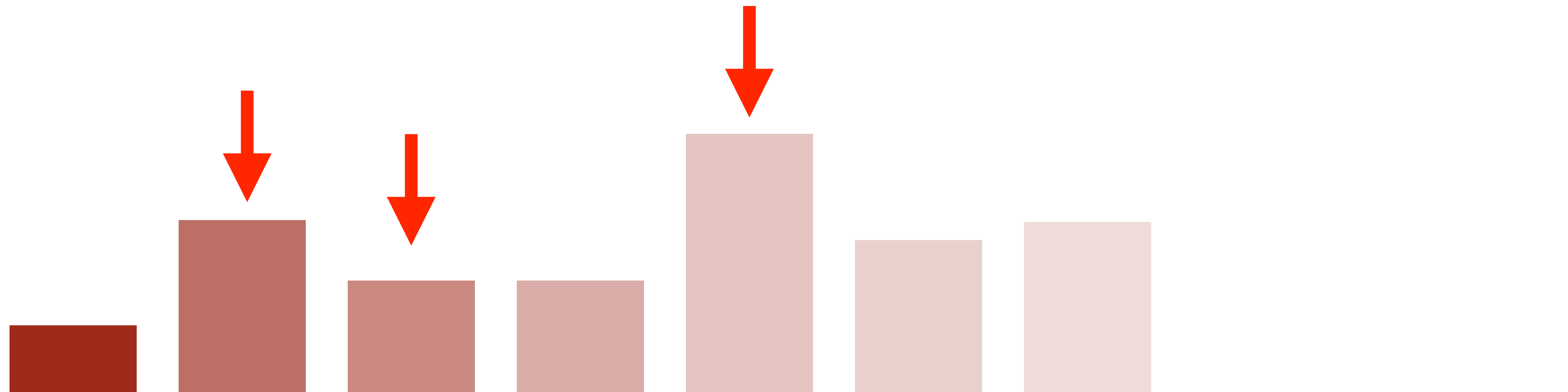
Gütegarantie

Satz 4.7. GREEDY_k ist ein $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert $\text{OPT}(I)$ gilt: $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$ (dabei ist $c \leq 1$)

Betrachte OPT mit $S^* \subseteq \{1, \dots, n\}$.

