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

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

3 Branch-and-Bound

Algorithmen und Datenstrukturen 2
Sommer 2021

Prof. Dr. Sándor Fekete

3.2 Methodik

Ideen

Ideen

Grundideen:

Ideen

Grundideen:

1. Enumeriere die möglichen Teilmengen in einem Enumerationsbaum

Ideen

Grundideen:

1. Enumeriere die möglichen Teilmengen in einem Enumerationsbaum
2. Behalte den Zielfunktionswert im Auge:

Ideen

Grundideen:

1. Enumeriere die möglichen Teilmengen in einem Enumerationsbaum
2. Behalte den Zielfunktionswert im Auge:
    - Untere Schranke: Erreichter Zielfunktionswert im ganzen Baum

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

Ideen

Grundideen:

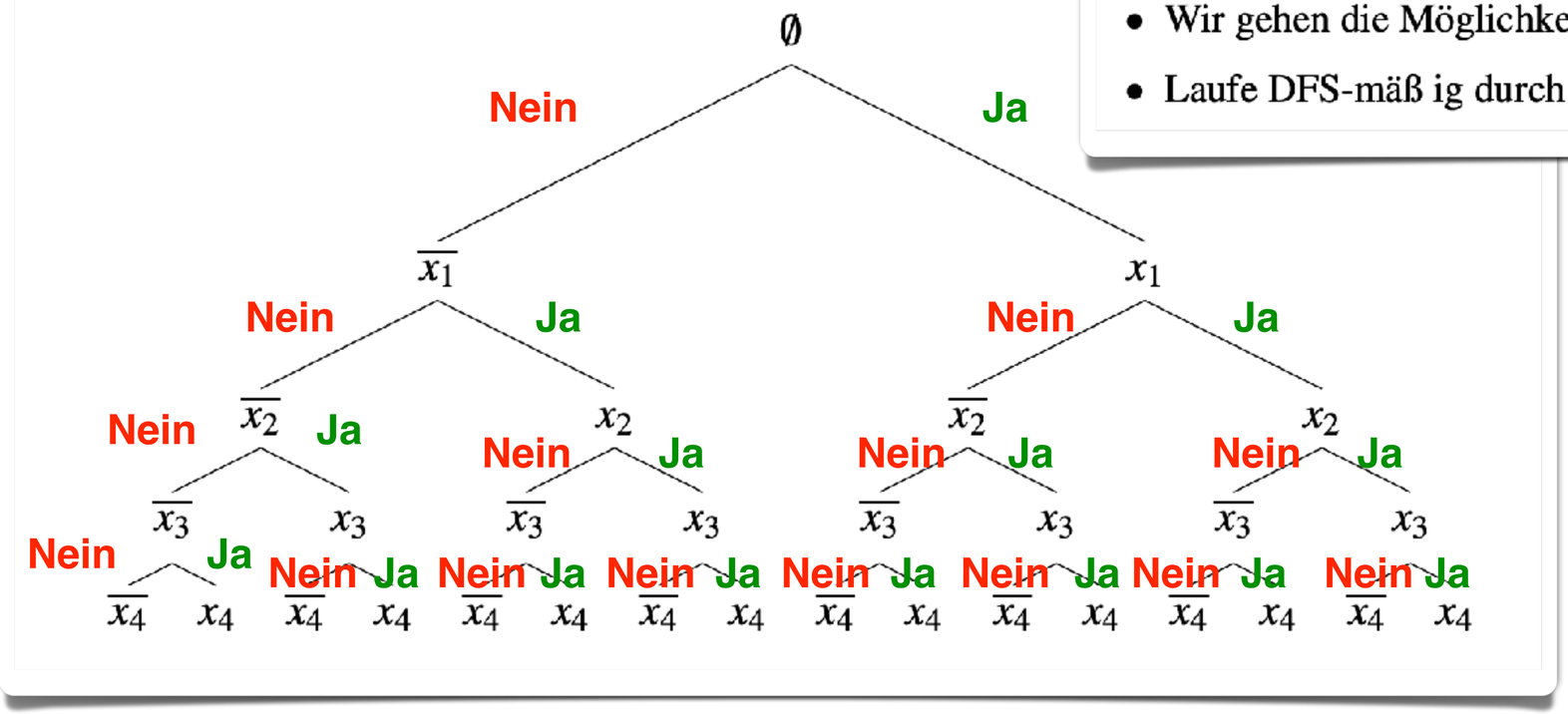
1. Enumeriere die möglichen Teilmengen in einem Enumerationsbaum
2. Behalte den Zielfunktionswert im Auge:
   - Untere Schranke: Erreichter Zielfunktionswert im ganzen Baum
   - Obere Schranke: Erreichbarer Zielfunktionswert im jeweiligen Baum
3. Beide Schranken sollten möglichst einfach und schnell zu berechnen sein

Ideen

Grundideen:

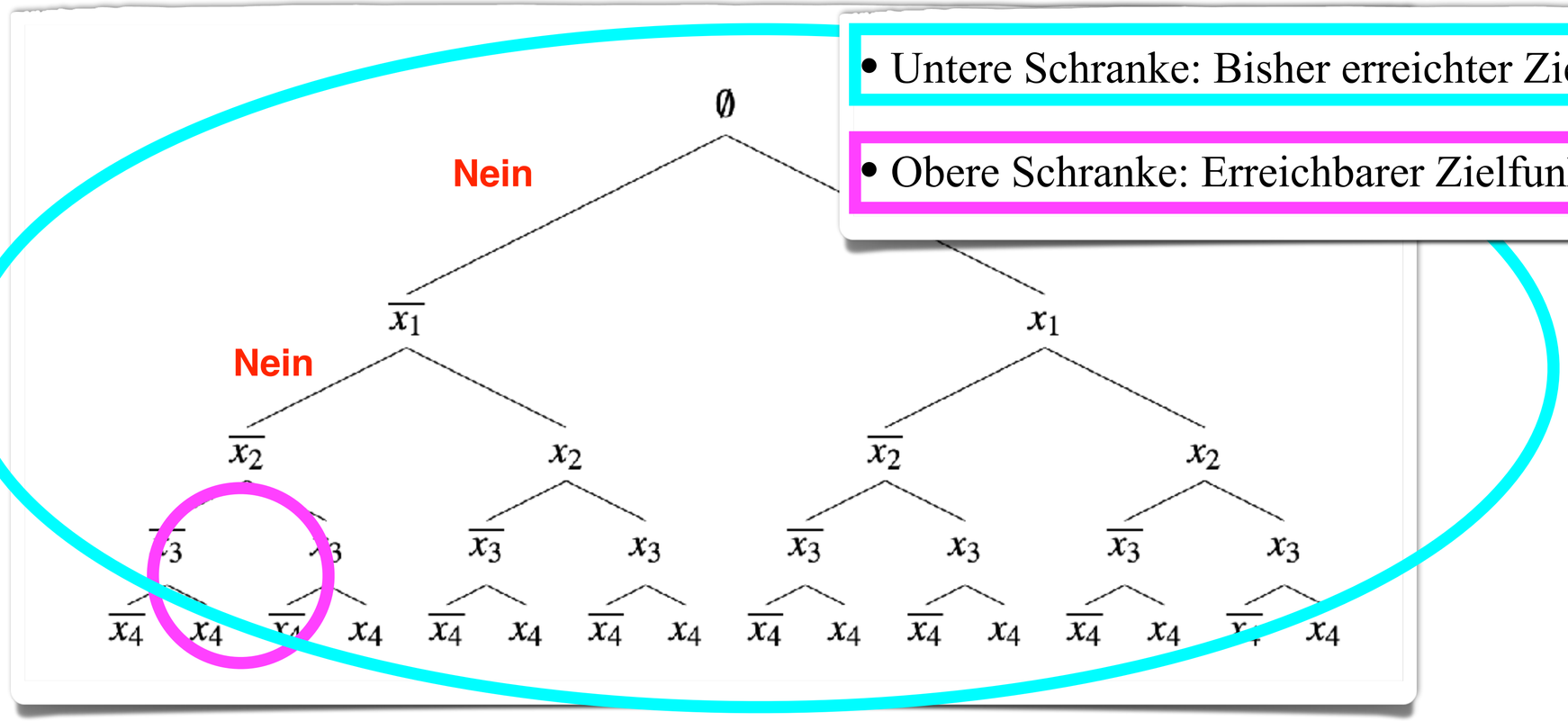
1. Enumeriere die möglichen Teilmengen in einem Enumerationsbaum
2. Behalte den Zielfunktionswert im Auge:
   - Untere Schranke: Erreichter Zielfunktionswert im ganzen Baum
   - Obere Schranke: Erreichbarer Zielfunktionswert im jeweiligen Baum
3. Beide Schranken sollten möglichst einfach und schnell 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

1. Enumerationsbaum

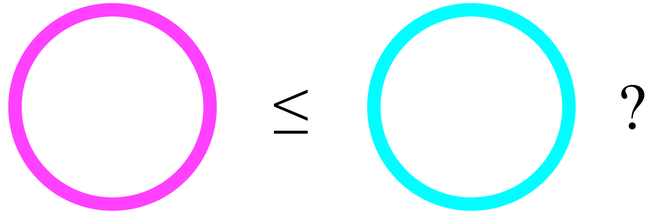


- Probiere nacheinander für $i = 1, ..., 7$, ob $x_i = 0$ oder $x_i = 1$
- Wir gehen die Möglichkeiten baumartig durch:
- Laufe DFS-mäß ig durch den Baum, d.h. arbeite die Entscheidungen im Stack ab.

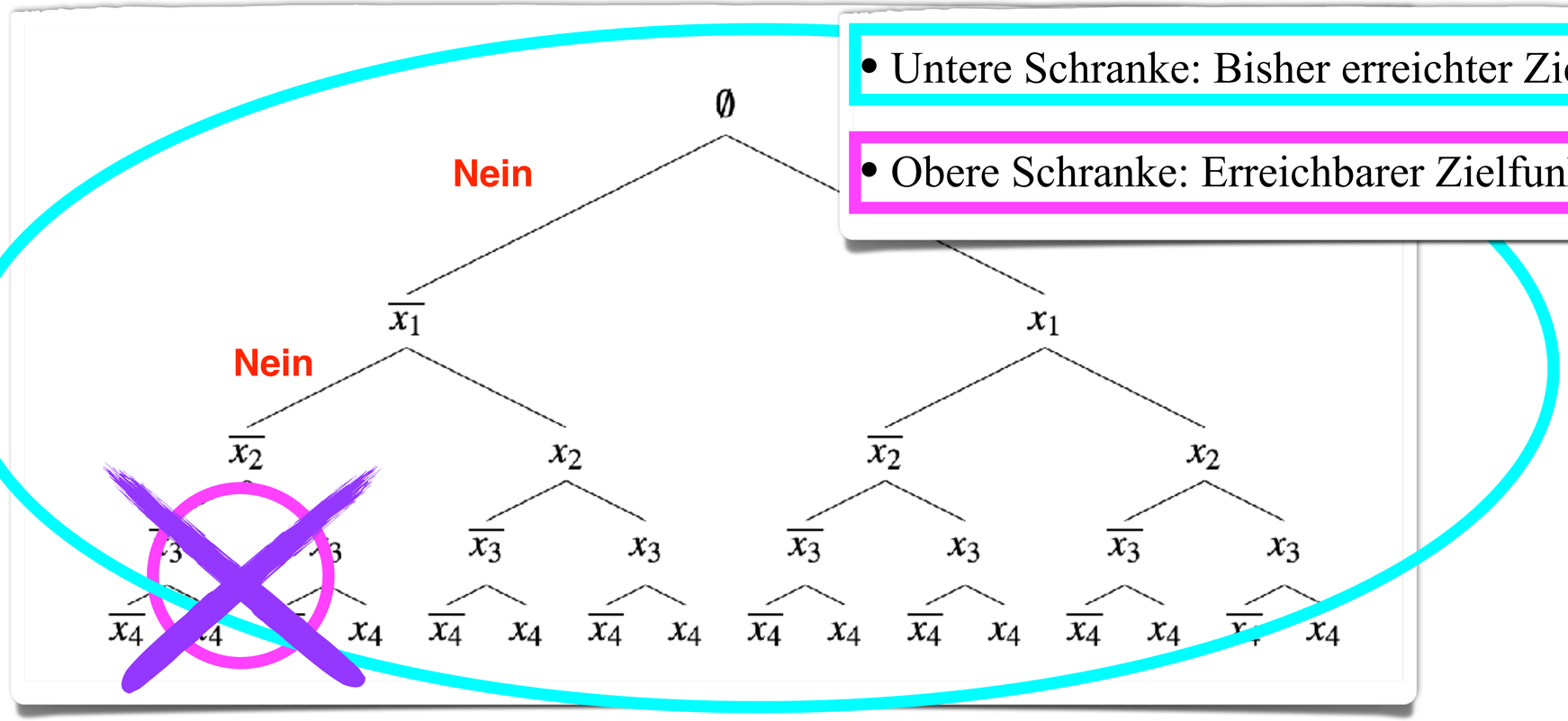
4. Einsatz von Schranken - Grundidee



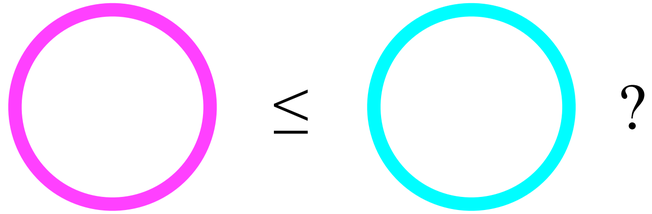
- Untere Schranke: Bisher erreichter Zielfunktionswert im ganzen Baum
- Obere Schranke: Erreichbarer Zielfunktionswert im jeweiligen Teilbaum



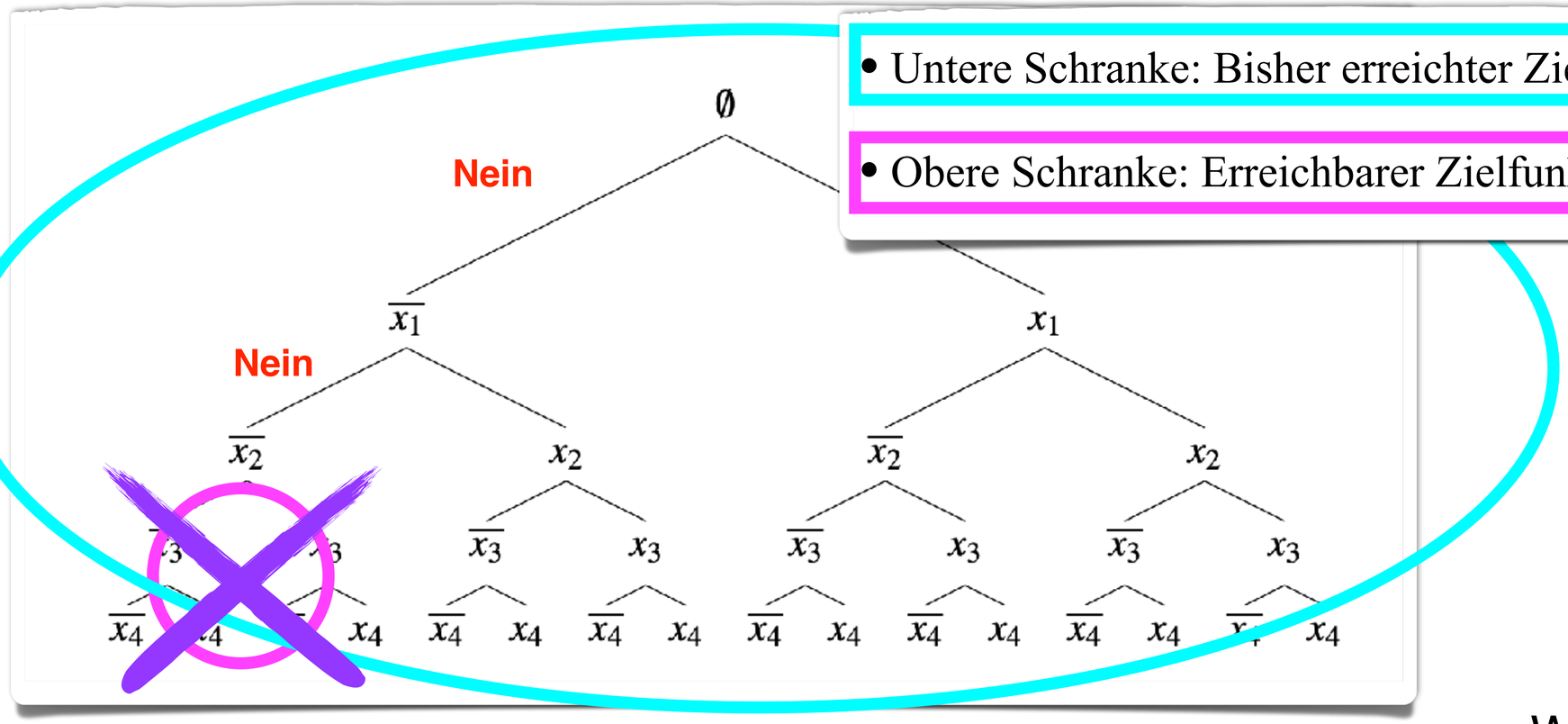
4. Einsatz von Schranken - Grundidee



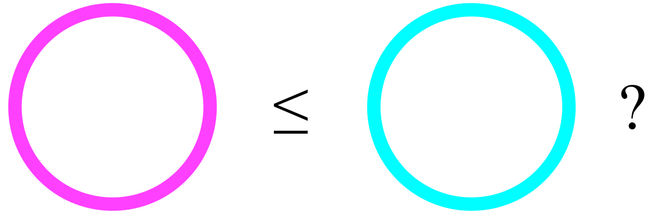
- Untere Schranke: Bisher erreichter Zielfunktionswert im ganzen Baum
- Obere Schranke: Erreichbarer Zielfunktionswert im jeweiligen Teilbaum



4. Einsatz von Schranken - Grundidee

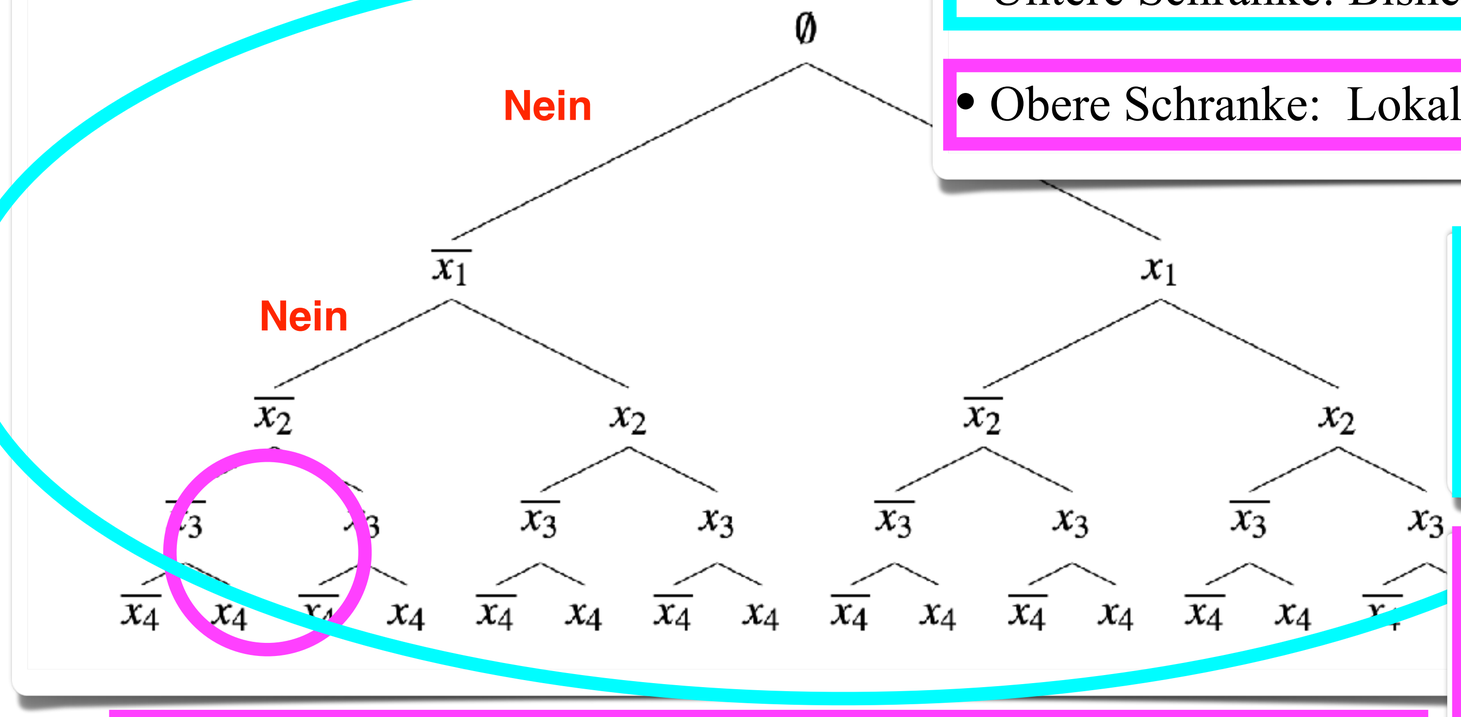


- Untere Schranke: Bisher erreichter Zielfunktionswert im ganzen Baum
- Obere Schranke: Erreichbarer Zielfunktionswert im jeweiligen Teilbaum



Was sind geeignete Schranken?

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

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

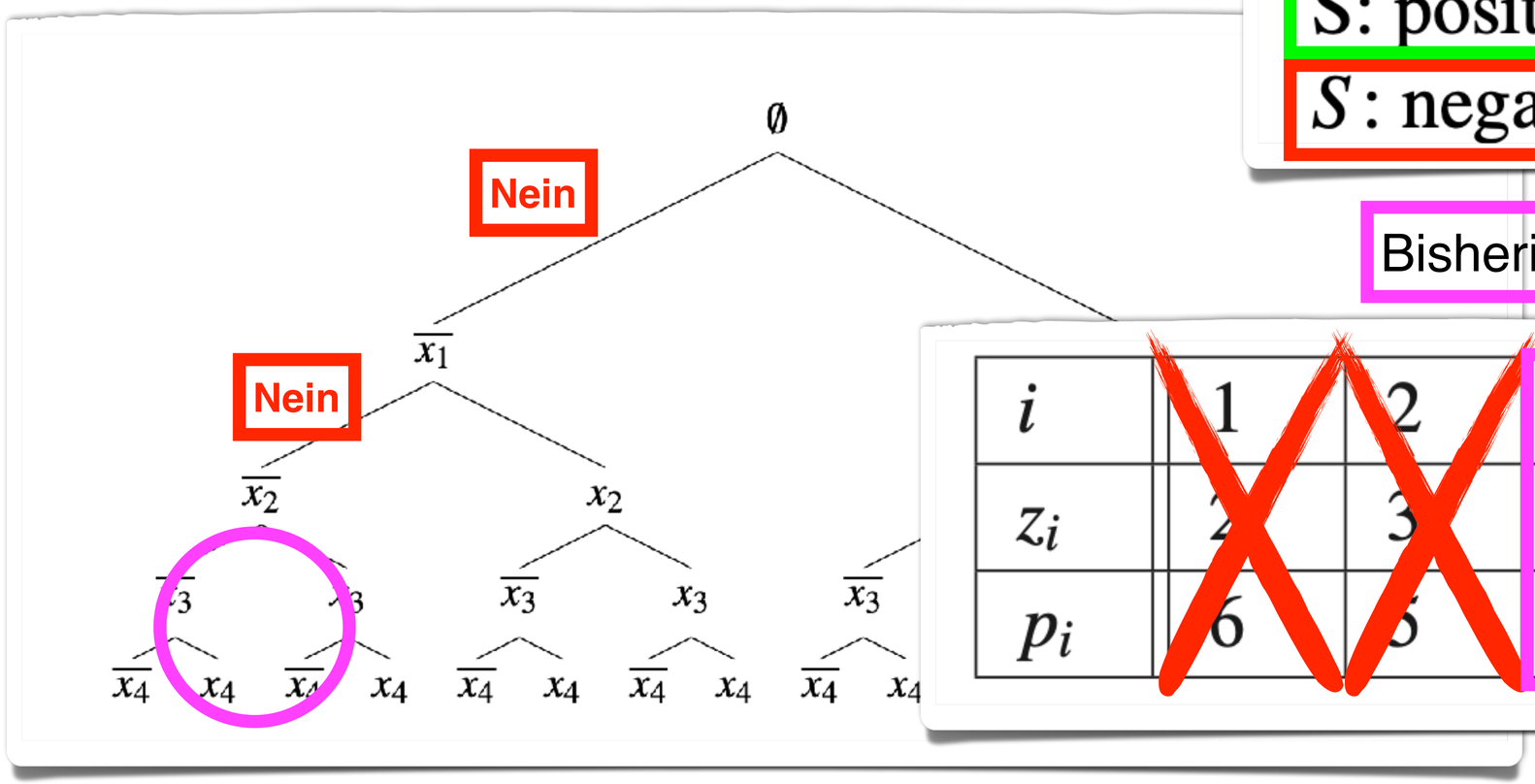
i	6	4	13	12	9	3	15	8	16	10	1	14	5	11	2	7
z_i	4	8	16	20	8	40	40	40	24	32	20	20	16	28	32	32
p_i	4	5	10	9	2	10	10	9	4	5	3	3	2	3	3	2

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

Heuristik: Schnell zu lösen, aber nicht notwendig optimal

Relaxierung: Schnell und optimal zu lösen durch größere Lösungsmenge

3. Berechnung von Schranken II



S: positiv fixiert ($x_i = 1$)

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

3.3 Durchführung

Beispiel: Knapsack

Beispiel 2.4 (Knapsackproblem).

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

1	2	3	4	5	6	7
2	3	6	7	5	9	4
6	5	8	9	6	7	3

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

Untere Schranke: Greedy (ganzzahlig)

Untere Schranke: Greedy (ganzzahlig)

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

Untere Schranke: Greedy (ganzzahlig)

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

Z=9

Untere Schranke: Greedy (ganzzahlig)

S=Ø

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

Z=9

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

Untere Schranke: Greedy (ganzzahlig)

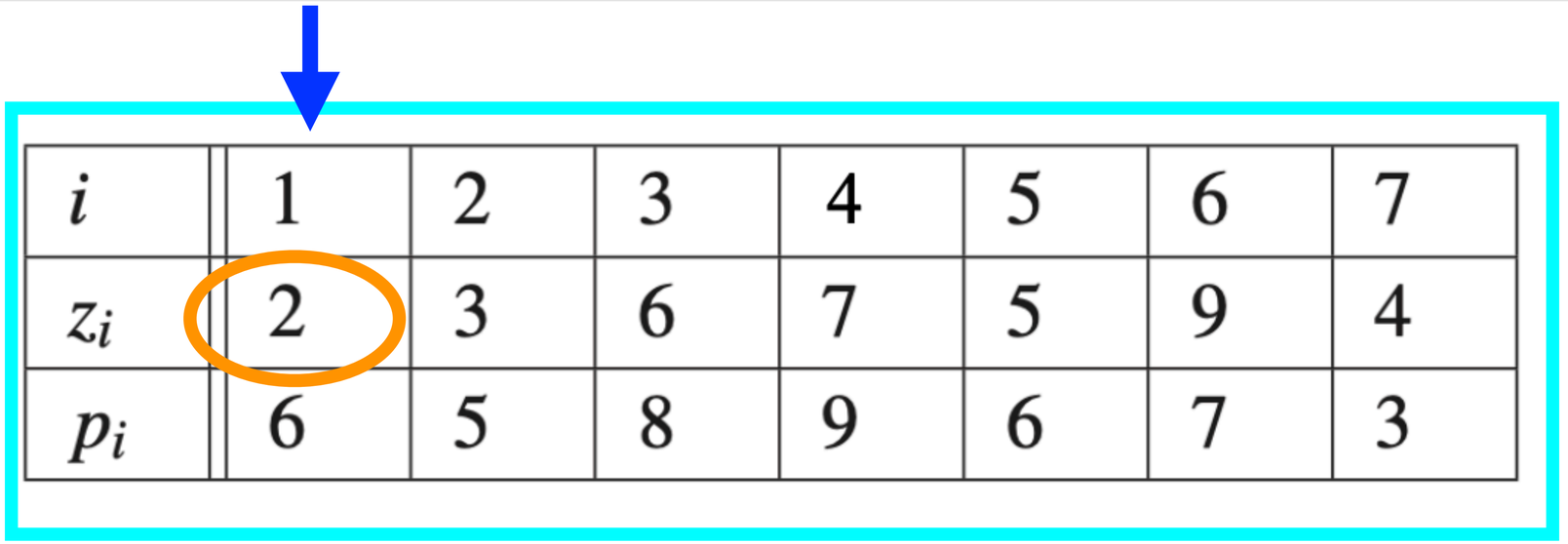
S=Ø

$\bar{S}$=Ø

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

Z=9

Untere Schranke: Greedy (ganzzahlig)



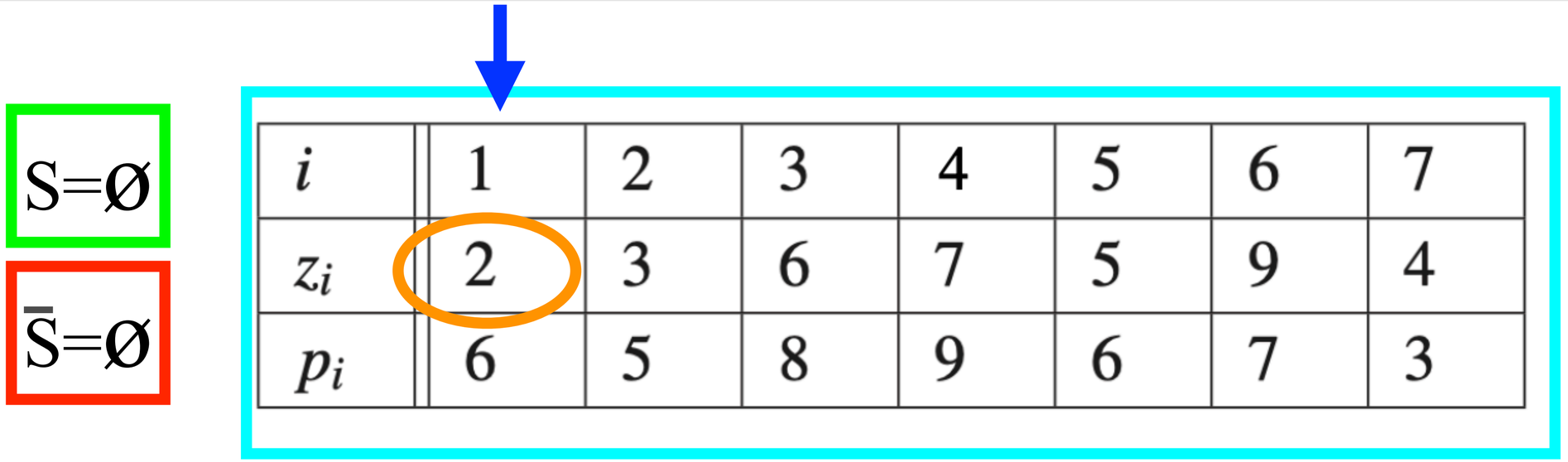
$S=\emptyset$

$\bar{S}=\emptyset$

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

$Z=9$

Untere Schranke: Greedy (ganzzahlig)



S=Ø

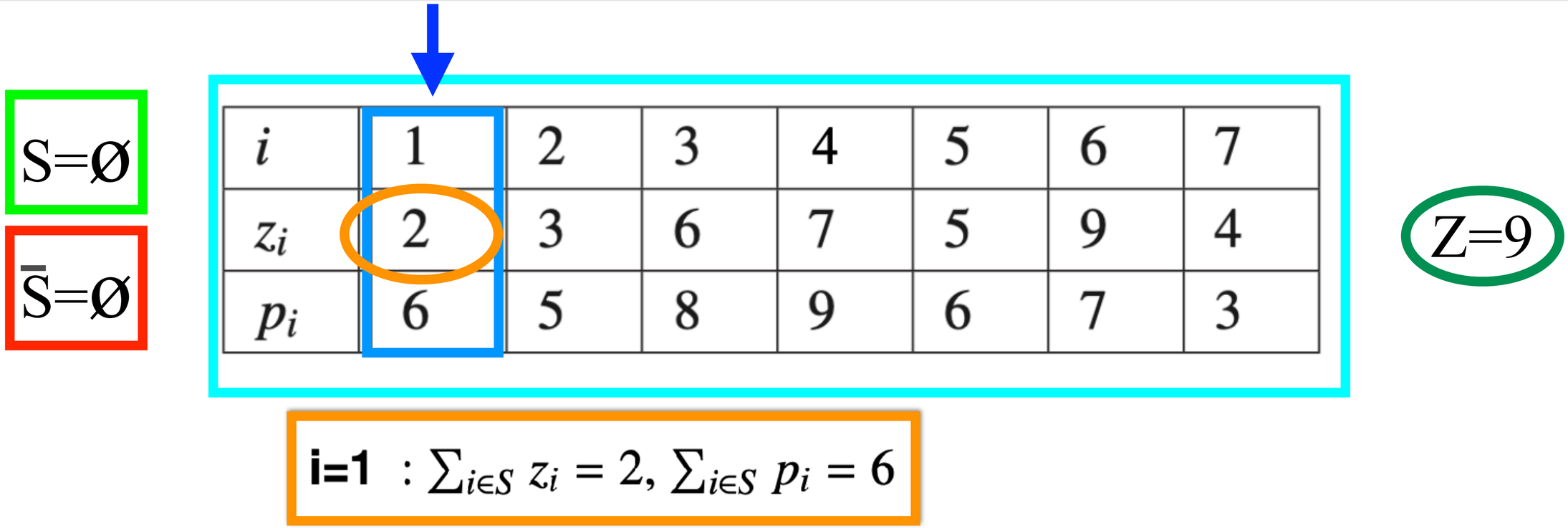
$\bar{S}$=Ø

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

Z=9

i=1 : $\sum_{i \in S} z_i = 2, \sum_{i \in S} p_i = 6$

Untere Schranke: Greedy (ganzzahlig)



S=Ø

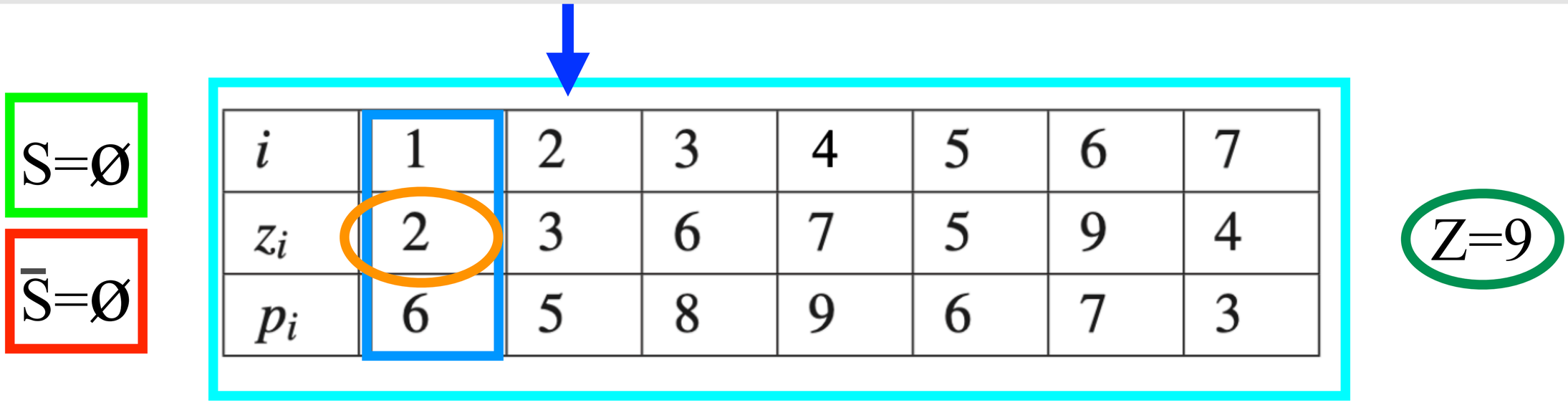
$\bar{S}$=Ø

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

Z=9

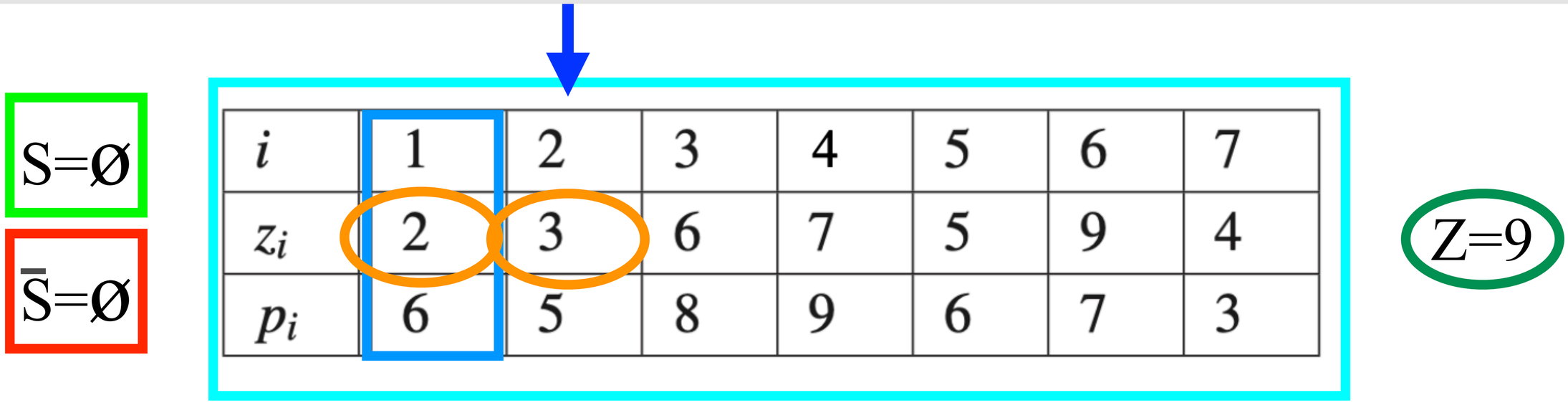
i=1 : $\sum_{i \in S} z_i = 2, \sum_{i \in S} p_i = 6$

Untere Schranke: Greedy (ganzzahlig)

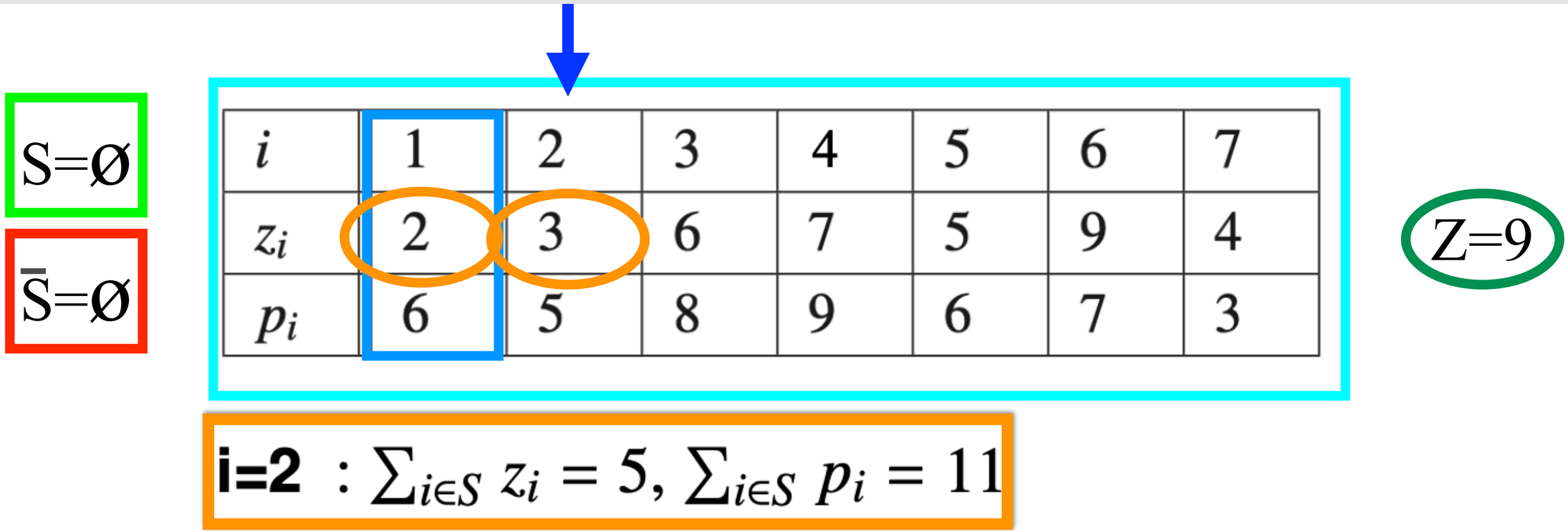


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

Untere Schranke: Greedy (ganzzahlig)



Untere Schranke: Greedy (ganzzahlig)



S=Ø

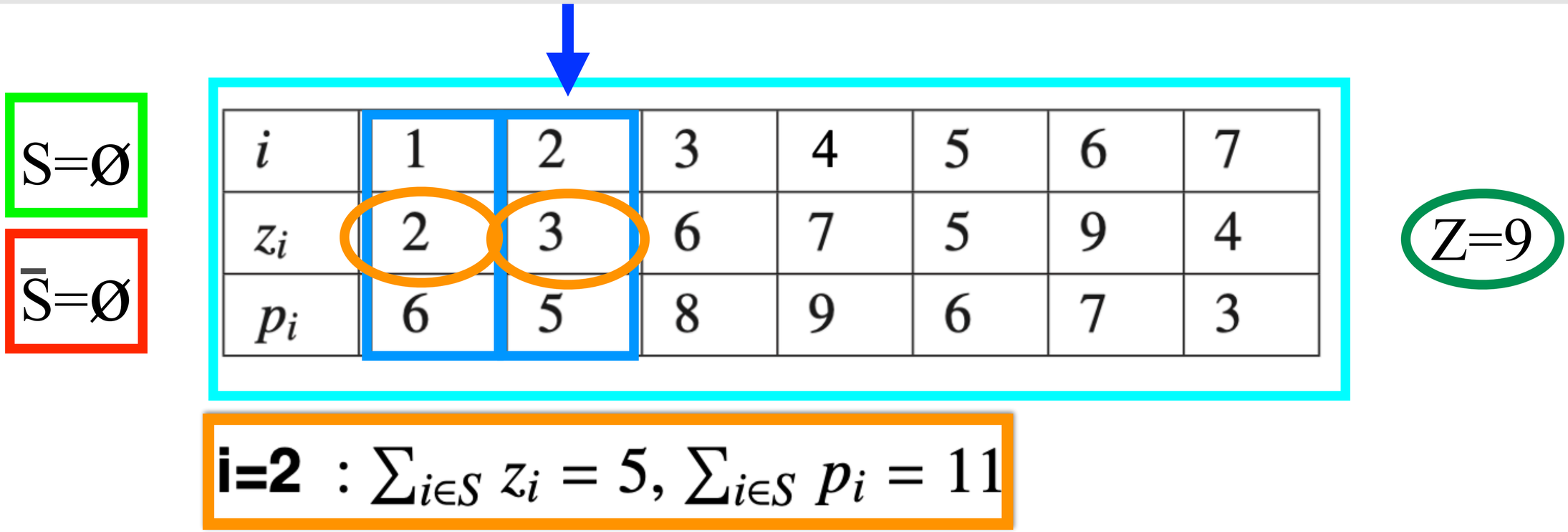
$\bar{S}$=Ø

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

Z=9

i=2 : $\sum_{i \in S} z_i = 5, \sum_{i \in S} p_i = 11$

Untere Schranke: Greedy (ganzzahlig)



Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=3 : $\sum_{i \in S} z_i = 11 > 9$, X

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

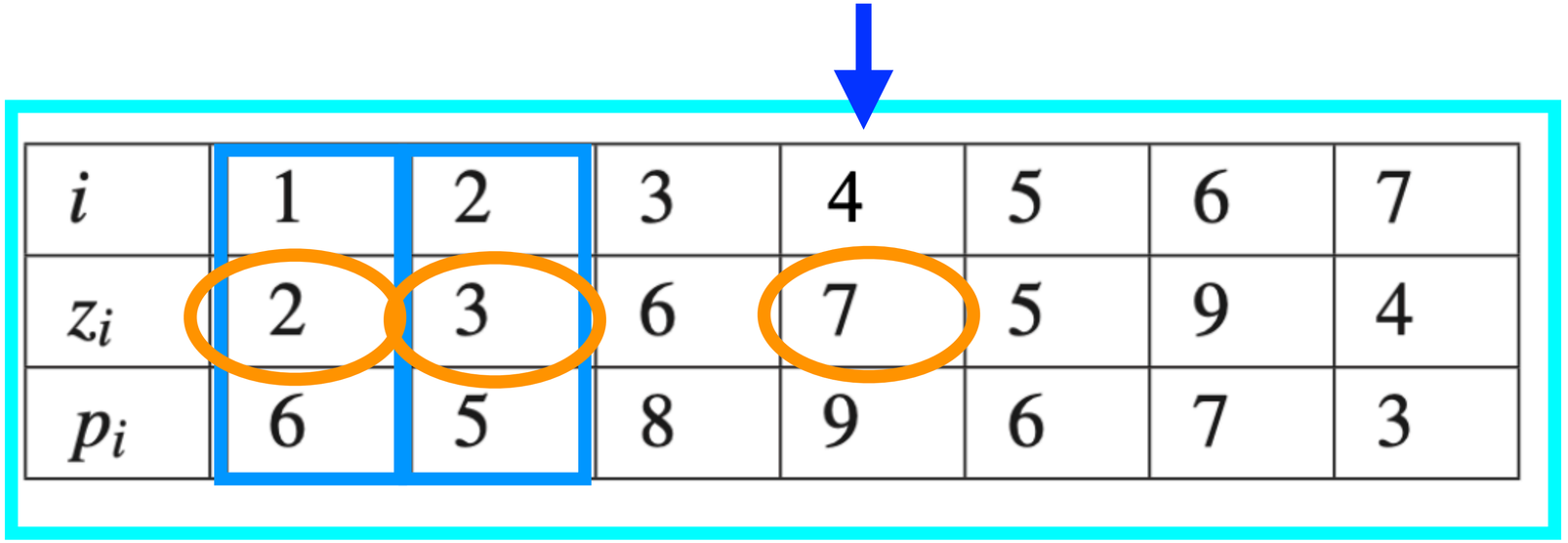
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

Z=9

Untere Schranke: Greedy (ganzzahlig)

$S=\emptyset$

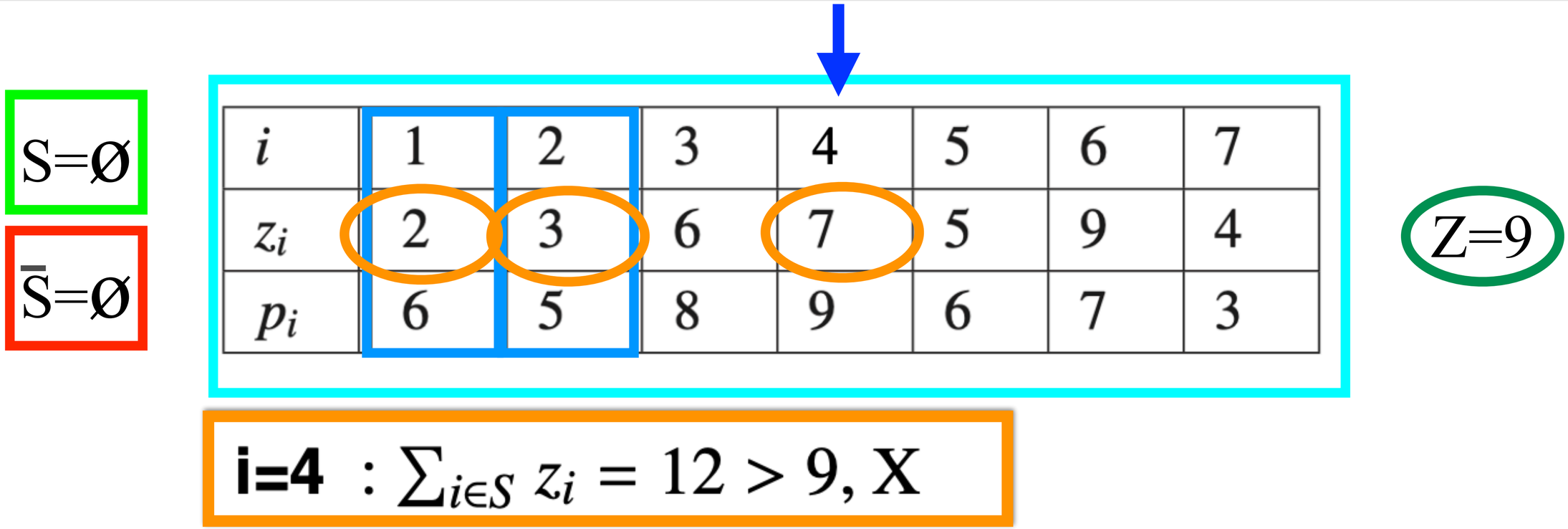
$\bar{S}=\emptyset$



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

$Z=9$

Untere Schranke: Greedy (ganzzahlig)



Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=5 : $\sum_{i \in S} z_i = 10 > 9, \text{X}$

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=6 : $\sum_{i \in S} z_i = 14 > 9, \text{X}$

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

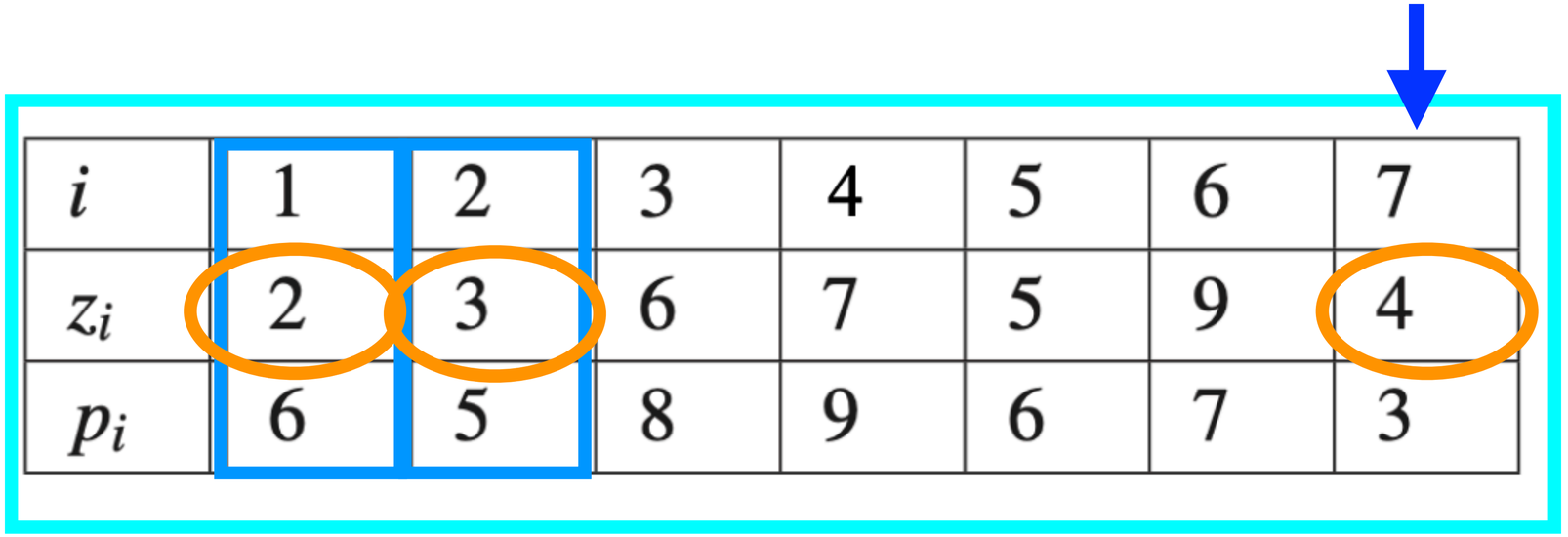
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

Z=9

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø



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

Z=9

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=7 : $\sum_{i \in S} z_i = 9, \sum_{i \in S} p_i = 14$

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=7 : $\sum_{i \in S} z_i = 9, \sum_{i \in S} p_i = 14$

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=7 : $\sum_{i \in S} z_i = 9, \sum_{i \in S} p_i = 14$

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=7 : $\sum_{i \in S} z_i = 9, \sum_{i \in S} p_i = 14$

Untere Schranke: Greedy (ganzzahlig)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=7 : $\sum_{i \in S} z_i = 9, \sum_{i \in S} p_i = 14$

LB=14

Obere Schranke: Greedy (fraktional)

Obere Schranke: Greedy (fraktional)

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

Obere Schranke: Greedy (fraktional)

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

Z=9

Obere Schranke: Greedy (fraktional)

S=Ø

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

Z=9

Obere Schranke: Greedy (fraktional)

S=Ø

$\bar{S}$=Ø

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

Z=9

Obere Schranke: Greedy (fraktional)

$S=\emptyset$

$\bar{S}=\emptyset$

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

$Z=9$

Obere Schranke: Greedy (fraktional)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=1 : $\sum_{i \in S} z_i = 2, \sum_{i \in S} p_i = 6$

Obere Schranke: Greedy (fraktional)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=1 : $\sum_{i \in S} z_i = 2, \sum_{i \in S} p_i = 6$

Obere Schranke: Greedy (fraktional)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=1 : $\sum_{i \in S} z_i = 2, \sum_{i \in S} p_i = 6$

Obere Schranke: Greedy (fraktional)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=2 : $\sum_{i \in S} z_i = 5, \sum_{i \in S} p_i = 11$

Obere Schranke: Greedy (fraktional)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=2 : $\sum_{i \in S} z_i = 5, \sum_{i \in S} p_i = 11$

Obere Schranke: Greedy (fraktional)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=2 : $\sum_{i \in S} z_i = 5, \sum_{i \in S} p_i = 11$

Obere Schranke: Greedy (fraktional)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=2 : $\sum_{i \in S} z_i = 5, \sum_{i \in S} p_i = 11$

Bleibt: Z-$\sum_{i \in S} z_i = 4$, also $x_3 = \frac{2}{3} (= \frac{4}{6} = \frac{Z - \sum_{i \in S} z_i}{z_3})$

Obere Schranke: Greedy (fraktional)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=2 : $\sum_{i \in S} z_i = 5, \sum_{i \in S} p_i = 11$

Bleibt: $Z - \sum_{i \in S} z_i = 4$, also $x_3 = \frac{2}{3} (= \frac{4}{6} = \frac{Z - \sum_{i \in S} z_i}{z_3})$

Obere Schranke: Greedy (fraktional)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=2 : $\sum_{i \in S} z_i = 5, \sum_{i \in S} p_i = 11$

Bleibt: $Z - \sum_{i \in S} z_i = 4$, also $x_3 = \frac{2}{3} (= \frac{4}{6} = \frac{Z - \sum_{i \in S} z_i}{z_3})$

Obere Schranke: Greedy (fraktional)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=2 : $\sum_{i \in S} z_i = 5, \sum_{i \in S} p_i = 11$

Bleibt: $Z - \sum_{i \in S} z_i = 4$; also $x_3 = \frac{2}{3} (= \frac{4}{6} = \frac{Z - \sum_{i \in S} z_i}{z_3})$

Obere Schranke: Greedy (fraktional)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=2 : $\sum_{i \in S} z_i = 5, \sum_{i \in S} p_i = 11$

Bleibt: $Z - \sum_{i \in S} z_i = 4$; also $x_3 = \frac{2}{3} (= \frac{4}{6} = \frac{Z - \sum_{i \in S} z_i}{z_3})$

Obere Schranke: Greedy (fraktional)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=2 : $\sum_{i \in S} z_i = 5, \sum_{i \in S} p_i = 11$

Bleibt: $Z - \sum_{i \in S} z_i = 4$; also $x_3 = \frac{2}{3} (= \frac{4}{6} = \frac{Z - \sum_{i \in S} z_i}{z_3})$

Obere Schranke: Greedy (fraktional)

$S=\emptyset$

$\bar{S}=\emptyset$

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

$Z=9$

i=2 : $\sum_{i\in S} z_i = 5, \sum_{i\in S} p_i = 11$

Bleibt: $Z-\sum_{i\in S} z_i = 4$; also $x_3 = \frac{2}{3}\left(= \frac{4}{6} = \frac{Z-\sum_{i\in S} z_i}{z_3}\right)$

Obere Schranke: Greedy (fraktional)

$S=\emptyset$

$\bar{S}=\emptyset$

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

$Z=9$

i=2 : $\sum_{i\in S} z_i = 5, \sum_{i\in S} p_i = 11$

Bleibt: $Z-\sum_{i\in S} z_i = 4$; also $x_3 = \frac{2}{3} (= \frac{4}{6} = \frac{Z-\sum_{i\in S} z_i}{z_3})$

Obere Schranke: Greedy (fraktional)

$S=\emptyset$

$\bar{S}=\emptyset$

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

$Z=9$

i=2 : $\sum_{i\in S} z_i = 5, \sum_{i\in S} p_i = 11$

Bleibt: $Z-\sum_{i\in S} z_i = 4$; also $x_3 = \frac{2}{3} (= \frac{4}{6} = \frac{Z-\sum_{i\in S} z_i}{z_3})$

Damit: $\sum_{i\in S} x_i z_i = 9, \sum_{i\in S} x_i p_i = 11 + \frac{2}{3} * 8 = 16,\overline{3}$

Obere Schranke: Greedy (fraktional)

$S=\emptyset$

$\bar{S}=\emptyset$

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

$Z=9$

i=2 : $\sum_{i\in S} z_i = 5, \sum_{i\in S} p_i = 11$

Bleibt: $Z-\sum_{i\in S} z_i = 4$; also $x_3 = \frac{2}{3} (= \frac{4}{6} = \frac{Z-\sum_{i\in S} z_i}{z_3})$

Damit: $\sum_{i\in S} x_i z_i = 9, \sum_{i\in S} x_i p_i = 11 + \frac{2}{3} * 8 = 16,\overline{3}$

Obere Schranke: Greedy (fraktional)

$S=\emptyset$

$\bar{S}=\emptyset$

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

$Z=9$

i=2 : $\sum_{i\in S} z_i = 5, \sum_{i\in S} p_i = 11$

Bleibt: $Z-\sum_{i\in S} z_i = 4$; also $x_3 = \frac{2}{3} (= \frac{4}{6} = \frac{Z-\sum_{i\in S} z_i}{z_3})$

Damit: $\sum_{i\in S} x_i z_i = 9, \sum_{i\in S} x_i p_i = 11 + \frac{2}{3} * 8 = 16,\overline{3}$

Obere Schranke: Greedy (fraktional)

S=Ø

$\bar{S}$=Ø

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

Z=9

i=2 : $\sum_{i\in S} z_i = 5, \sum_{i\in S} p_i = 11$

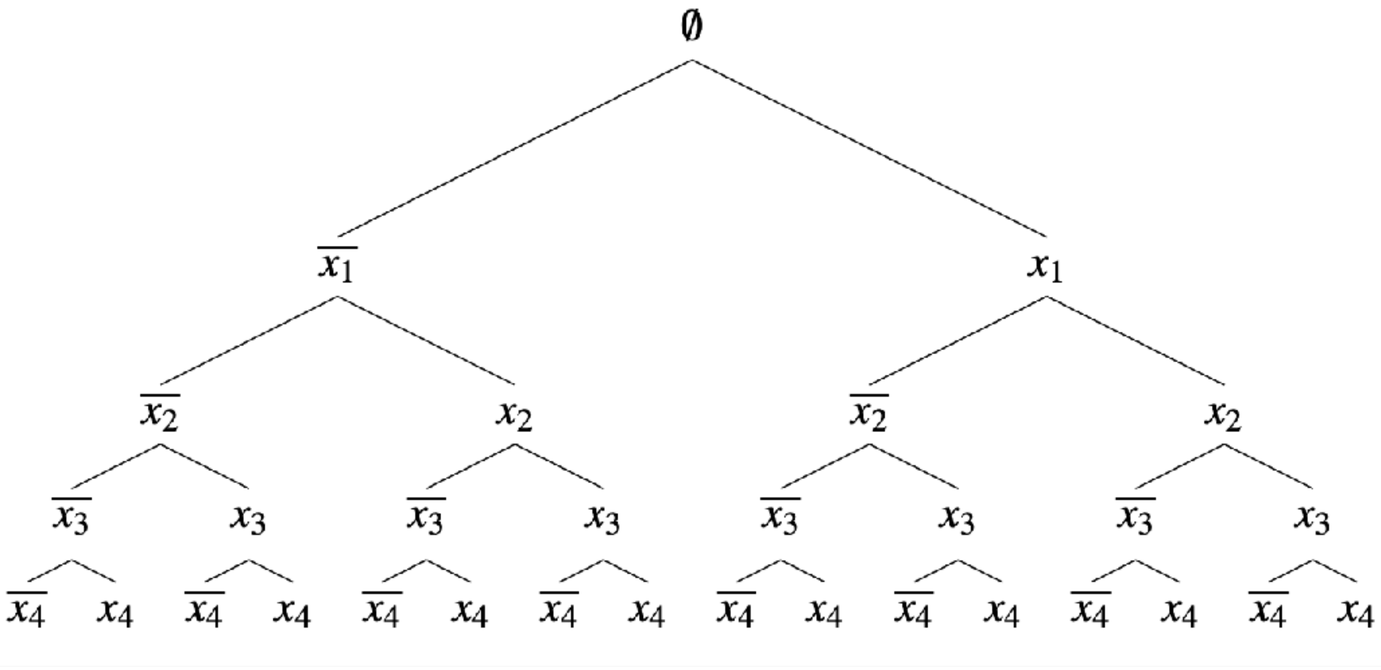
Bleibt: $Z - \sum_{i\in S} z_i = 4$; also $x_3 = \frac{2}{3} (= \frac{4}{6} = \frac{Z - \sum_{i\in S} z_i}{z_3})$

Damit: $\sum_{i\in S} x_i z_i = 9, \sum_{i\in S} x_i p_i = 11 + \frac{2}{3} * 8 = 16,\overline{3}$

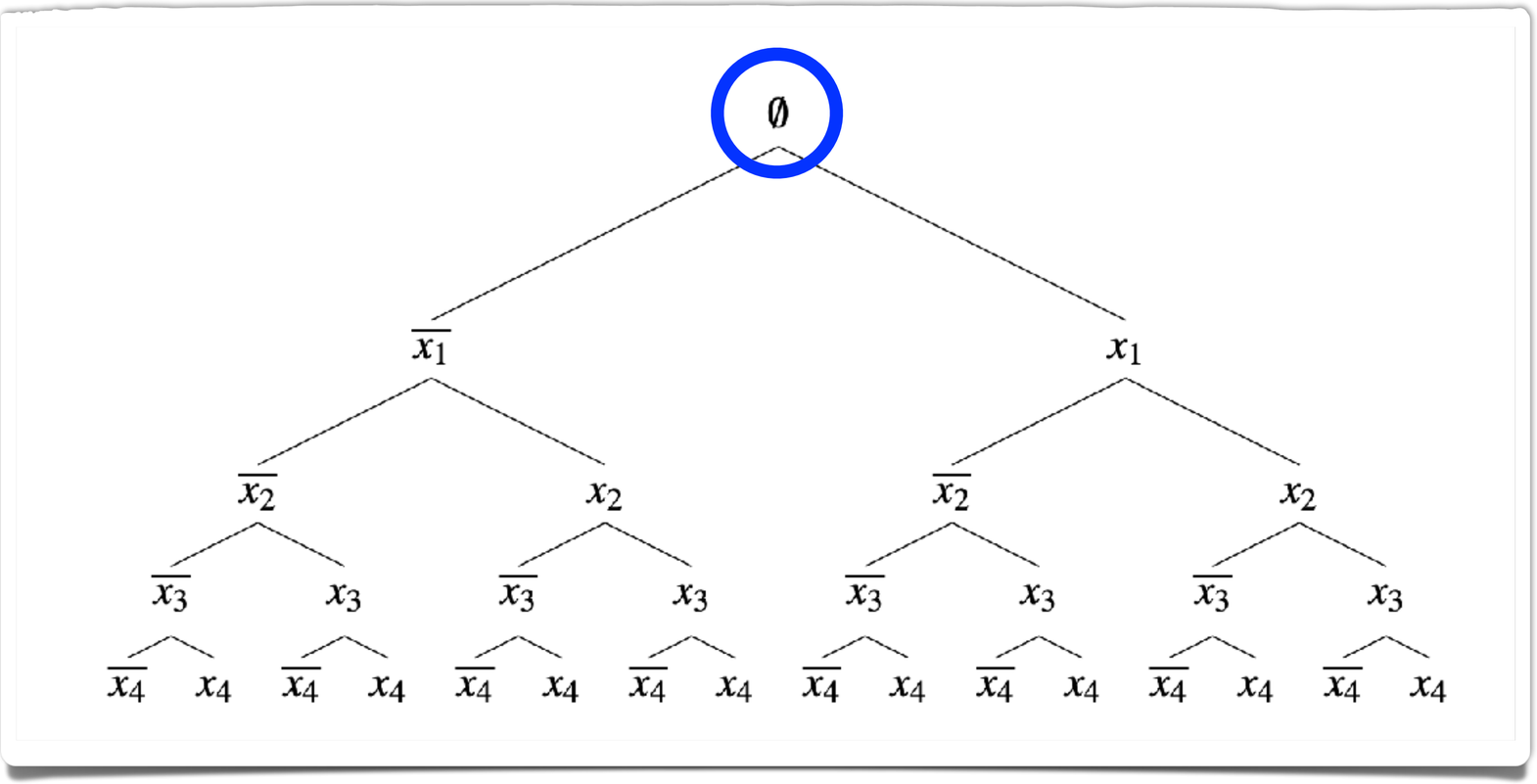
UB=16

Gesamtbeispiel

Gesamtbeispiel

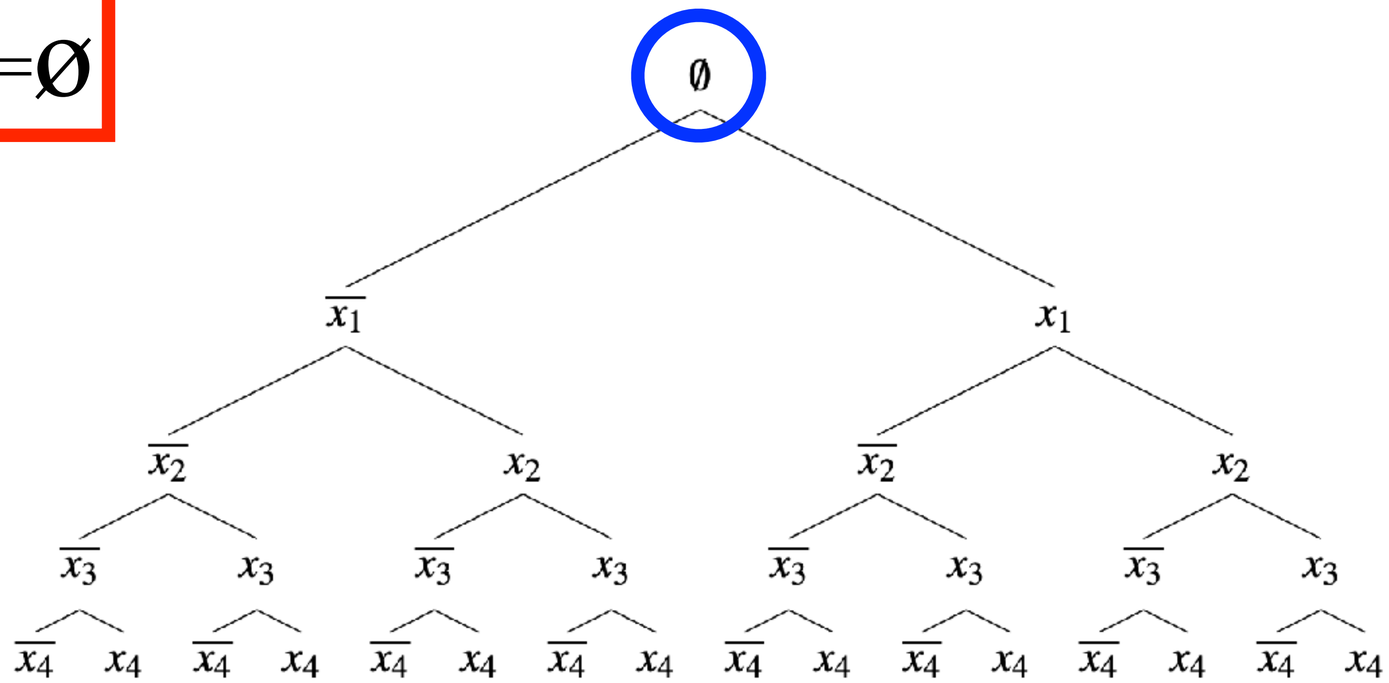


Gesamtbeispiel

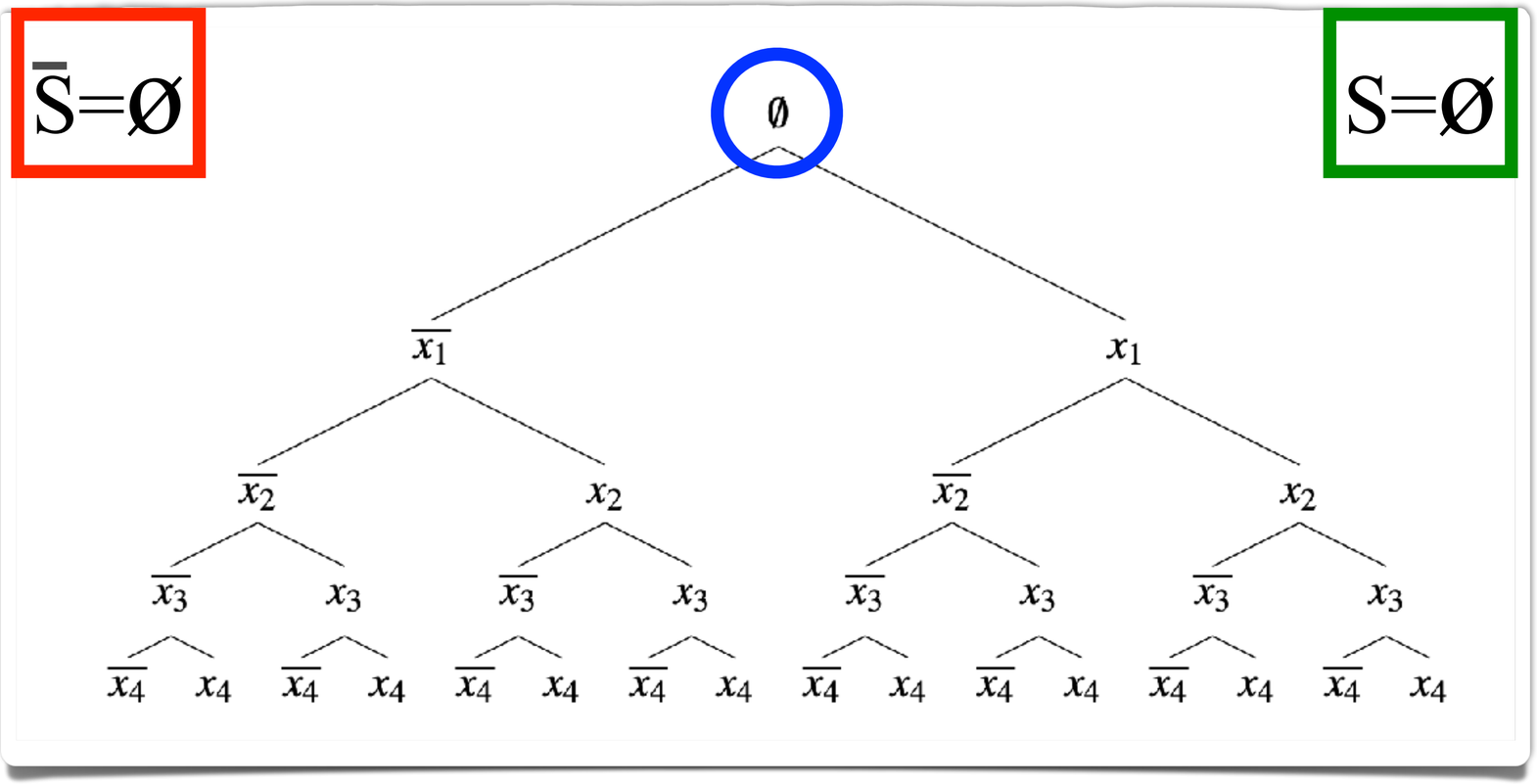


Gesamtbeispiel

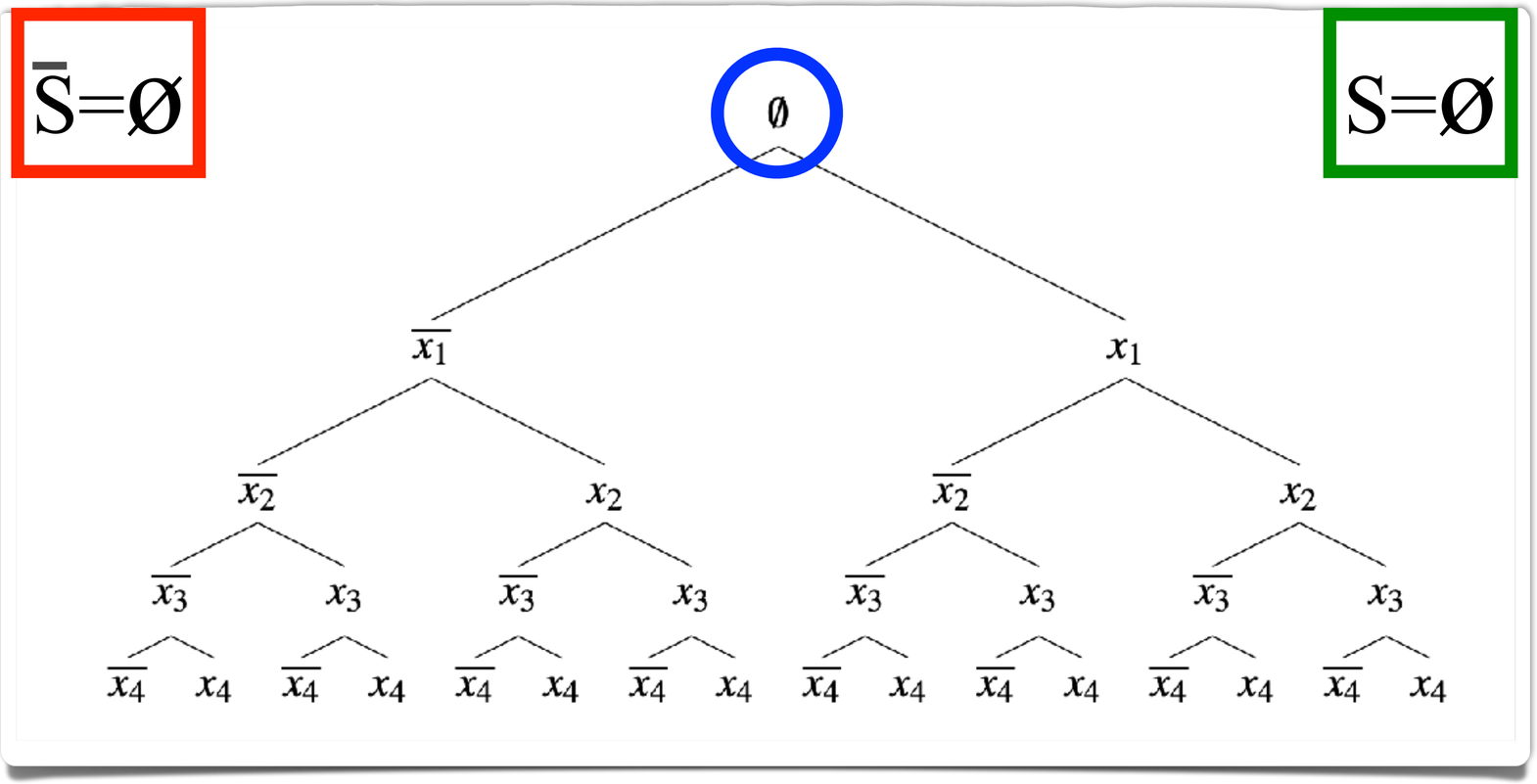
$\bar{S}=\emptyset$



Gesamtbeispiel



Gesamtbeispiel

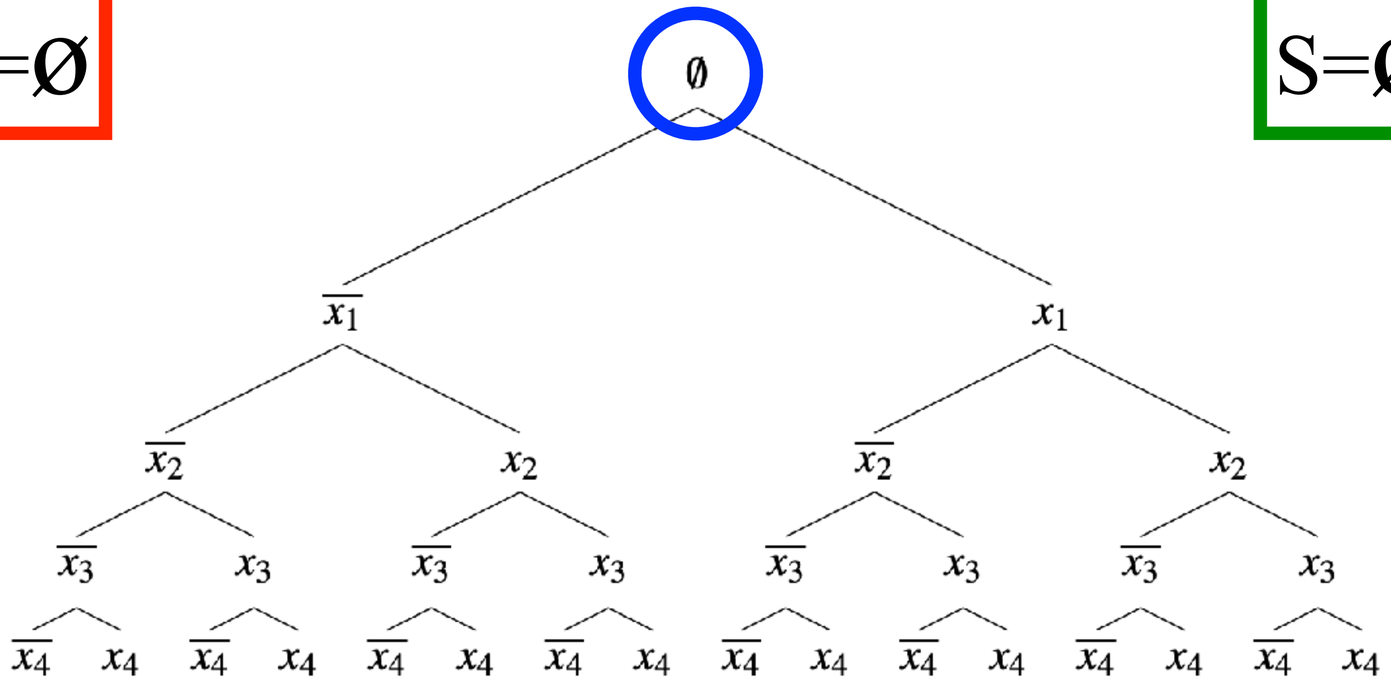


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

Gesamtbeispiel

$\bar{S}=\emptyset$

$S=\emptyset$

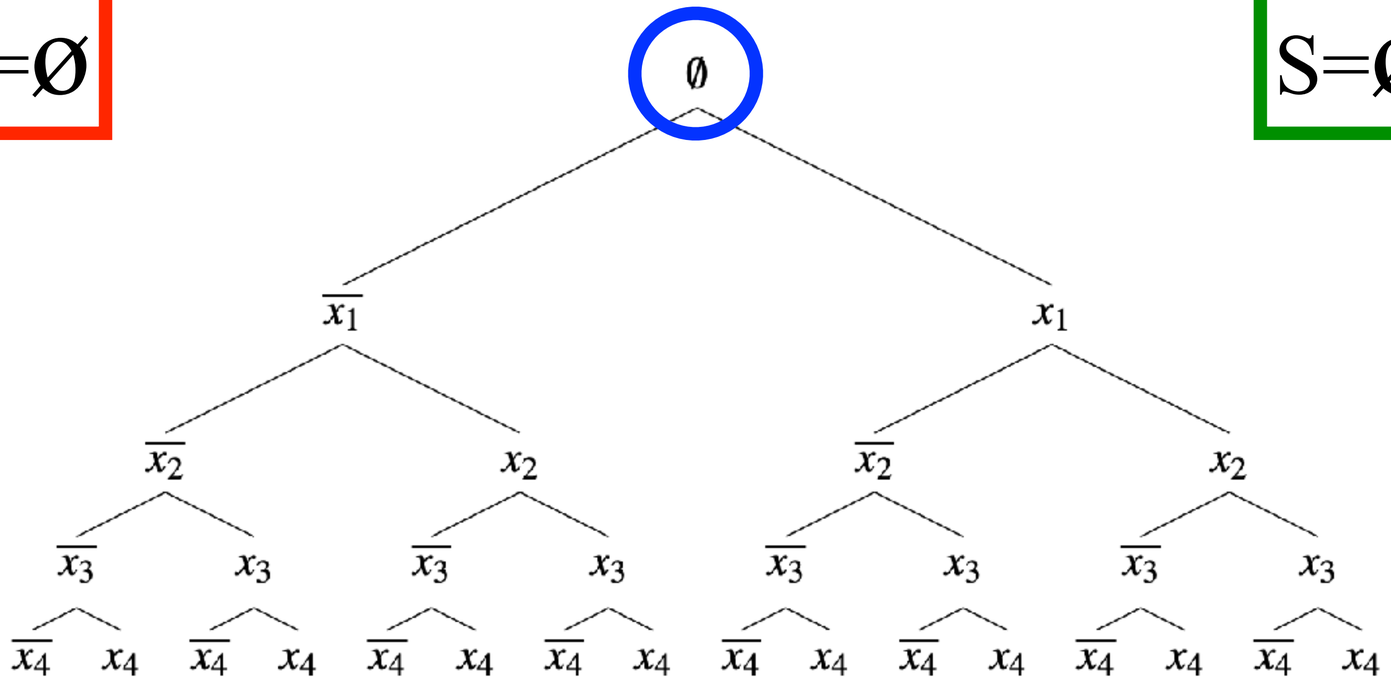


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

Gesamtbeispiel

$\bar{S}=\emptyset$

$S=\emptyset$

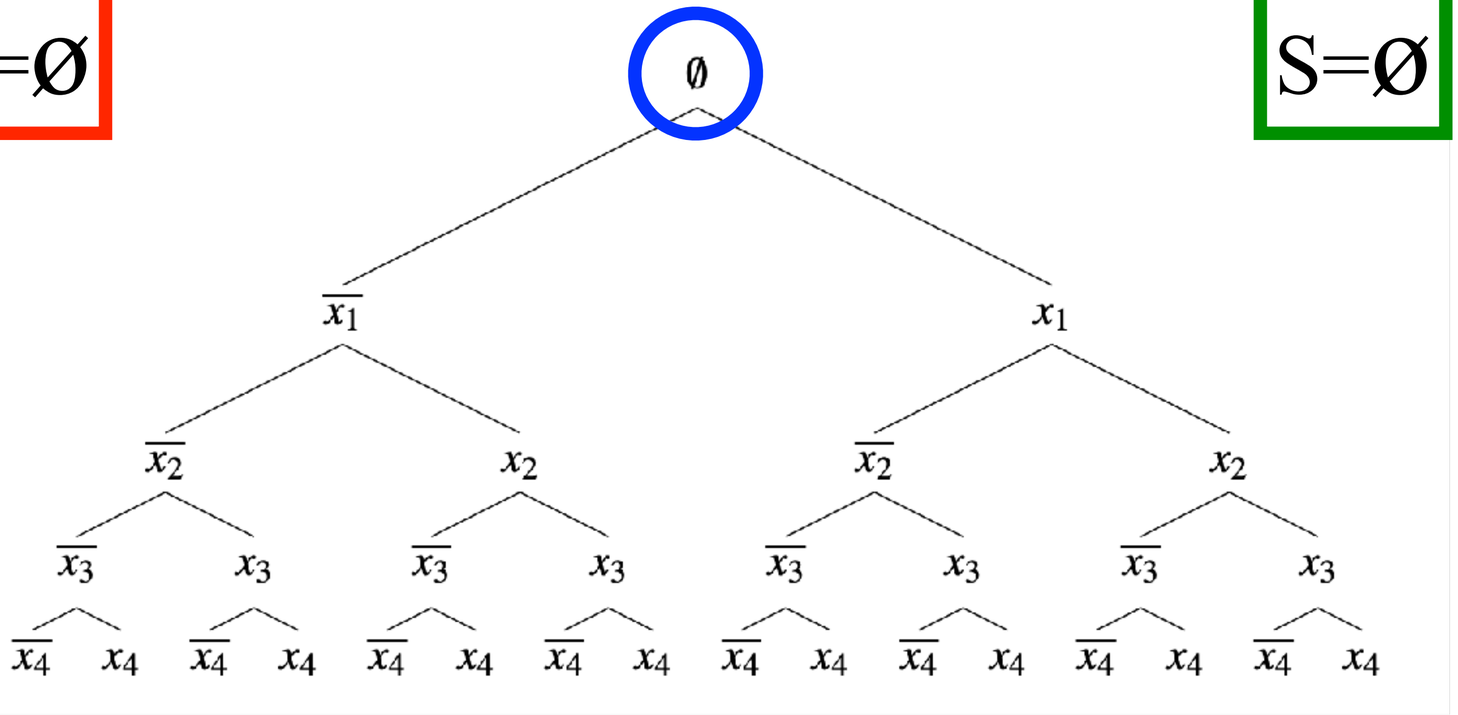


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

Gesamtbeispiel

$\bar{S}=\emptyset$

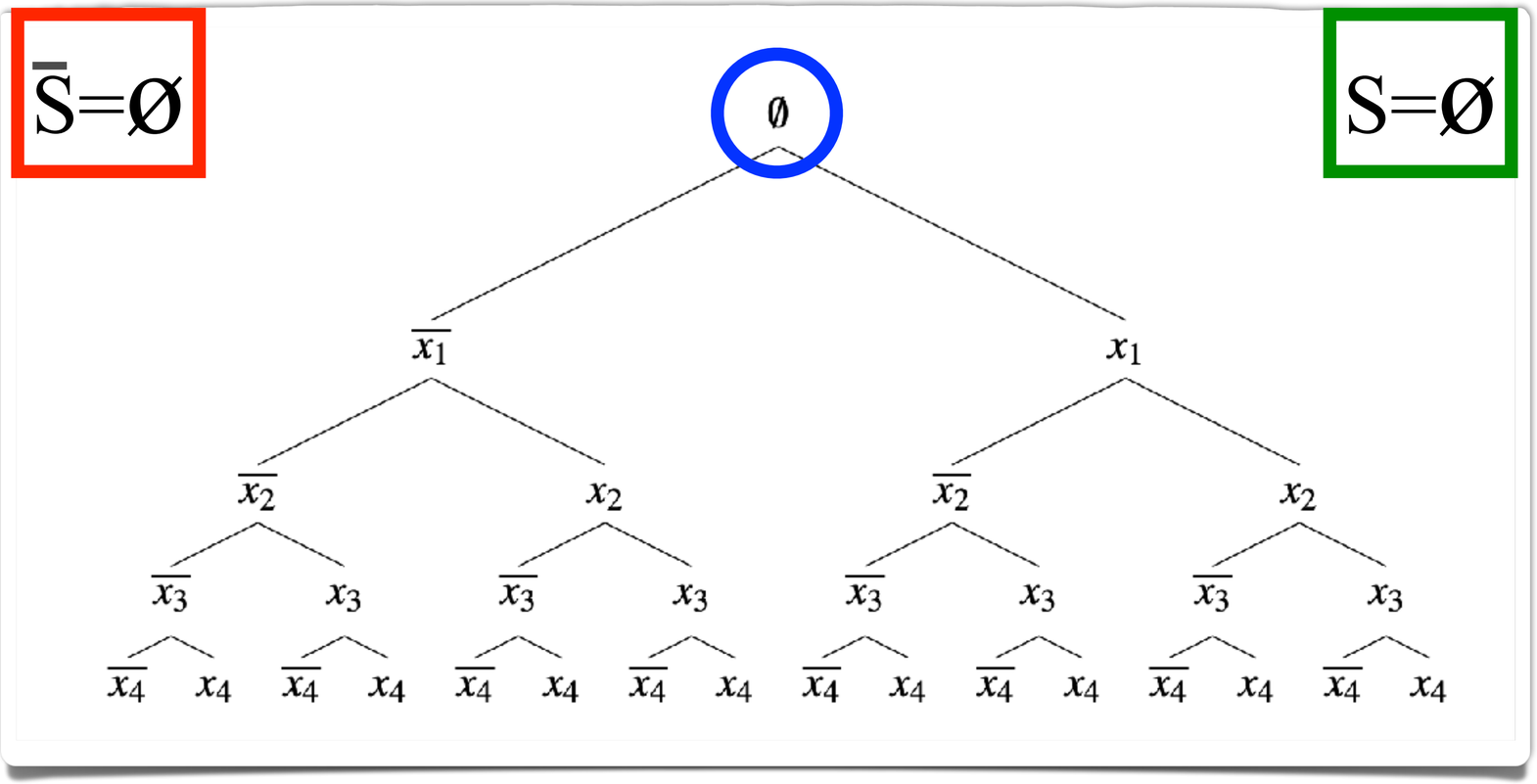
$S=\emptyset$



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

Gesamtbeispiel

$\bar{S}=\emptyset$



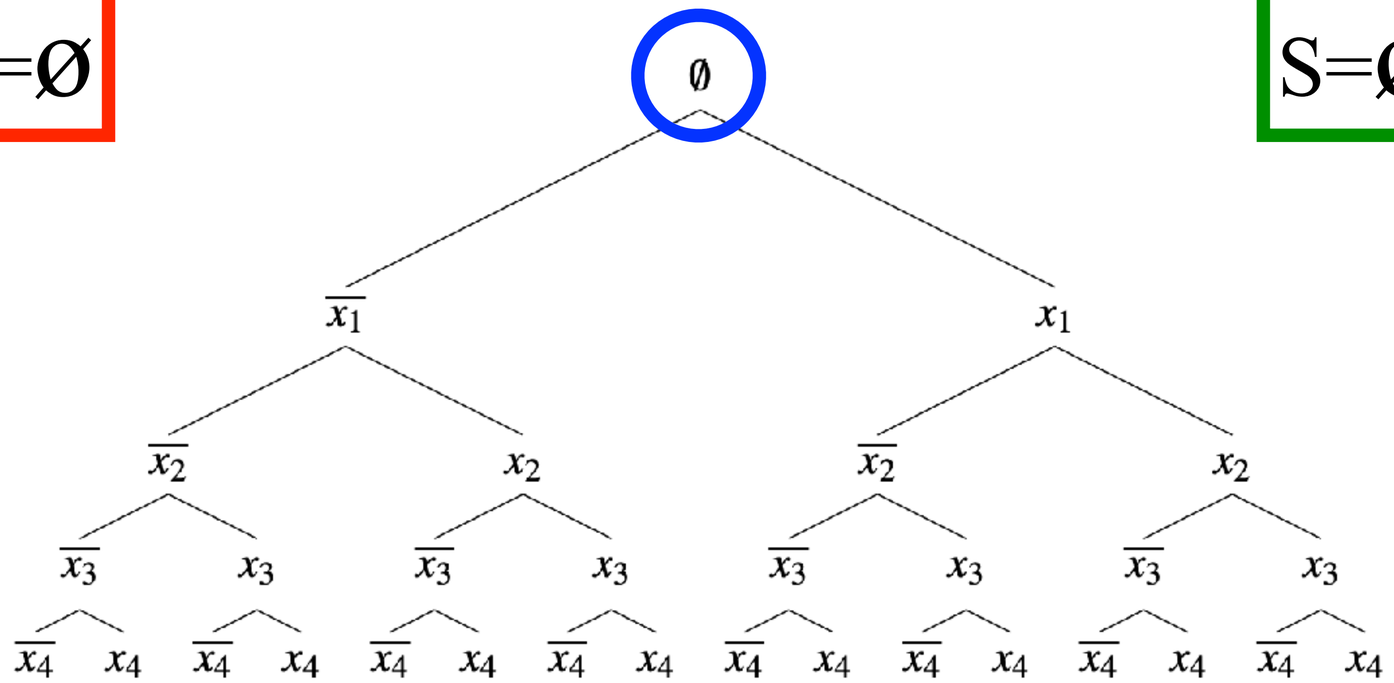
$S=\emptyset$

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

LB=14

Gesamtbeispiel

$\bar{S}=\emptyset$



$S=\emptyset$

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

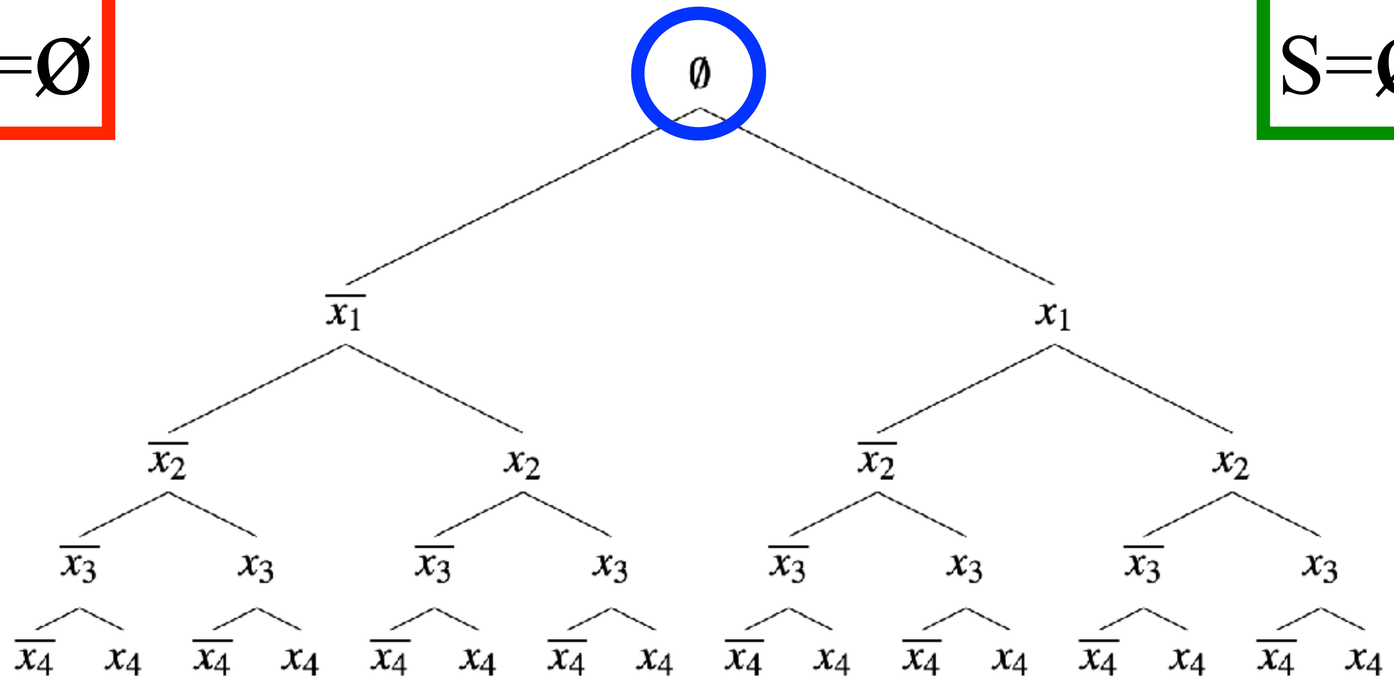
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

LB=14

Gesamtbeispiel

$\bar{S}=\emptyset$

$S=\emptyset$



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

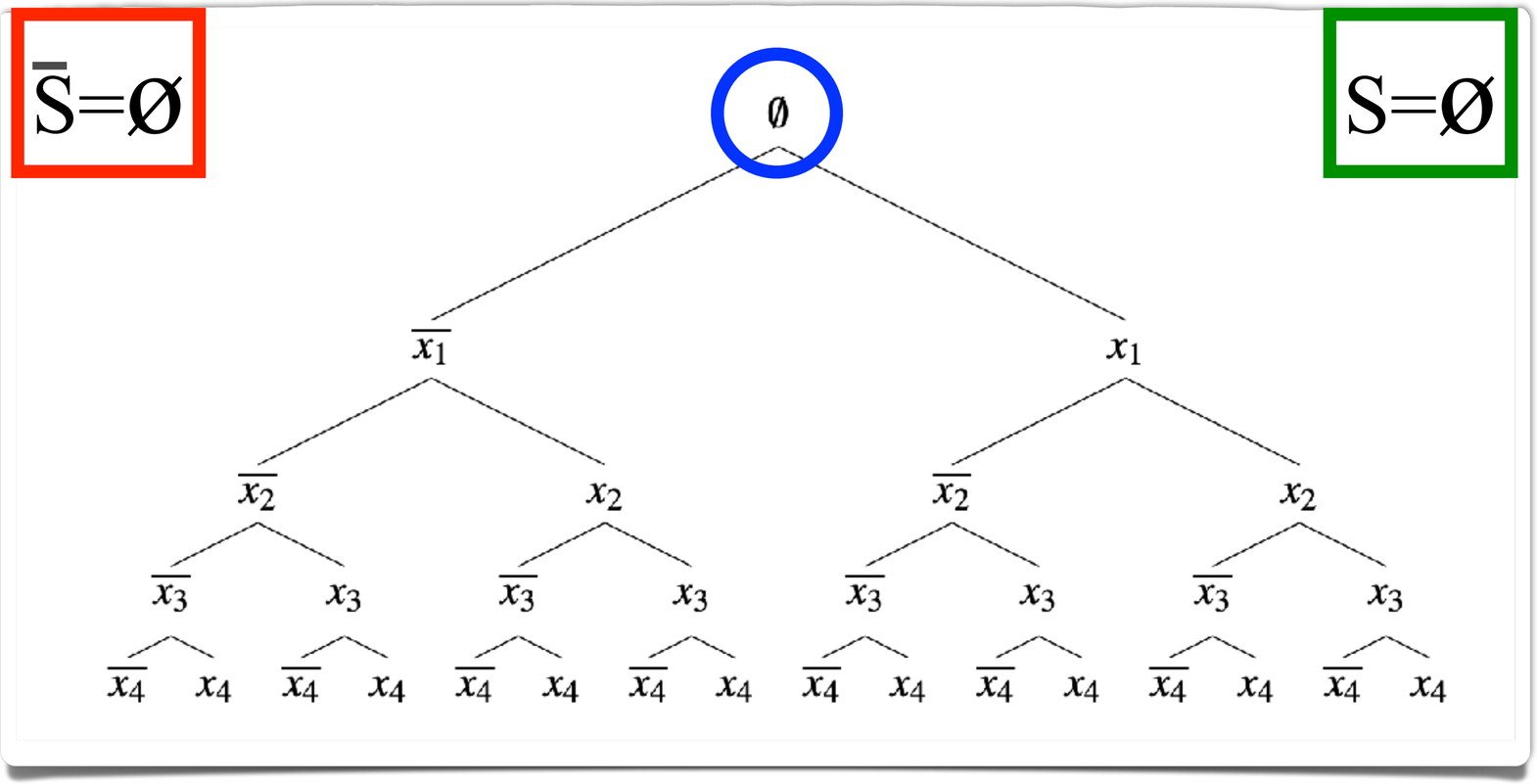
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

LB=14

Gesamtbeispiel

$\bar{S}=\emptyset$

$S=\emptyset$



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

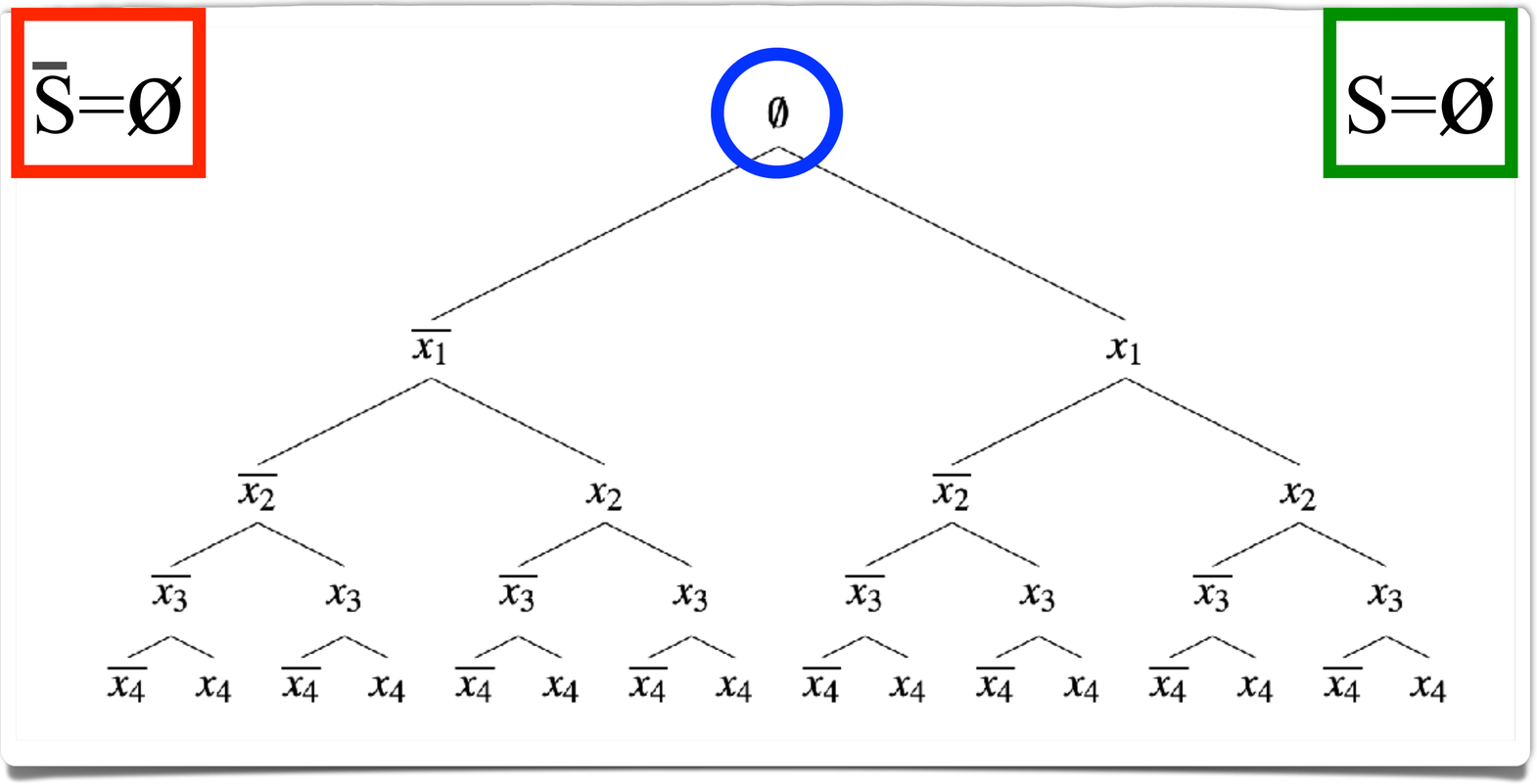
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