$|S^*| \leq k \implies$



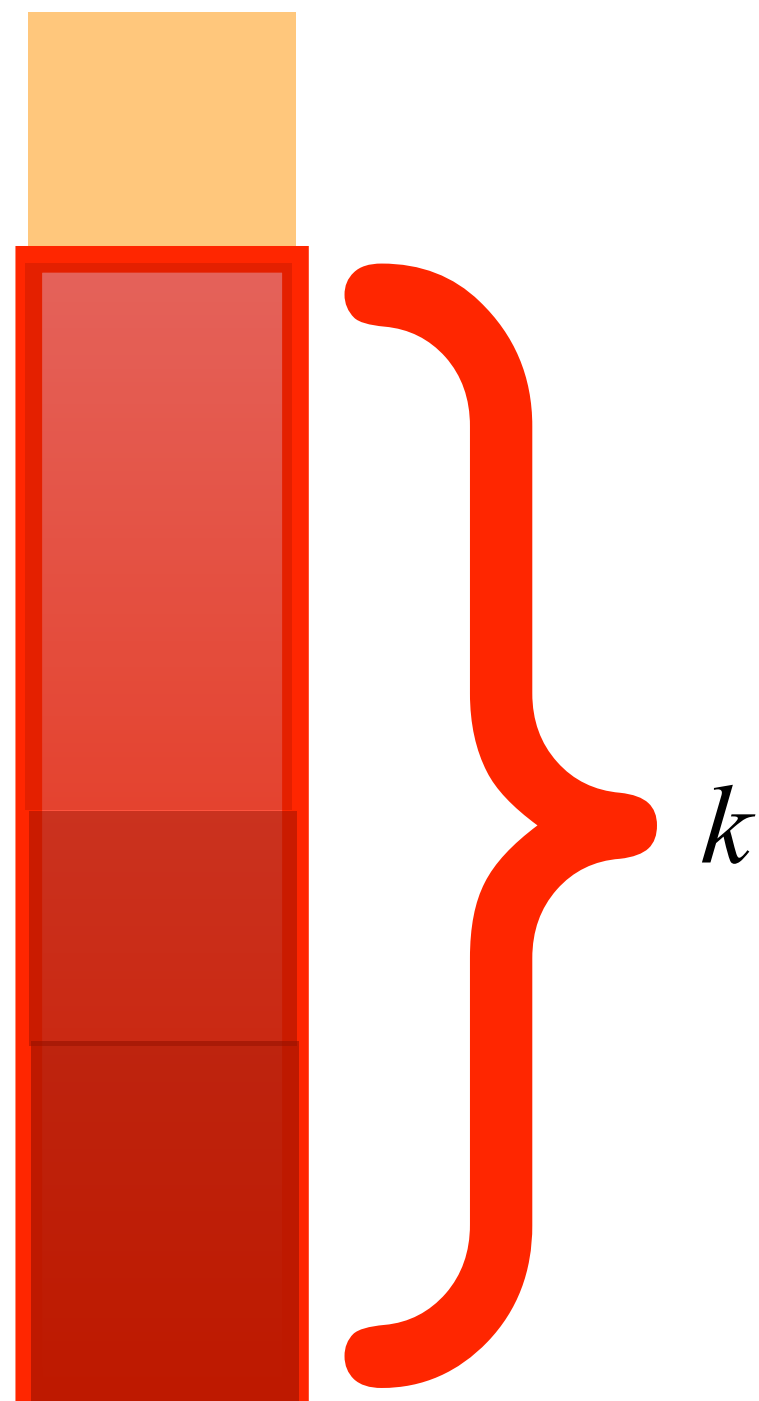
Gütegarantie

Satz 4.7. GREEDY_k ist ein $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert $\text{OPT}(I)$ gilt: $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$ (dabei ist $c \leq 1$)

Betrachte OPT mit $S^* \subseteq \{1, \dots, n\}$.

$|S^*| \leq k \implies \text{OPT wird gefunden.}$



Gütegarantie

Satz 4.7. GREEDY_k ist ein $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