LB=14

Gesamtbeispiel

$\bar{S}=\emptyset$

$S=\emptyset$



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

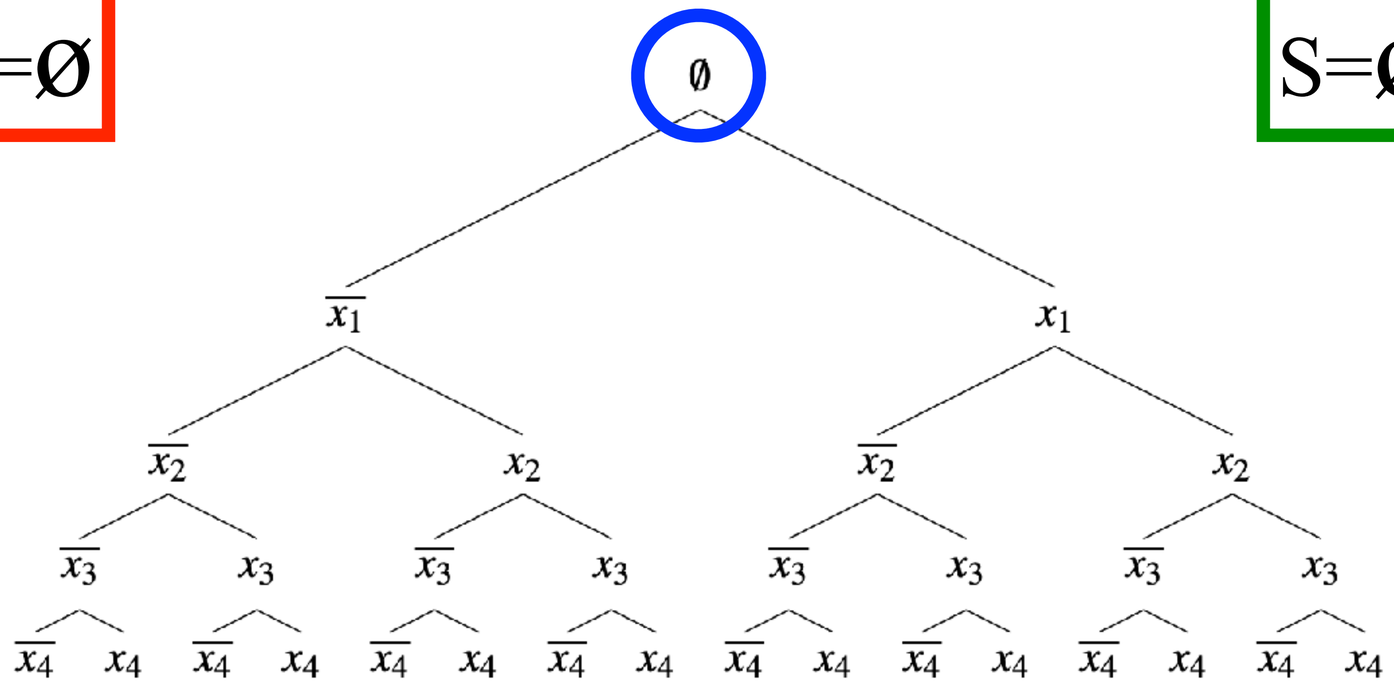
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

LB=14

Gesamtbeispiel

$\bar{S}=\emptyset$

$S=\emptyset$



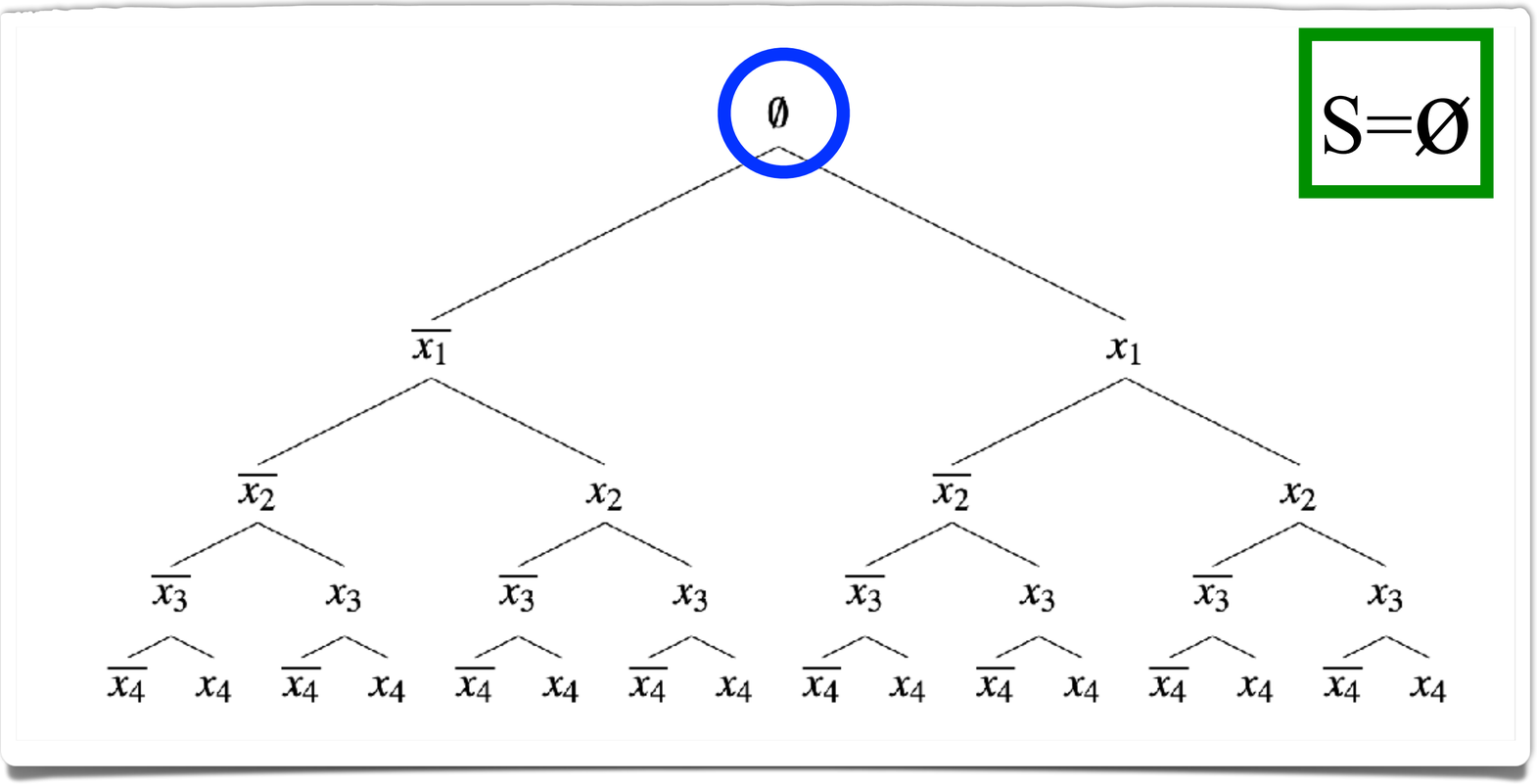
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

UB=16

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

LB=14

Gesamtbeispiel

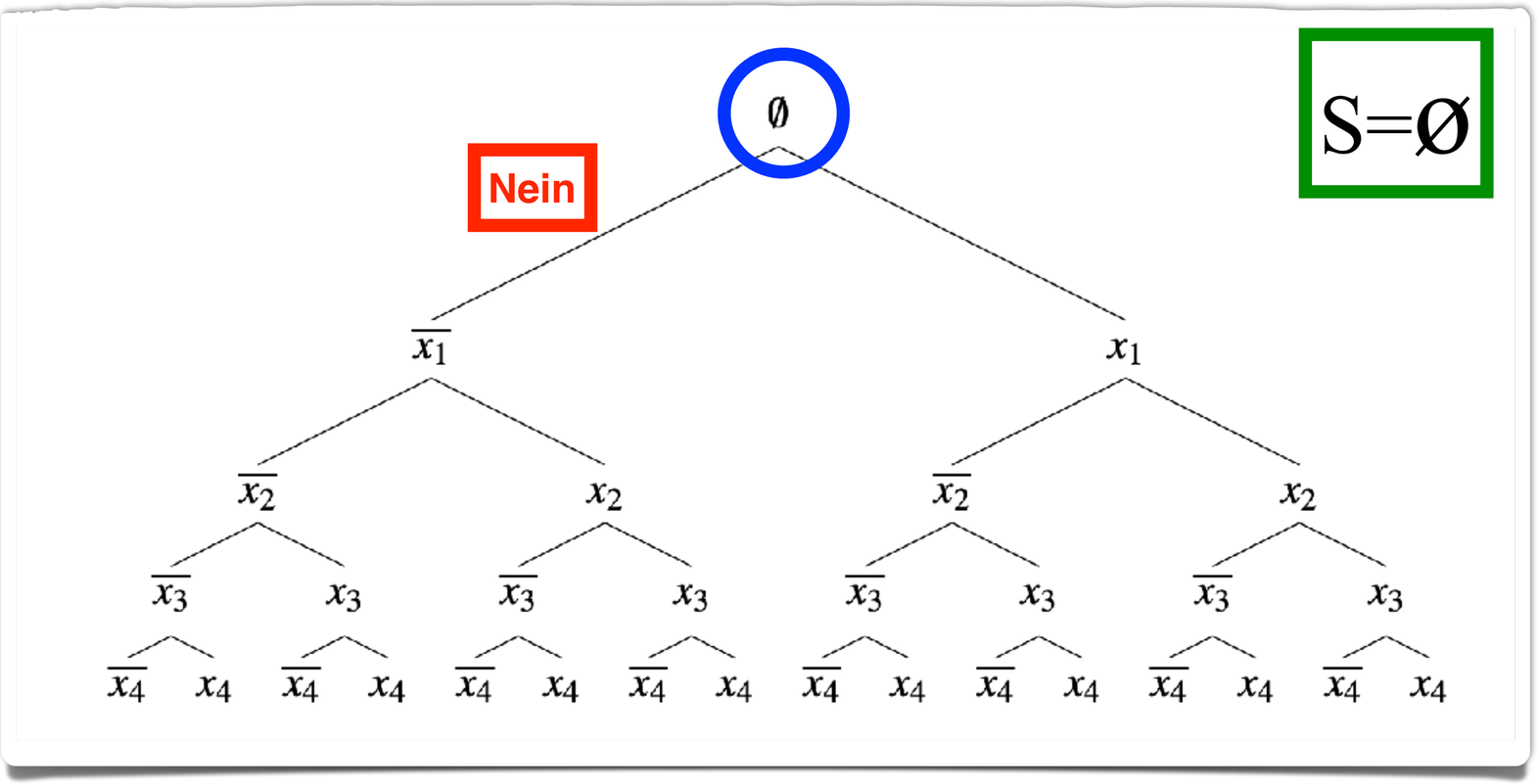


S=Ø

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

LB=14

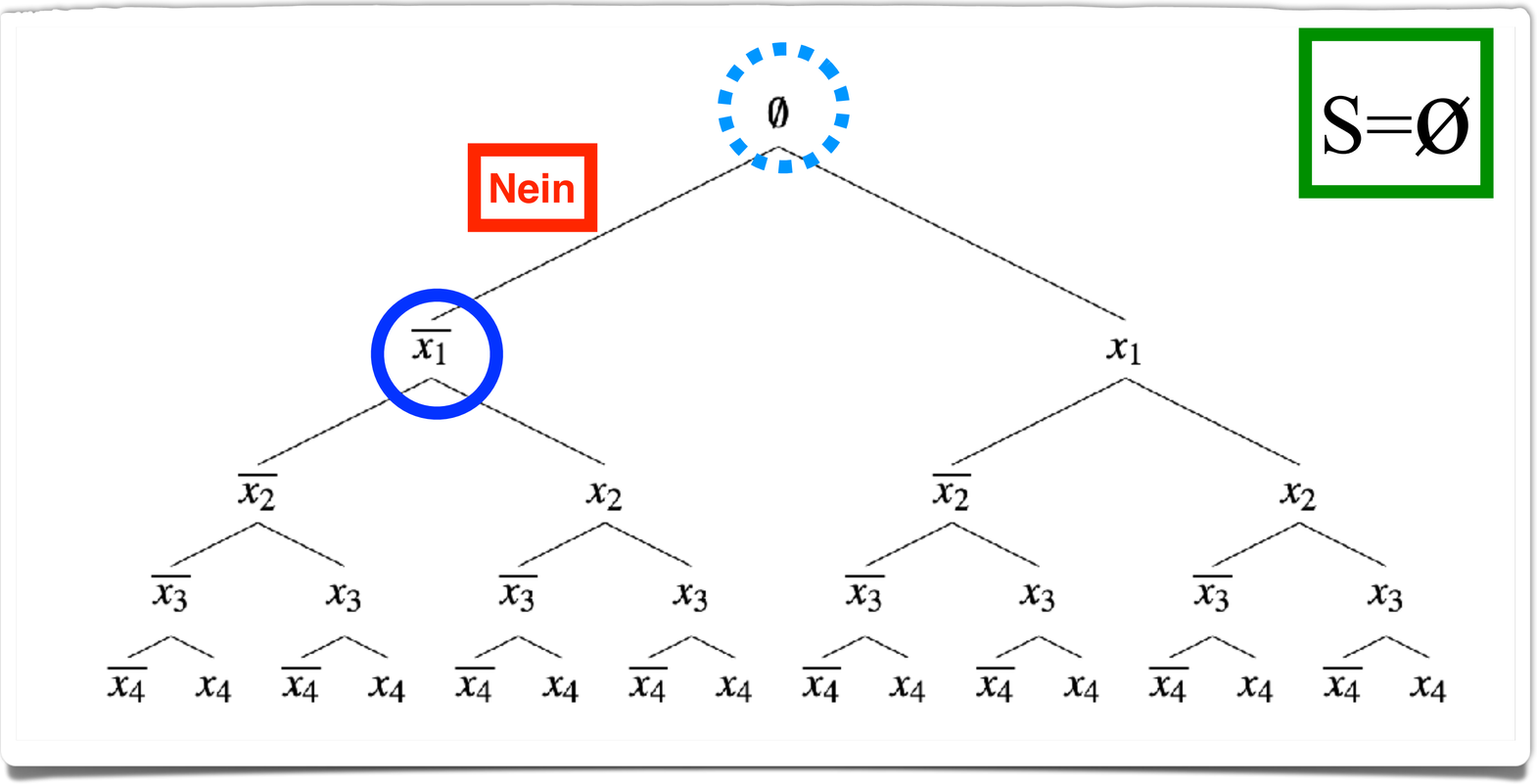
Gesamtbeispiel



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

LB=14

Gesamtbeispiel



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

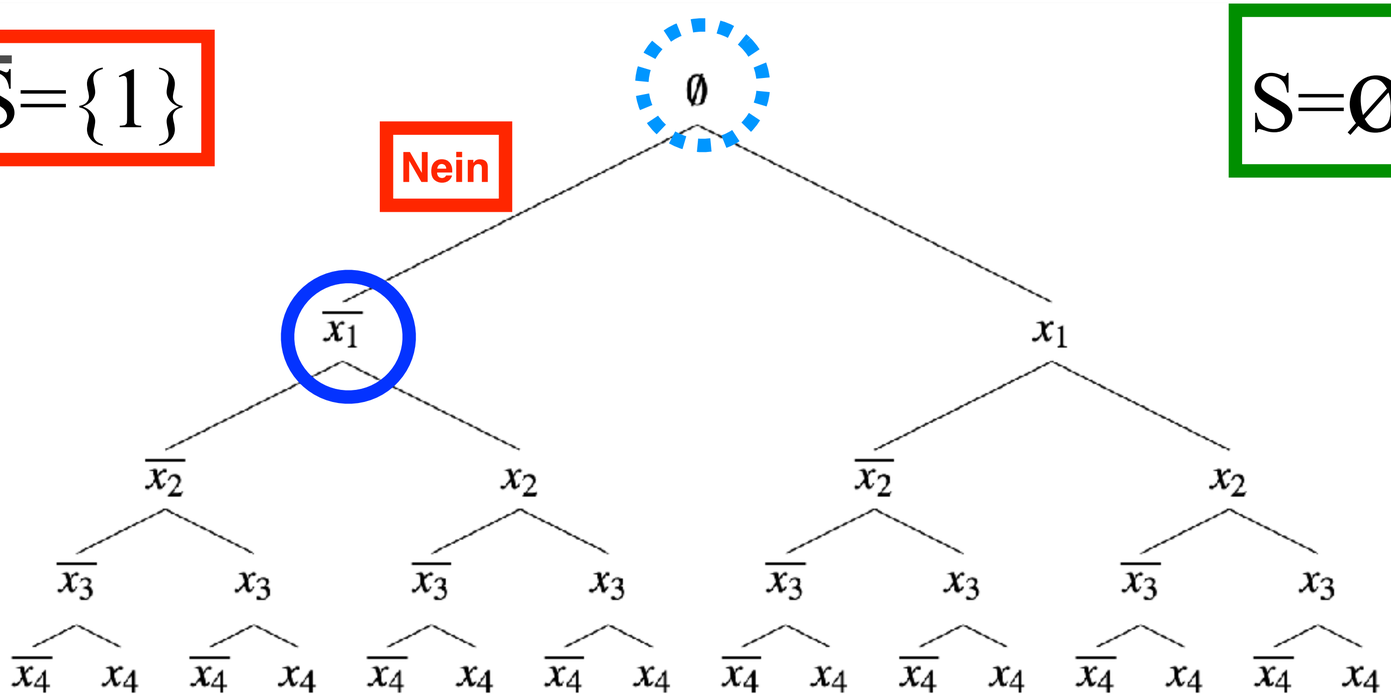
LB=14

Gesamtbeispiel

$\bar{S}=\{1\}$

Nein

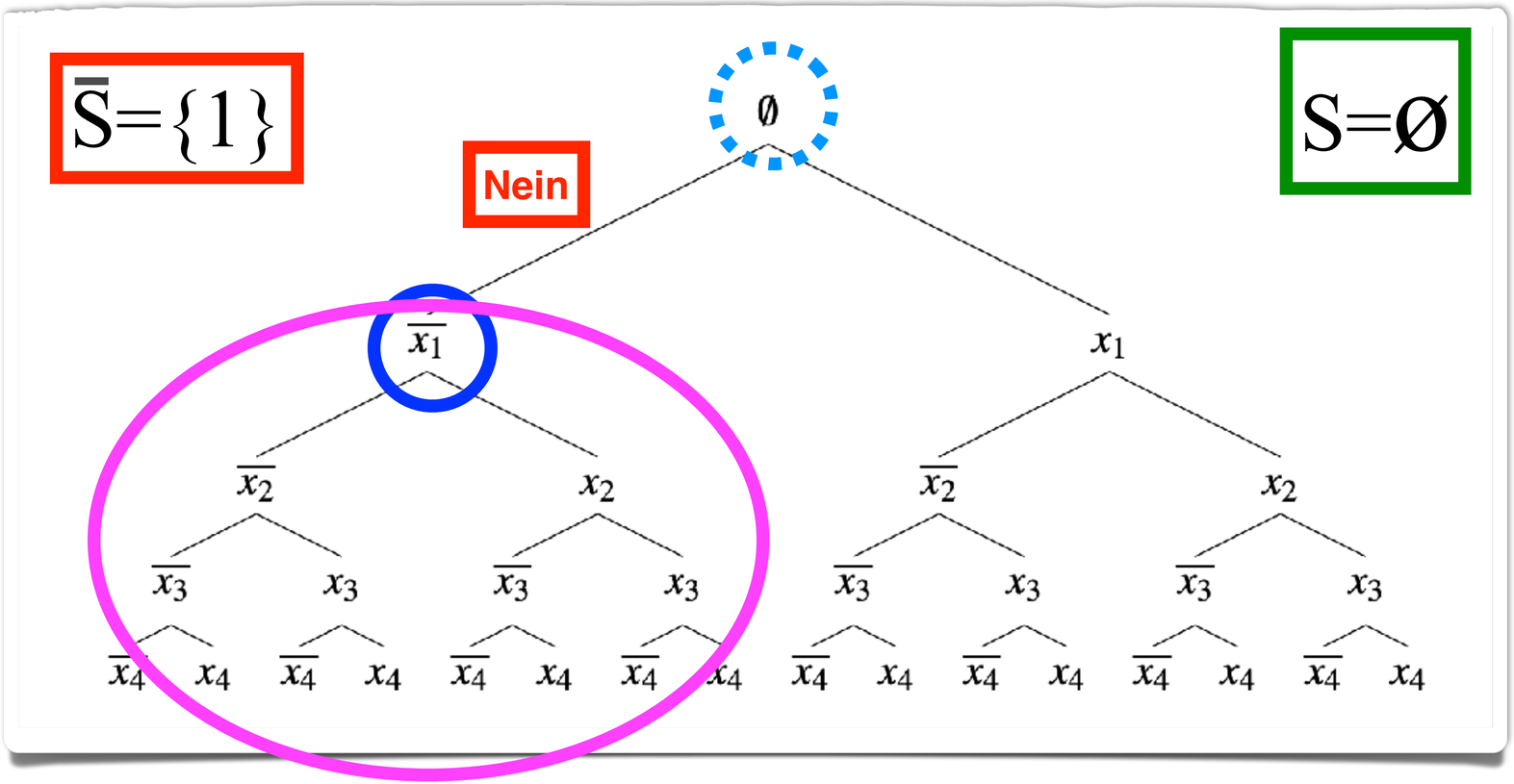
$S=\emptyset$



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

LB=14

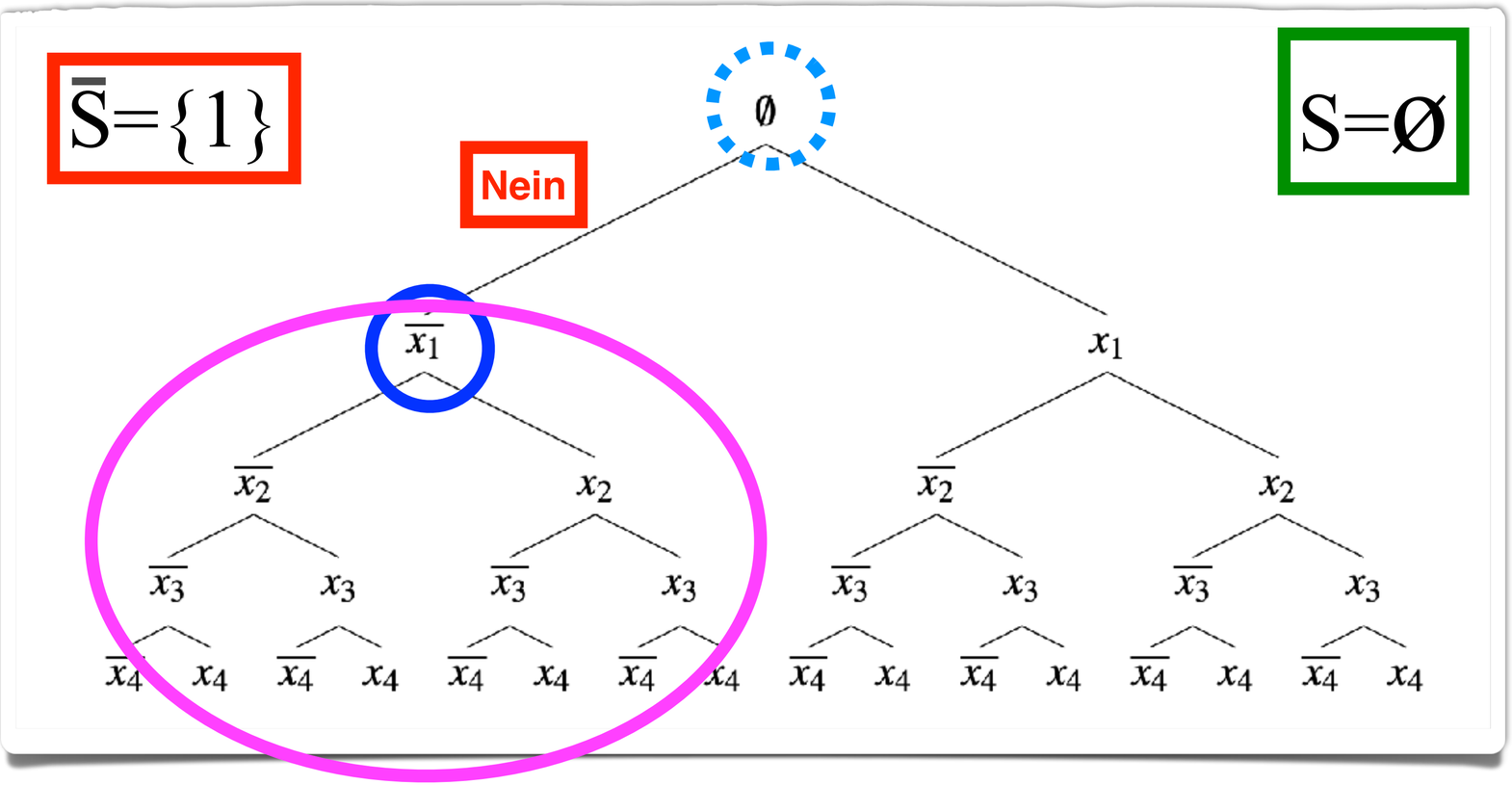
Gesamtbeispiel



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

LB=14

Gesamtbeispiel

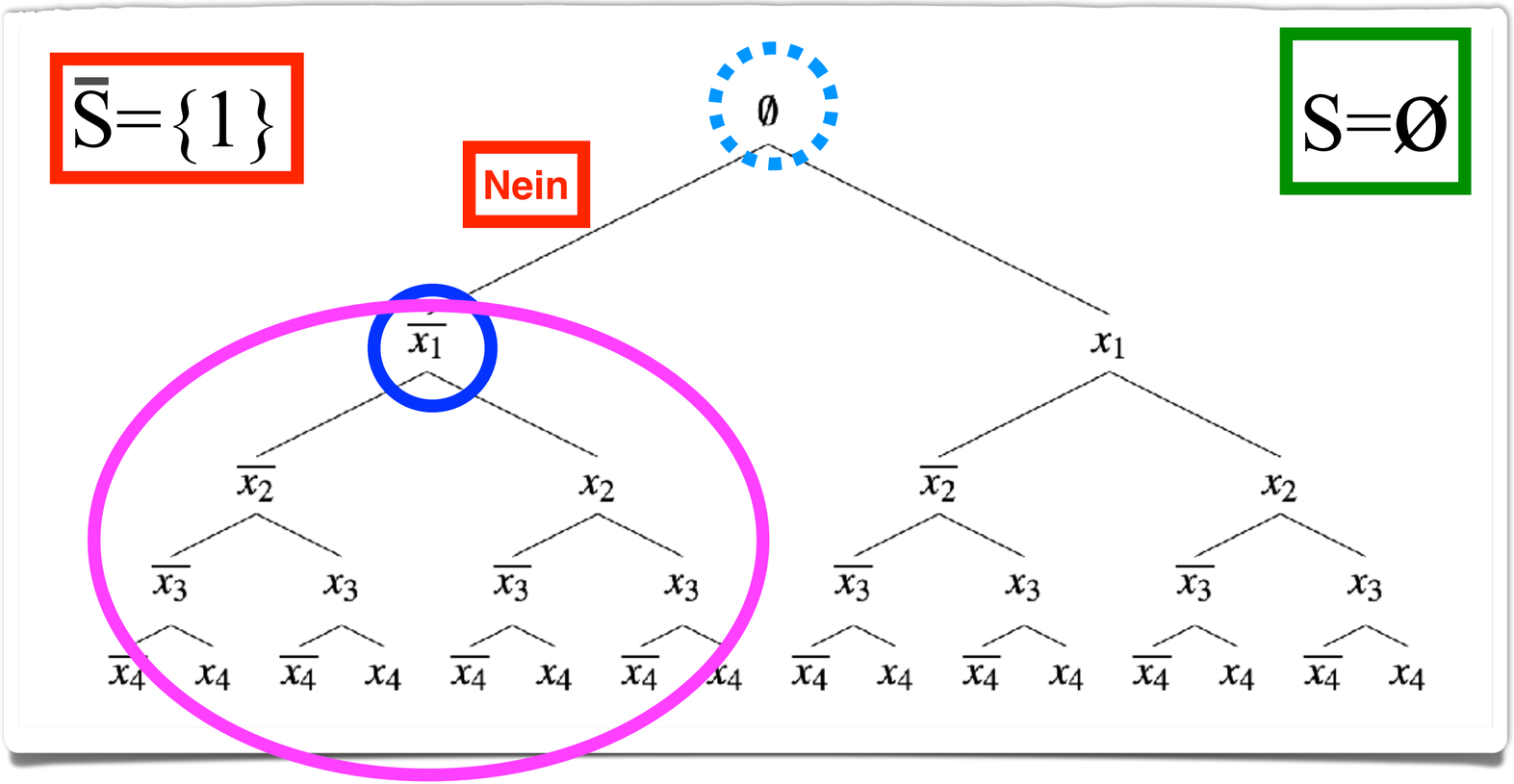


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=14

Gesamtbeispiel

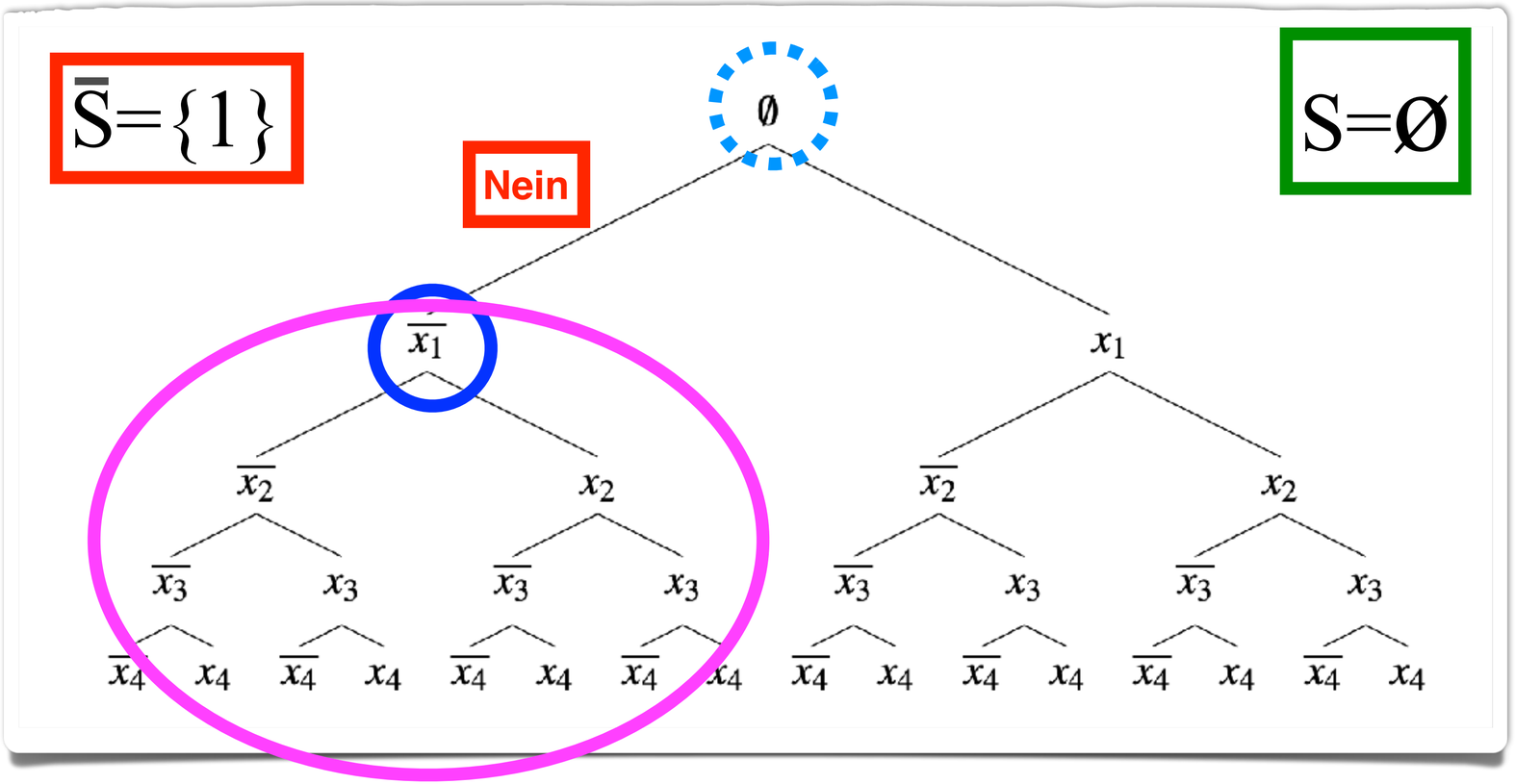


i	1	2	3	4	5	6	7
z_i		3	6	7	5	9	4
p_i		5	8	9	6	7	3

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

LB=14

Gesamtbeispiel



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

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

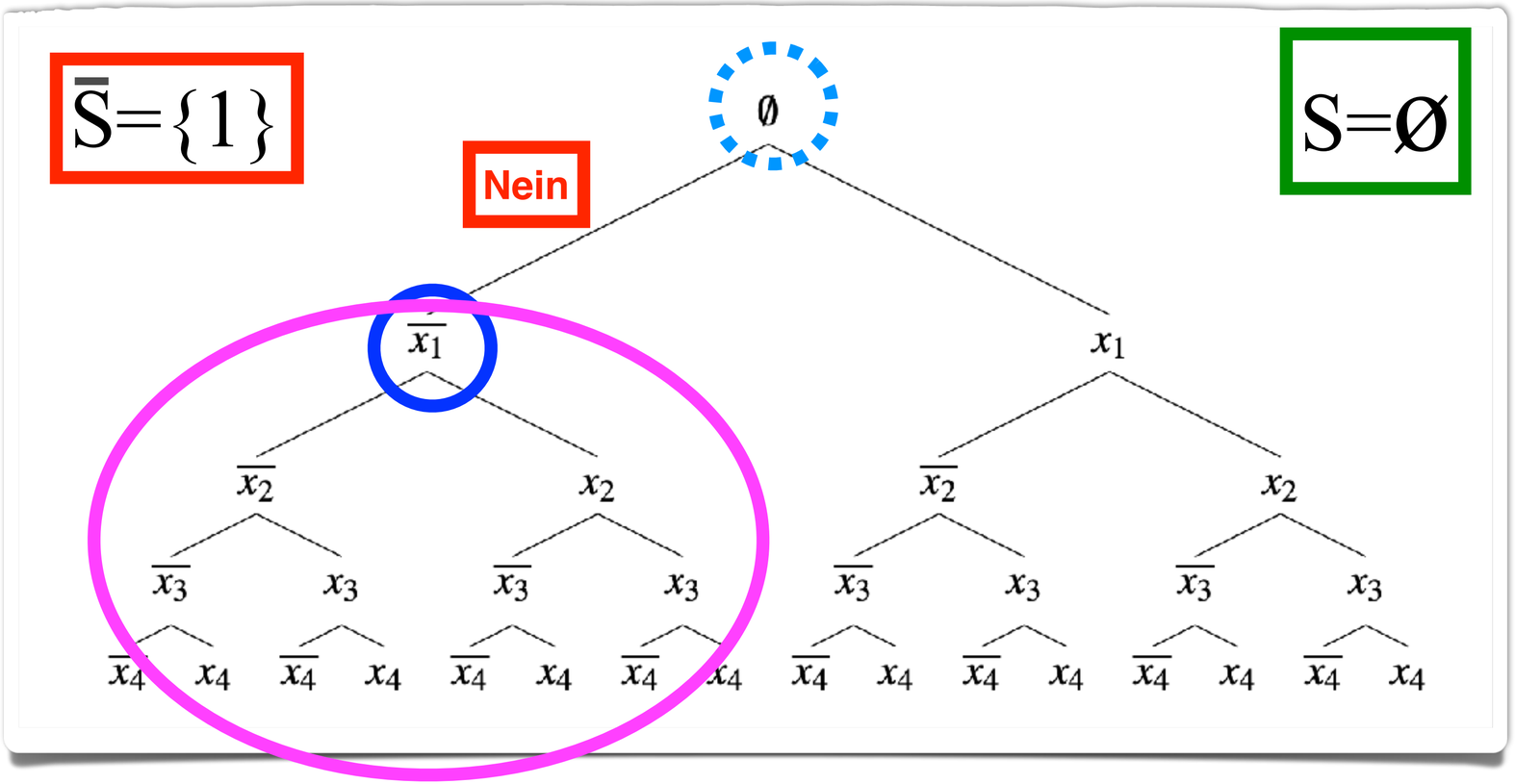
LB=14

Gesamtbeispiel

$\bar{S}=\{1\}$

Nein

$S=\emptyset$



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

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

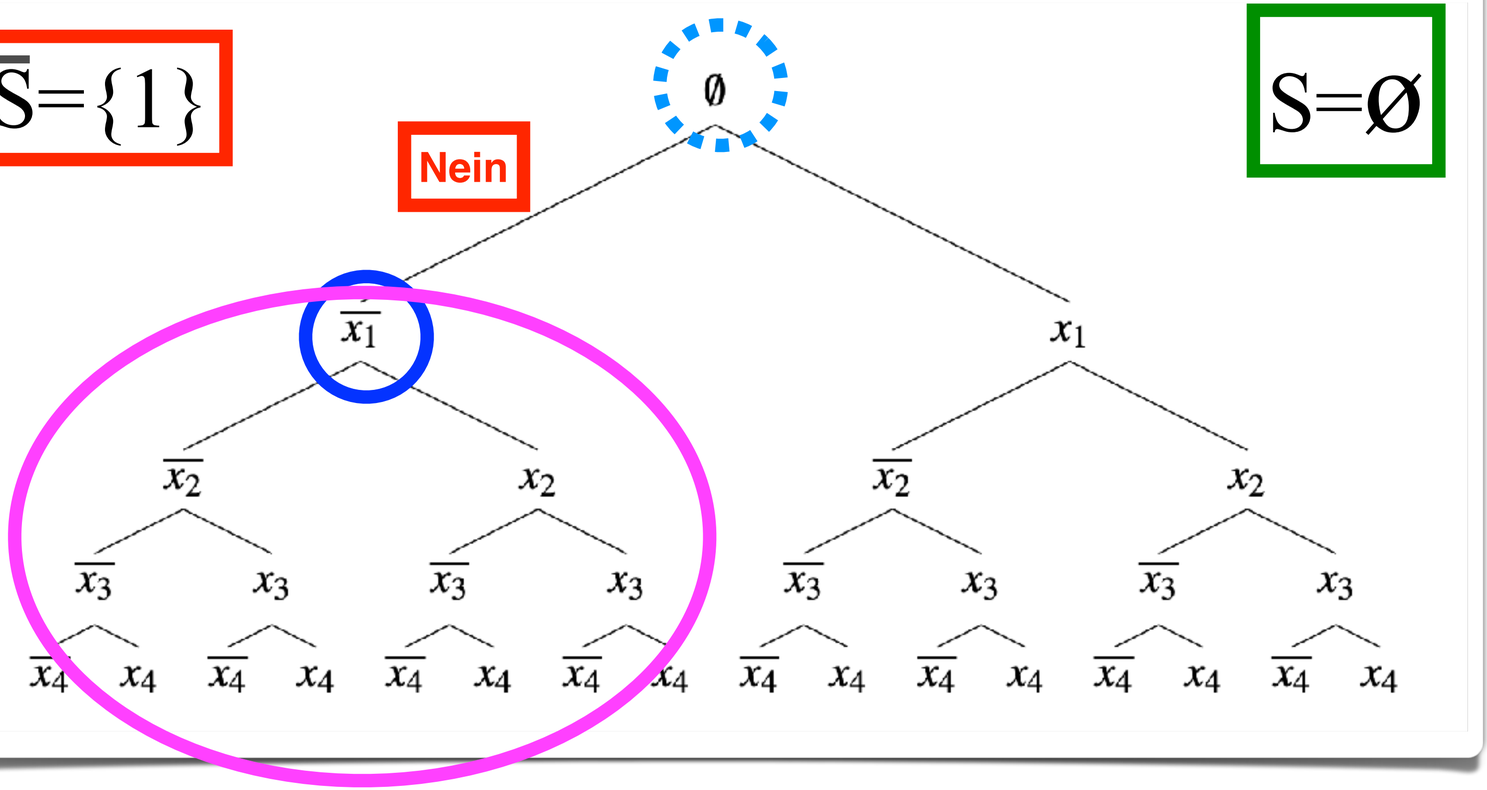
LB=14

Gesamtbeispiel

$\bar{S}=\{1\}$

Nein

$S=\emptyset$



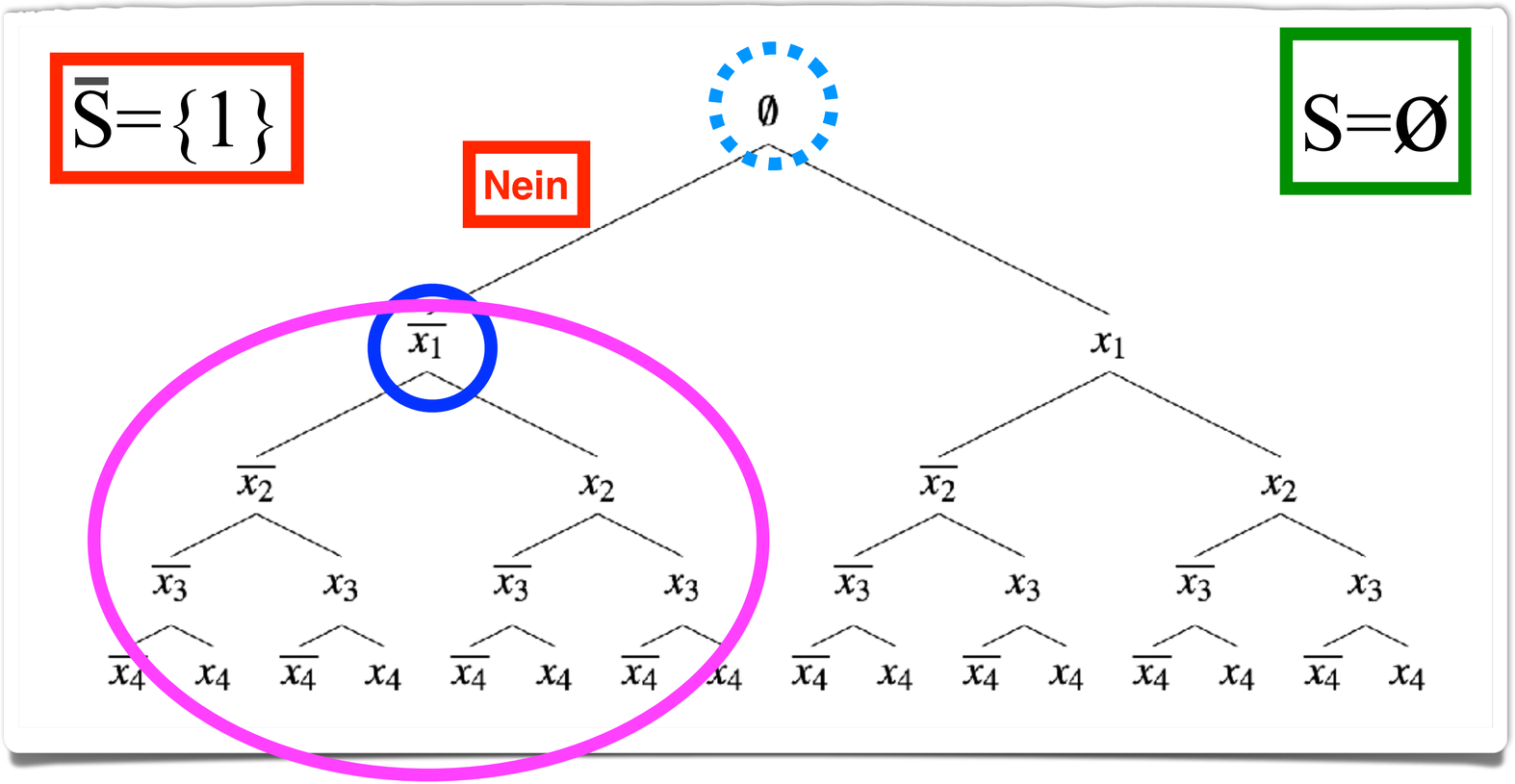
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

UB=13

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

LB=14

Gesamtbeispiel



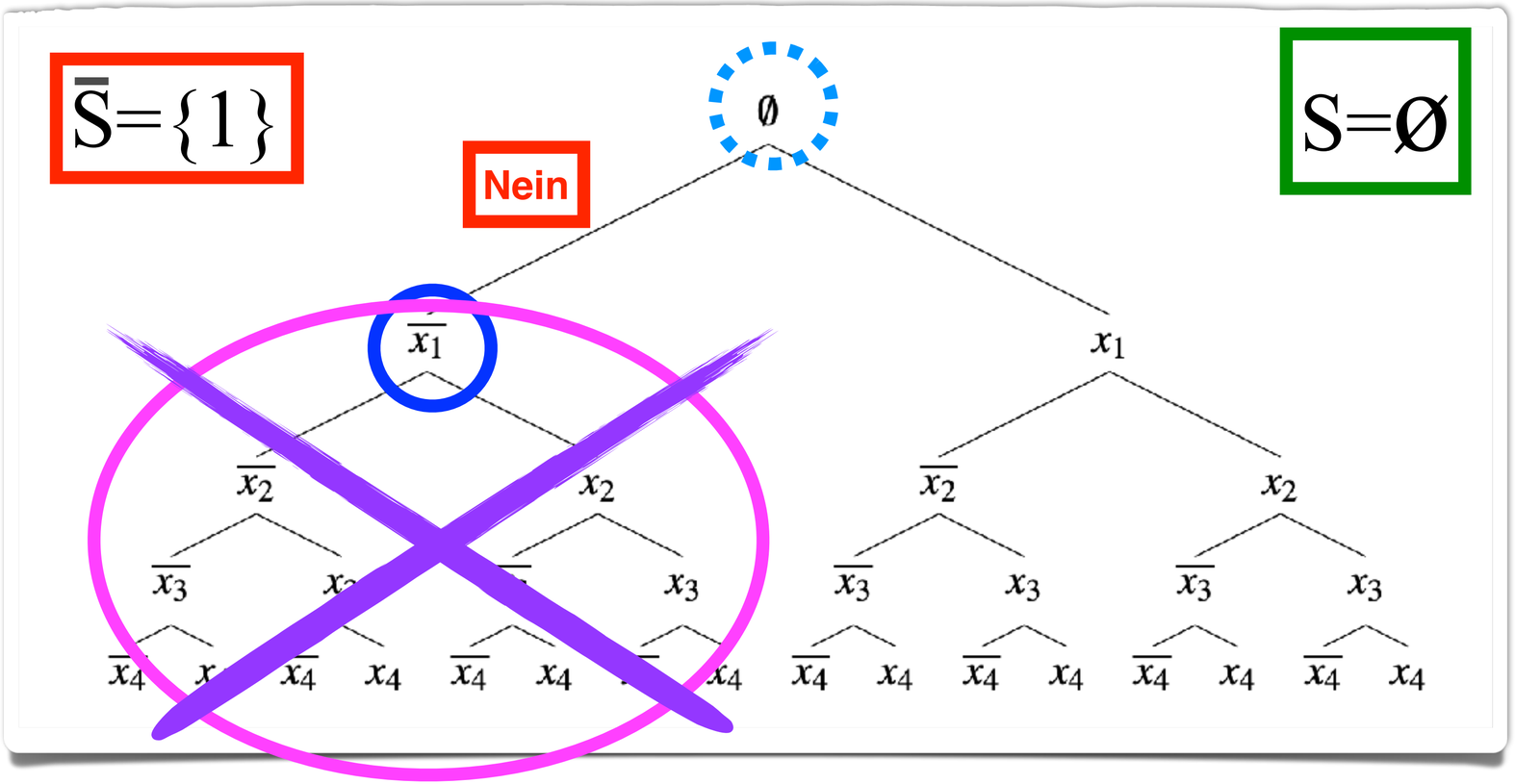
i	1	2	3	4	5	6	7
z_i		3	6	7	5	9	4
p_i		5	8	9	6	7	3

UB=13

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

LB=14

Gesamtbeispiel



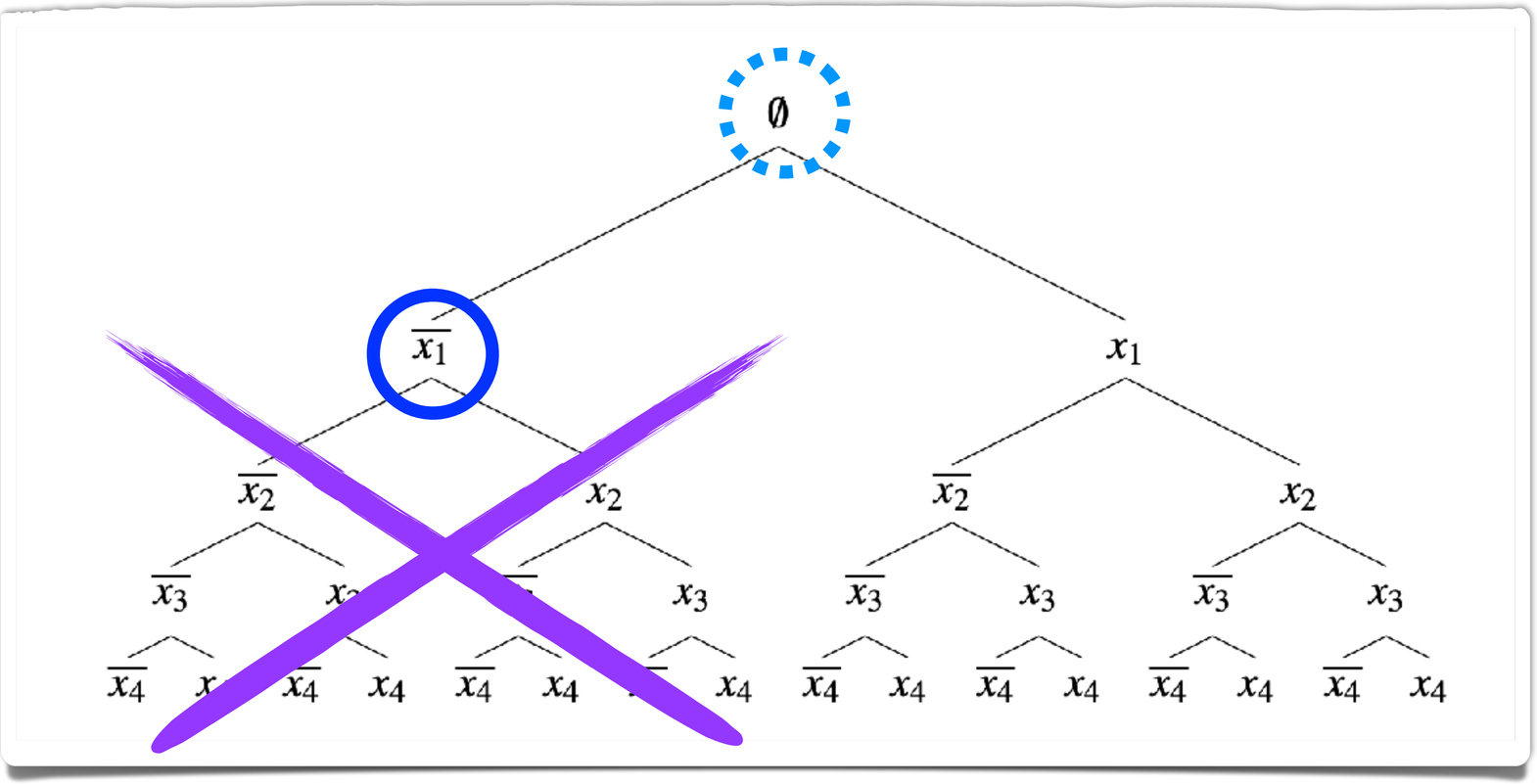
i	1	2	3	4	5	6	7
z_i		3	6	7	5	9	4
p_i		5	8	9	6	7	3

UB=13

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

LB=14

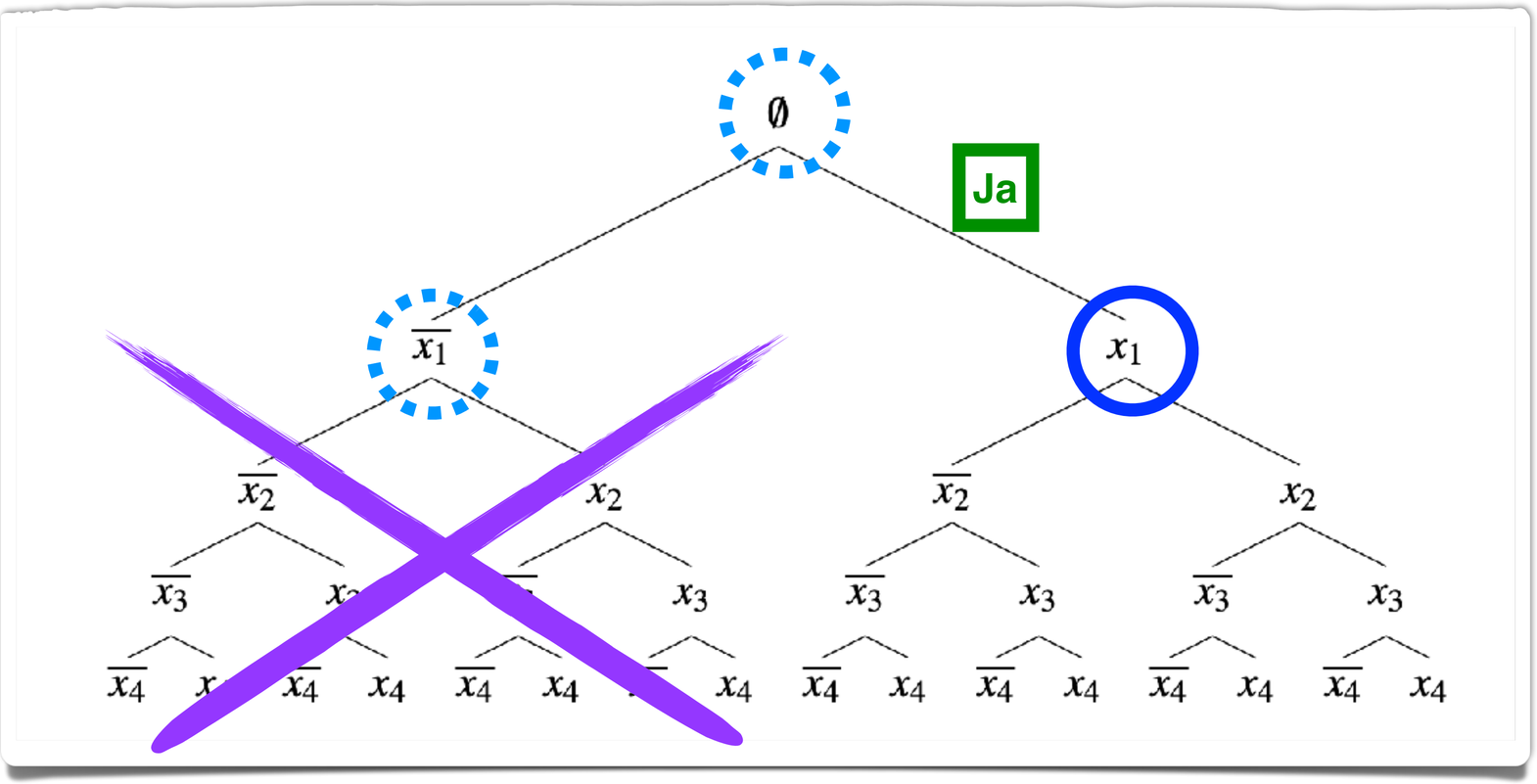
Gesamtbeispiel



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

LB=14

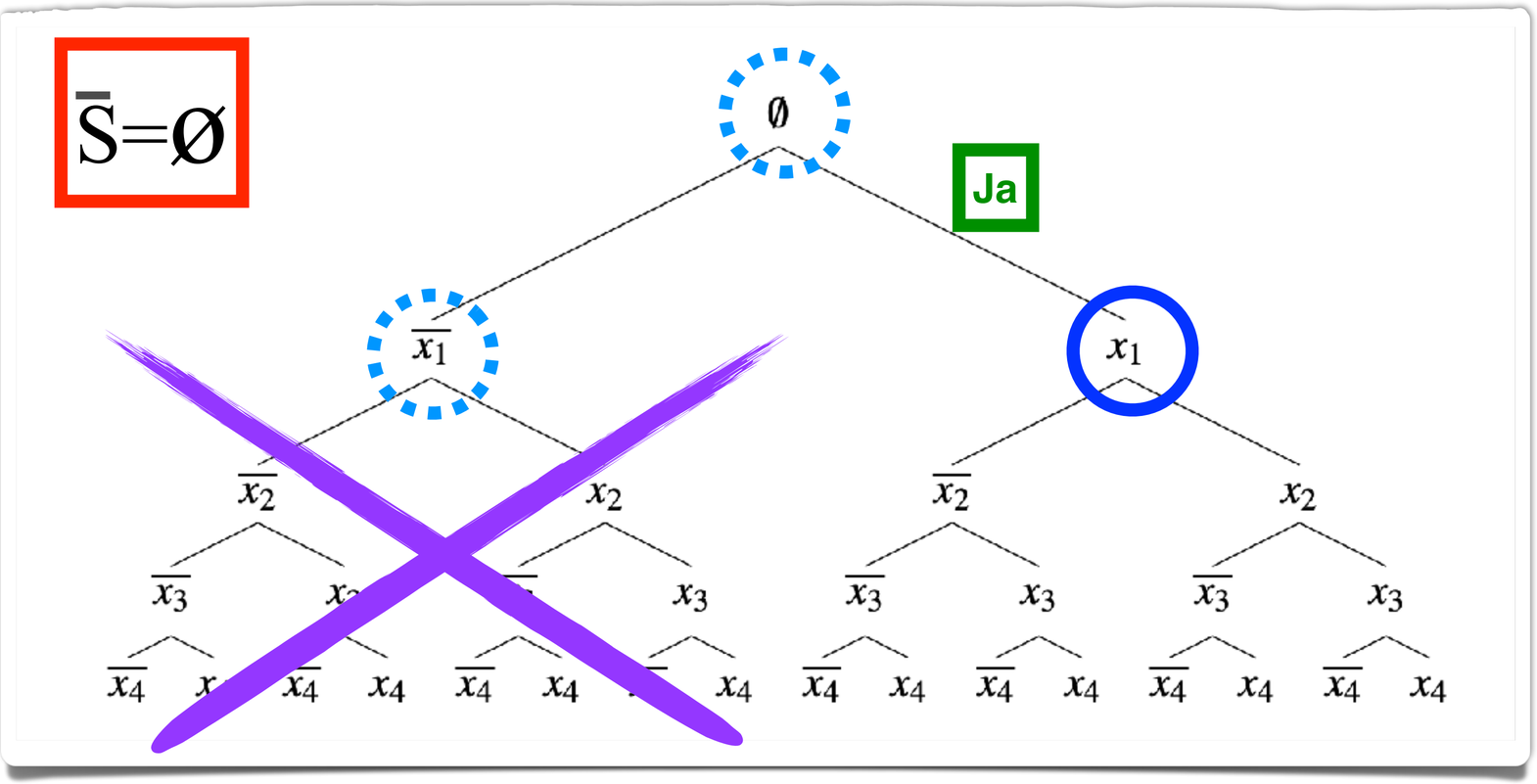
Gesamtbeispiel



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

LB=14

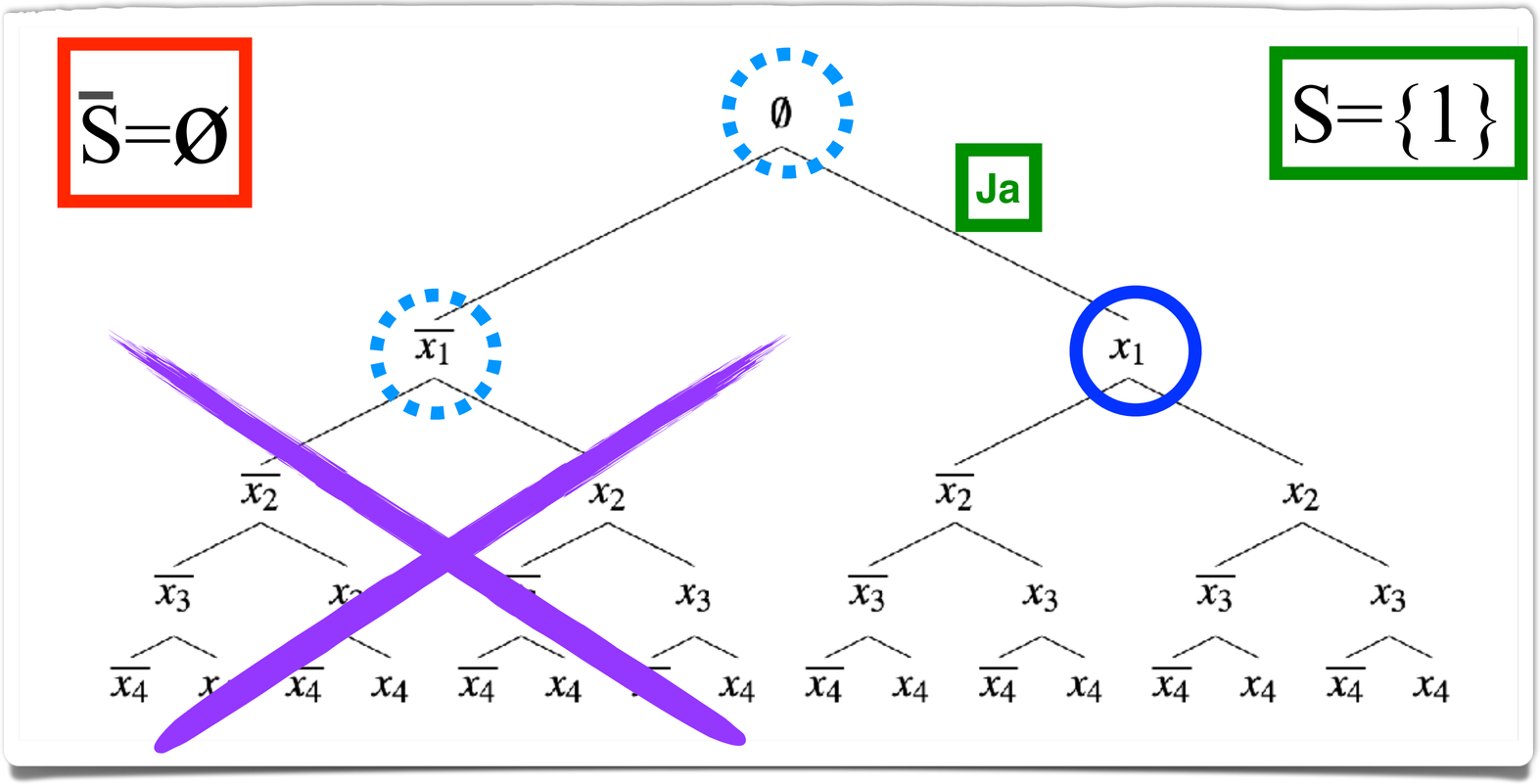
Gesamtbeispiel



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

LB=14

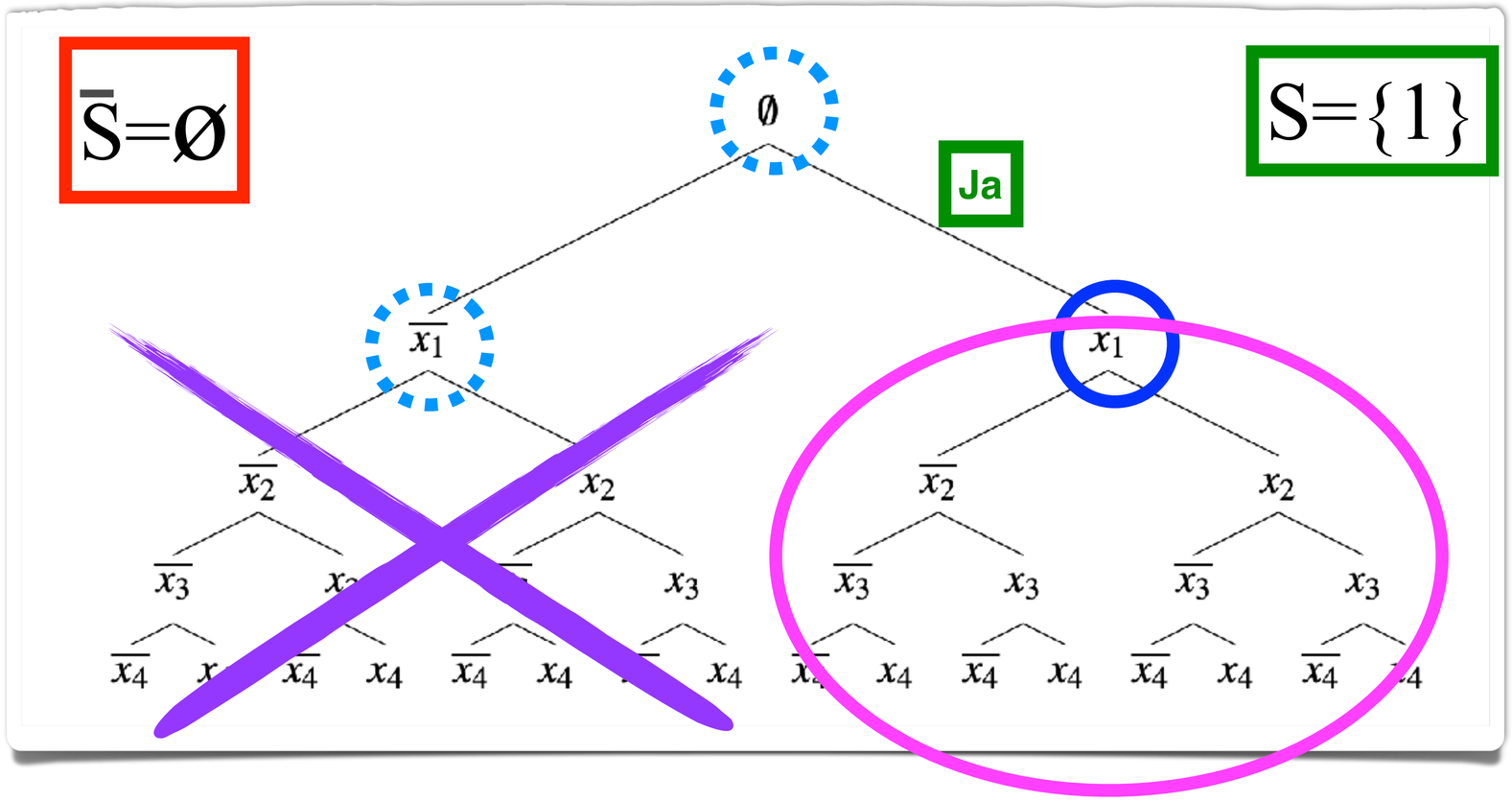
Gesamtbeispiel



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

LB=14

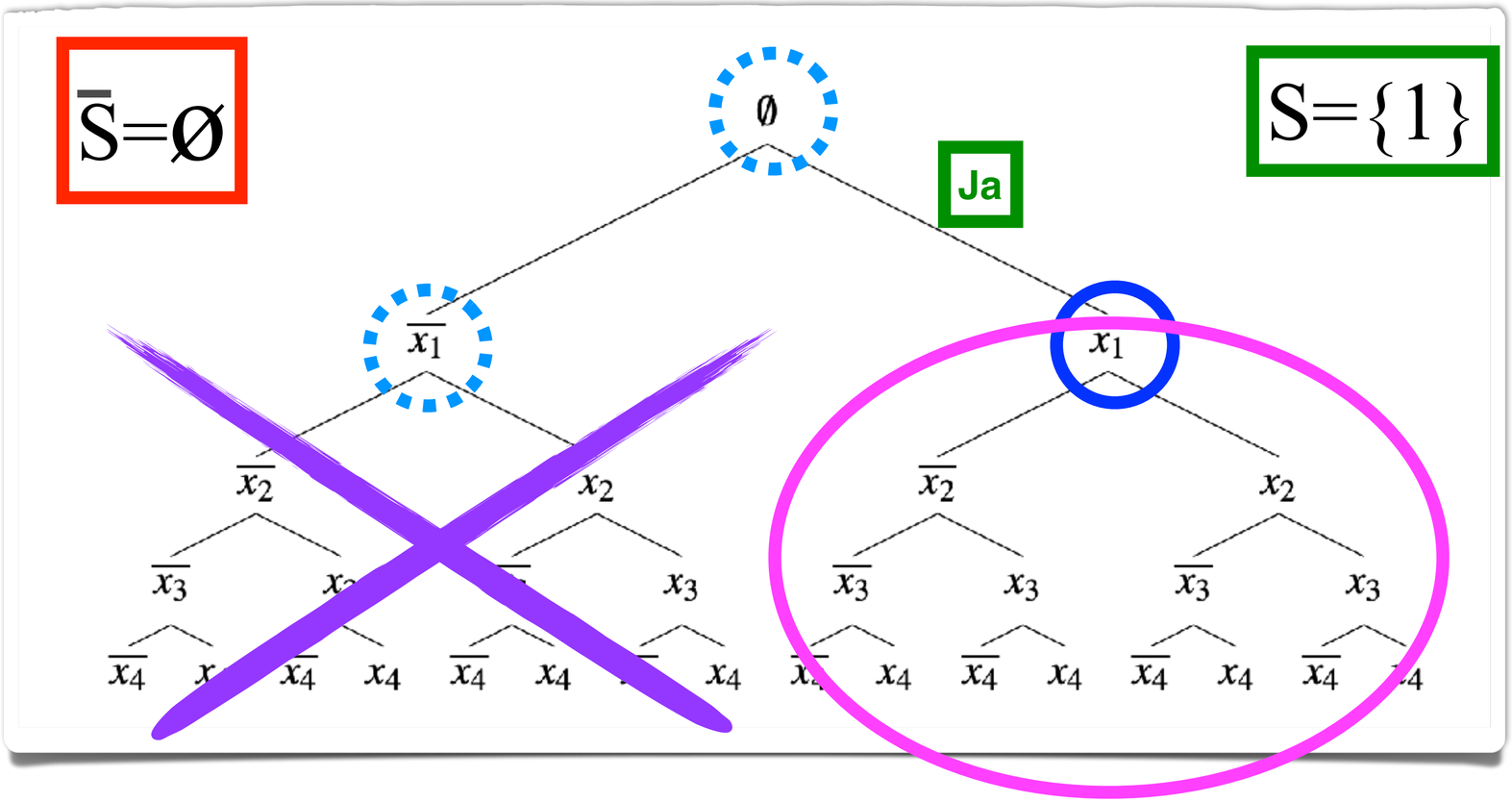
Gesamtbeispiel



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

LB=14

Gesamtbeispiel

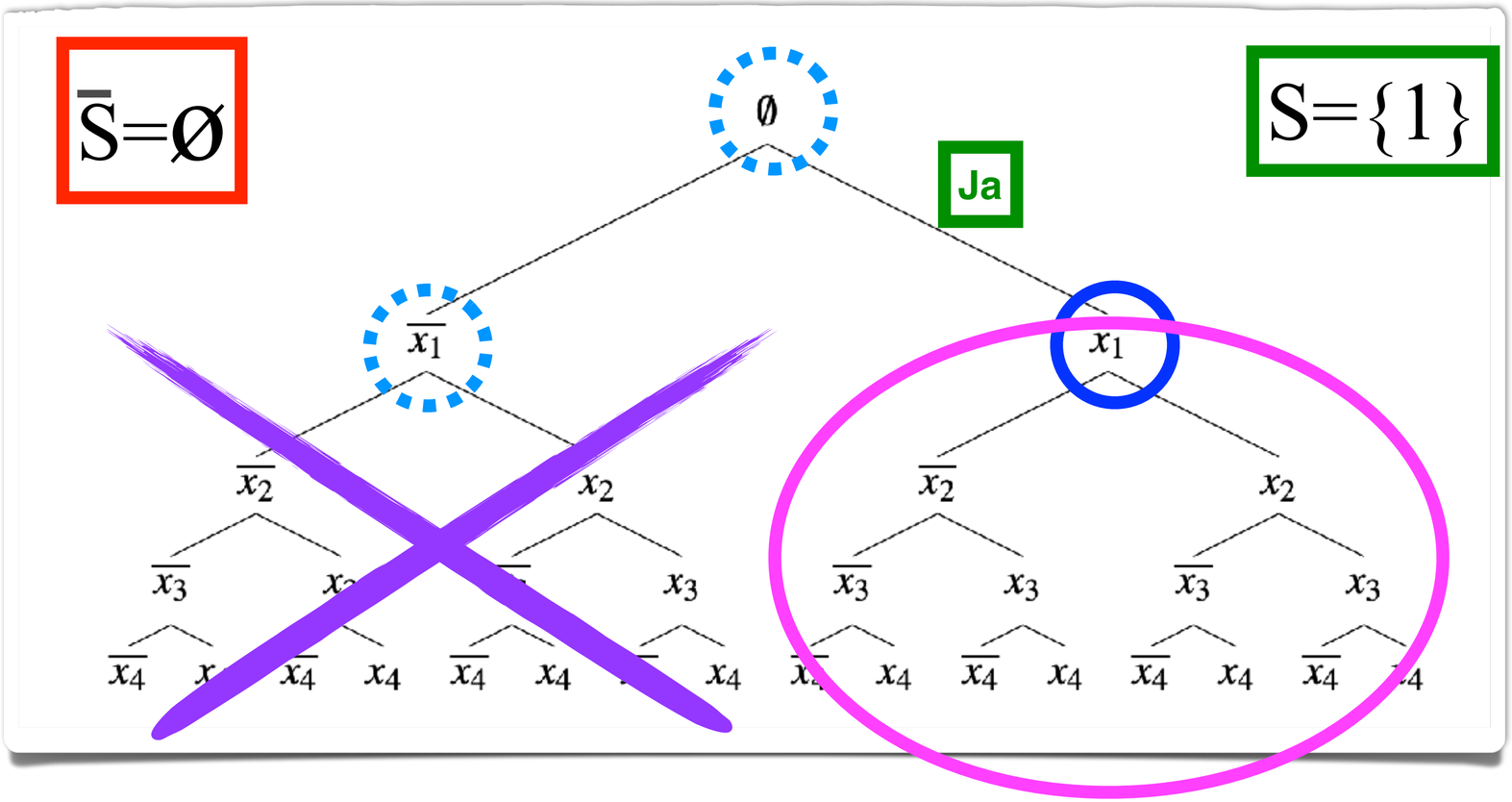


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=14

Gesamtbeispiel

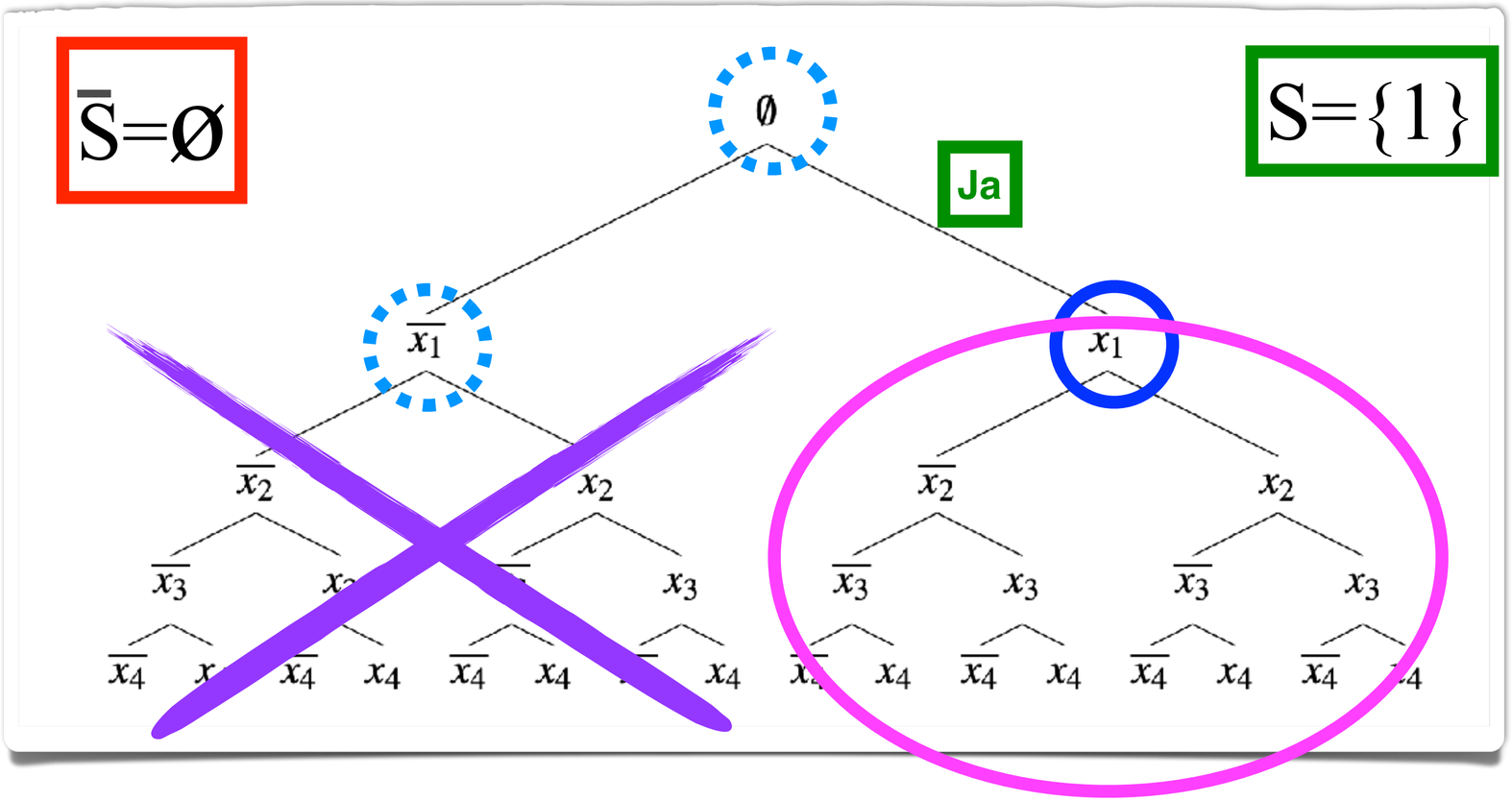


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=14

Gesamtbeispiel

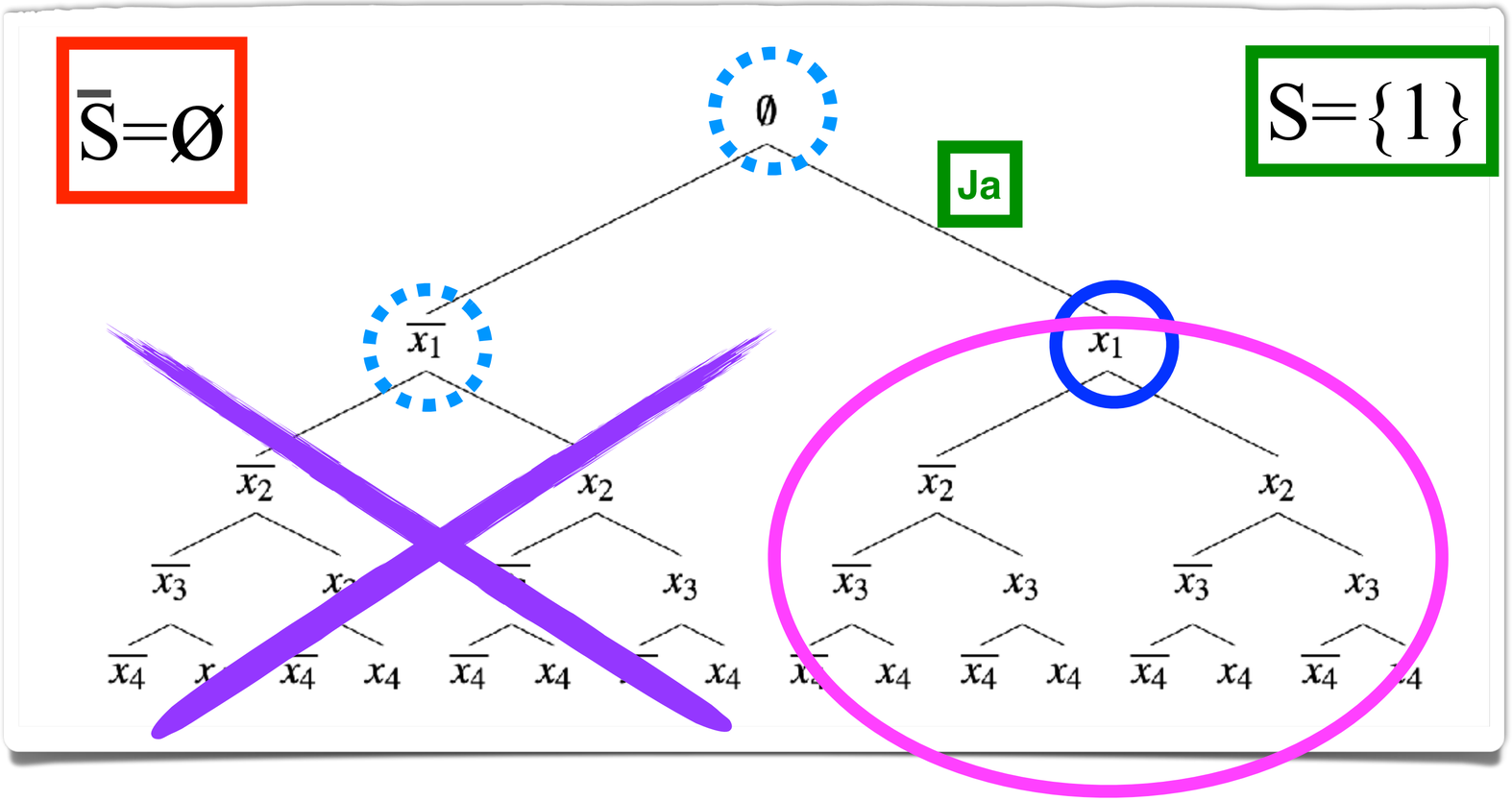


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=14

Gesamtbeispiel

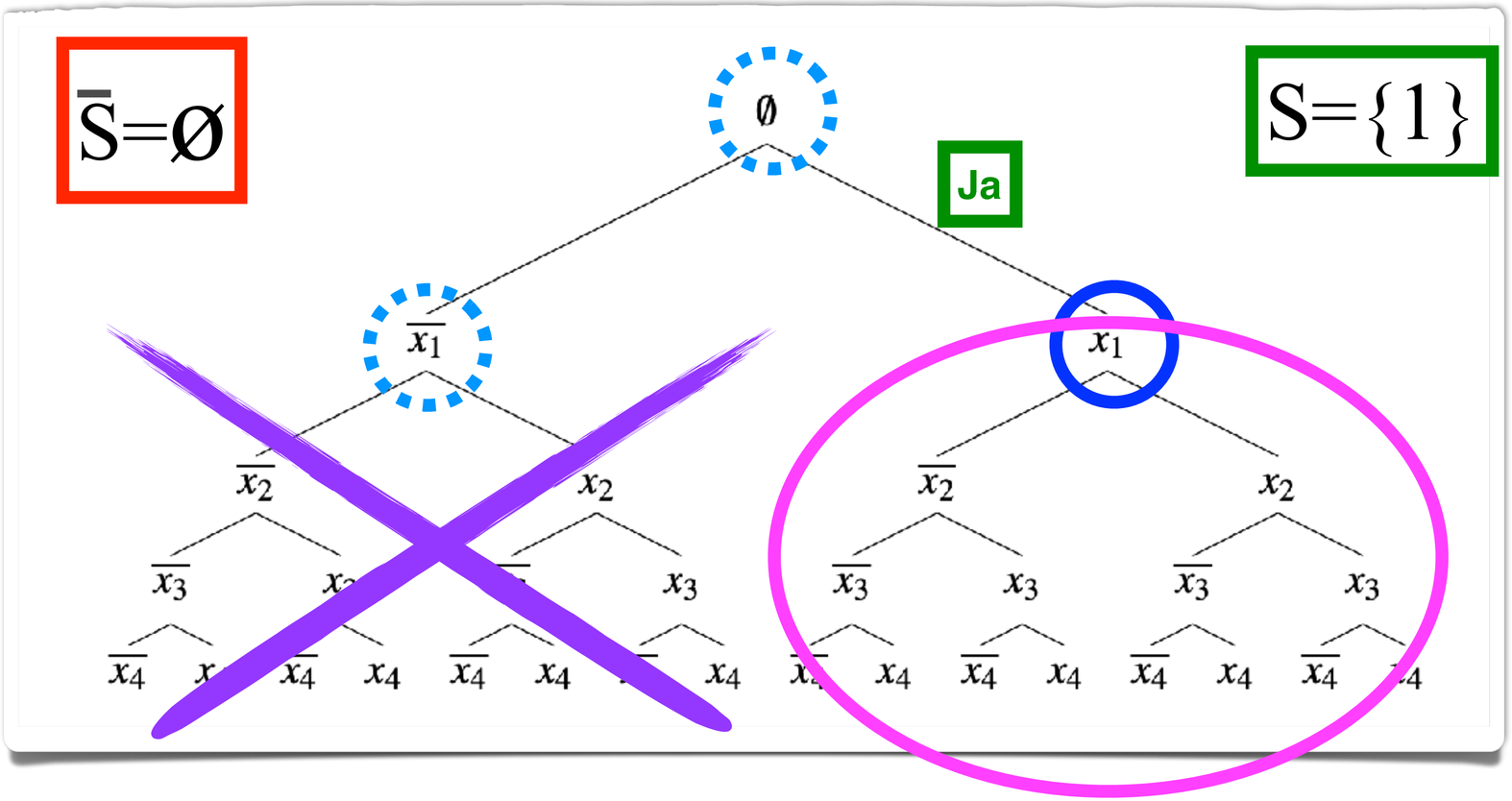


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=14

Gesamtbeispiel



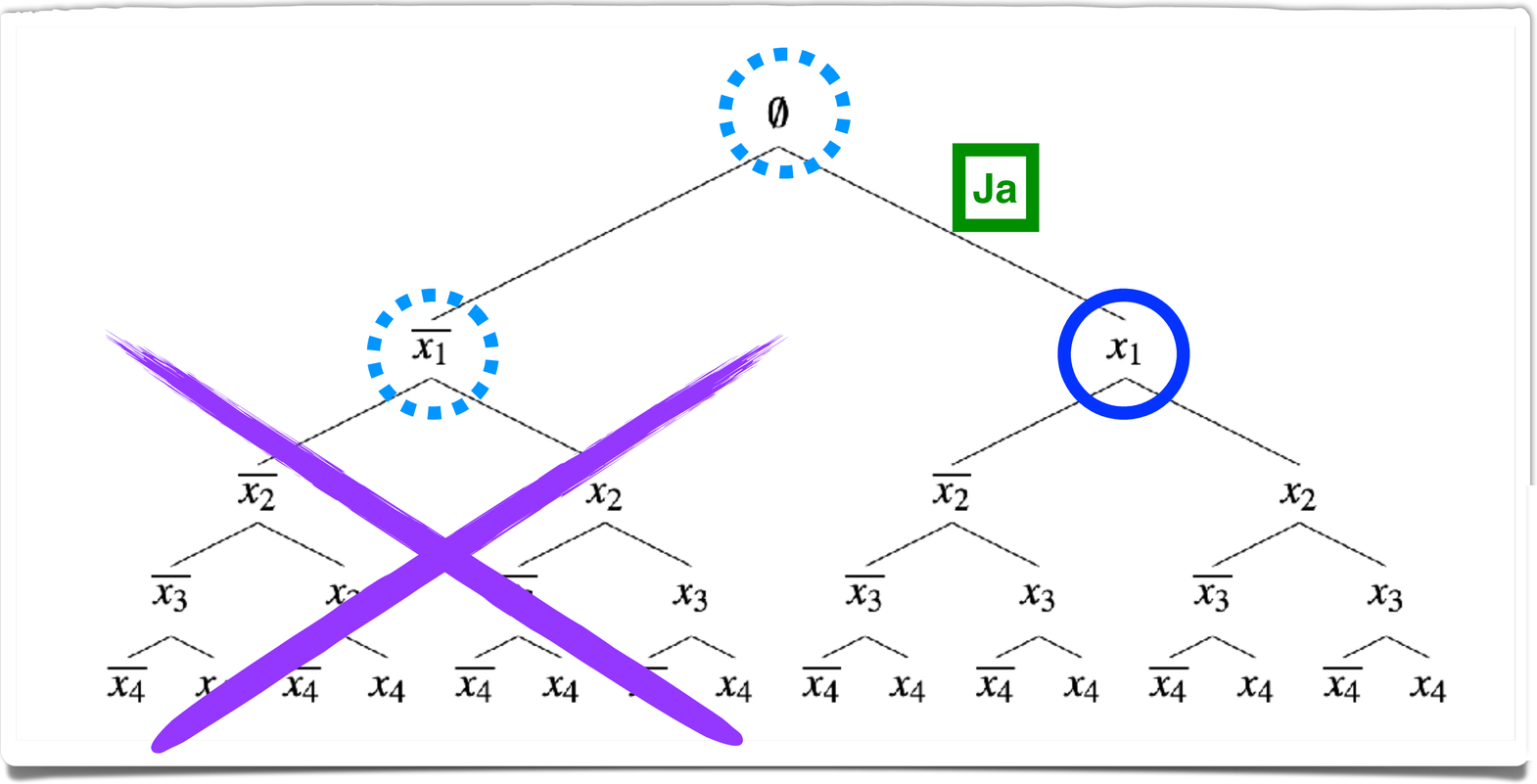
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