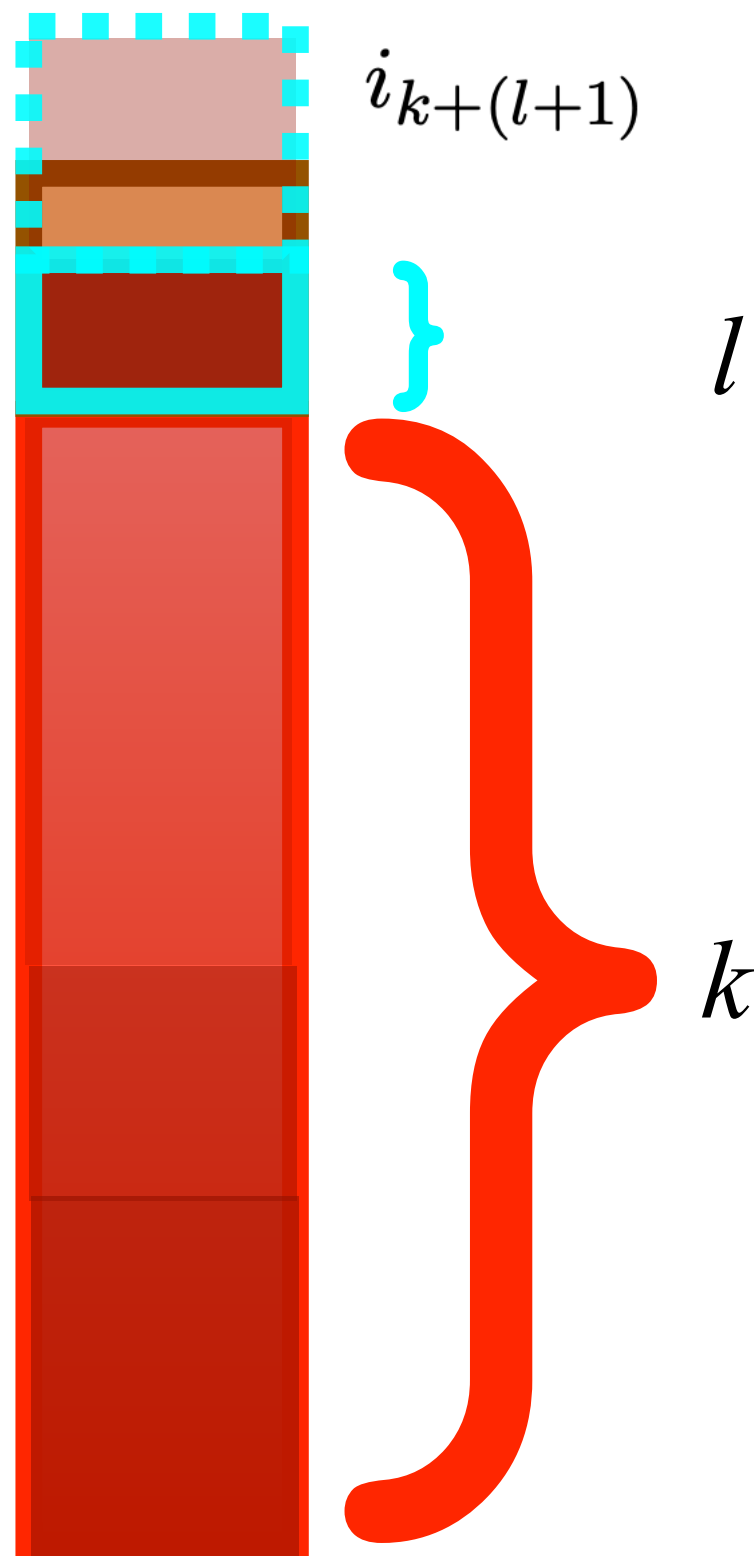
b) für den Vergleich mit dem zugehörigen Optimalwert $\text{OPT}(I)$ gilt: $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$ (dabei ist $c \leq 1$)

Betrachte OPT mit $S^* \subseteq \{1, \dots, n\}$.

$|S^*| > k \implies$

$$\begin{aligned} \text{a) } & \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT} \\ \text{b) } & \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k \\ \text{c) } & p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k \end{aligned}$$

$$\begin{aligned} & G_k + \frac{1}{k} G_k \geq \text{OPT} \\ \Leftrightarrow G_k & \geq \frac{k}{k+1} \text{OPT} \\ & = \left(1 - \frac{1}{k+1}\right) \text{OPT} \end{aligned}$$



Zusatzbemerkungen

Man kann zeigen:

- Für GREEDY_k ist der Faktor $(1 - \frac{1}{k+1})$ bestmöglich.
- Wenn man G_0 in Zeile 4 von Algorithmus 4.6 durch G'_0 ersetzt, bekommt man einen Approximationsalgorithmus mit Gütegarantie $(1 - \frac{1}{k+2})$.

Approximationschemata

Man sieht: Für jedes feste $\epsilon > 0$ gibt es einen polynomiellen Algorithmus für Knapsack, der eine $(1 - \epsilon)$ -Approximation liefert. Das motiviert:

Definition 4.8 (PTAS). Ein polynomielles Approximationsschema (Engl.: Polynomialtime approximation scheme. Kurz: PTAS) für ein Optimierungsproblem ist eine Familie von Algorithmen, die für jedes beliebige, aber feste $\epsilon > 0$ einen $(1 - \epsilon)$ -Approximationsalgorithmus (bzw. $(1 + \epsilon)$) liefert.

Korollar 4.9. $\{\text{GREEDY}_k \mid k \in \mathbb{N}\}$ ist ein PTAS für Knapsack.