UB=16

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

LB=14

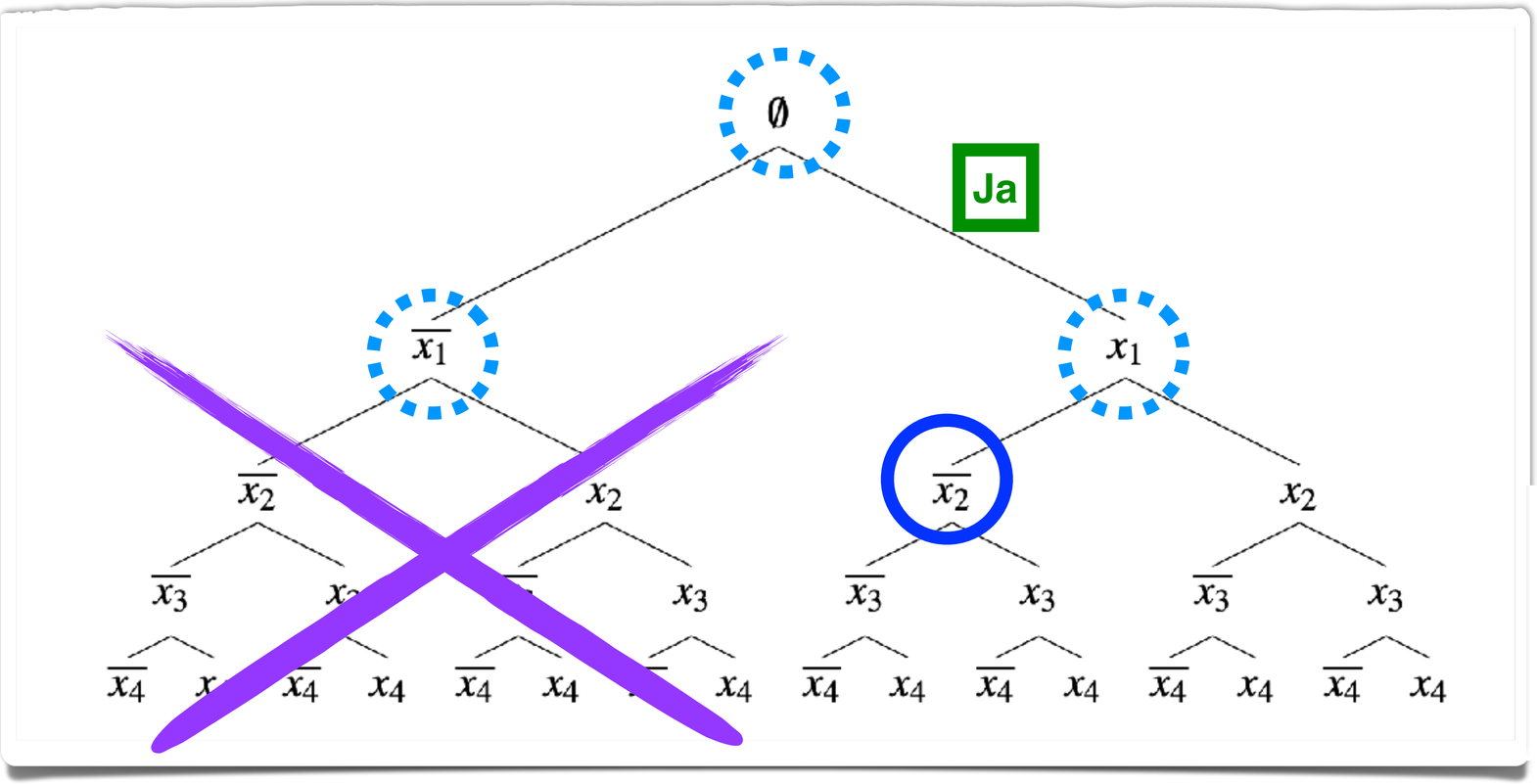
Gesamtbeispiel



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

LB=14

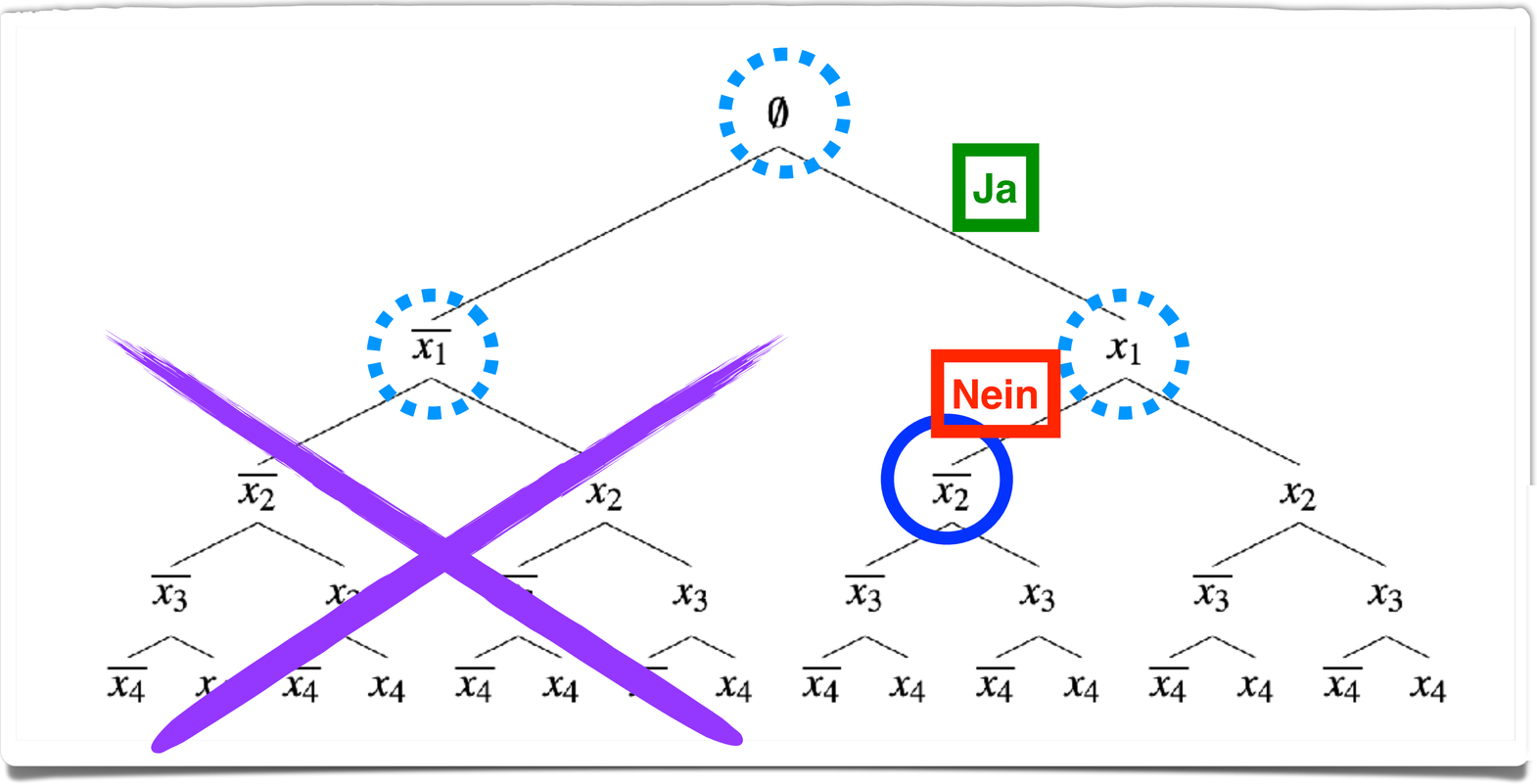
Gesamtbeispiel



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

LB=14

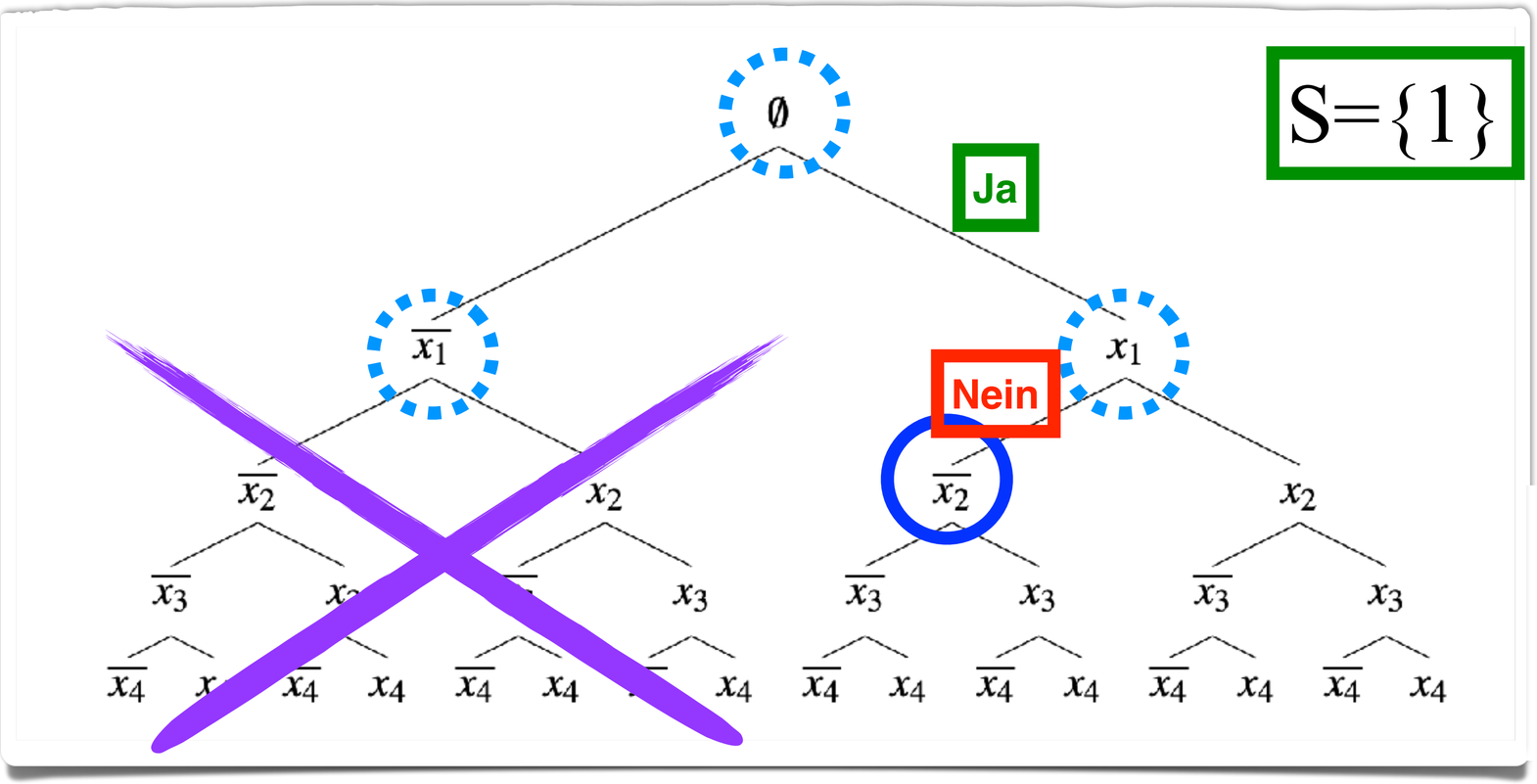
Gesamtbeispiel



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

LB=14

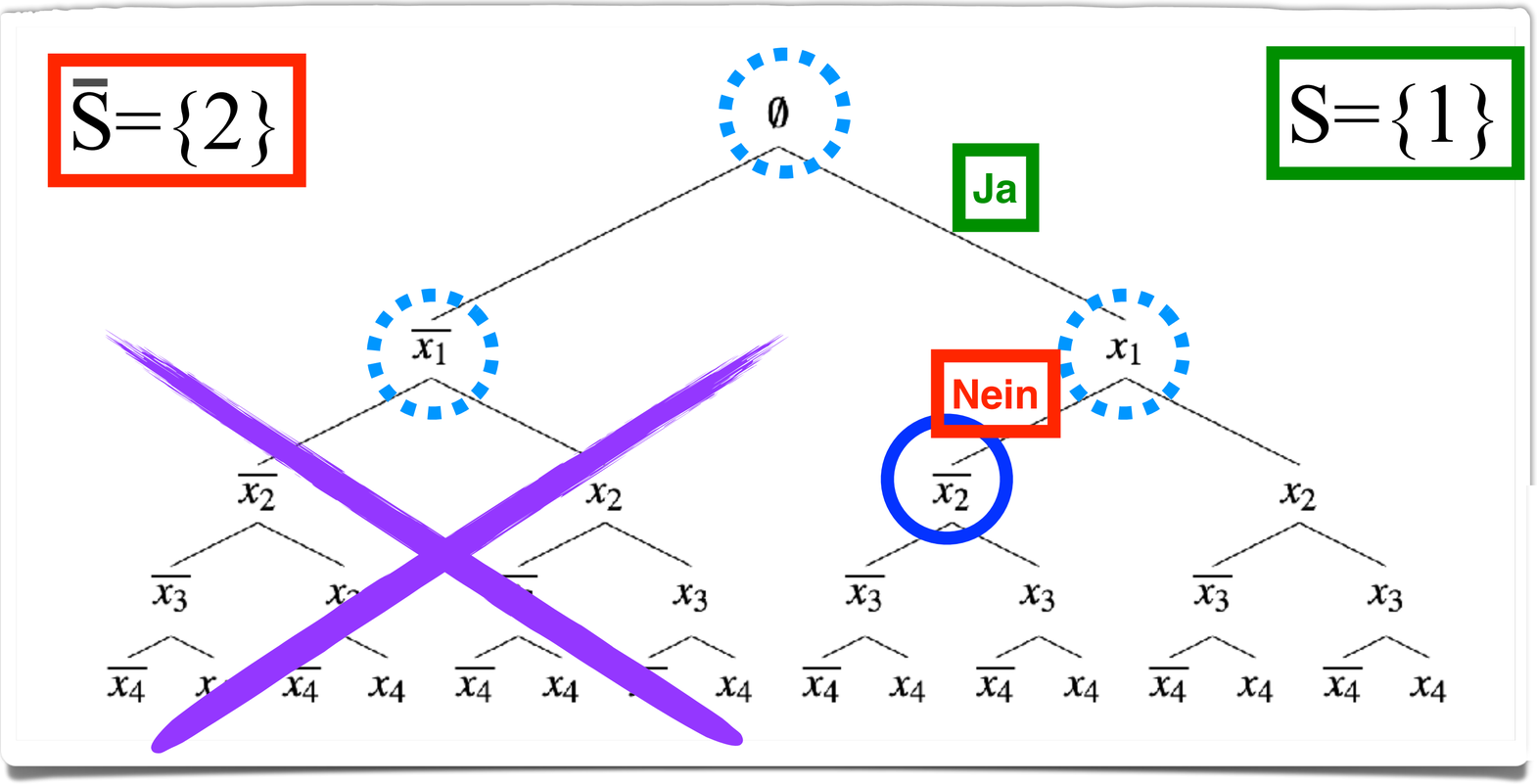
Gesamtbeispiel



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

LB=14

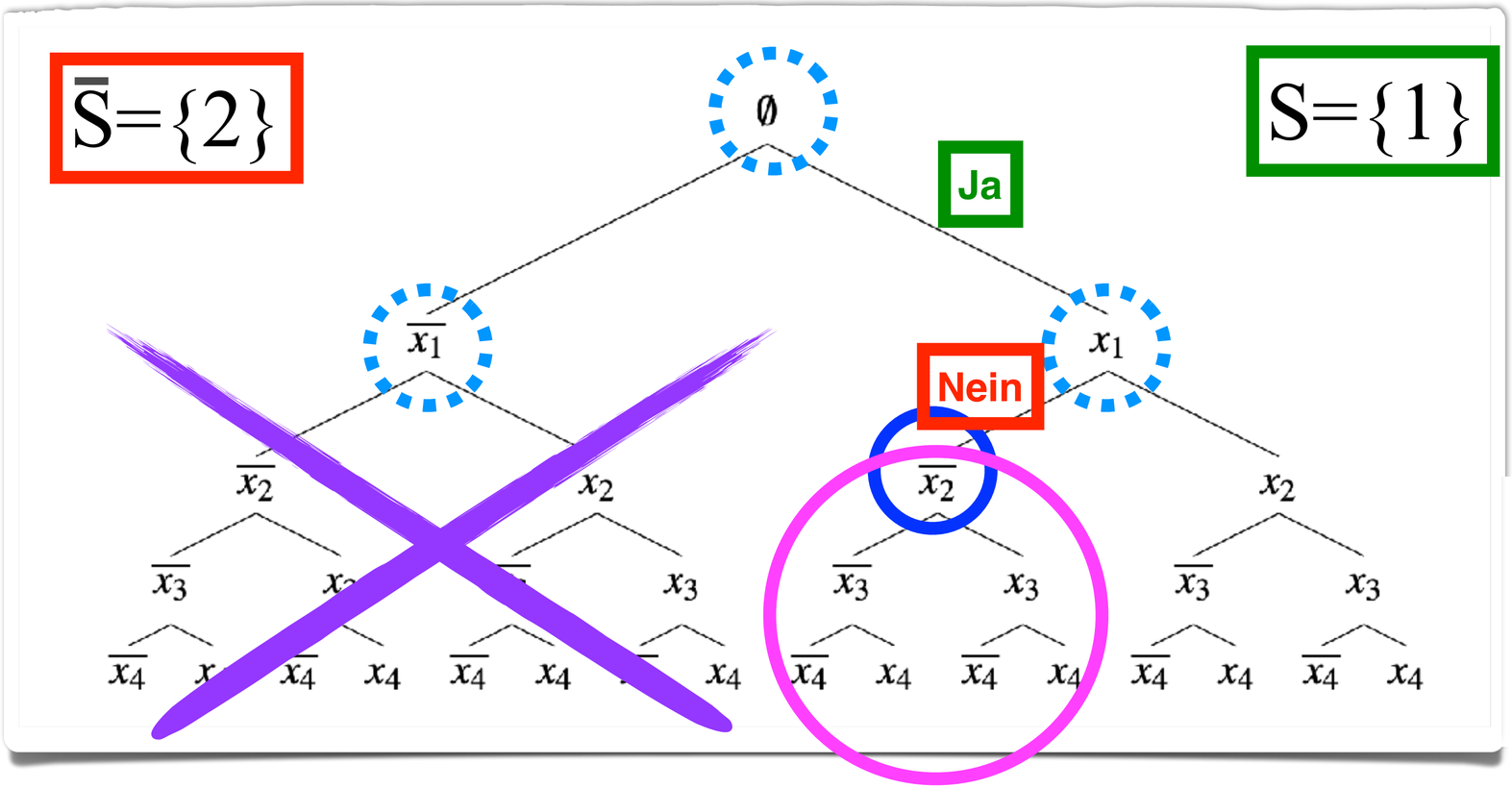
Gesamtbeispiel



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

LB=14

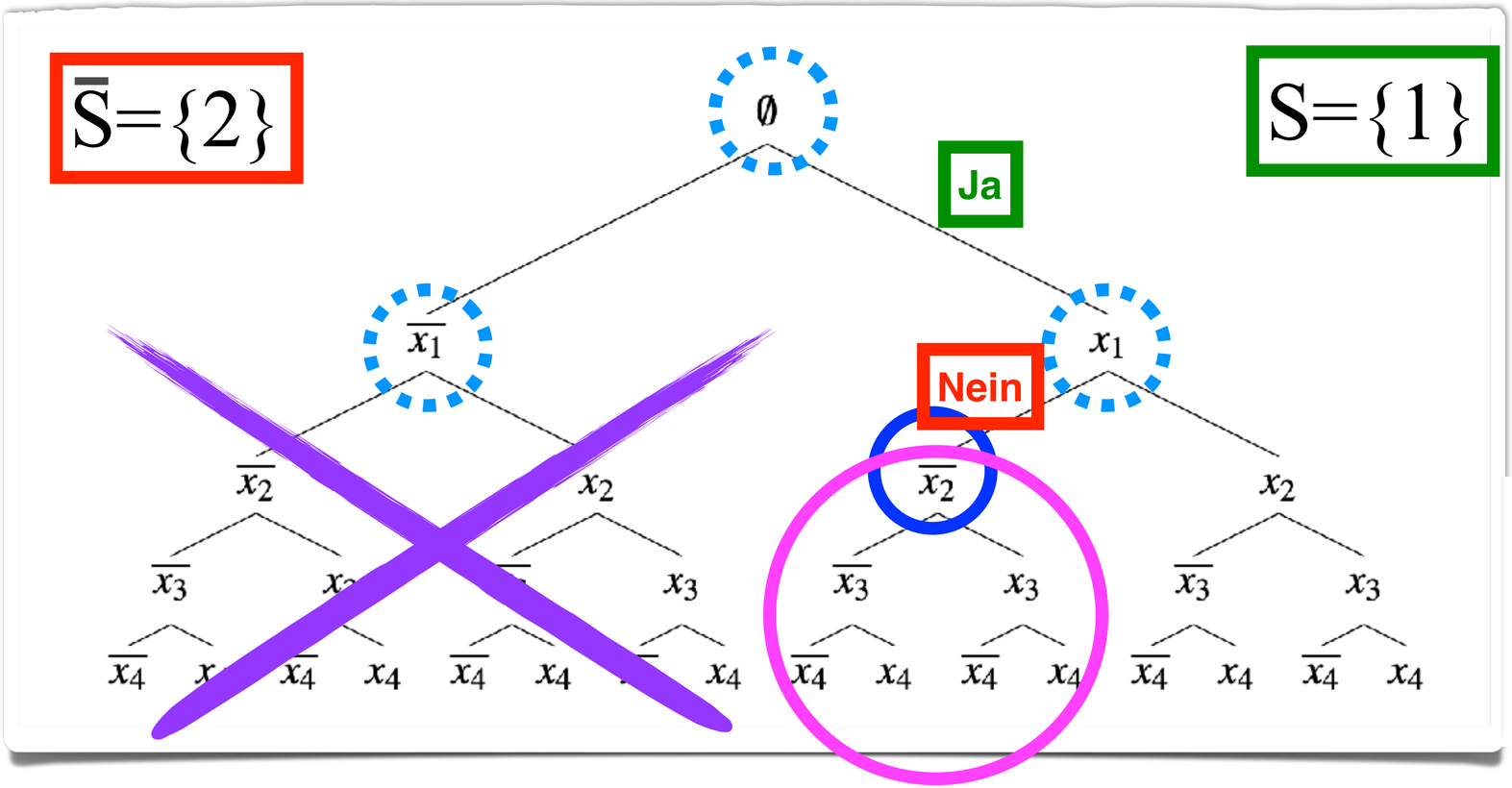
Gesamtbeispiel



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

LB=14

Gesamtbeispiel

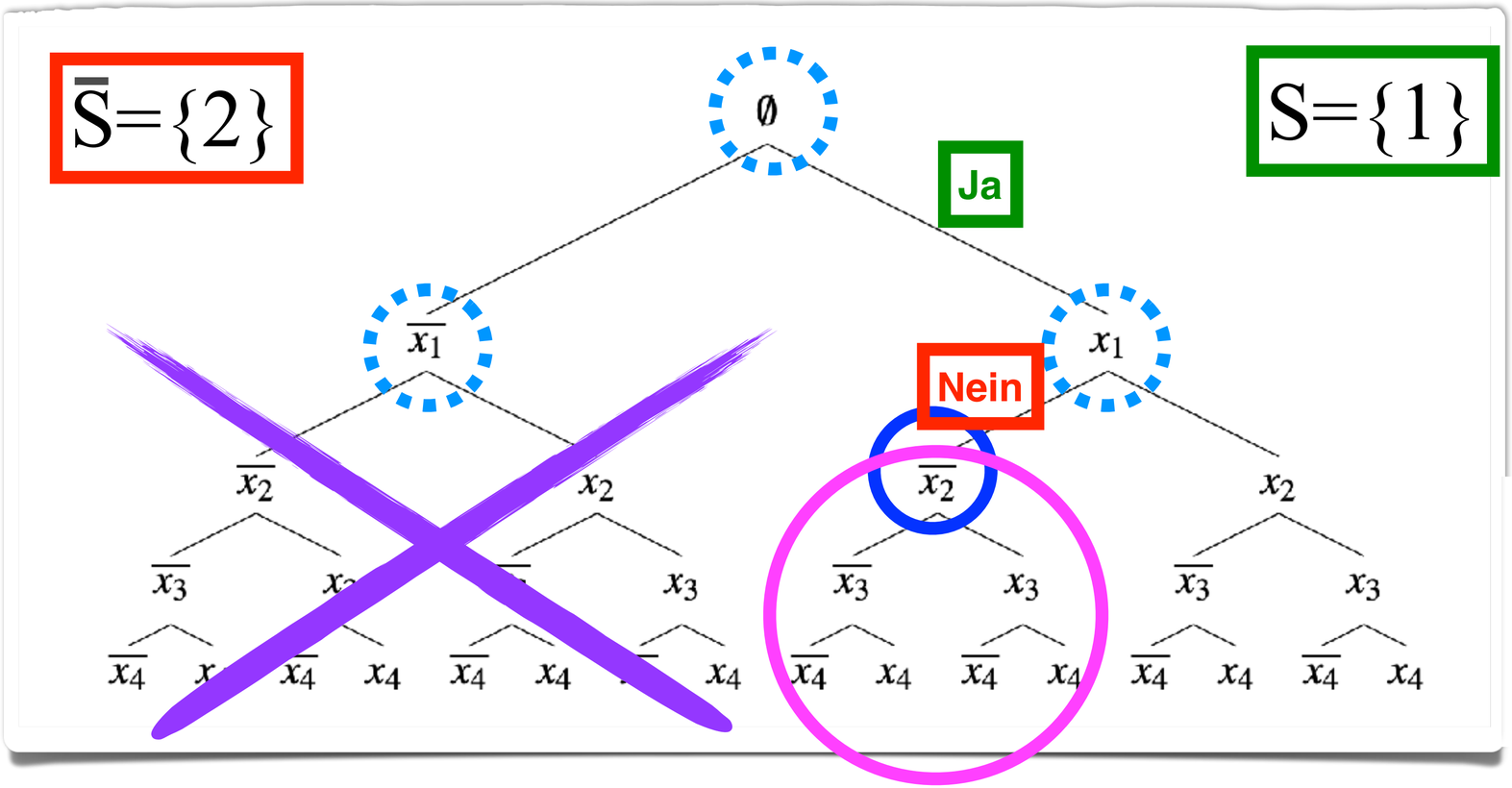


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=14

Gesamtbeispiel

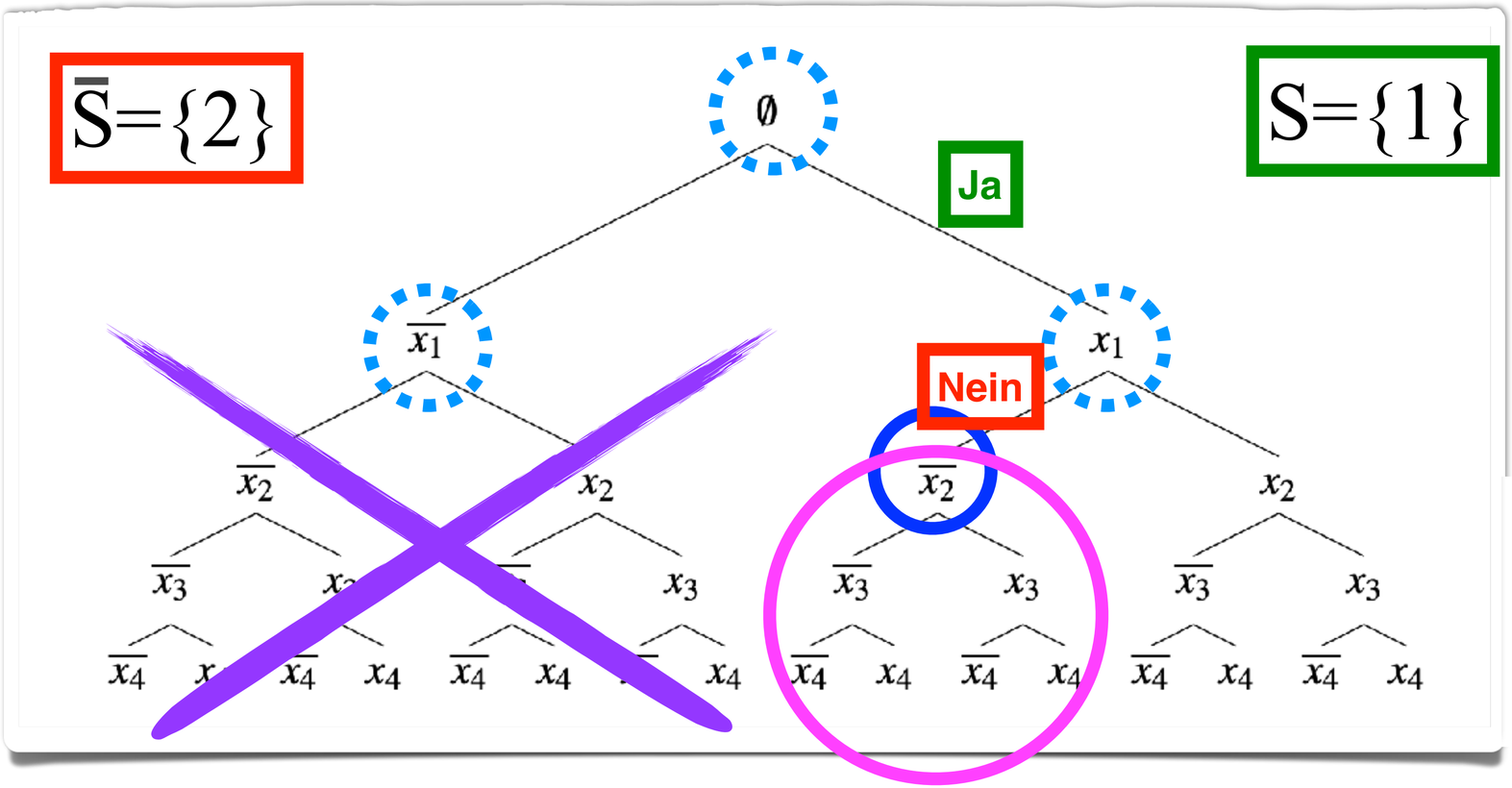


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=14

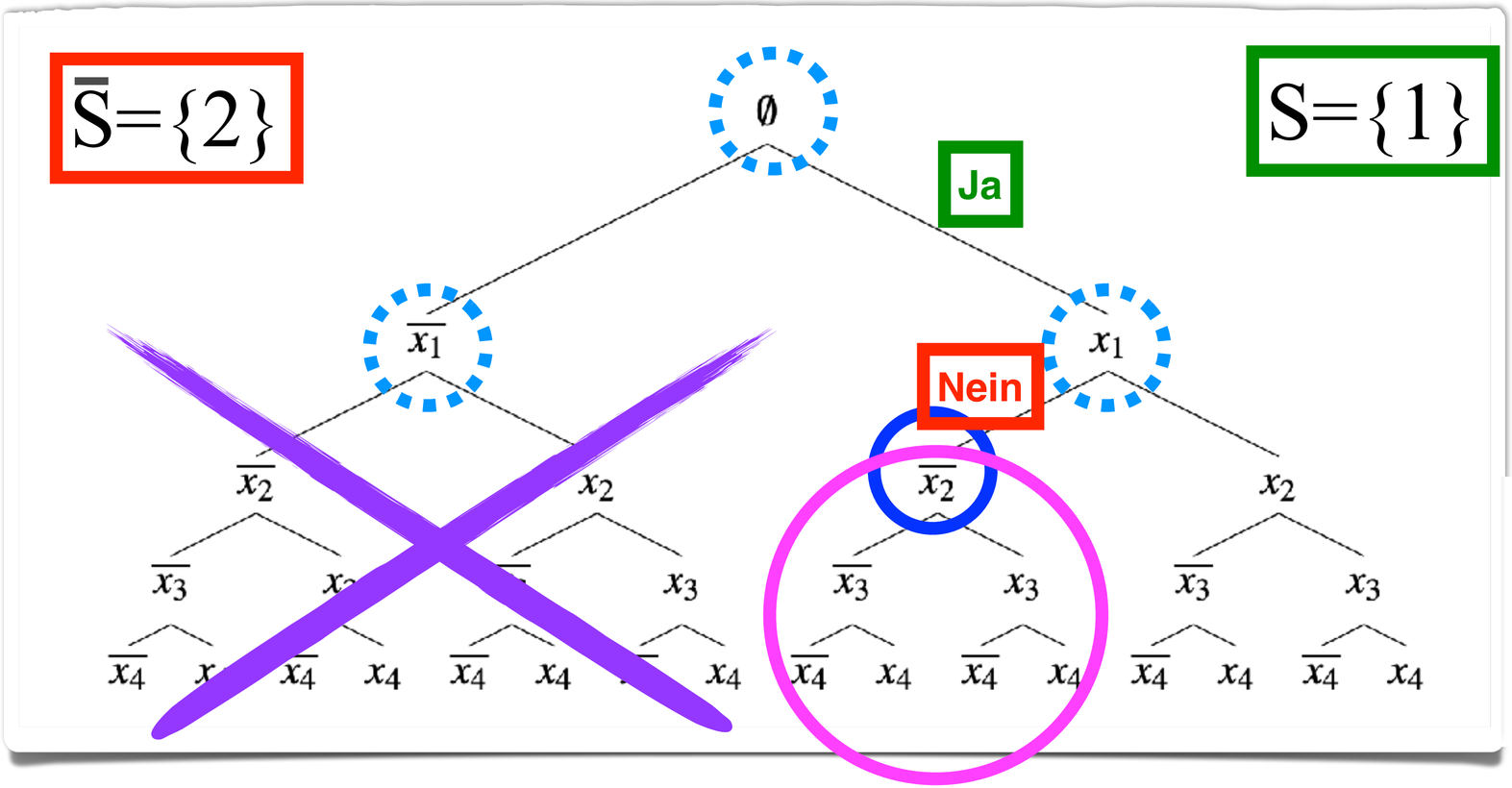
Gesamtbeispiel



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

LB=14

Gesamtbeispiel

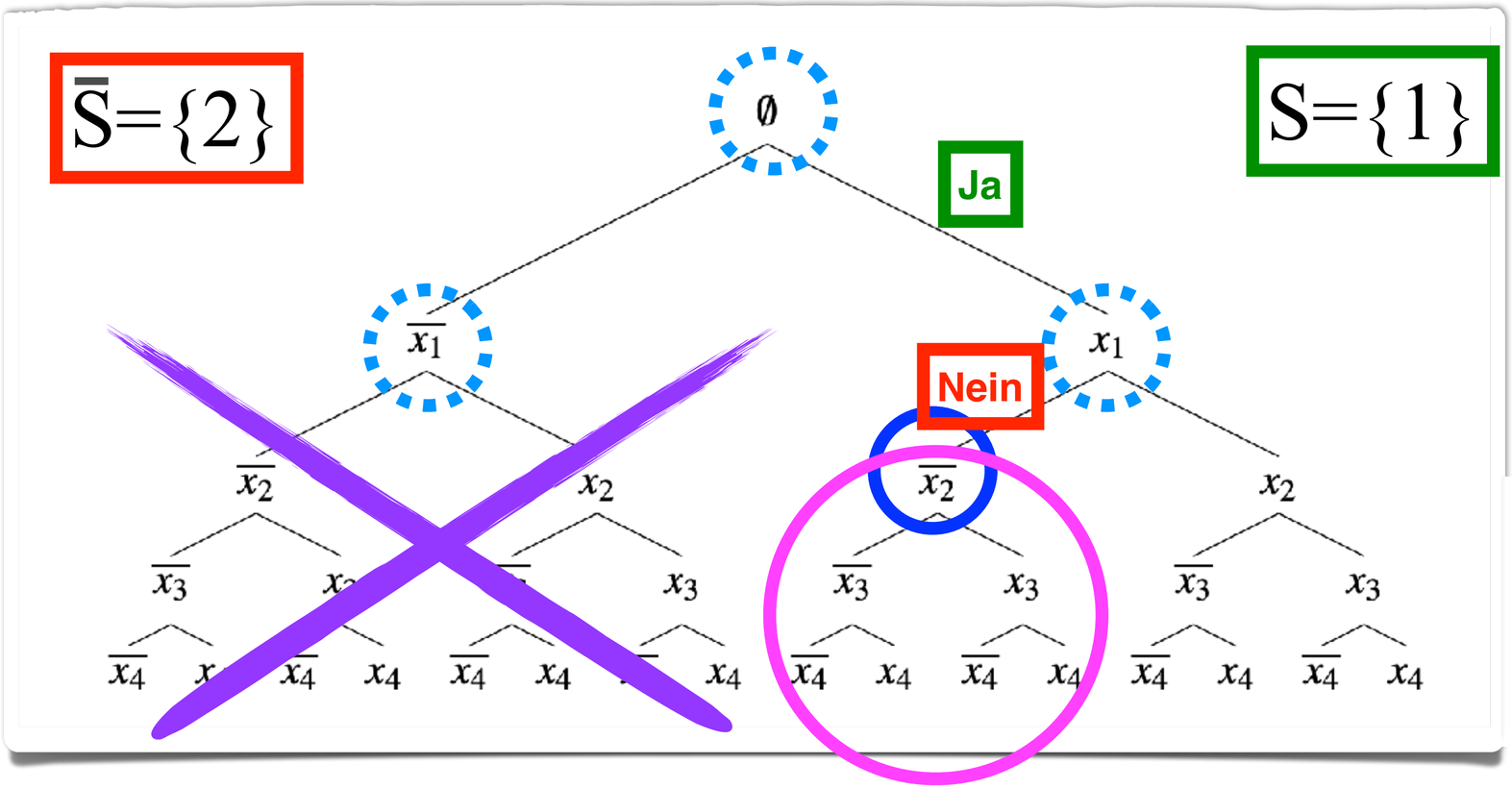


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=14

Gesamtbeispiel

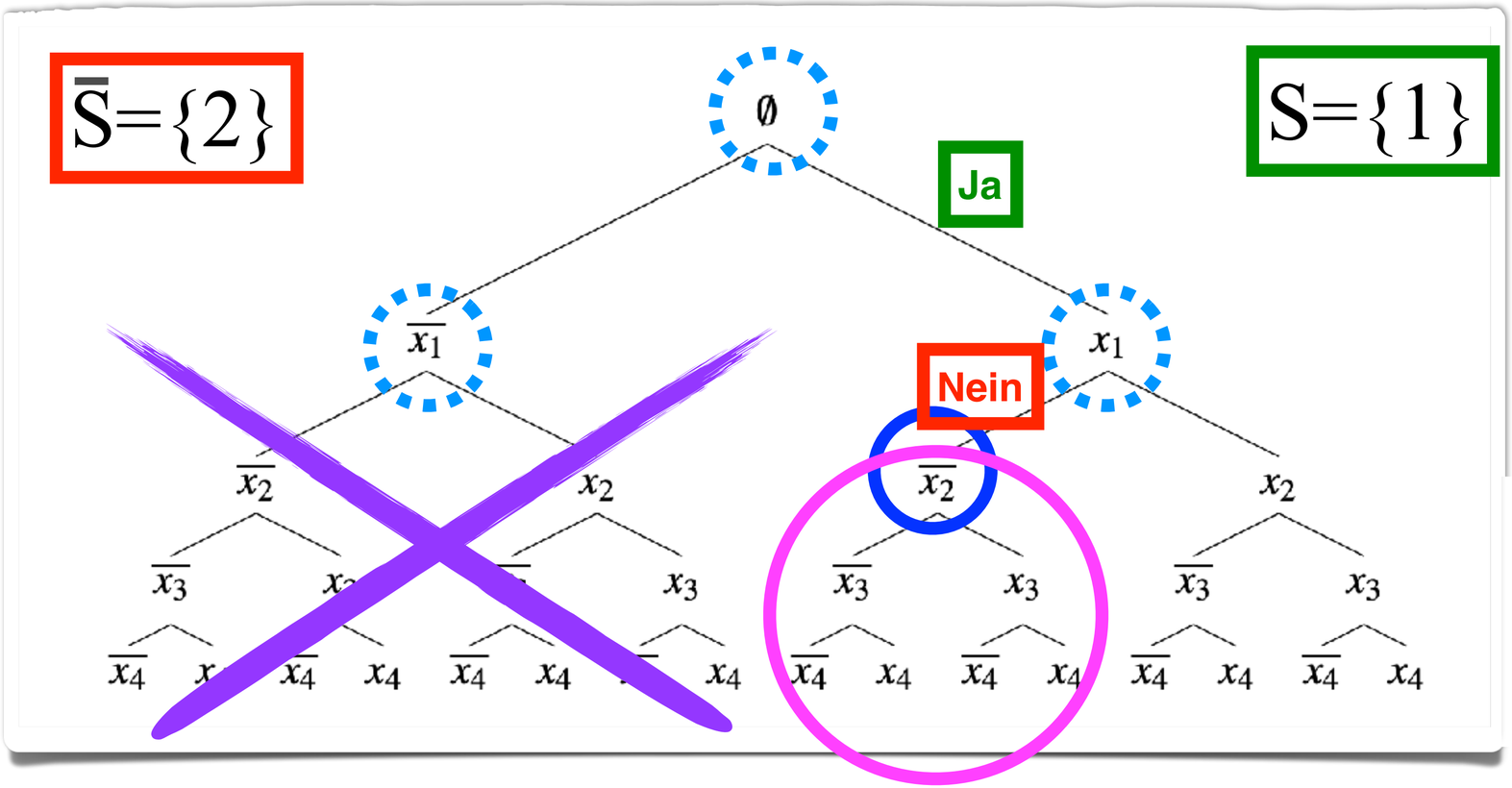


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=14

Gesamtbeispiel



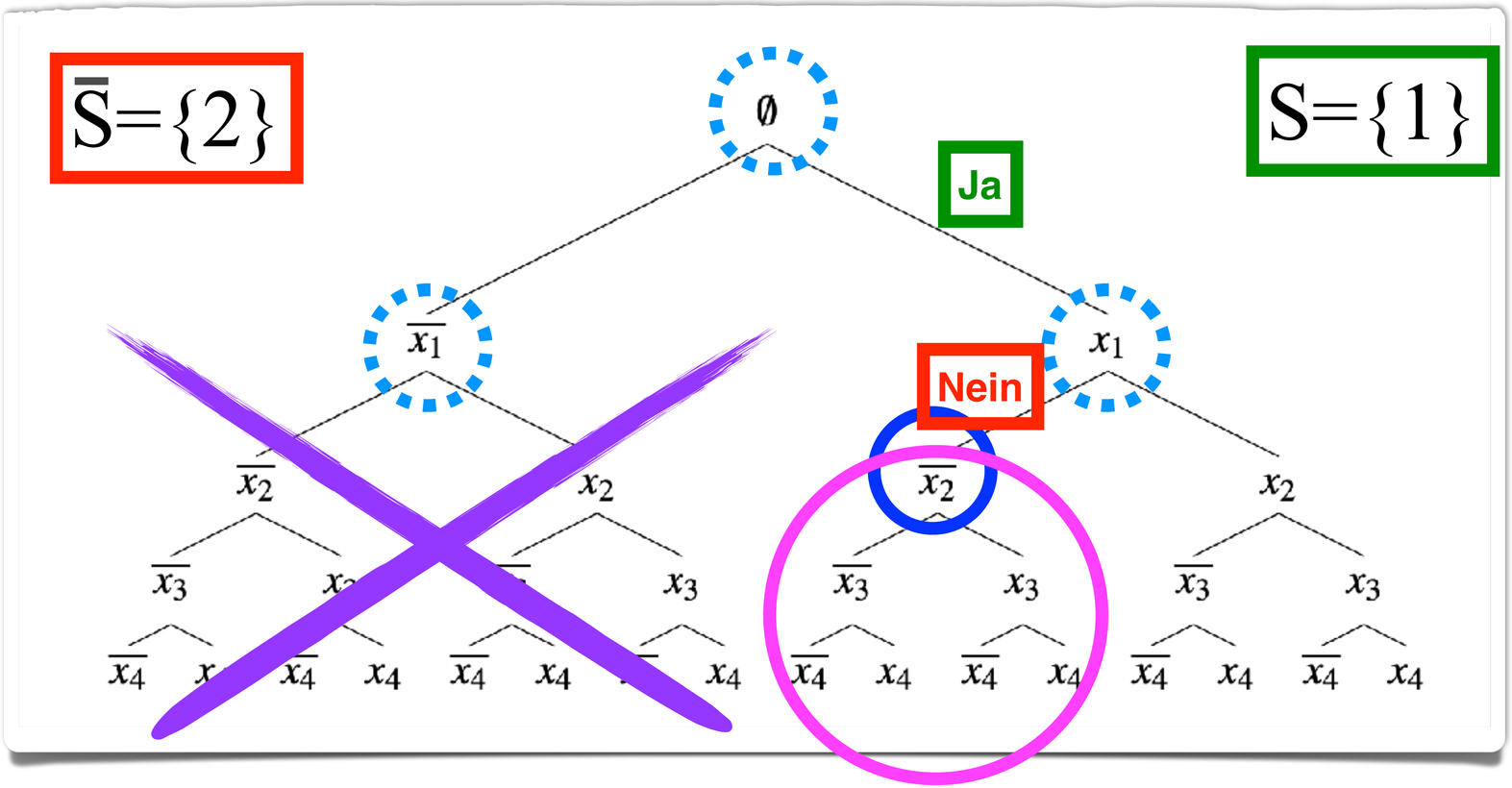
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

UB=15

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

LB=14

Gesamtbeispiel



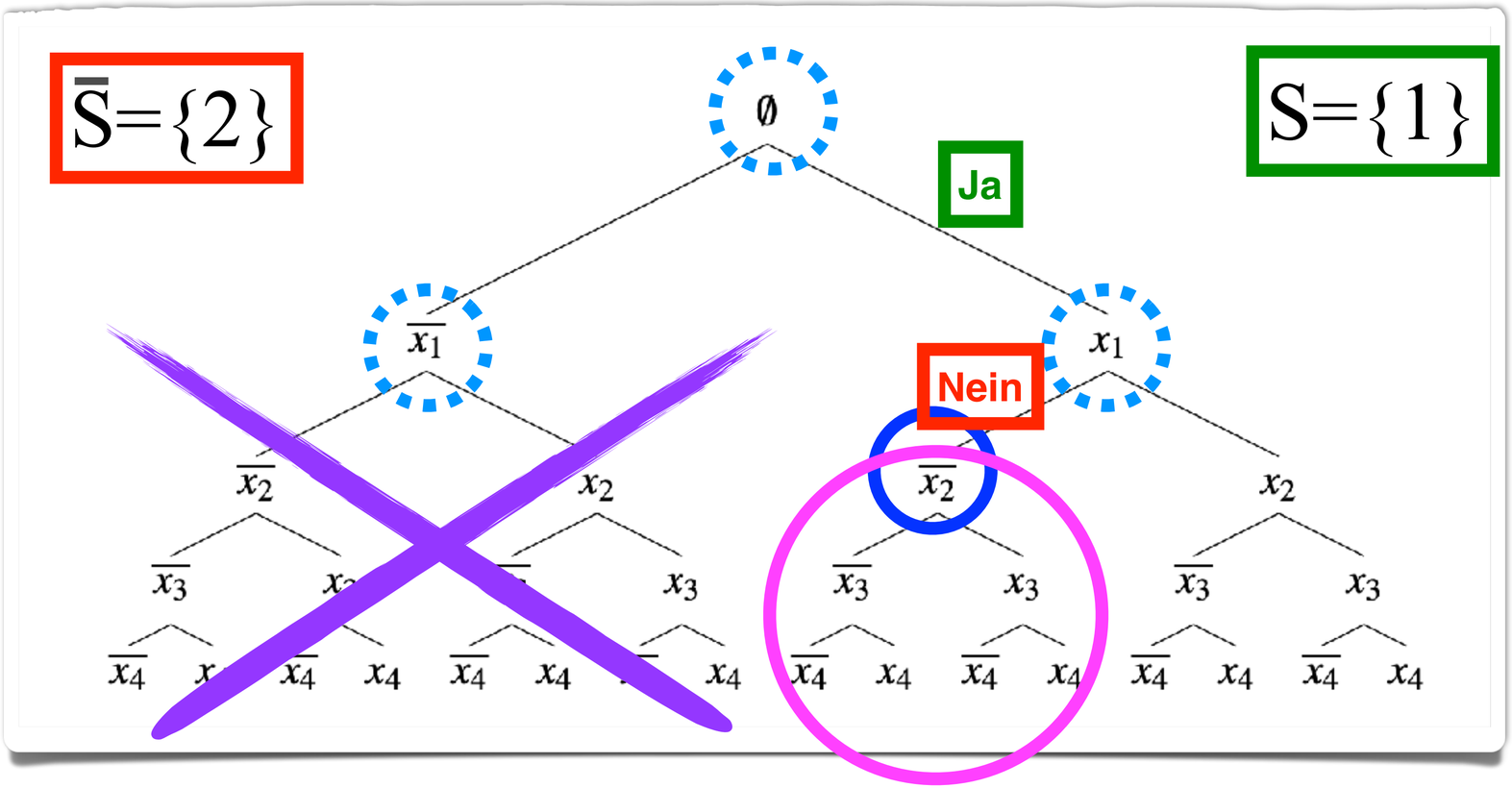
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	6	8	9	6	7	3

UB=15

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

LB=14

Gesamtbeispiel



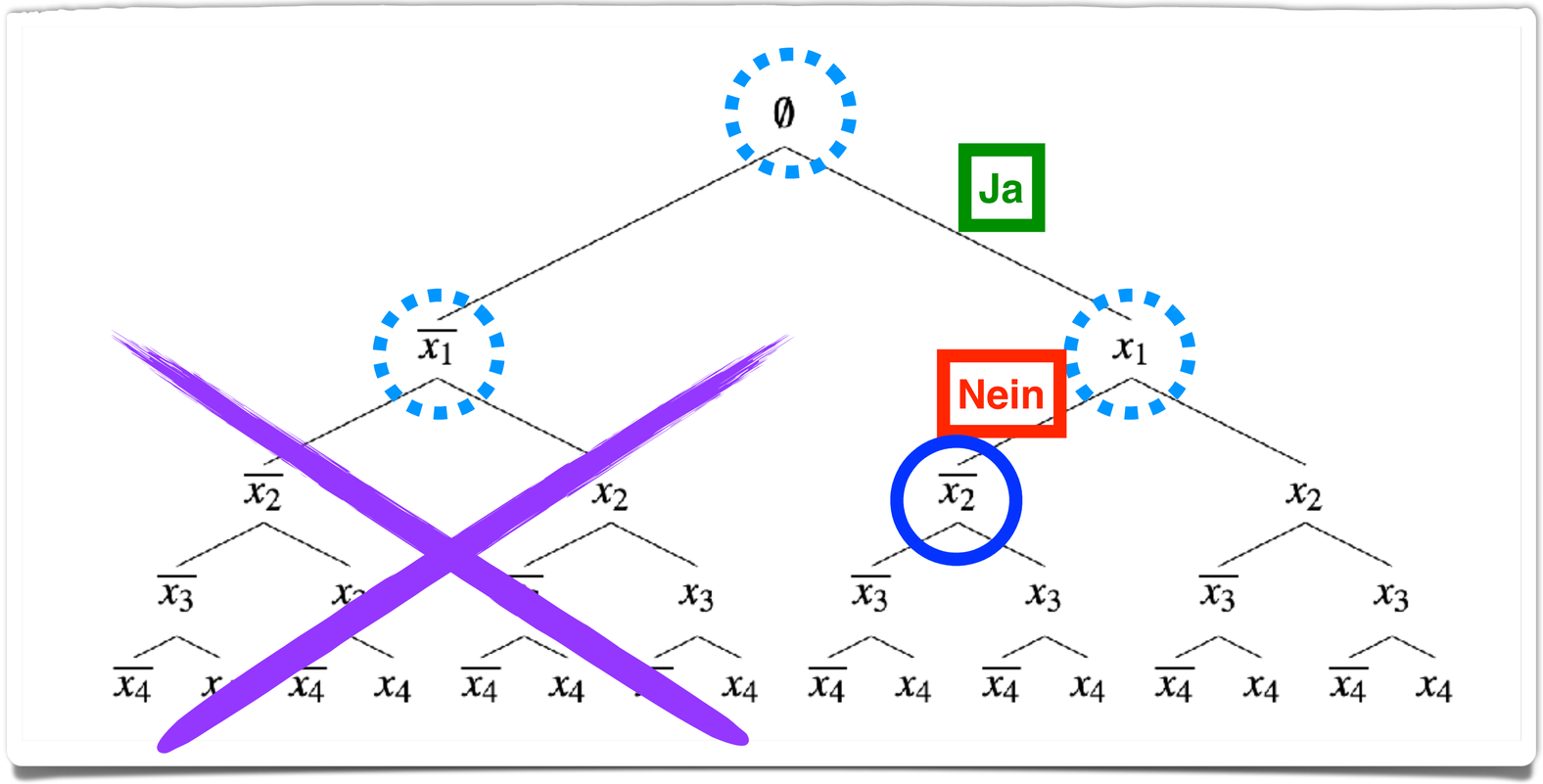
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	6	8	9	6	7	3

UB=15

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

LB=14

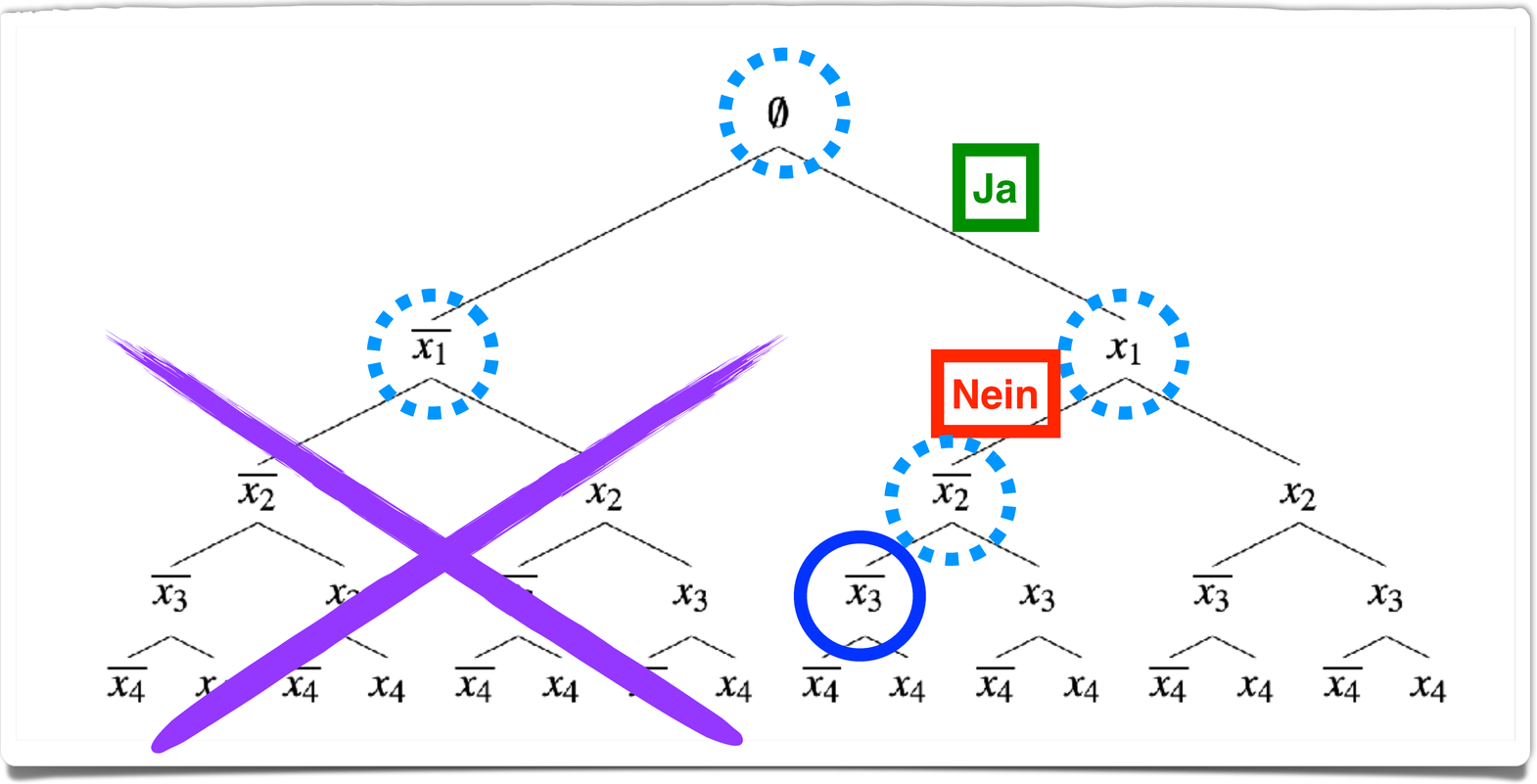
Gesamtbeispiel



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

LB=14

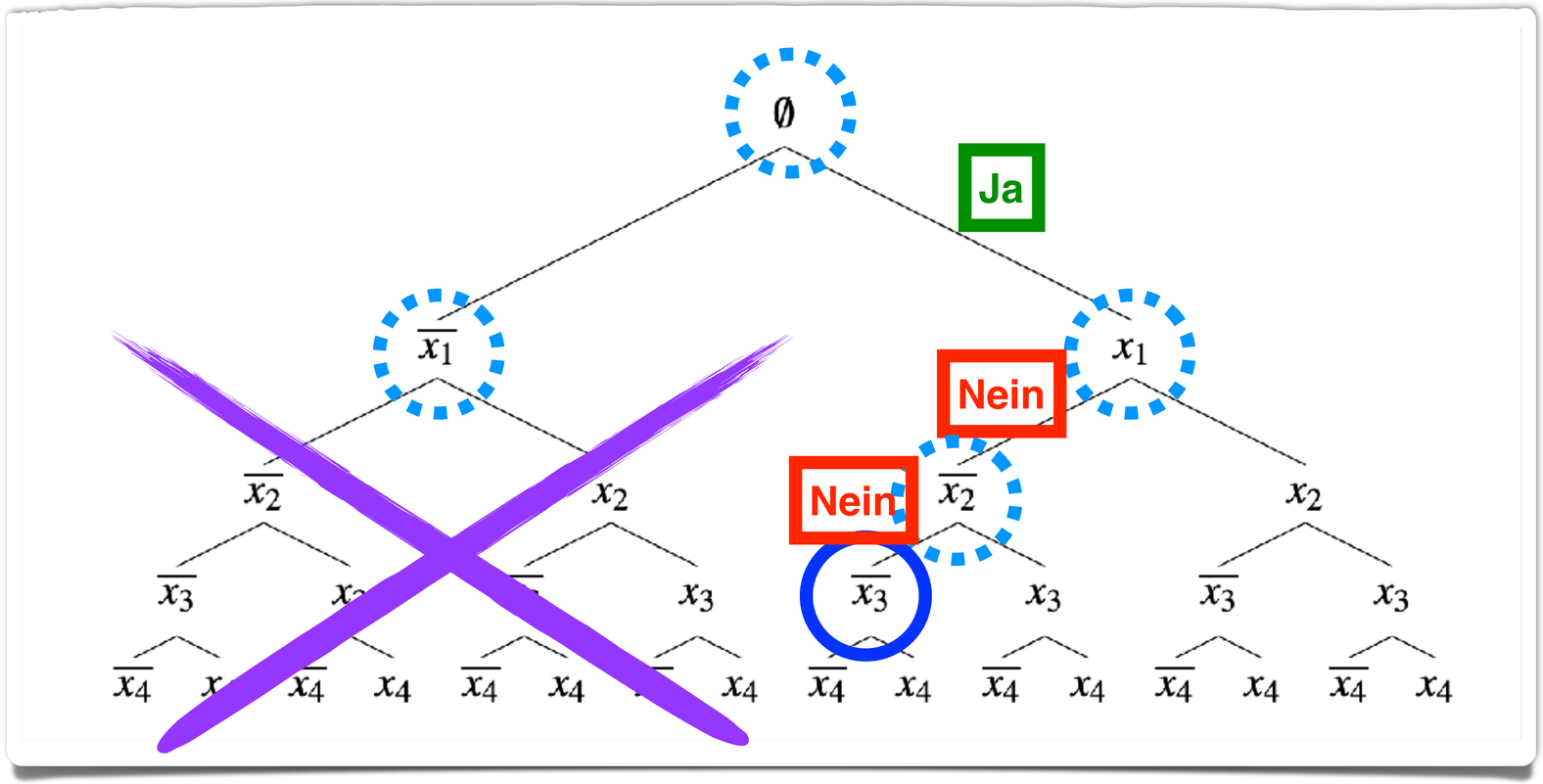
Gesamtbeispiel



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

LB=14

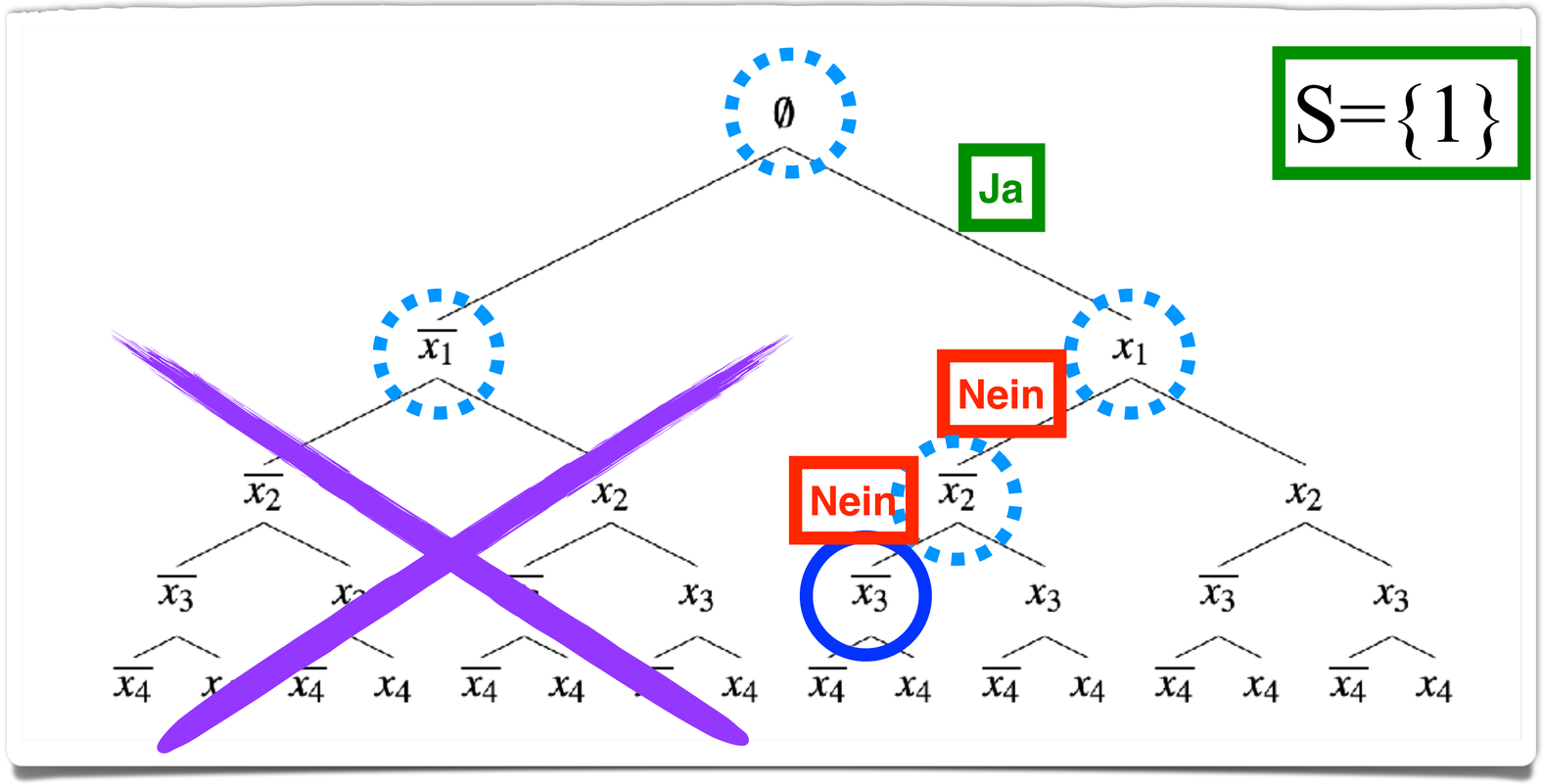
Gesamtbeispiel



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

LB=14

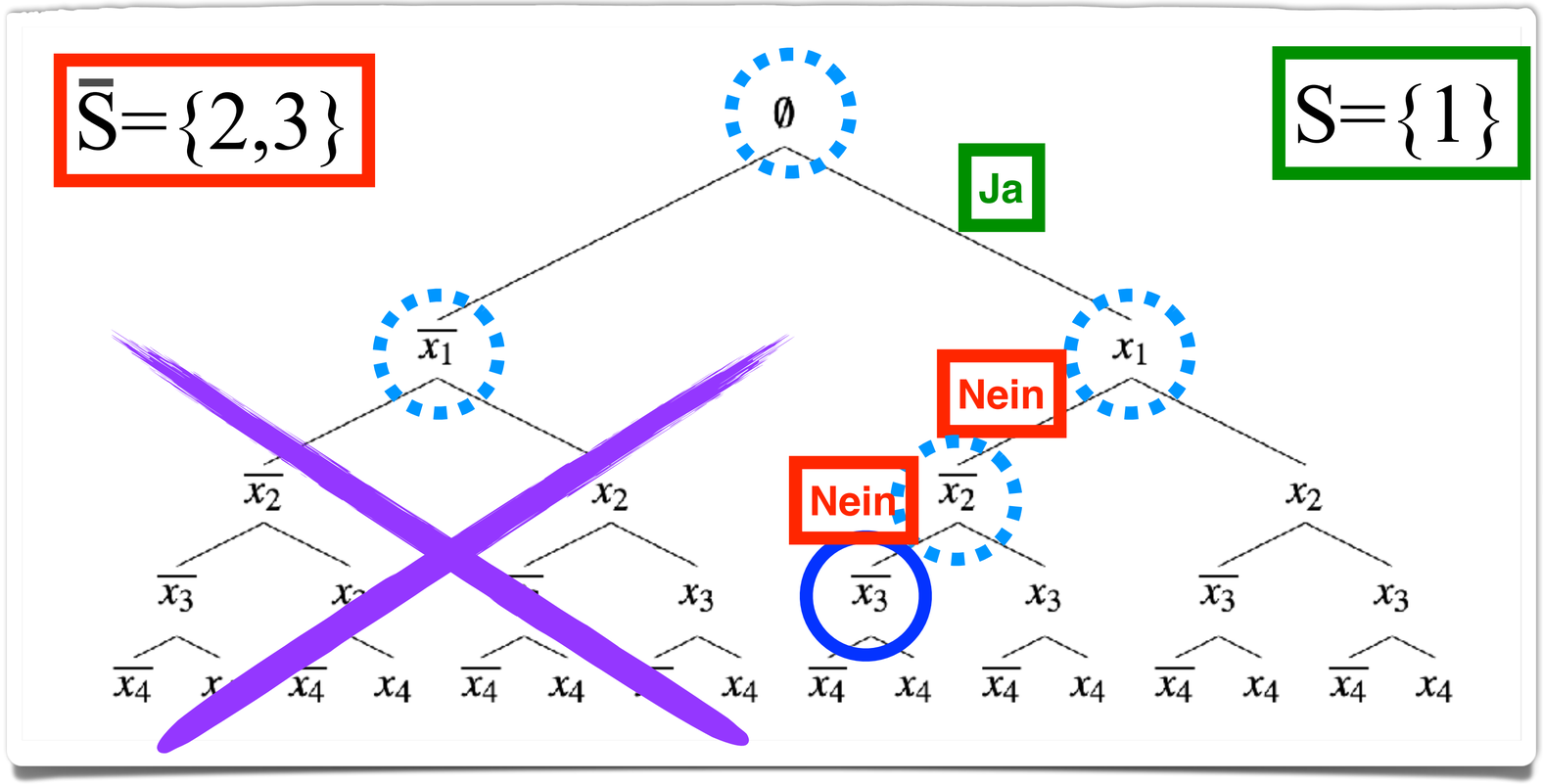
Gesamtbeispiel



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

LB=14

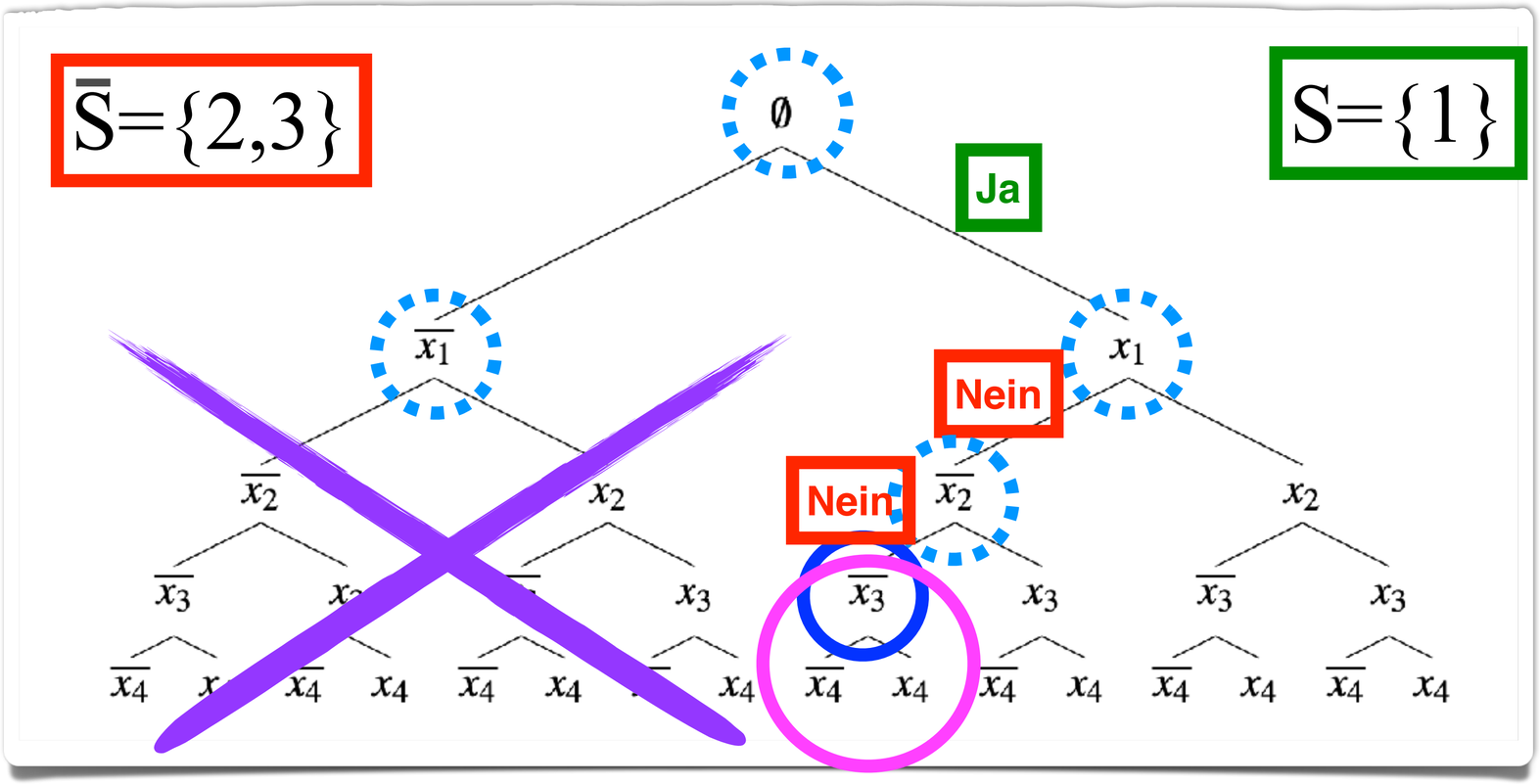
Gesamtbeispiel



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

LB=14

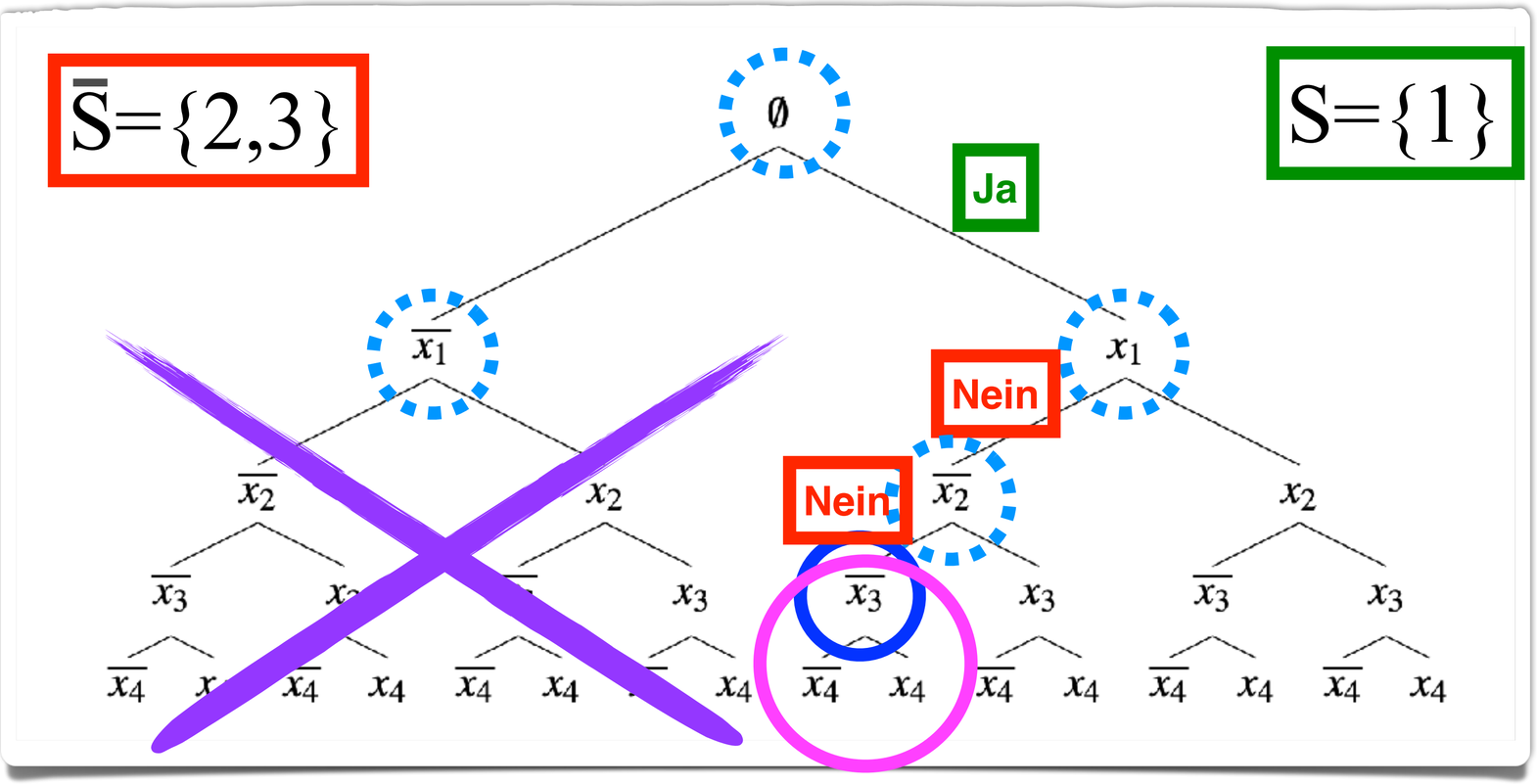
Gesamtbeispiel



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

LB=14

Gesamtbeispiel

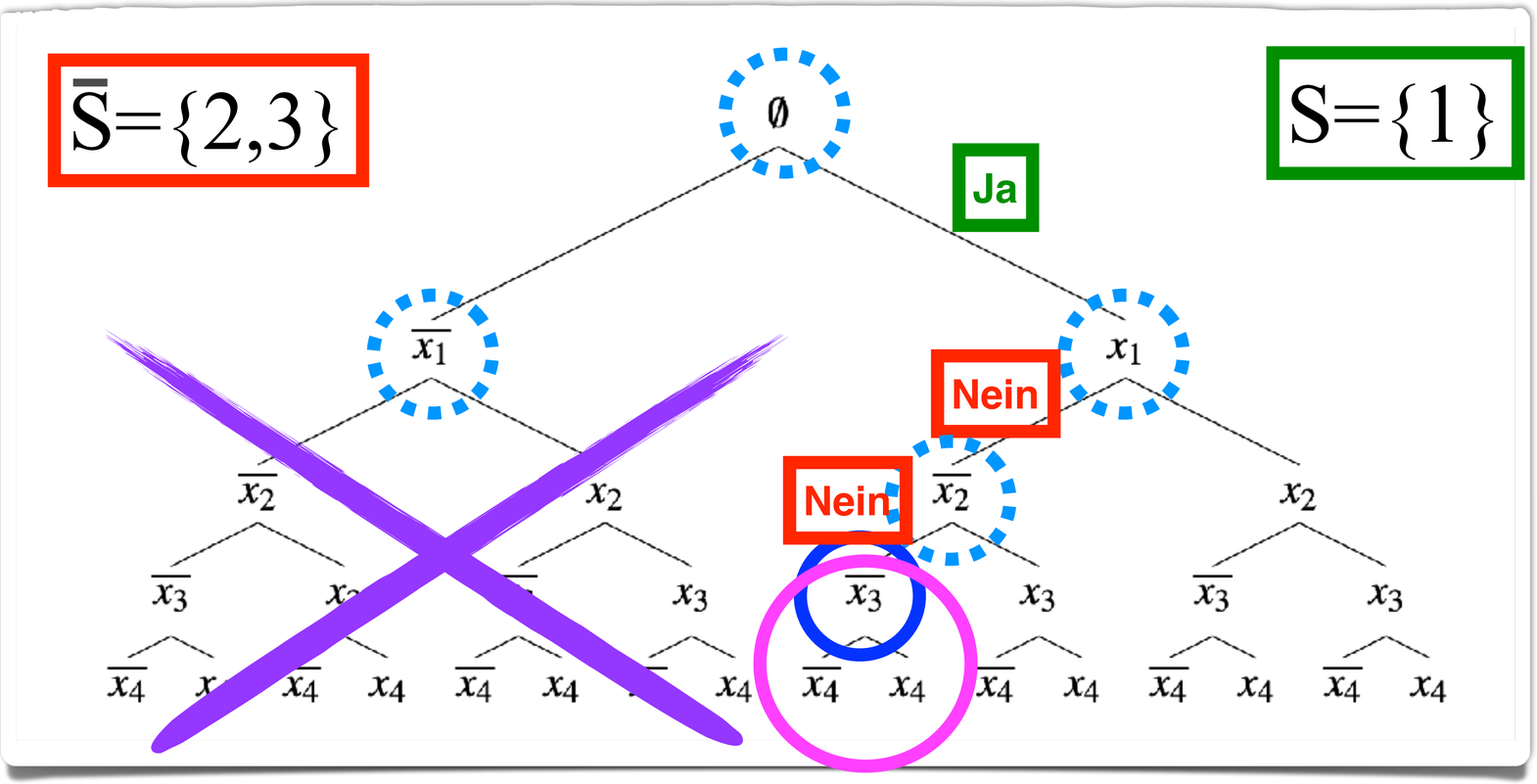


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=14

Gesamtbeispiel

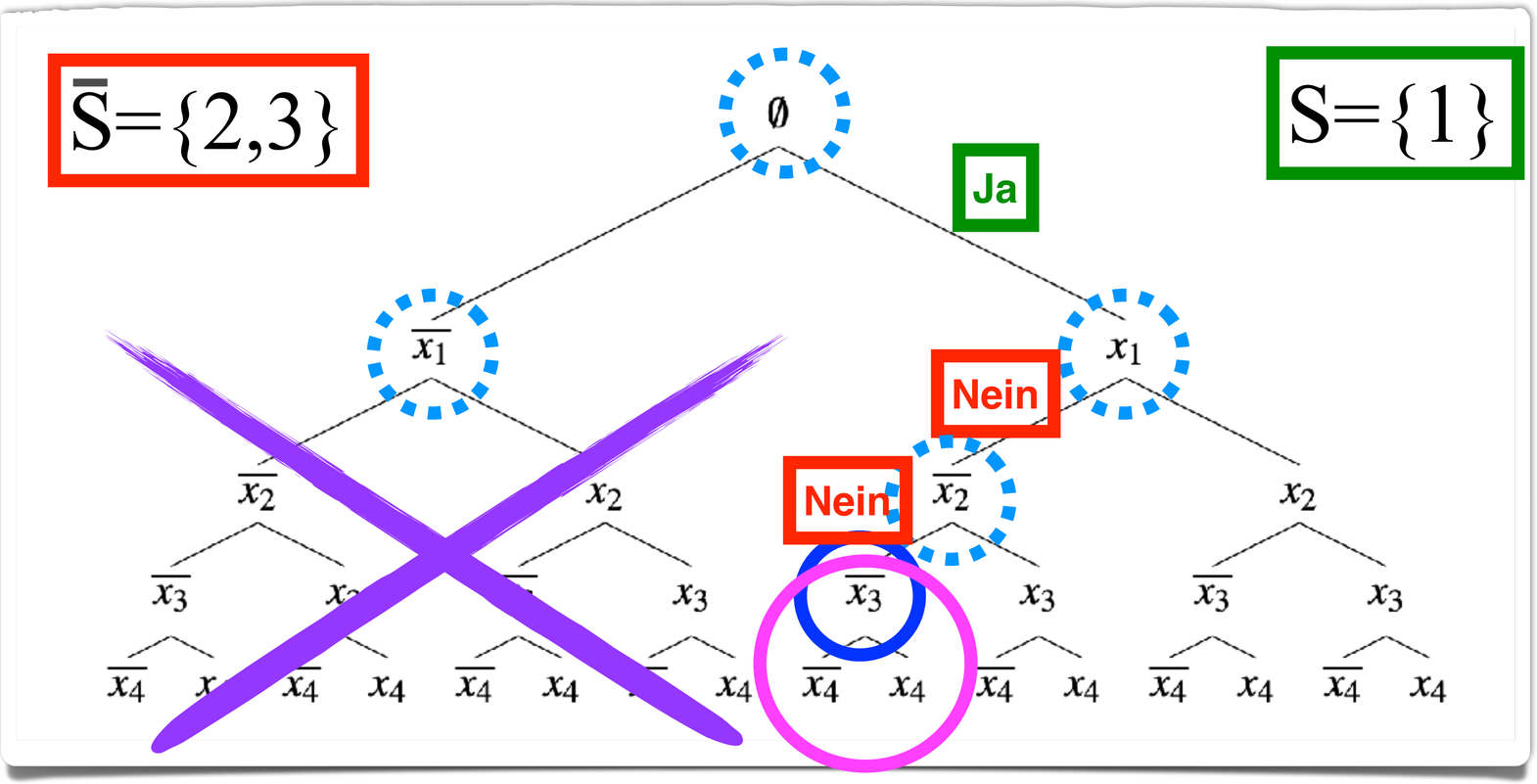


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=14

Gesamtbeispiel

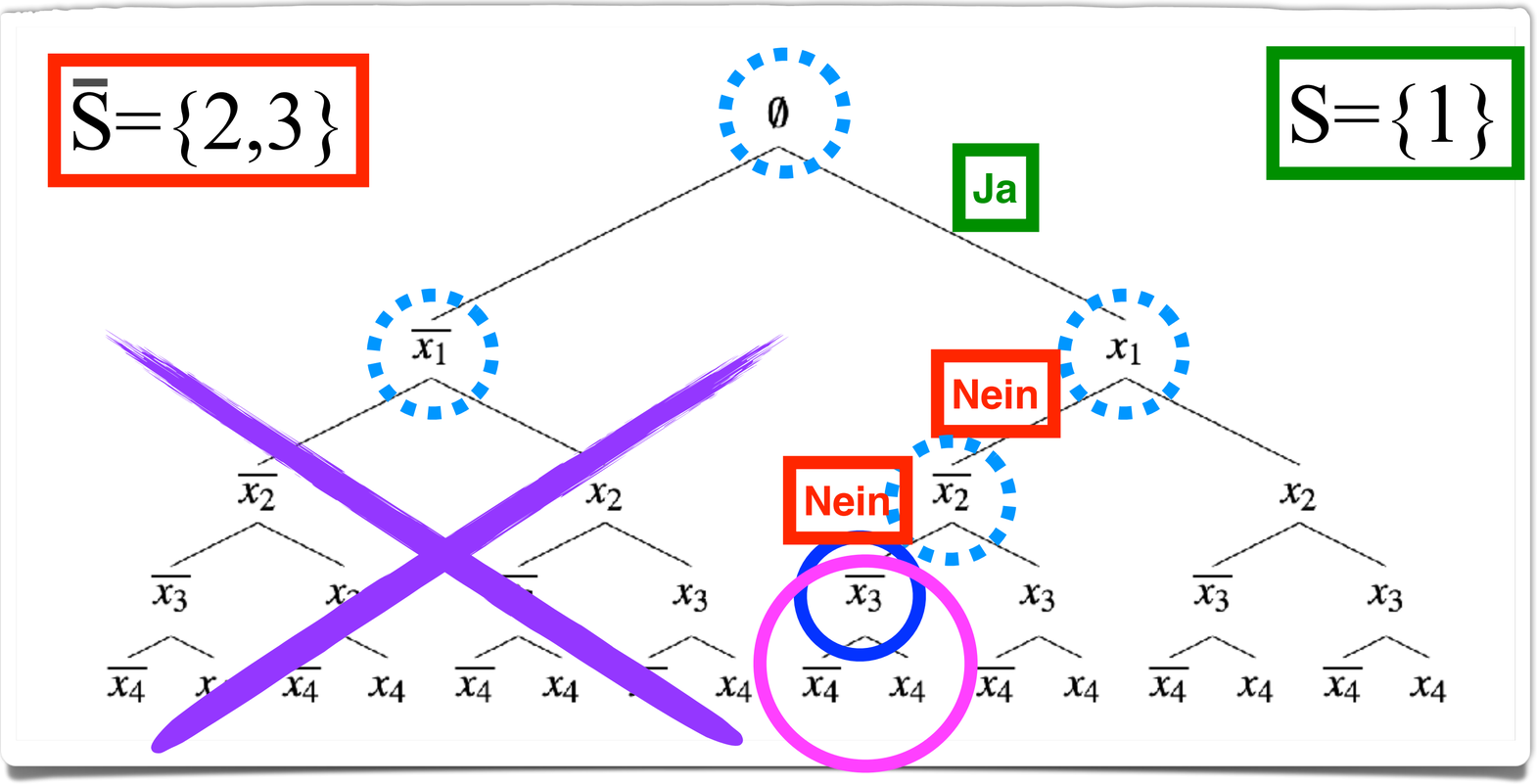


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=14

Gesamtbeispiel

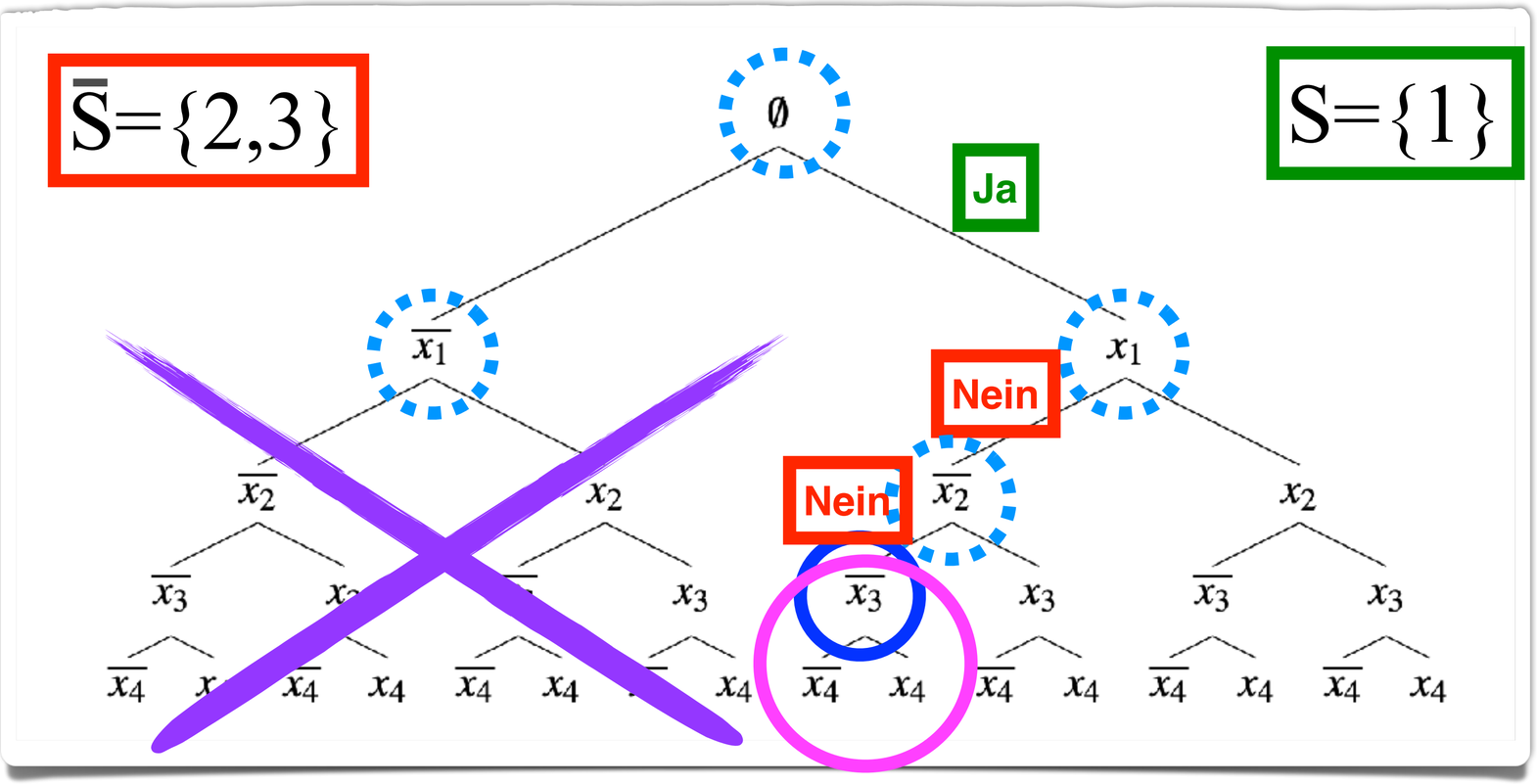


i	1	2	3	4	5	6	7
z_i	2			7	5	9	4
p_i	6			9	6	7	3

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

LB=14

Gesamtbeispiel

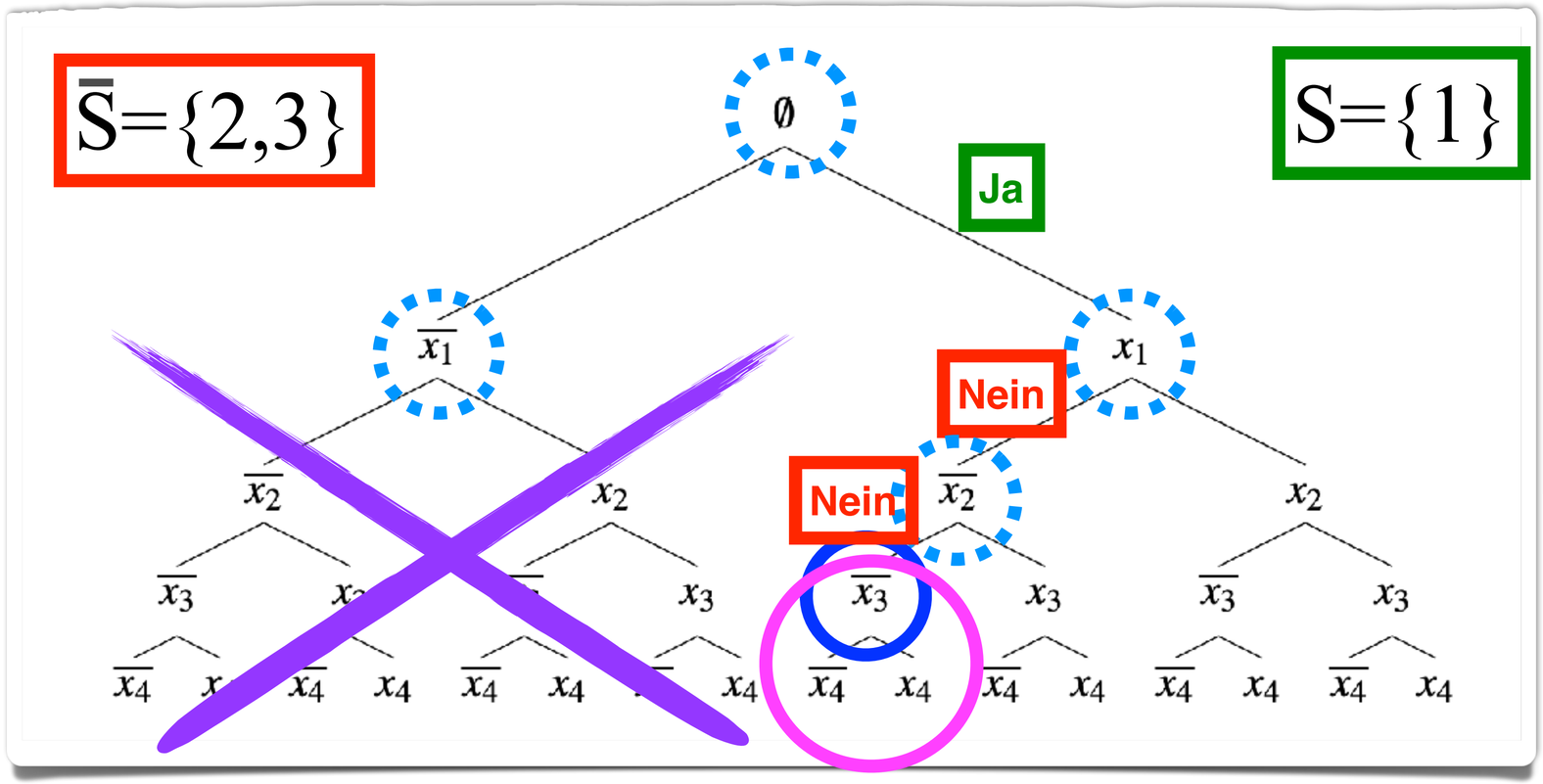


i	1	2	3	4	5	6	7
z_i	2			7	5	9	4
p_i	6			9	6	7	3

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

LB=14

Gesamtbeispiel



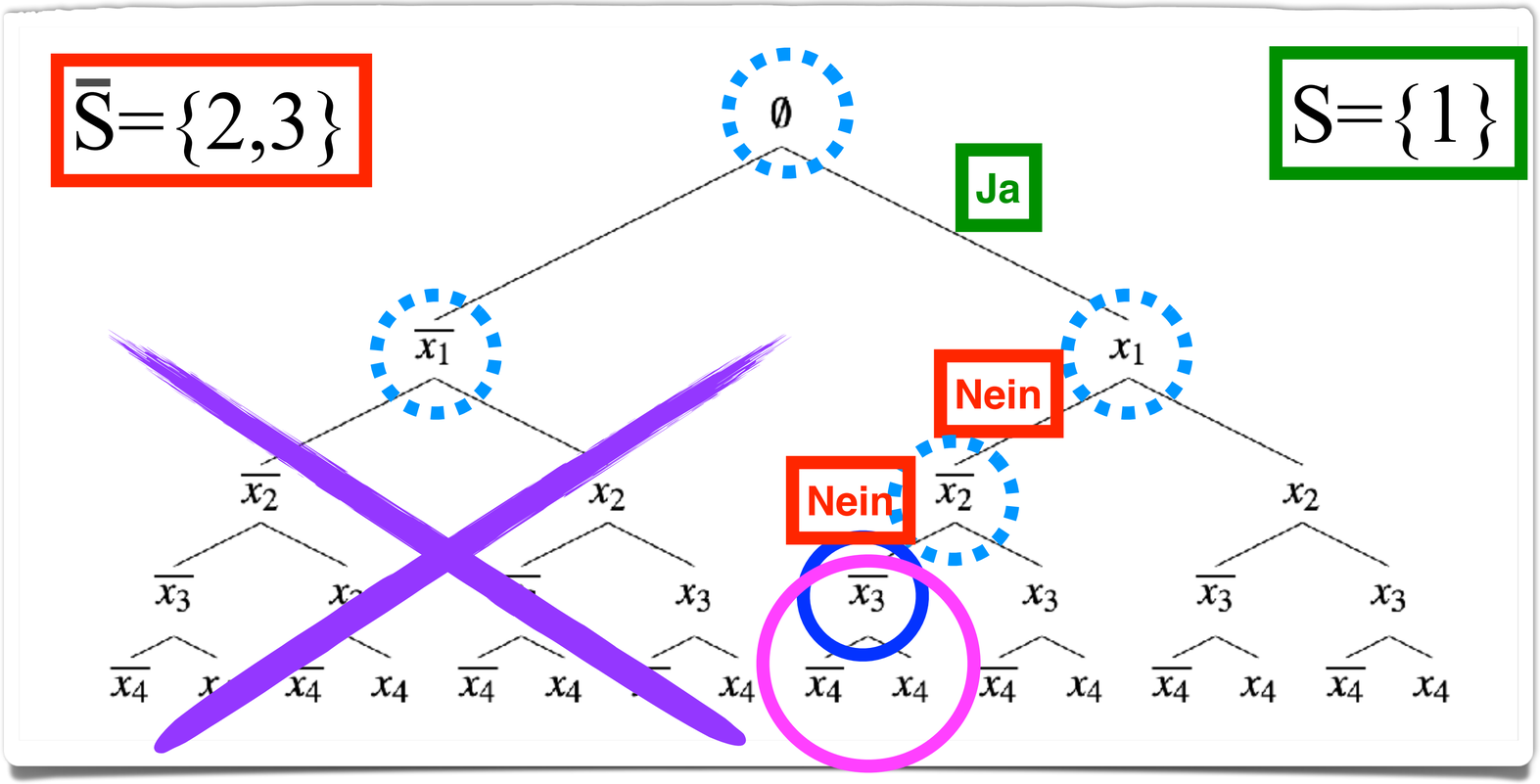
i	1	2	3	4	5	6	7
z_i	2			7	5	9	4
p_i	6			9	6	7	3

UB=15

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

LB=14

Gesamtbeispiel



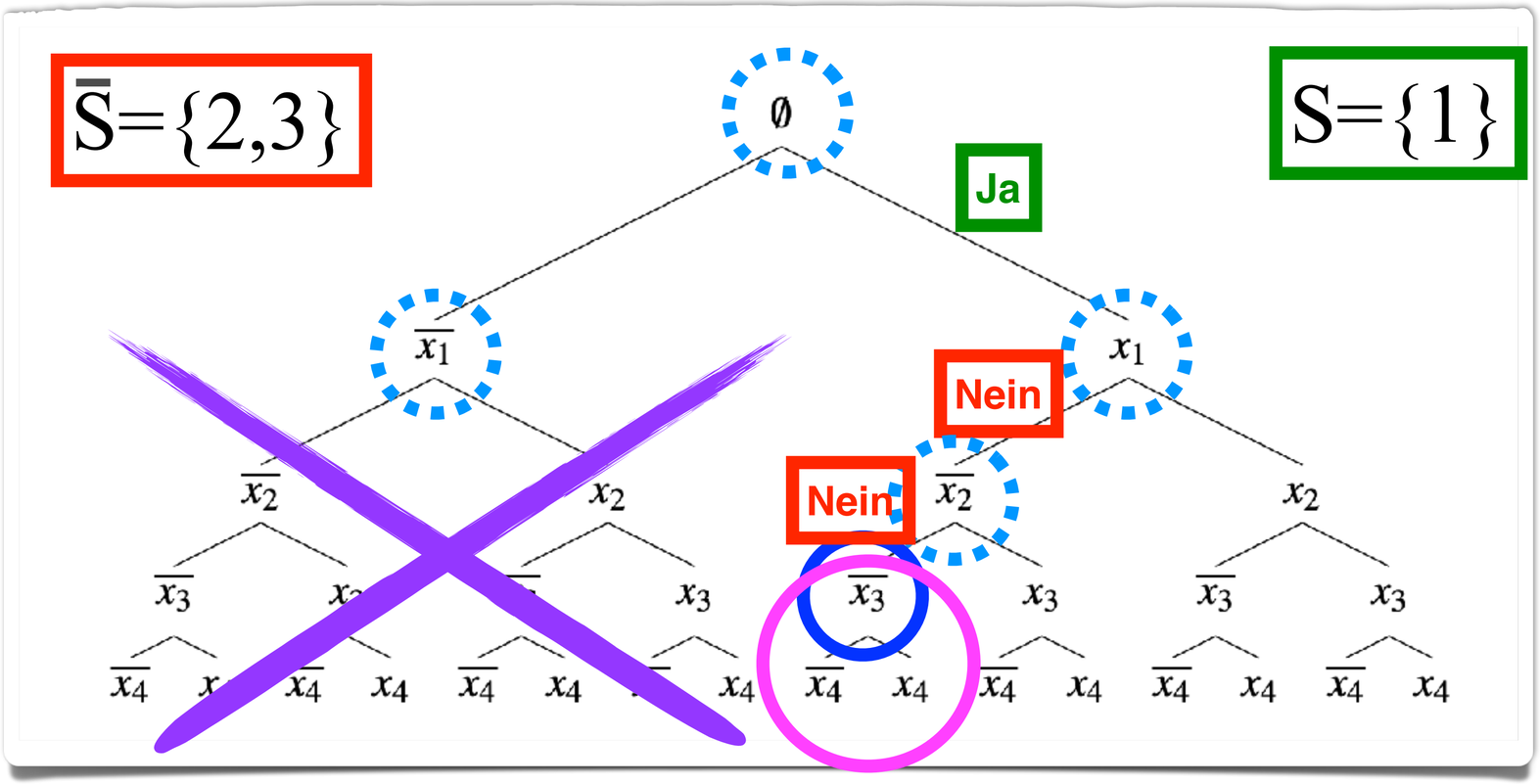
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	6	8	9	6	7	3

UB=15

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

LB=14

Gesamtbeispiel



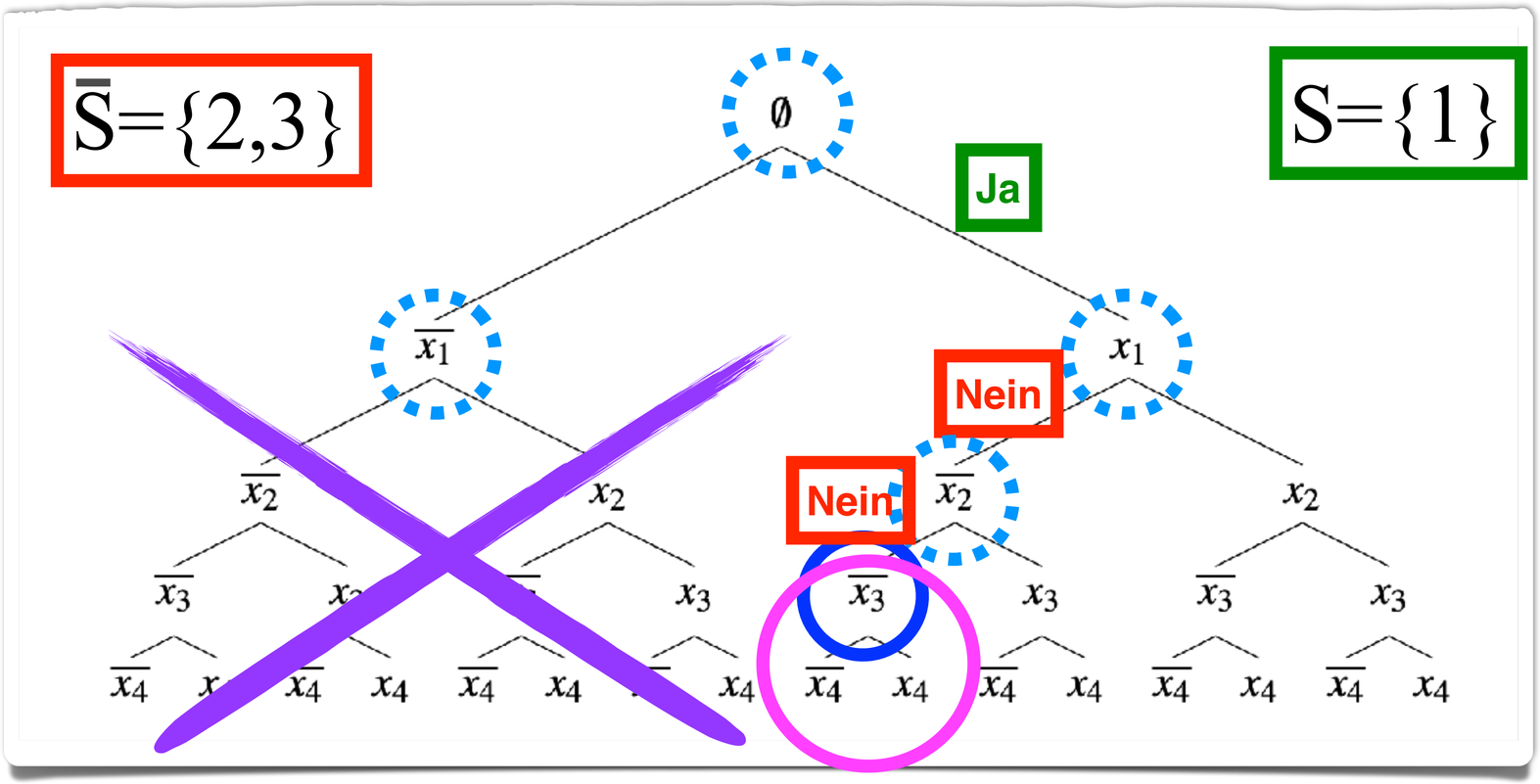
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	6	8	9	6	7	3

UB=15

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

LB=14

Gesamtbeispiel



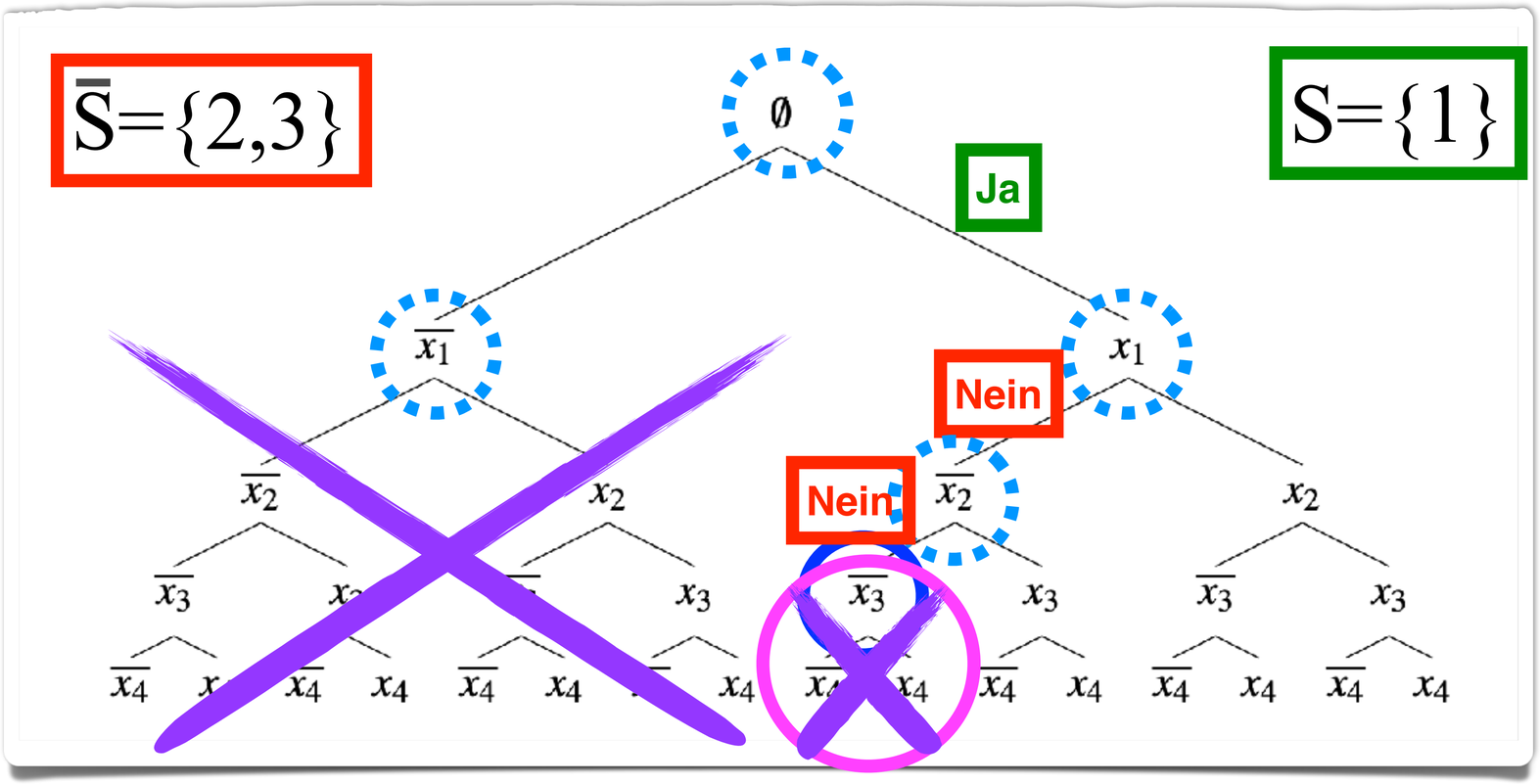
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	6	8	9	6	7	3

UB=15

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

LB=15

Gesamtbeispiel



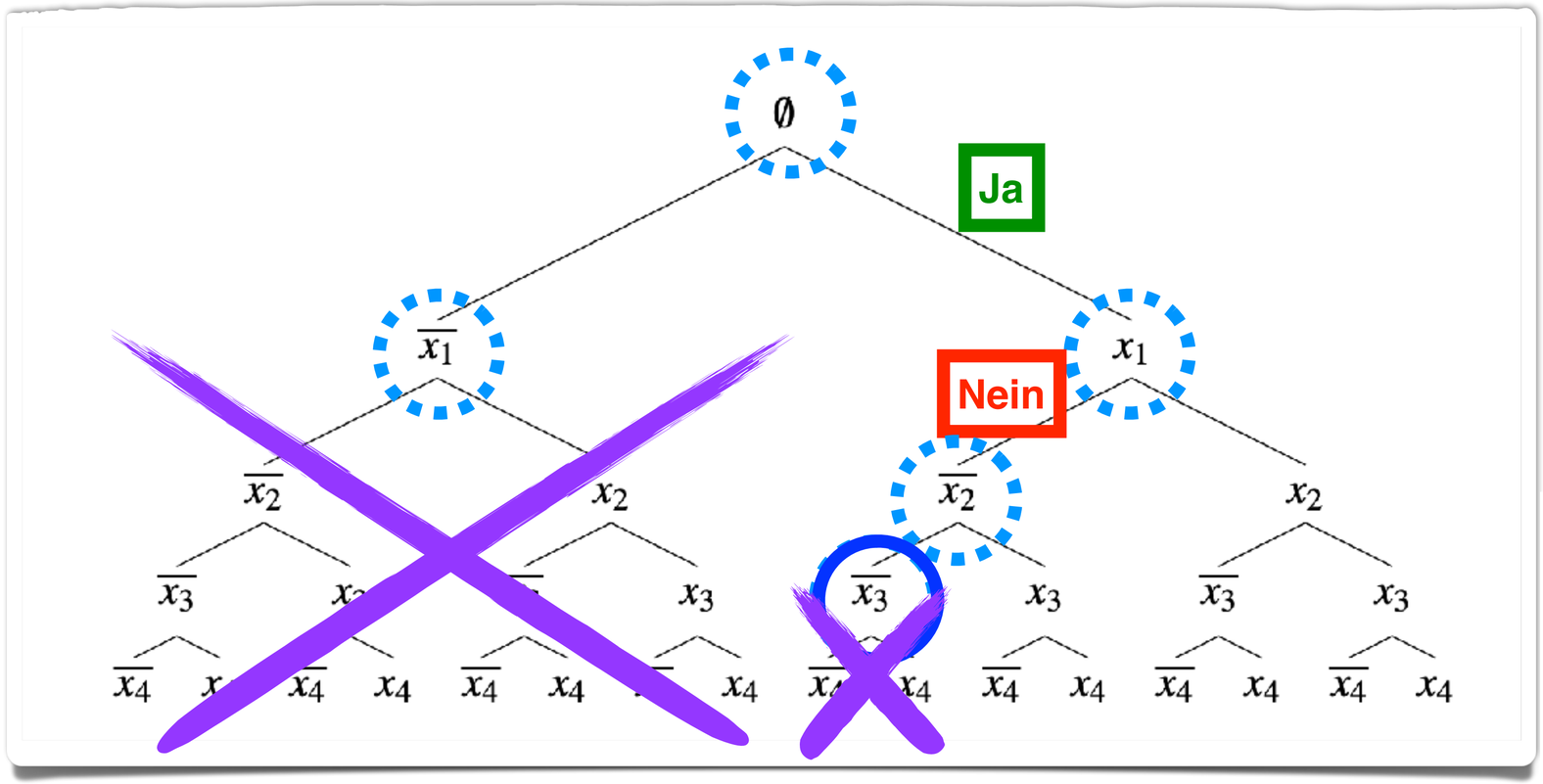
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	6	8	9	6	7	3