4.4 Ausblicke

Approximationsfaktoren

0-1 Knapsack

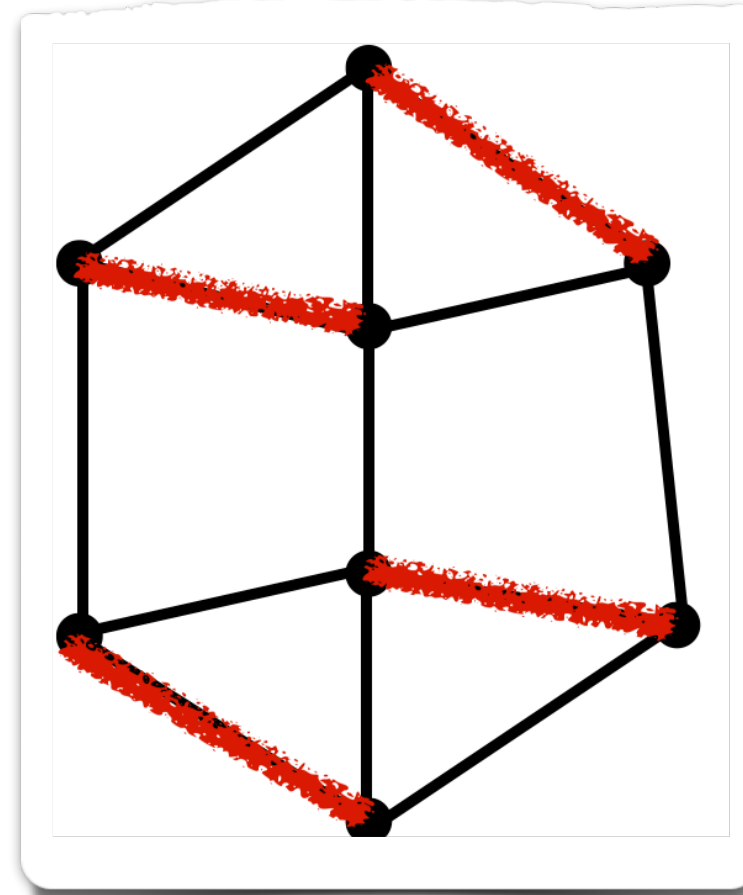
1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
8	10	32	11	12	13
40	5	28	20	16	16
9	14	3	9	24	10
32	20	15	40	4	
2	3	10			

Kann Knut die Klausur bestehen?