UB=15

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

LB=15

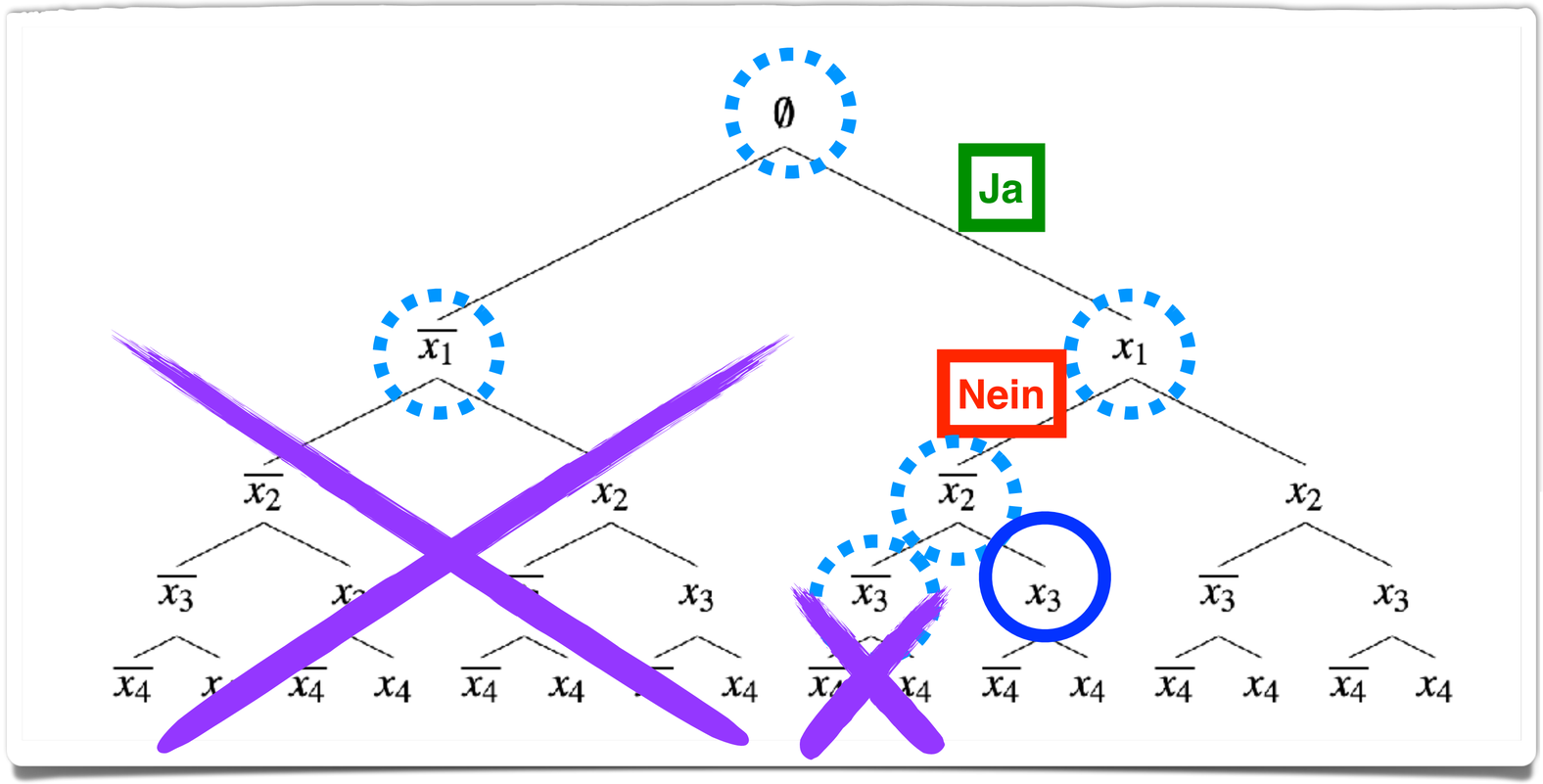
Gesamtbeispiel



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

LB=15

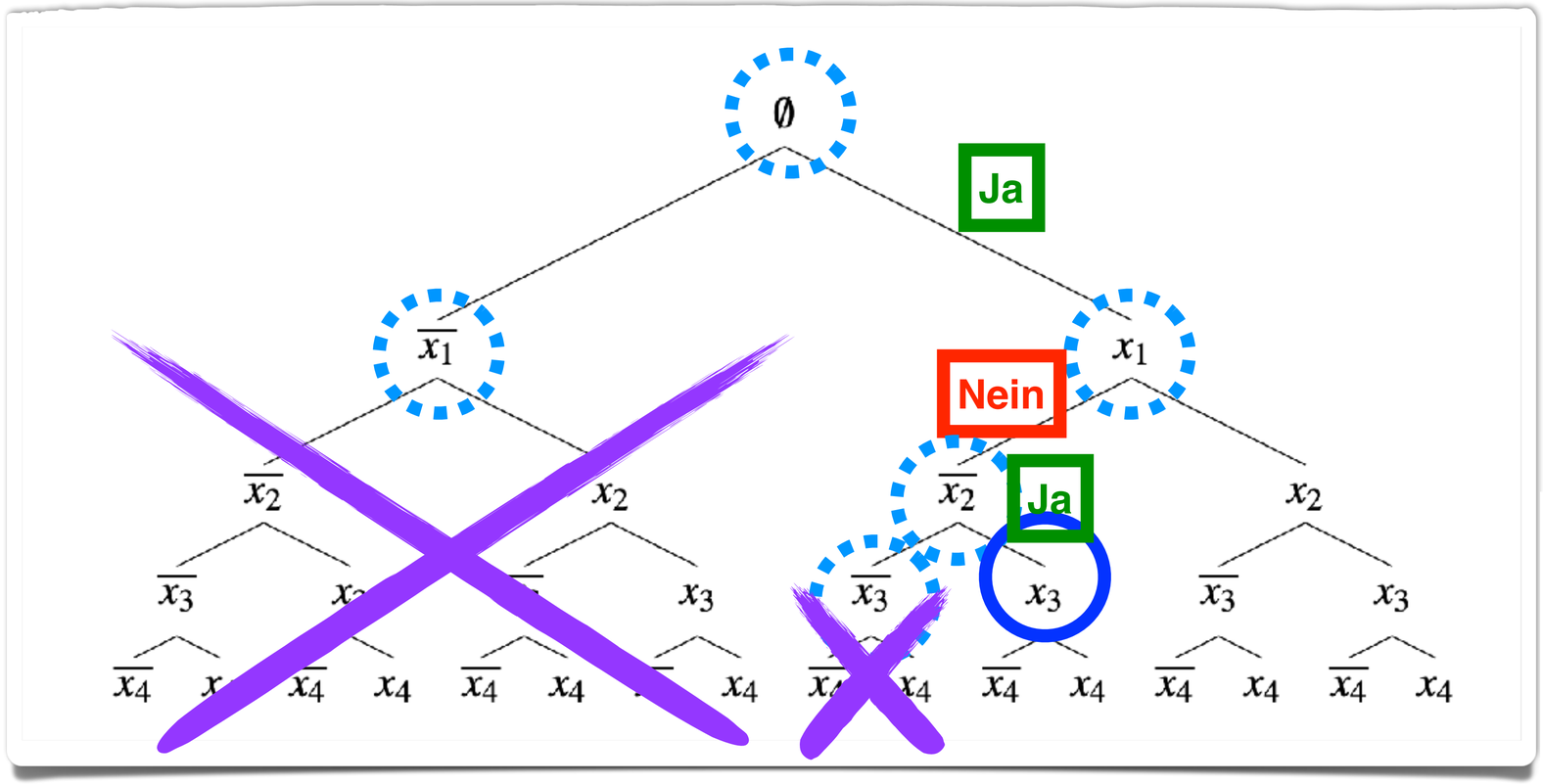
Gesamtbeispiel



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

LB=15

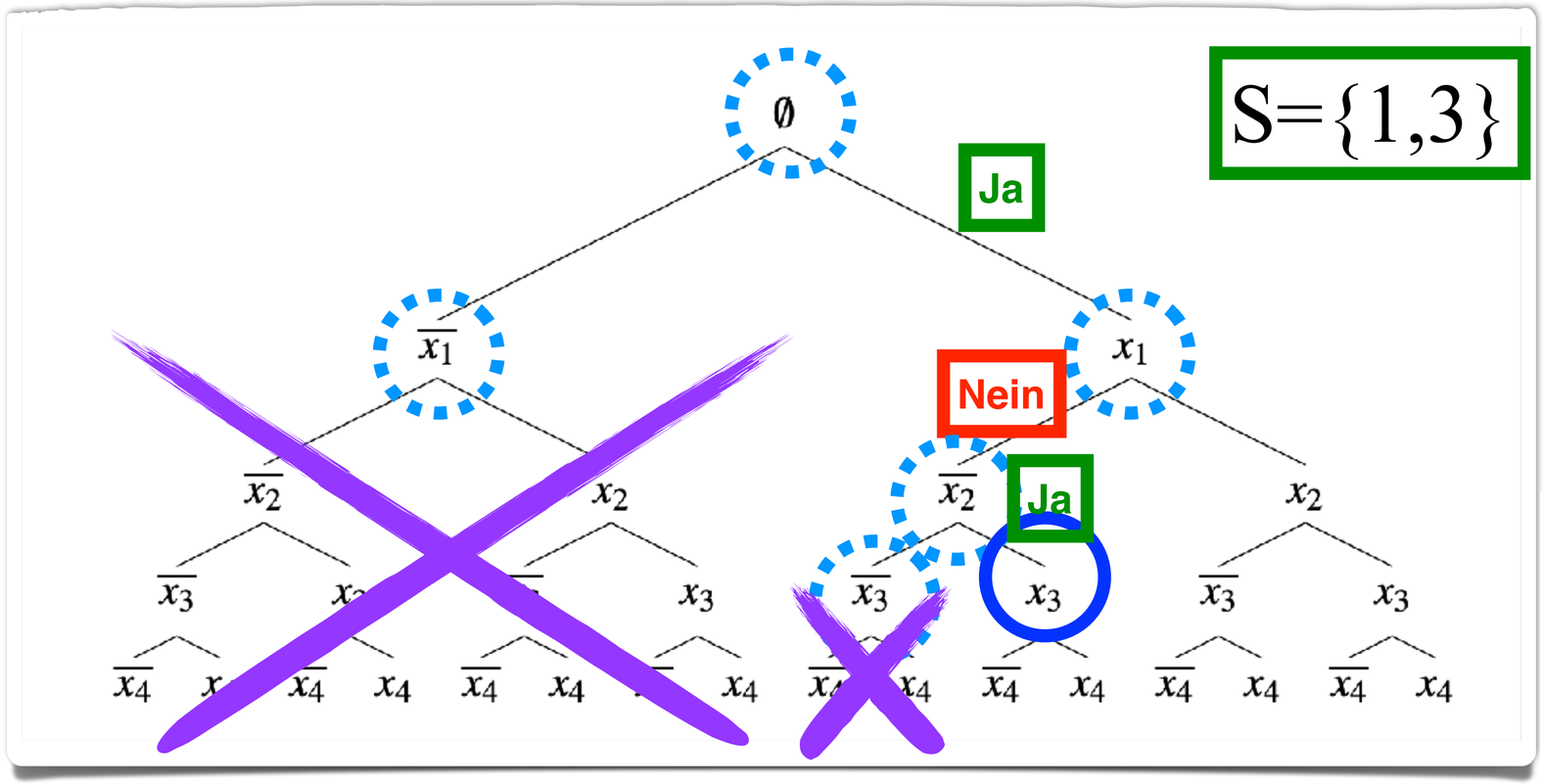
Gesamtbeispiel



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

LB=15

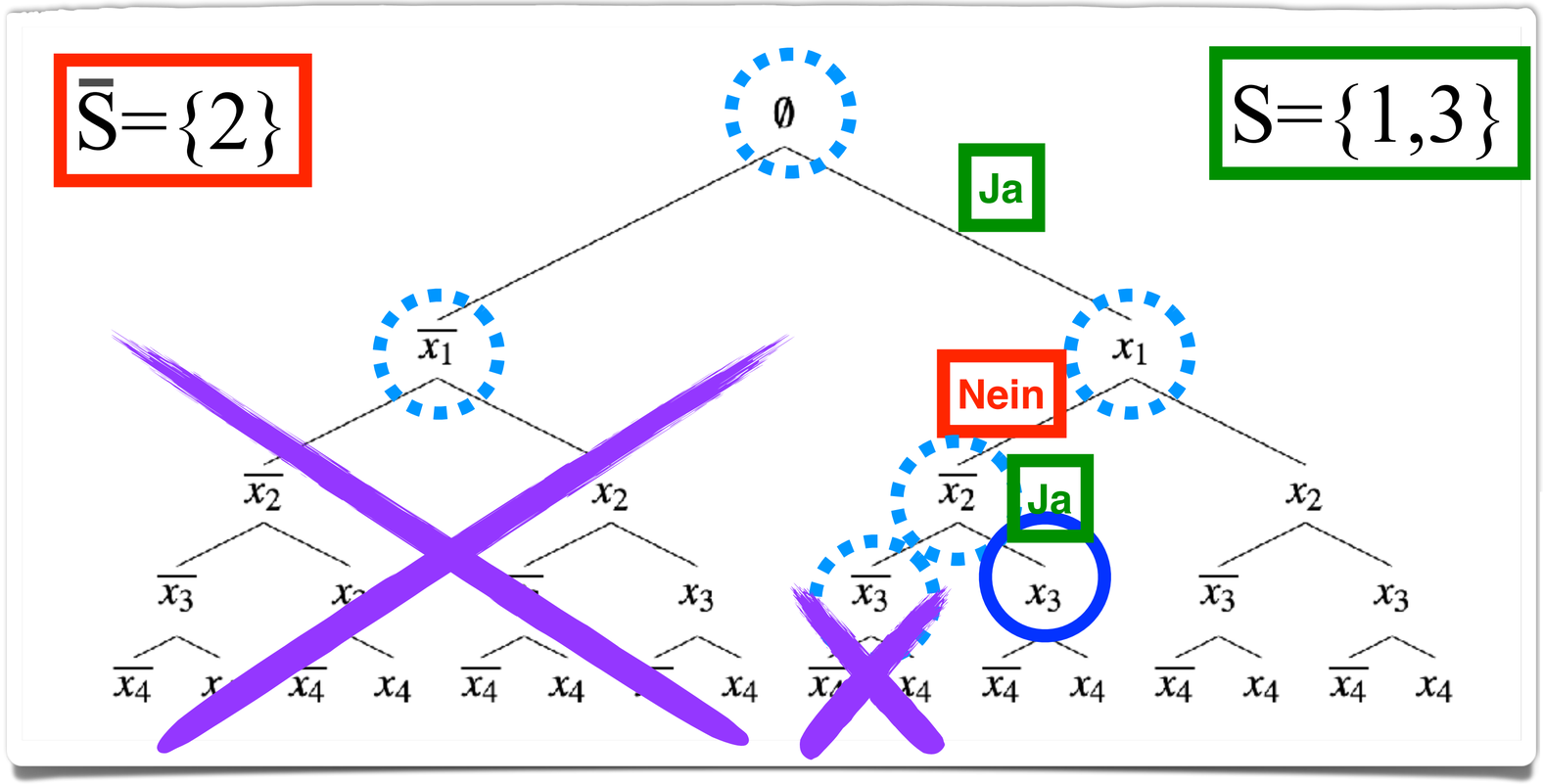
Gesamtbeispiel



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

LB=15

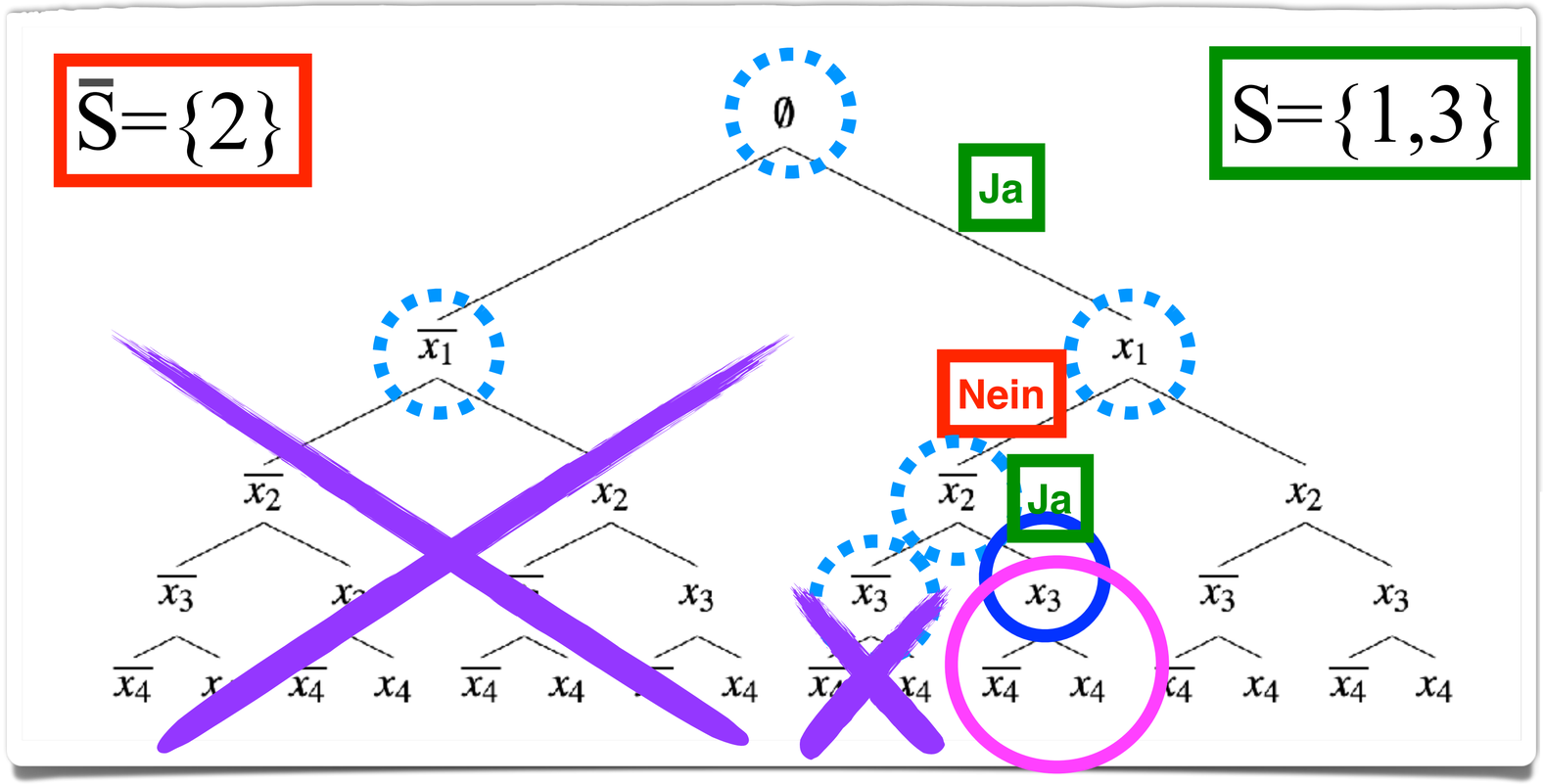
Gesamtbeispiel



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

LB=15

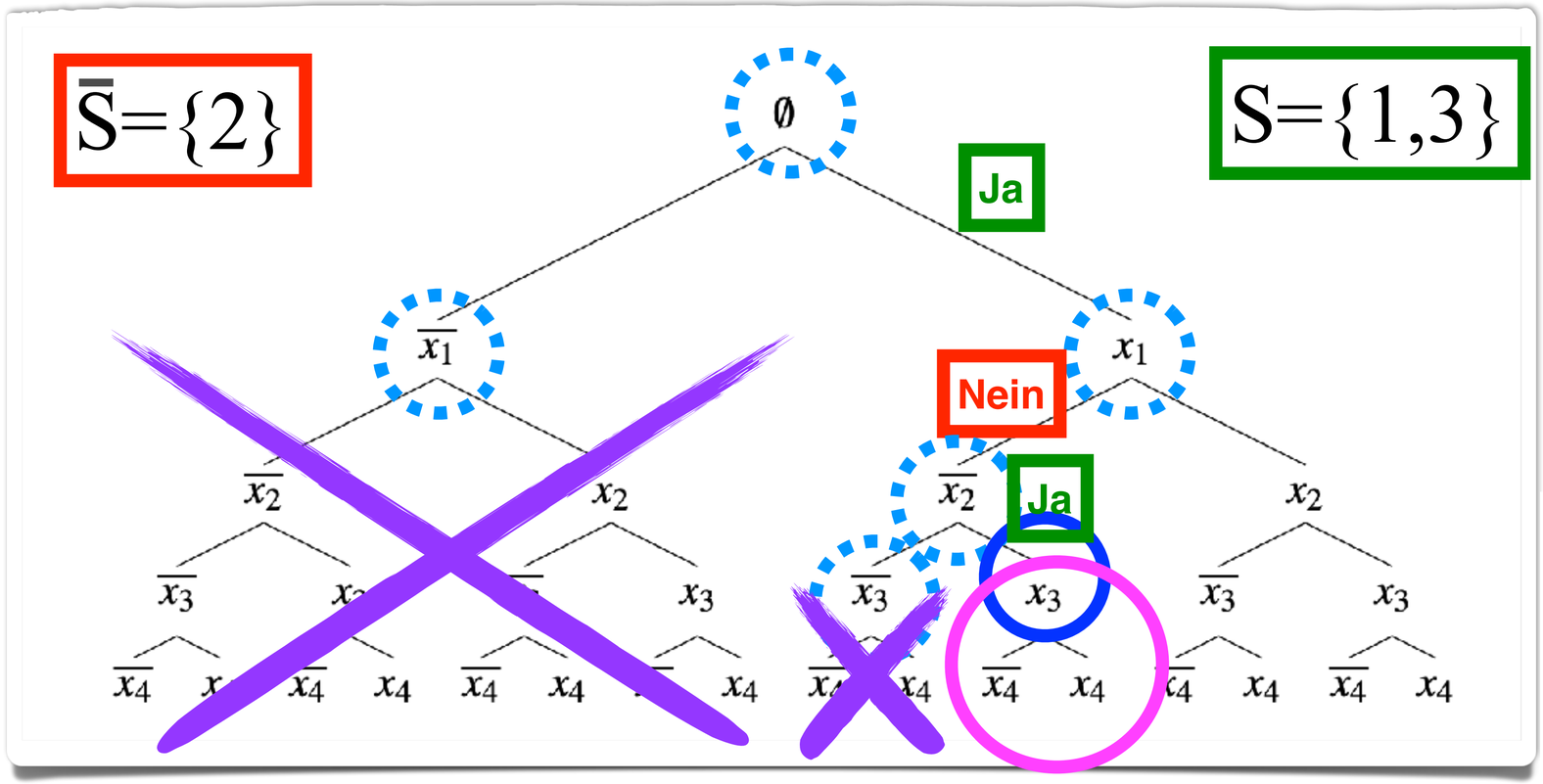
Gesamtbeispiel



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

LB=15

Gesamtbeispiel

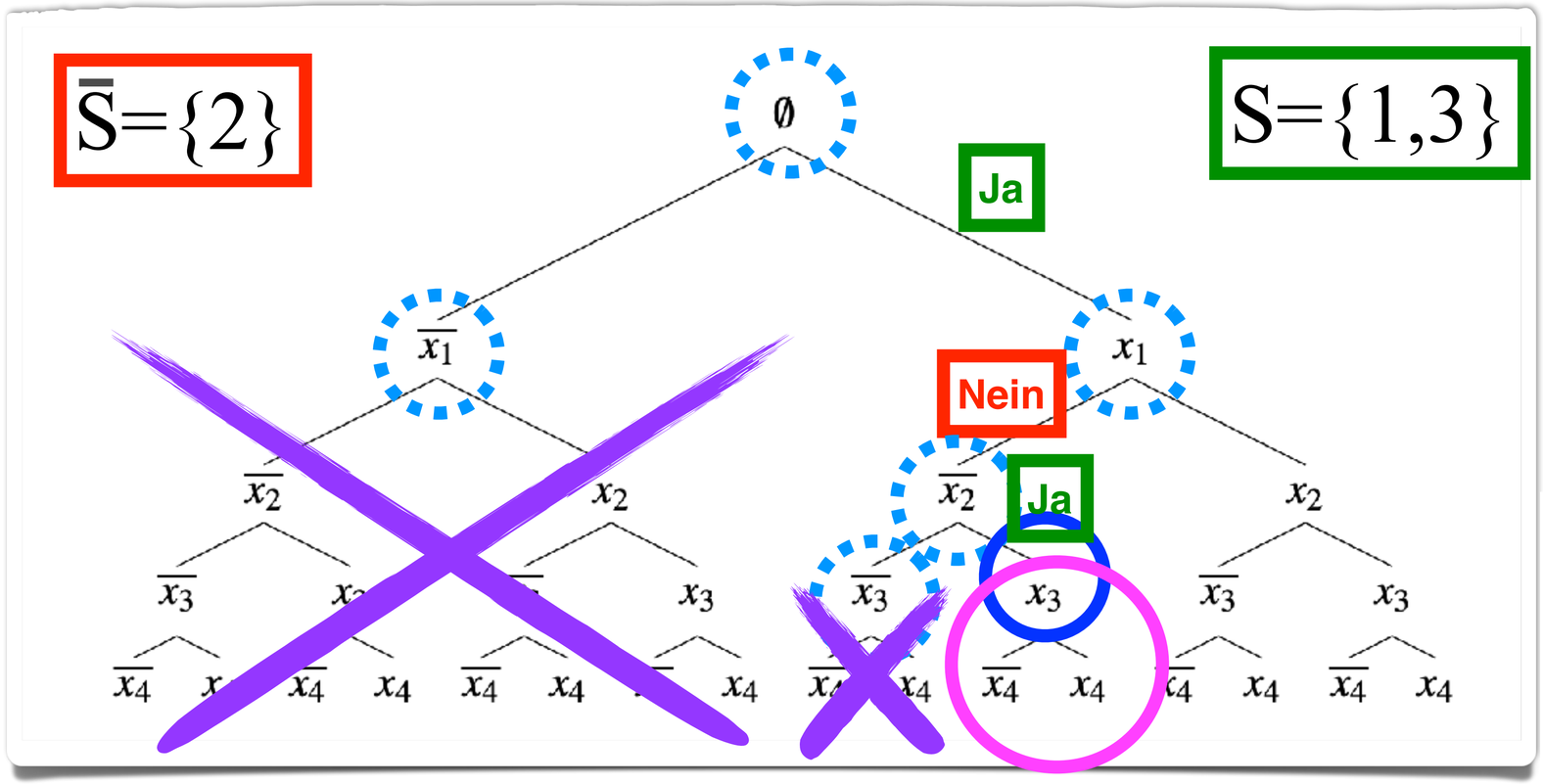


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=15

Gesamtbeispiel

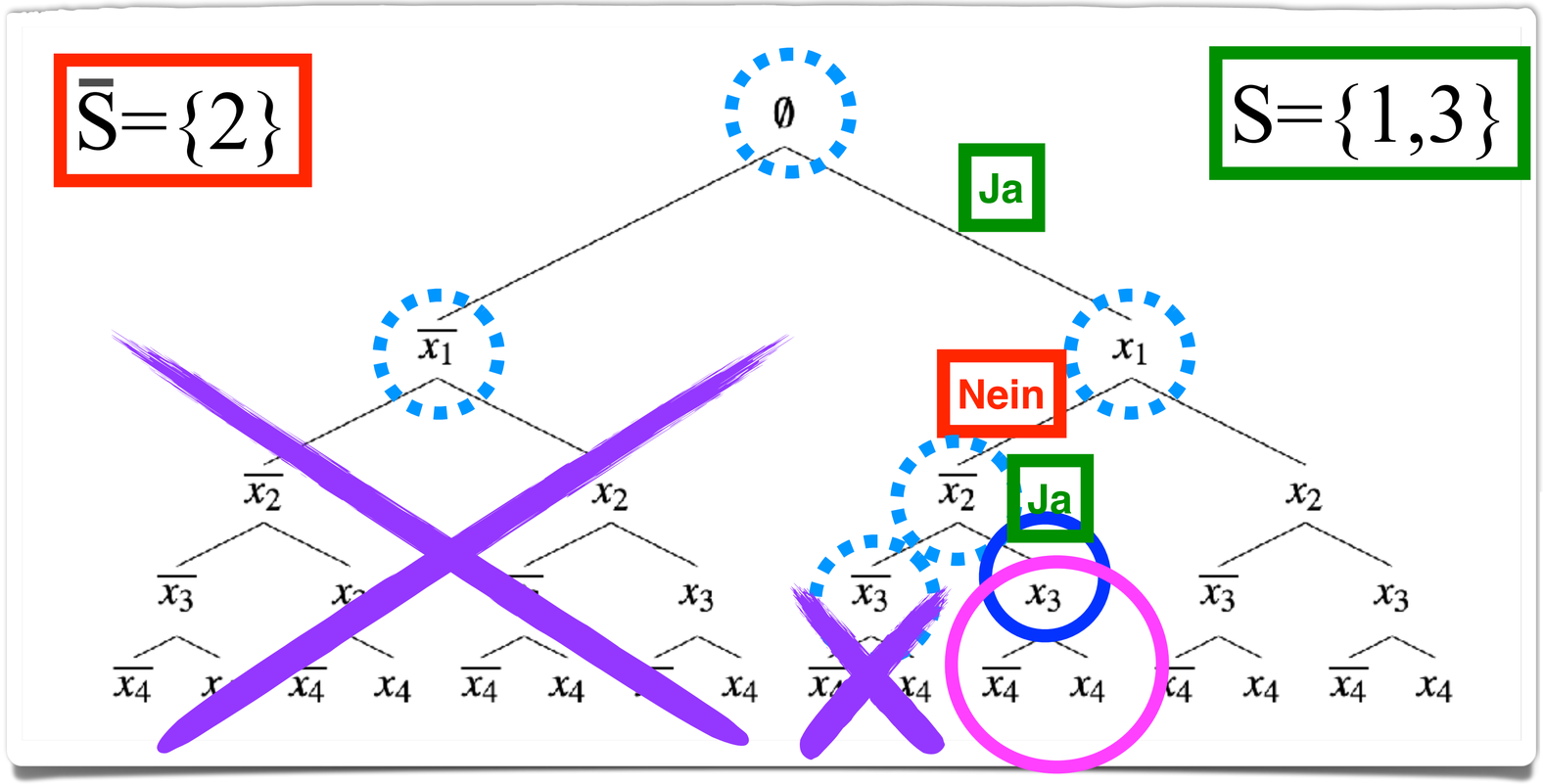


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=15

Gesamtbeispiel

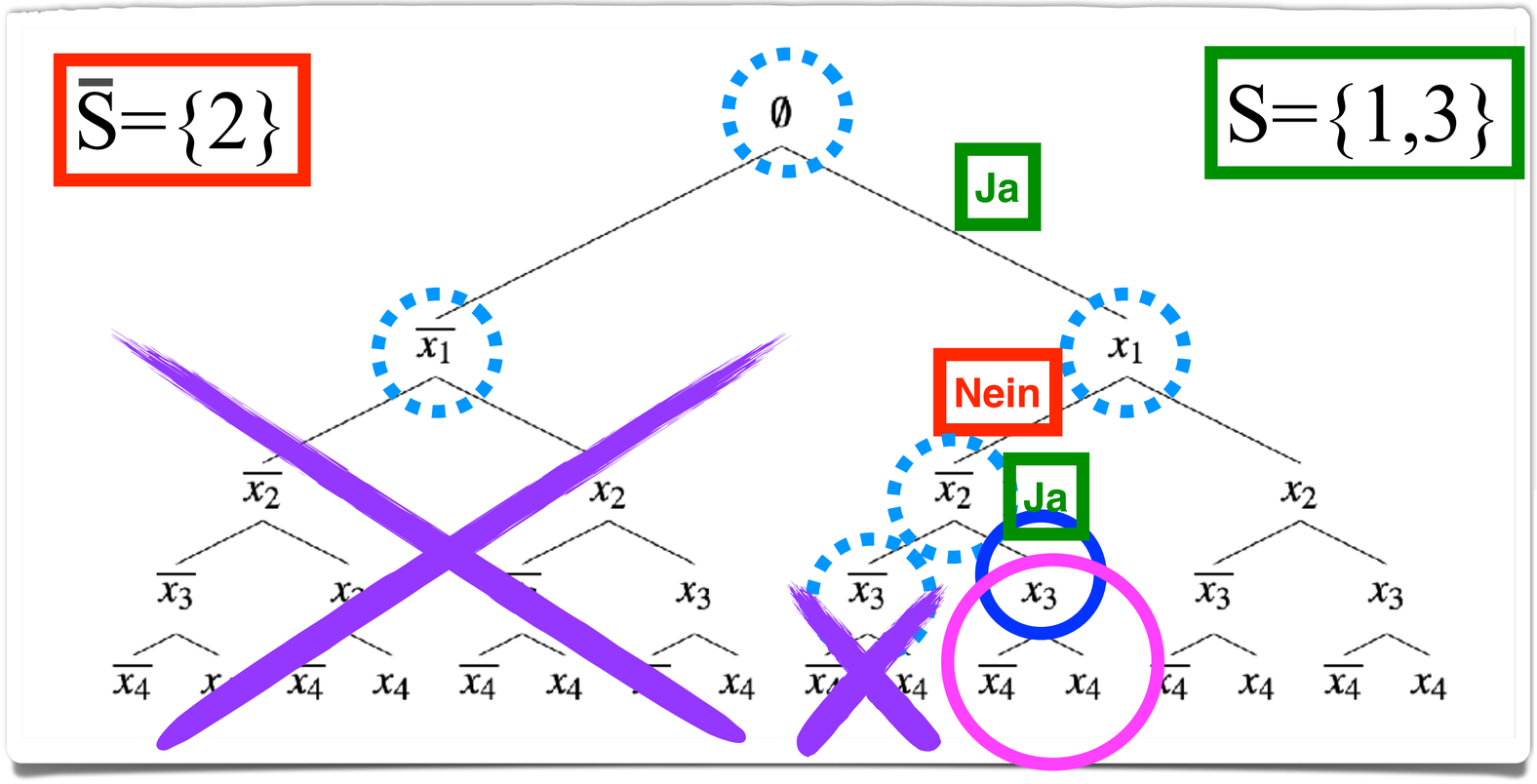


i	1	2	3	4	5	6	7
z_i	2		6	7	5	9	4
p_i	6		8	9	6	7	3

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

LB=15

Gesamtbeispiel

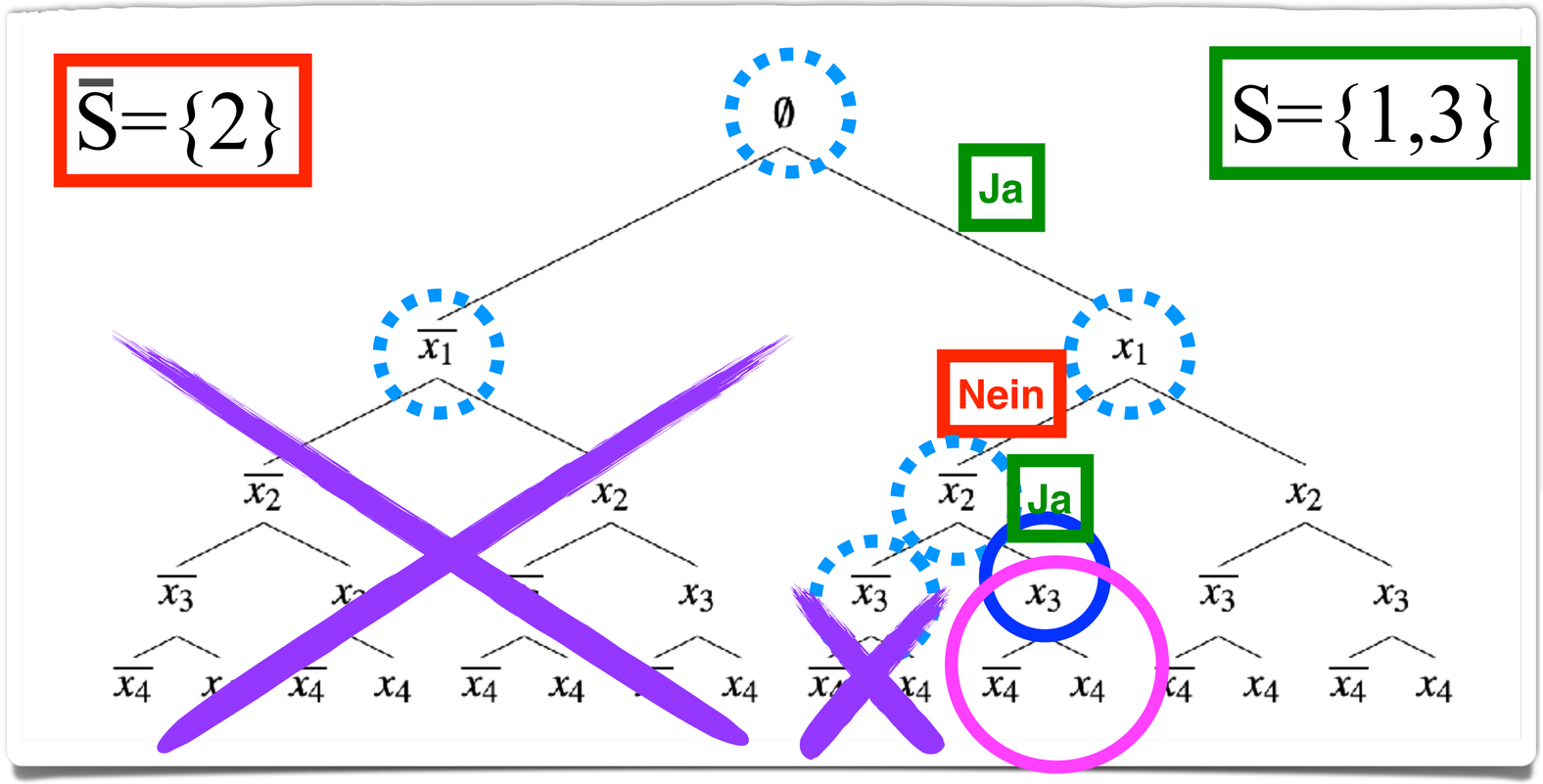


i	1	2	3	4	5	6	7
z_i	2		6	7	5	9	4
p_i	6		8	9	6	7	3

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

LB=15

Gesamtbeispiel

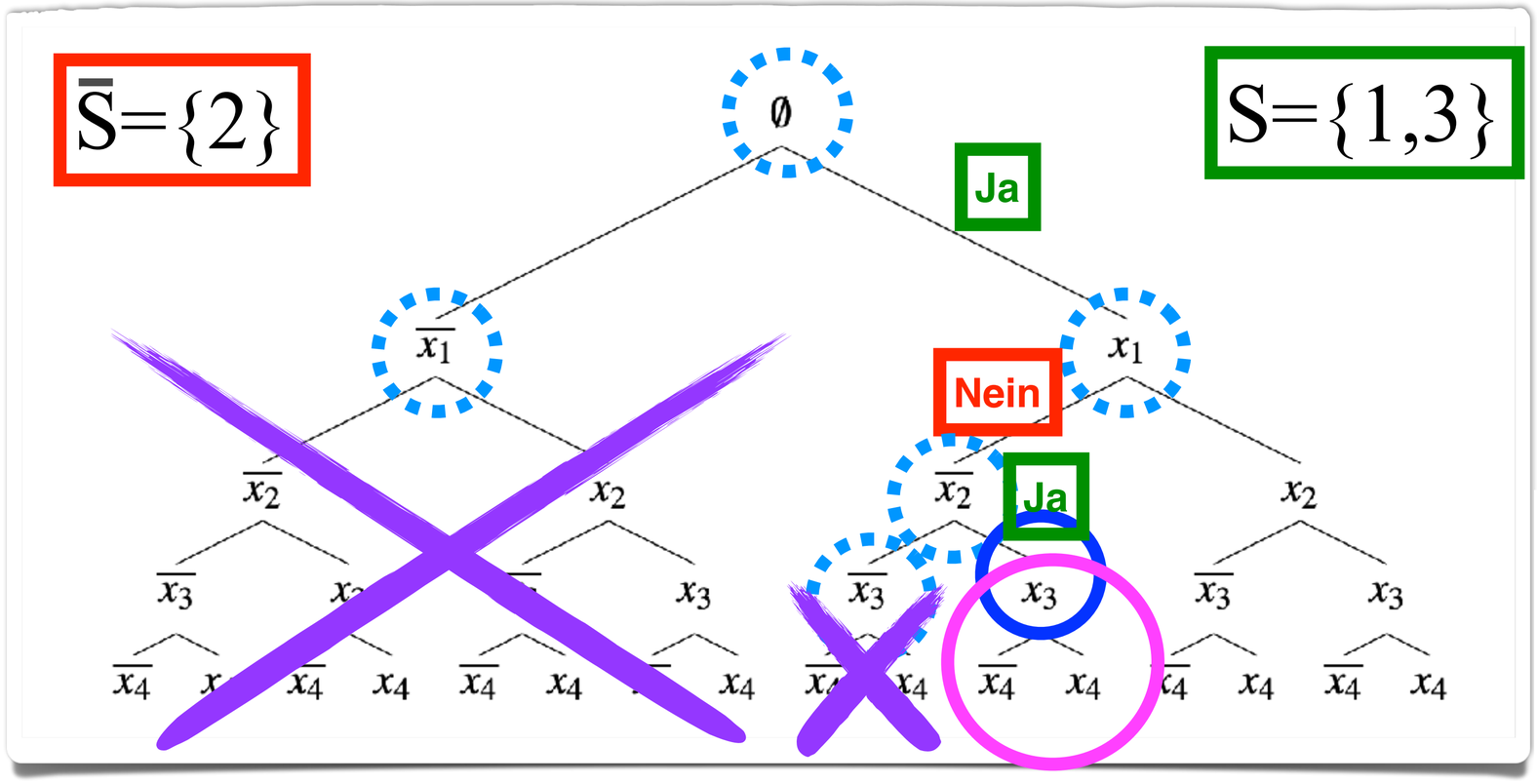


i	1	2	3	4	5	6	7
z_i	2		6	7	5	9	4
p_i	6		8	9	6	7	3

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

LB=15

Gesamtbeispiel



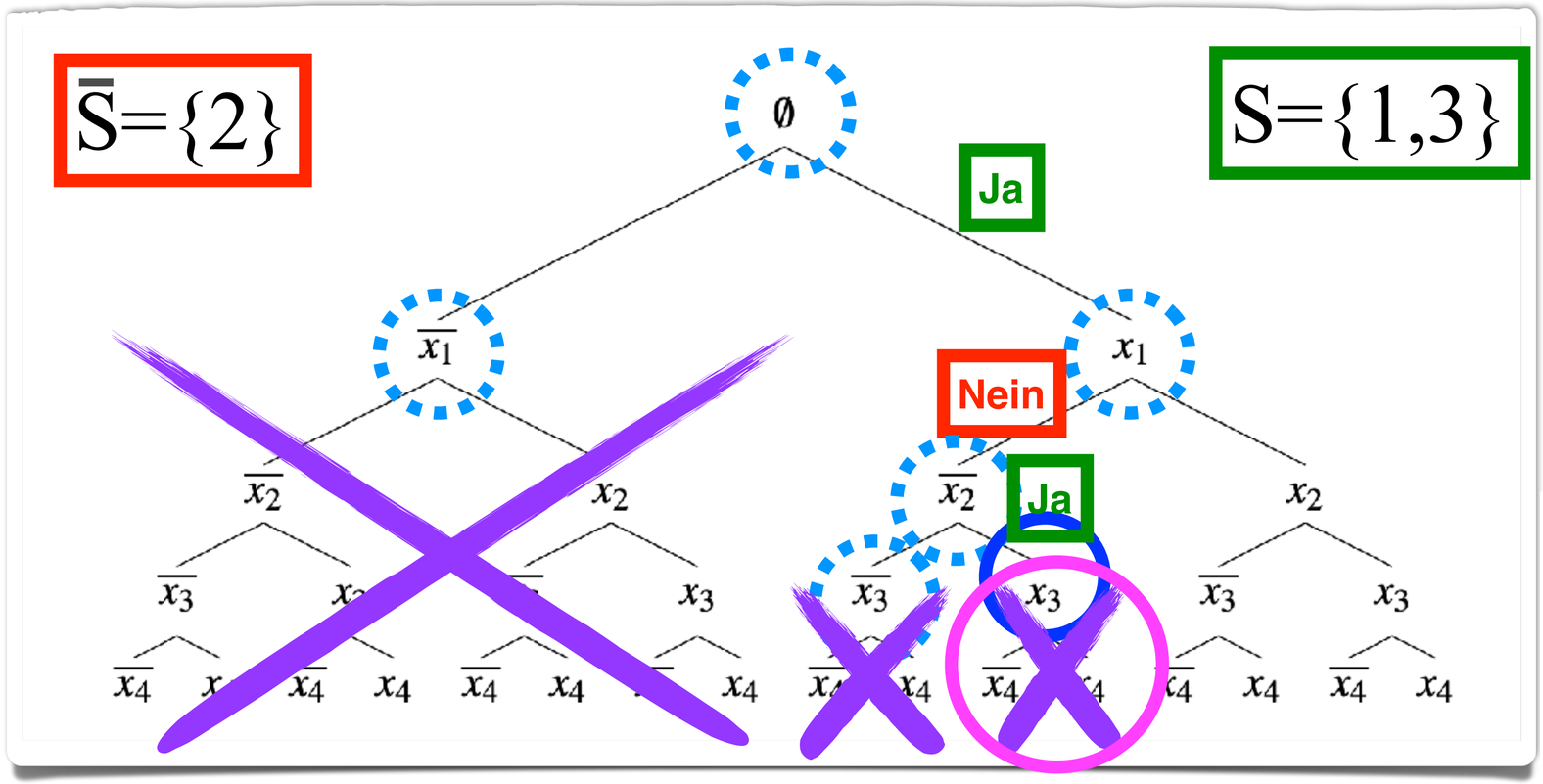
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	6	8	9	6	7	3

UB=15

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

LB=15

Gesamtbeispiel



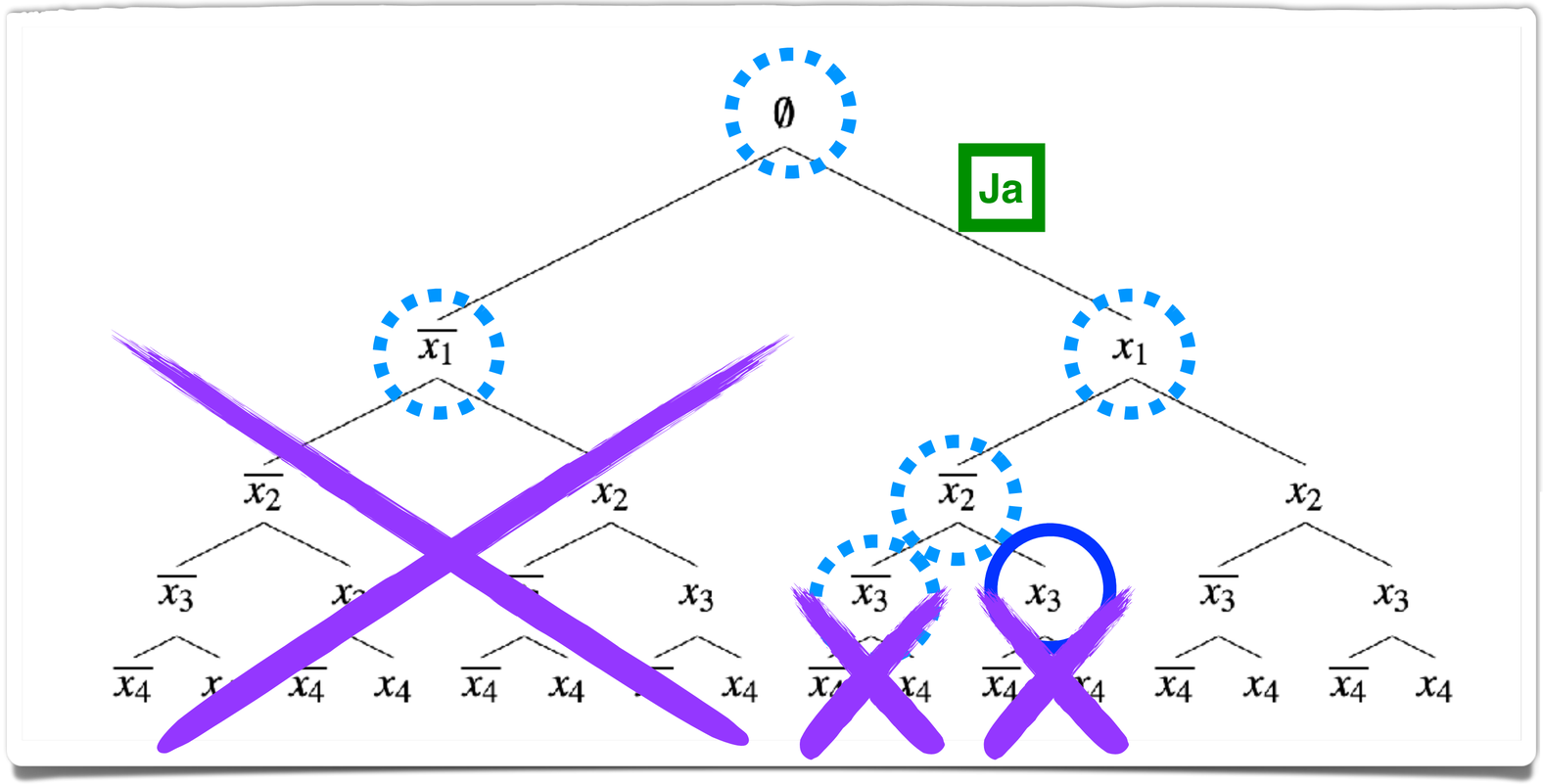
i	1	2	3	4	5	6	7
z_i	2		6	7	5	9	4
p_i	6		8	9	6	7	3