Max

$$1 - \frac{1}{k + 2}$$

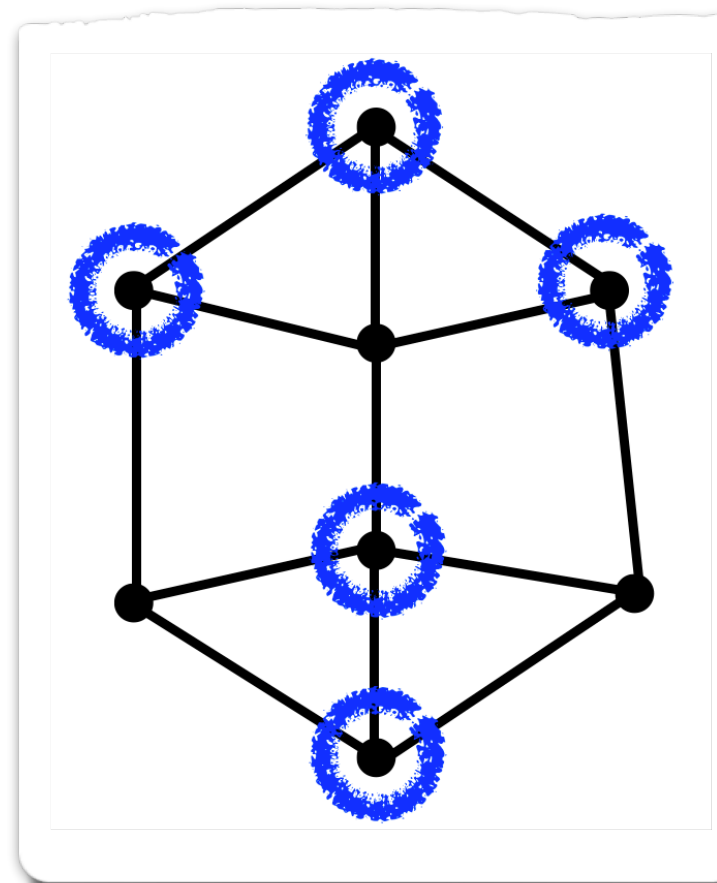
Matching



Max

1

Vertex Cover

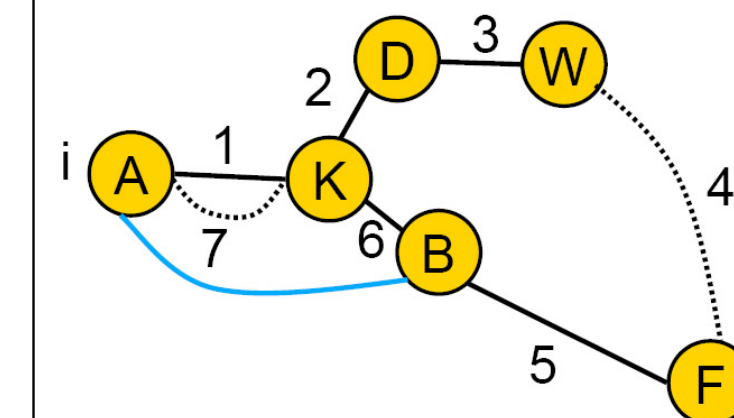


Min

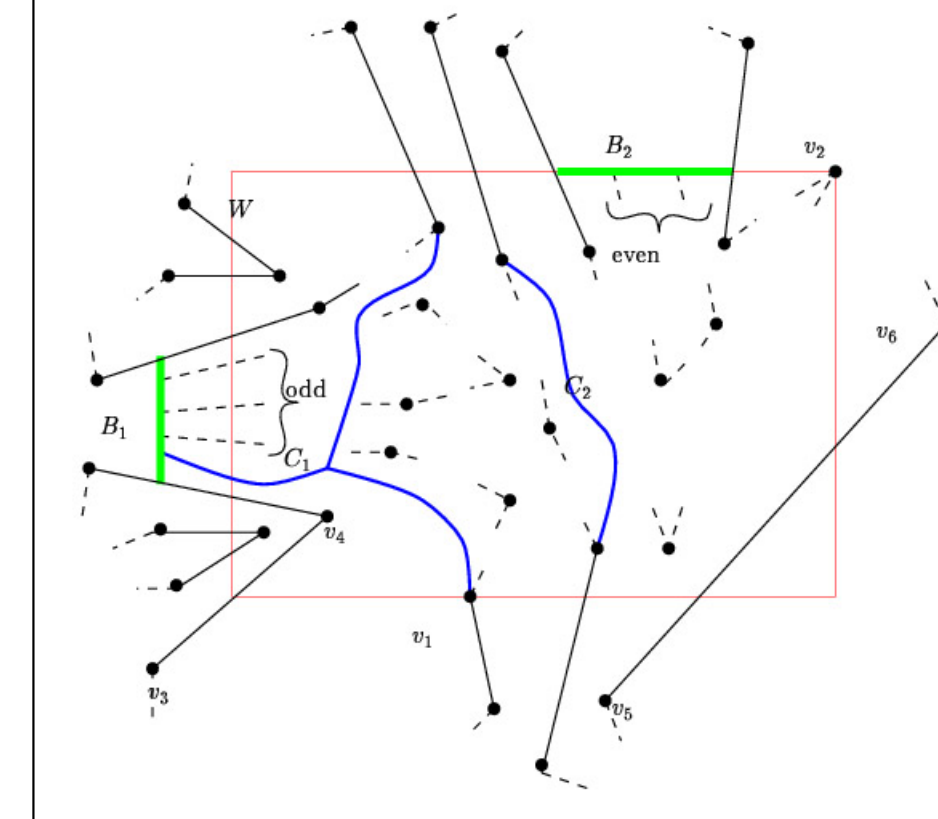
2

Traveling Salesman

Beispiel für CH-Heuristik



Example Subproblem (TSP)



Min

3
2

1 + ε

Vielen Dank!

s.fekete@tu-bs.de