UB=15

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

LB=15

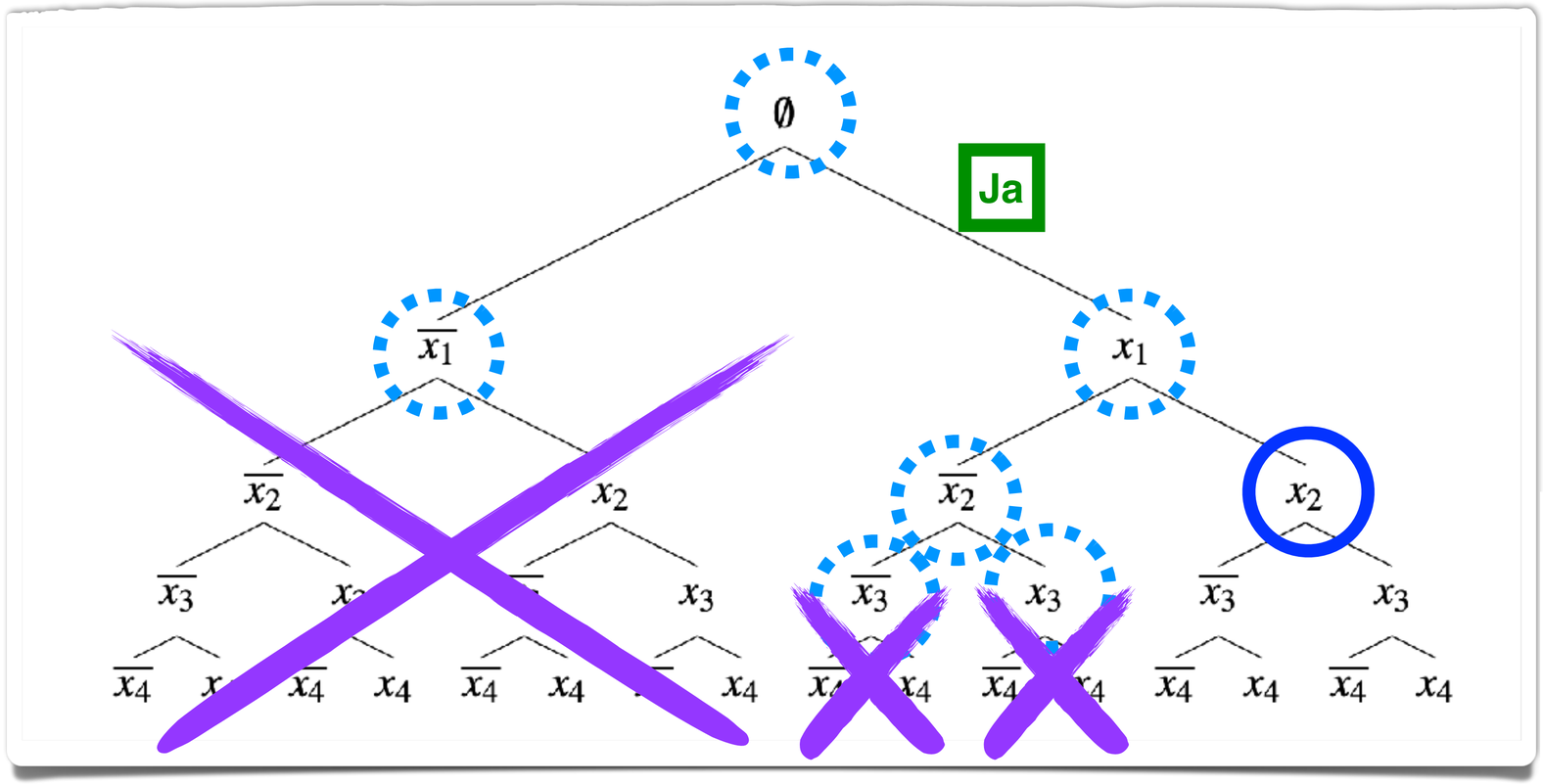
Gesamtbeispiel



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

LB=15

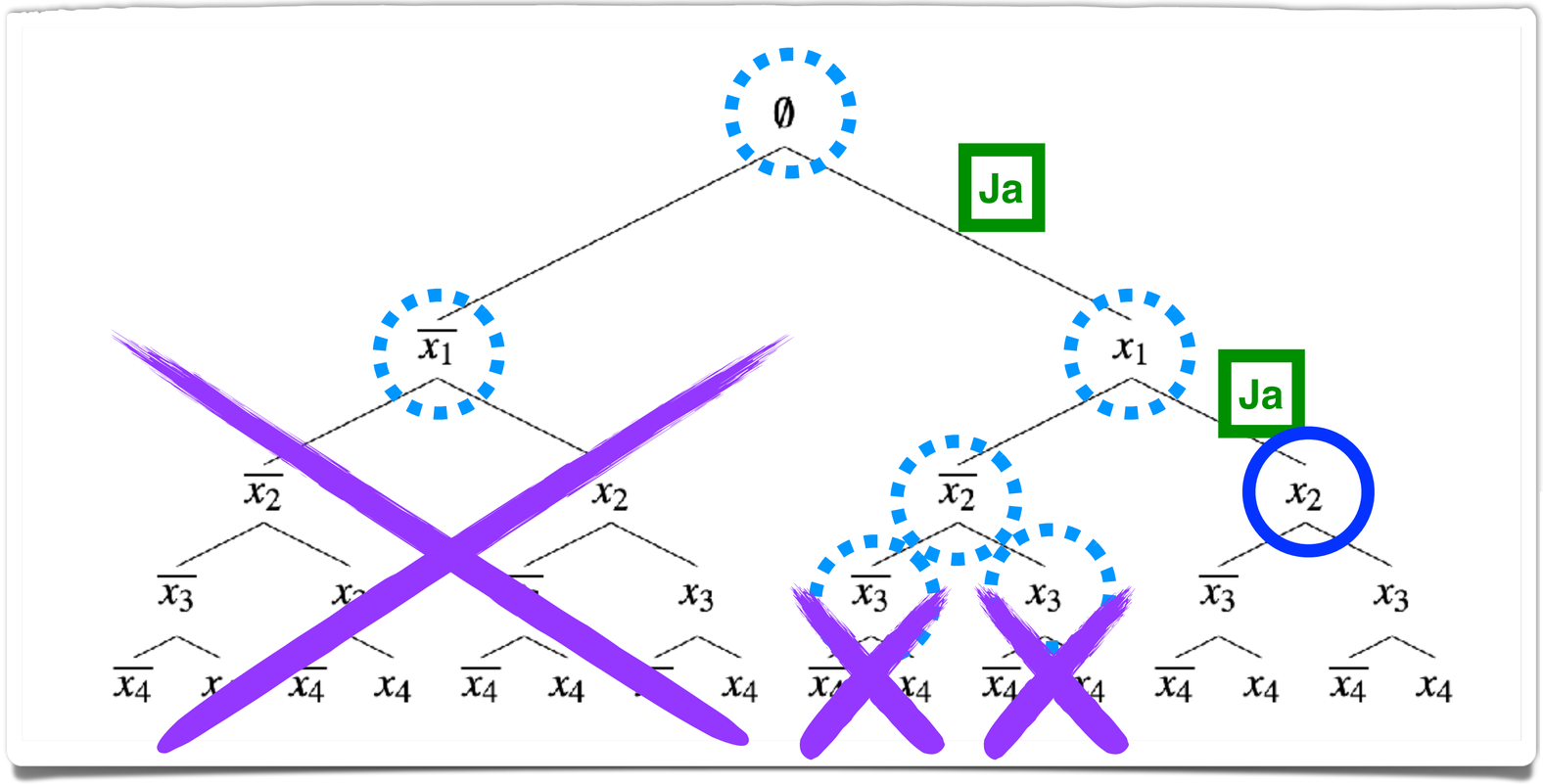
Gesamtbeispiel



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

LB=15

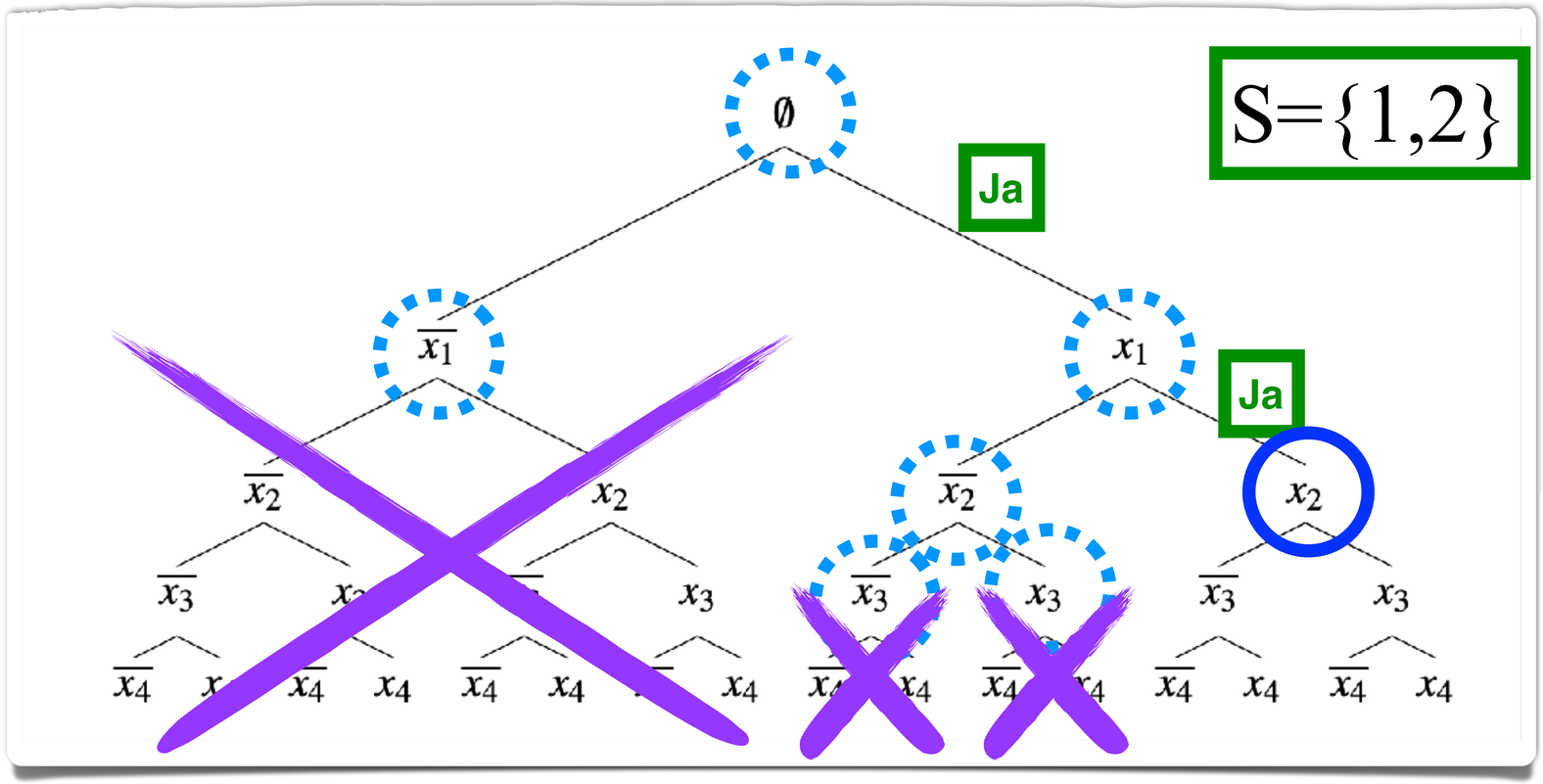
Gesamtbeispiel



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

LB=15

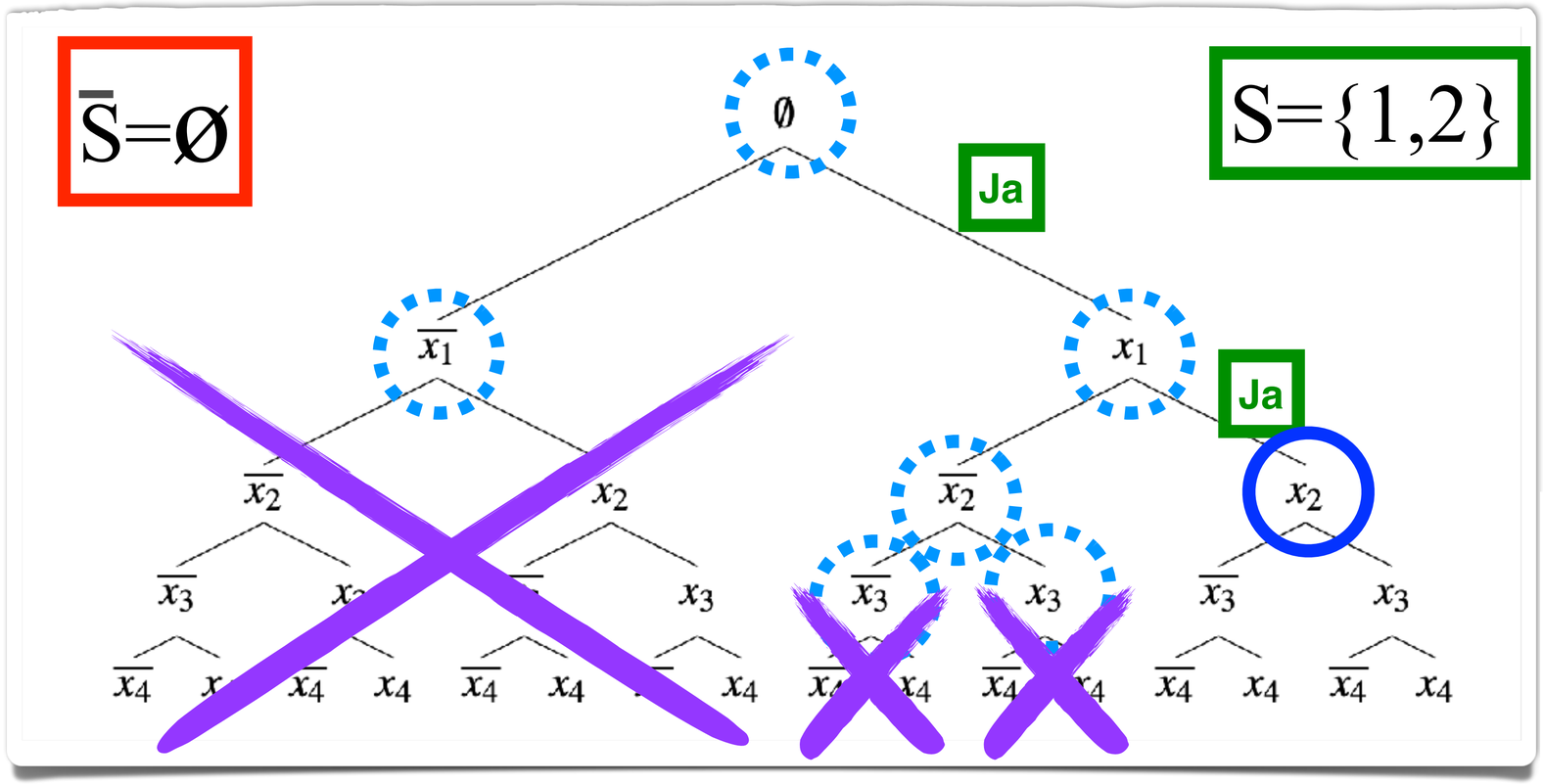
Gesamtbeispiel



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

LB=15

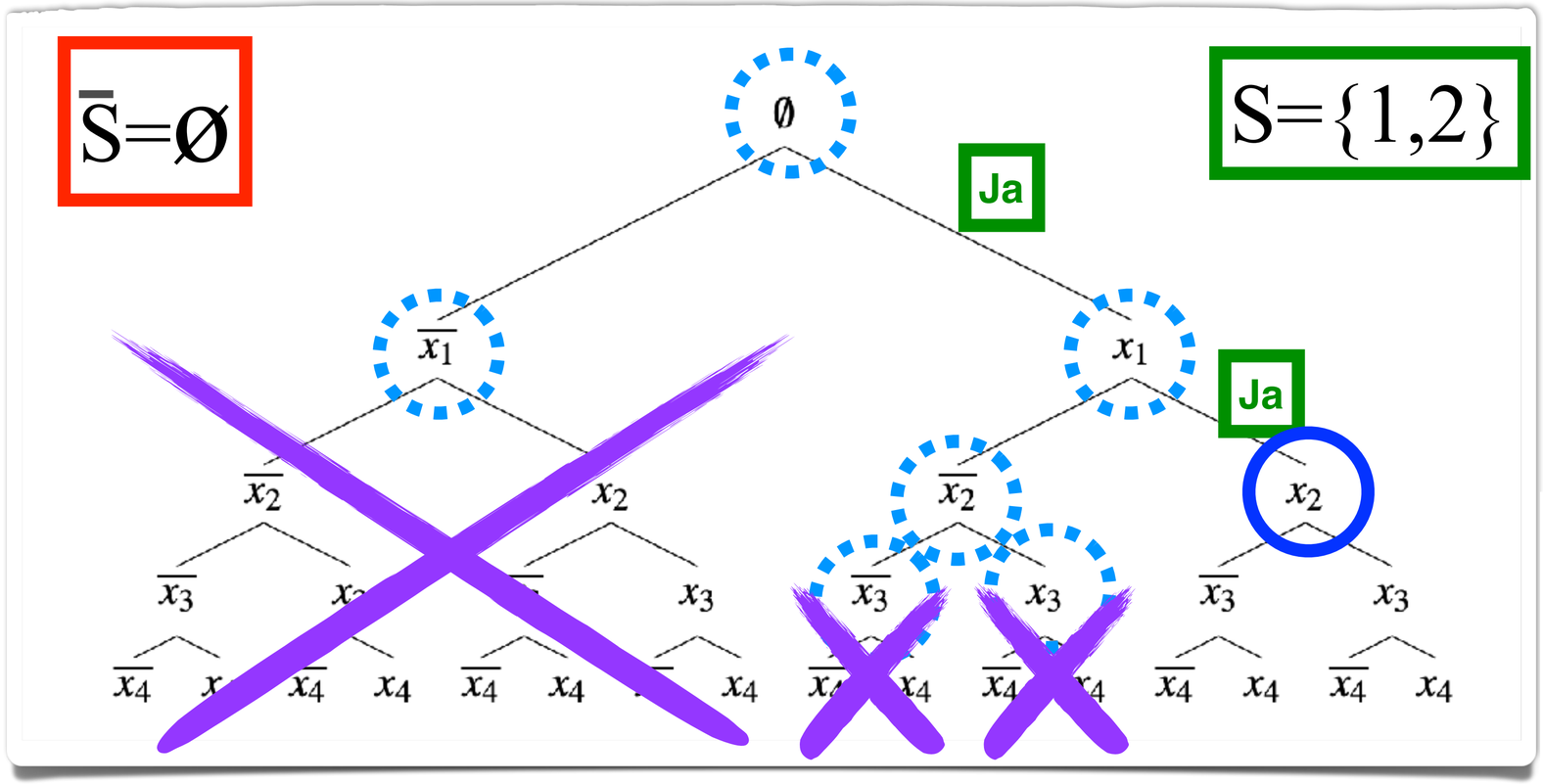
Gesamtbeispiel



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

LB=15

Gesamtbeispiel

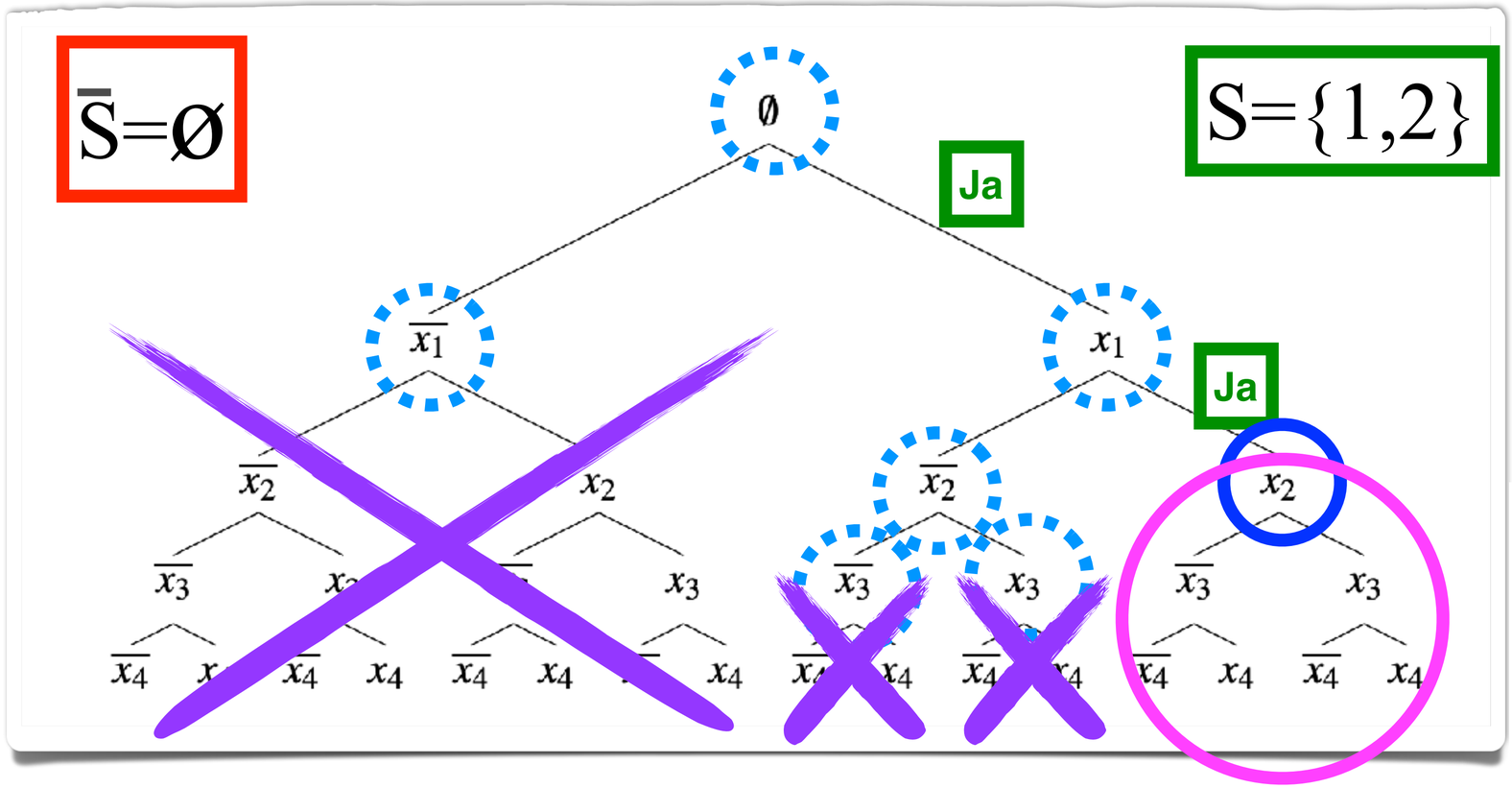


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=15

Gesamtbeispiel

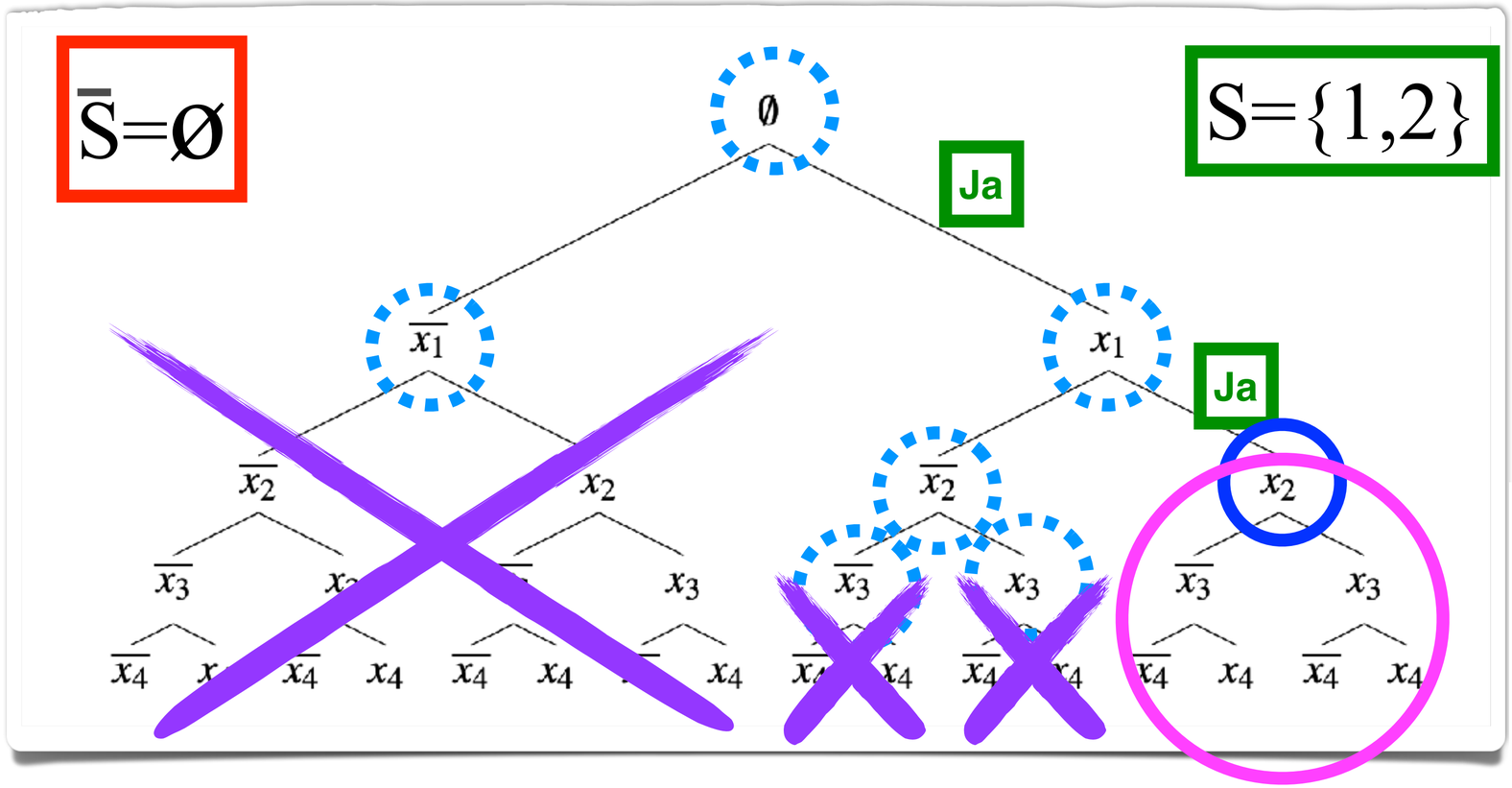


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=15

Gesamtbeispiel

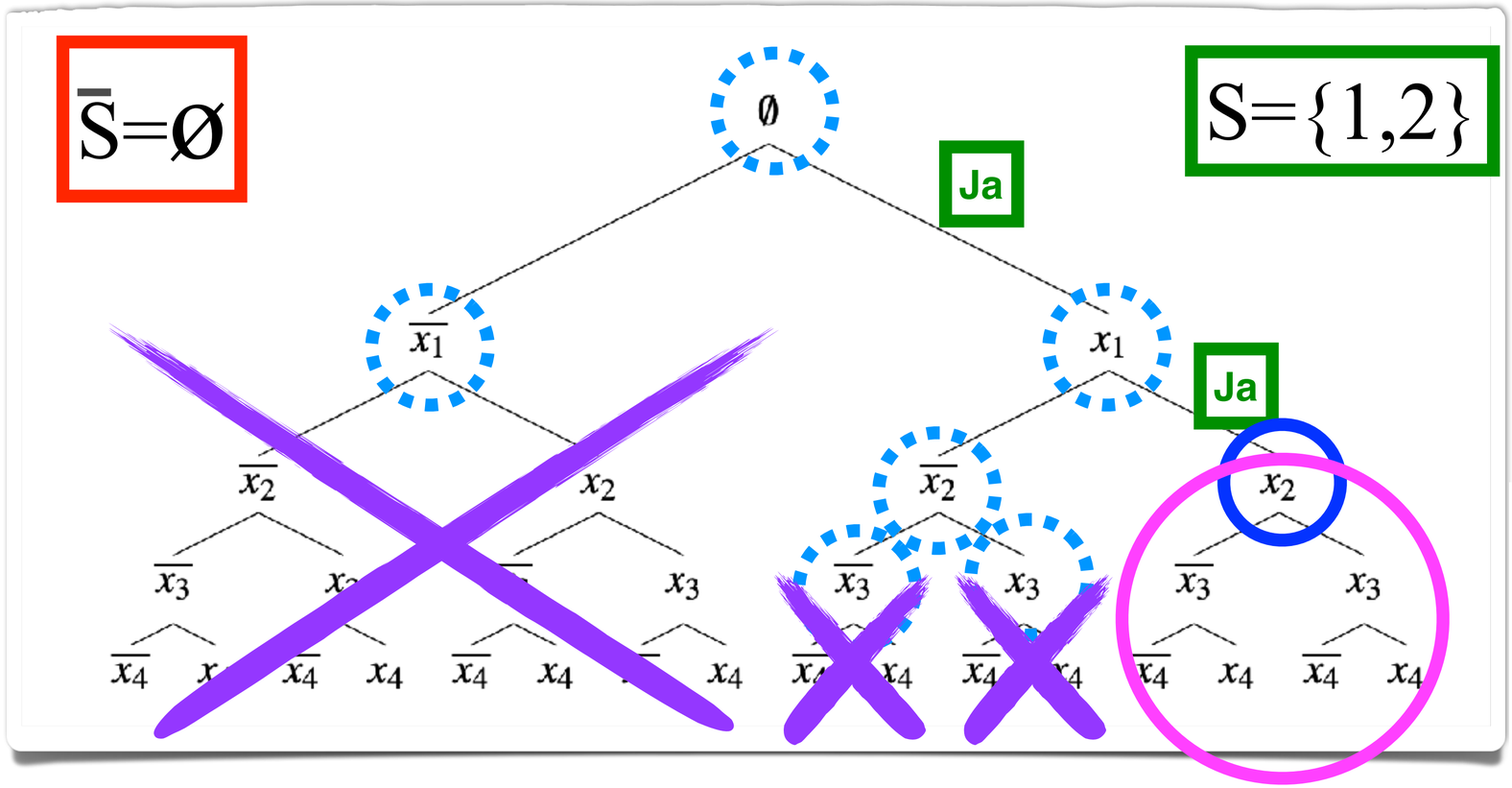


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=15

Gesamtbeispiel

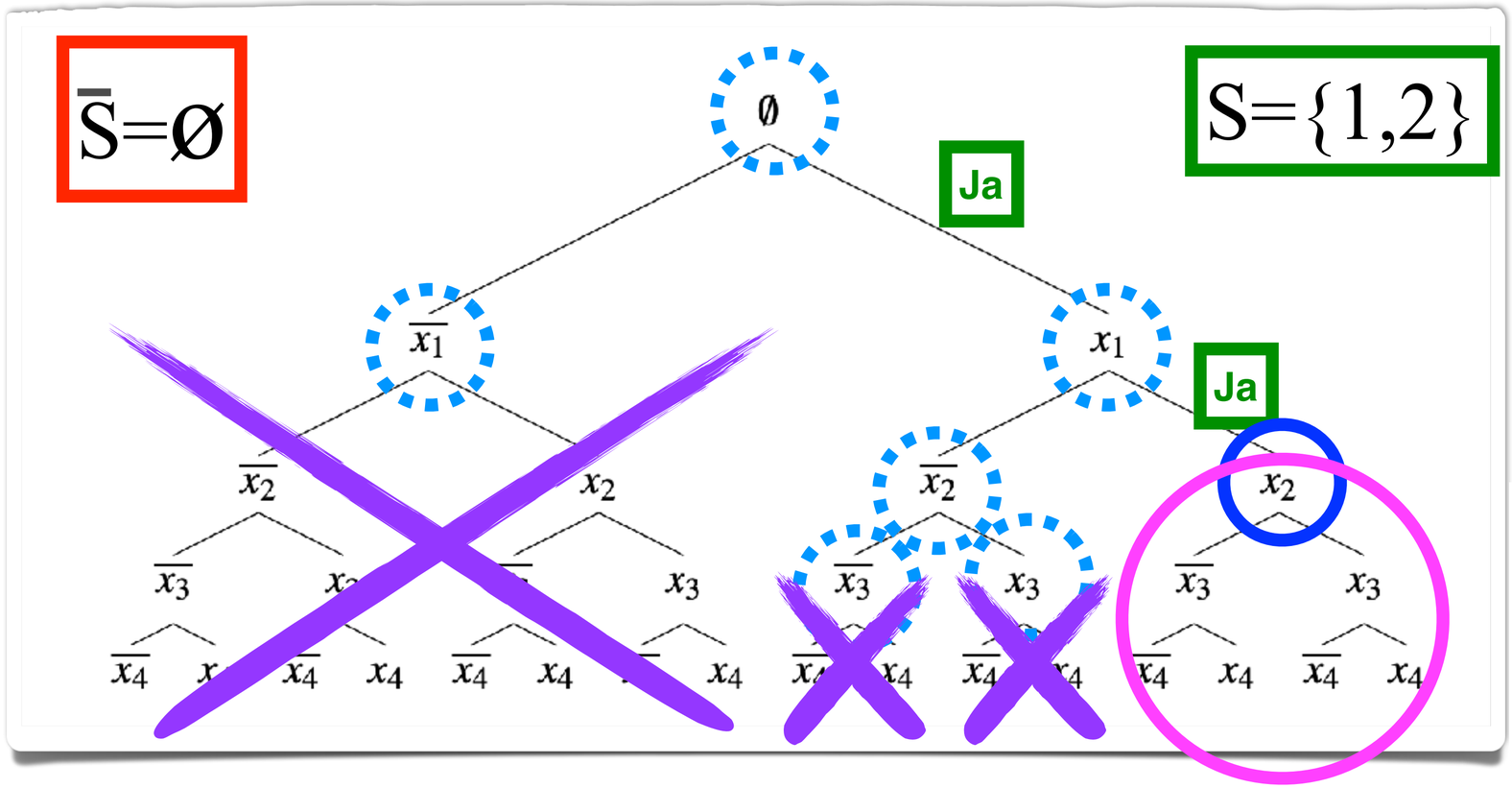


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=15

Gesamtbeispiel

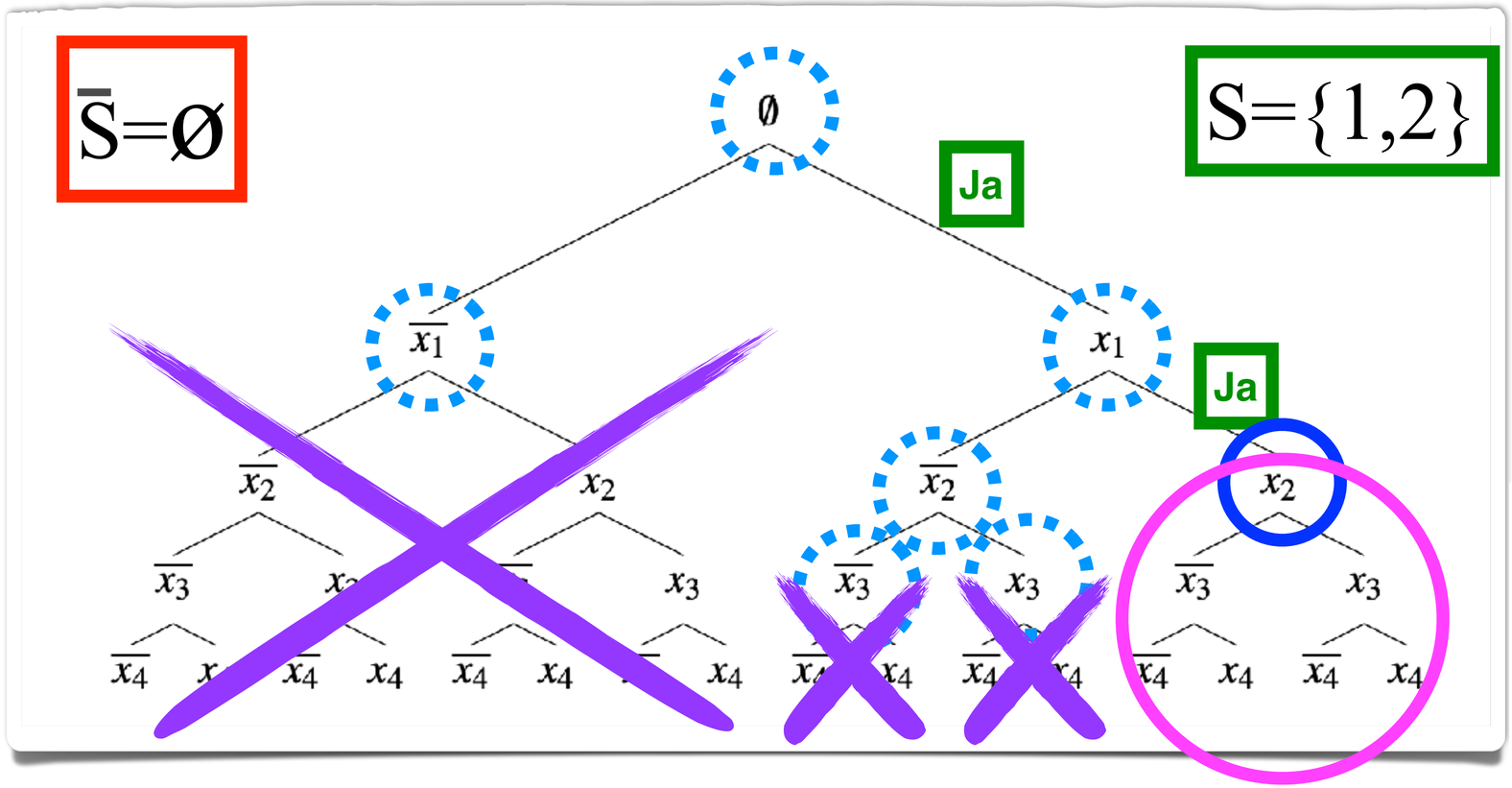


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=15

Gesamtbeispiel

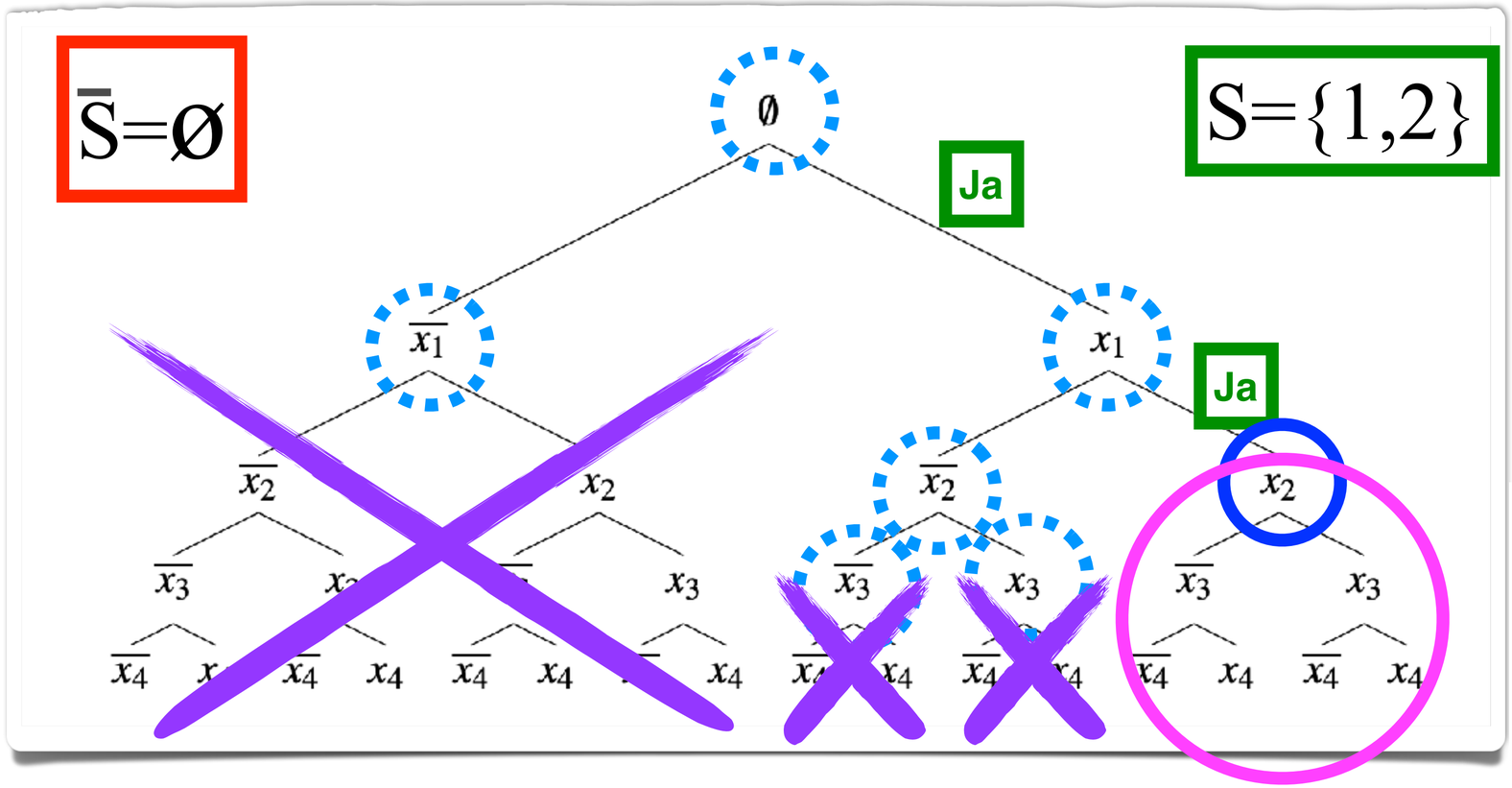


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=15

Gesamtbeispiel

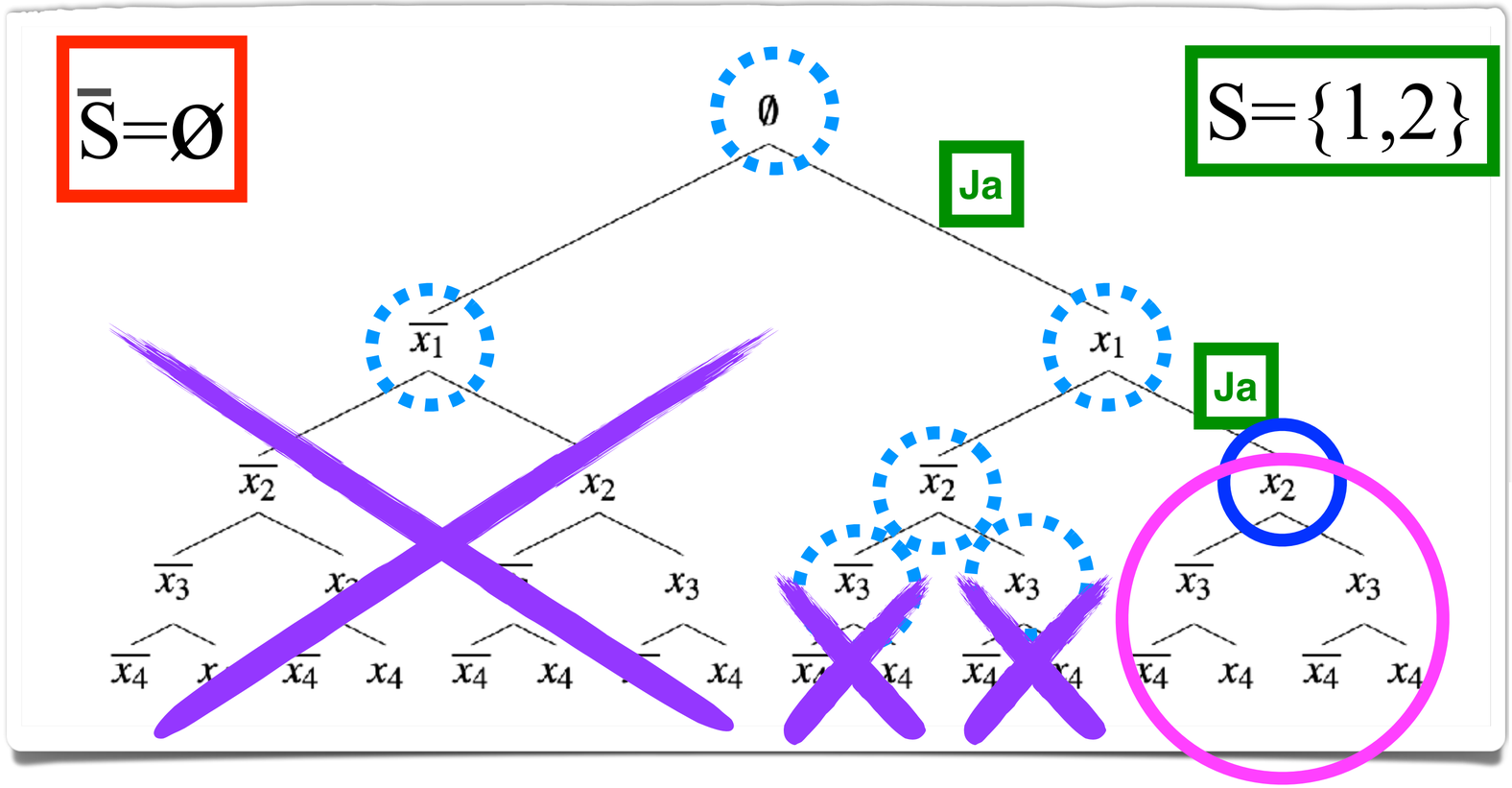


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=15

Gesamtbeispiel

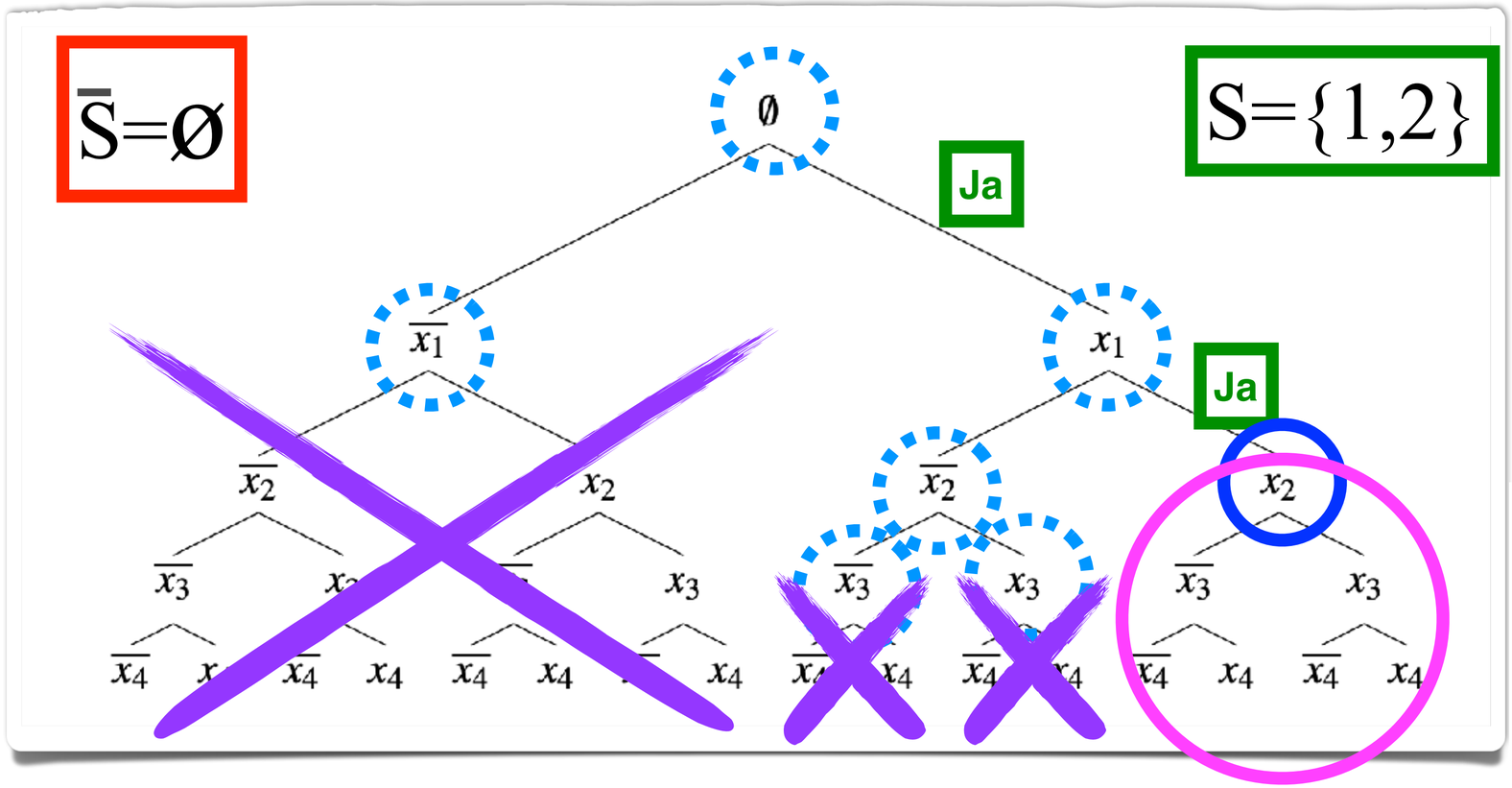


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=15

Gesamtbeispiel

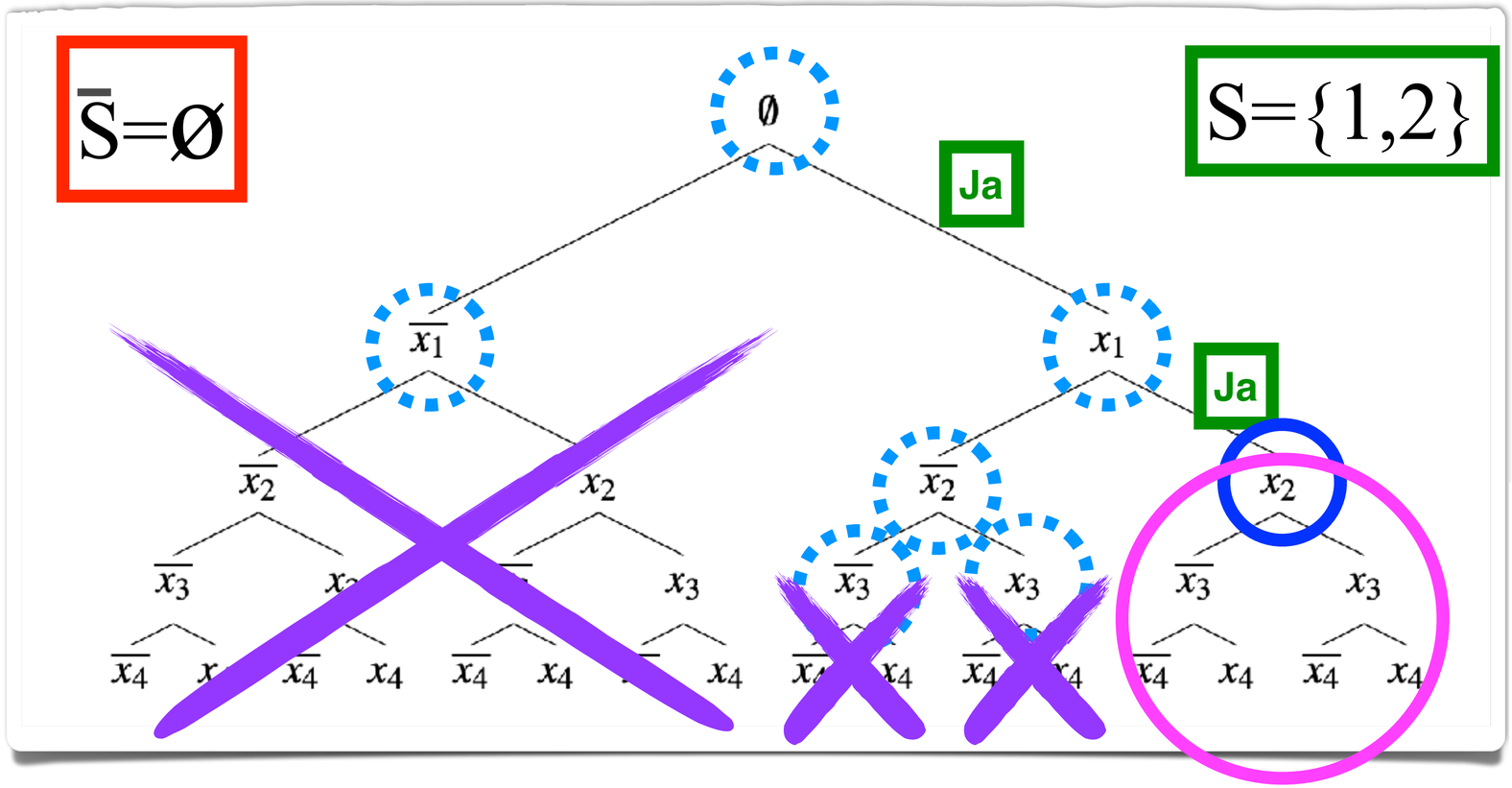


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=15

Gesamtbeispiel

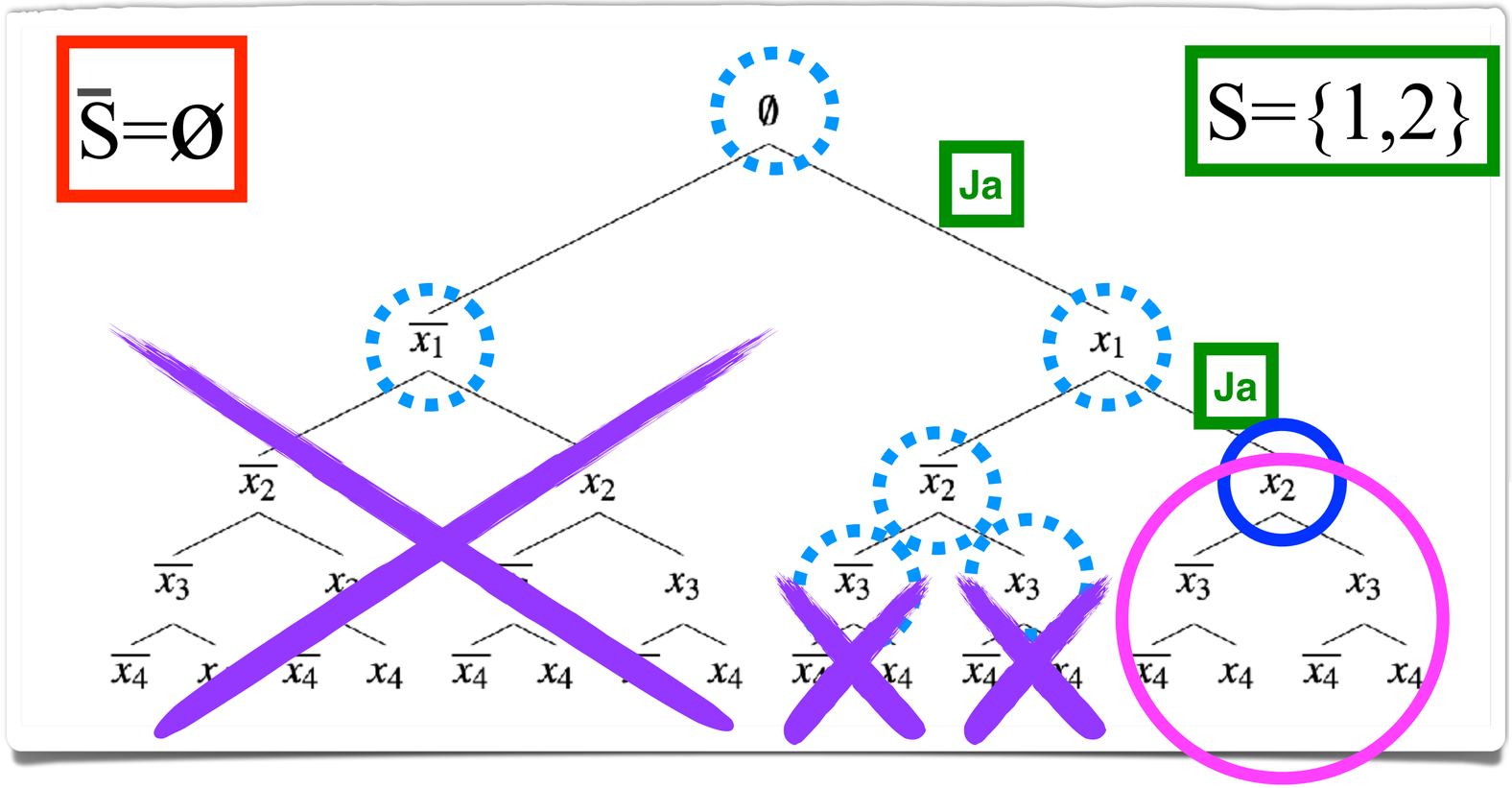


i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

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

LB=15

Gesamtbeispiel



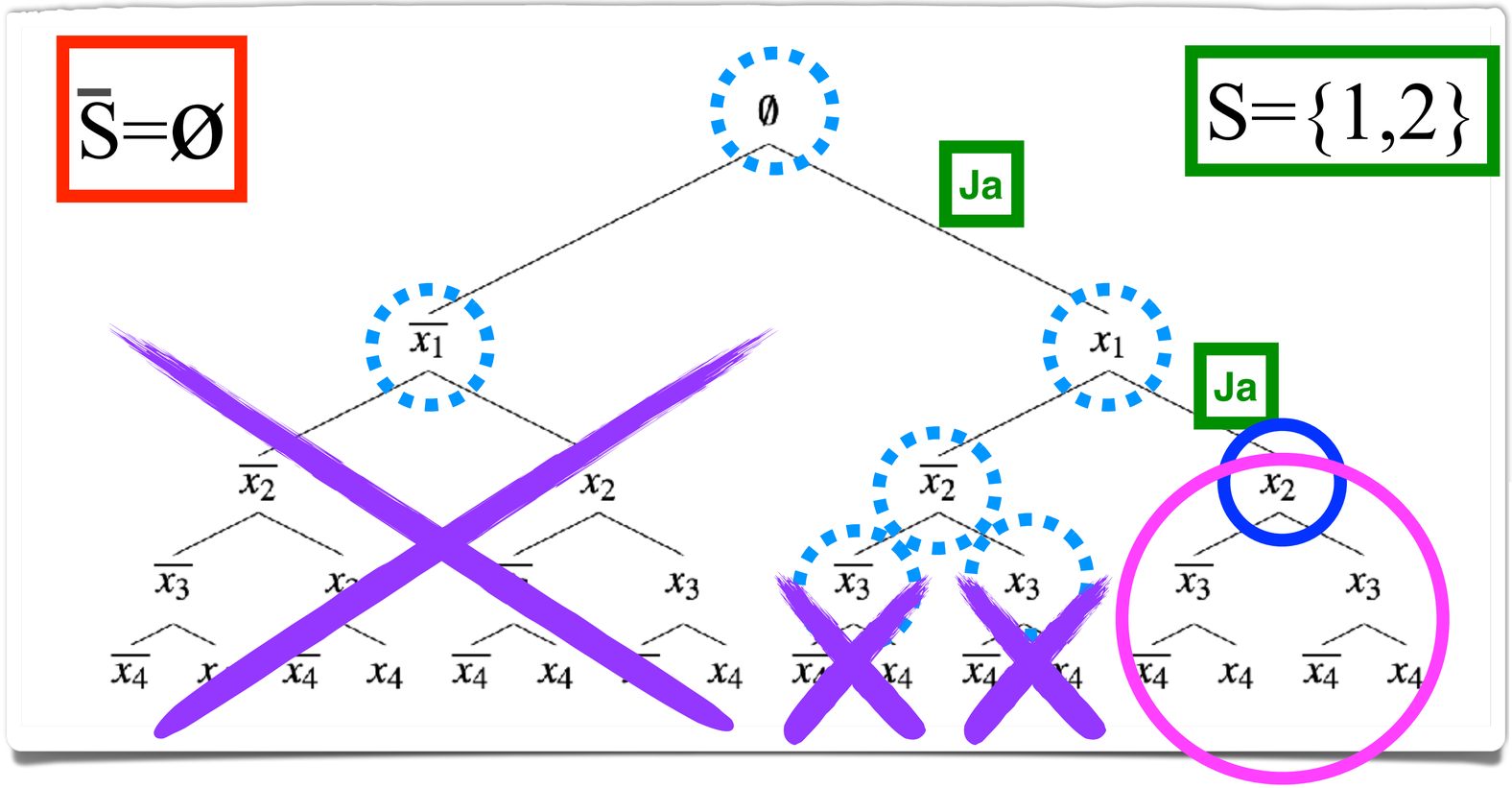
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

UB=14

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

LB=15

Gesamtbeispiel



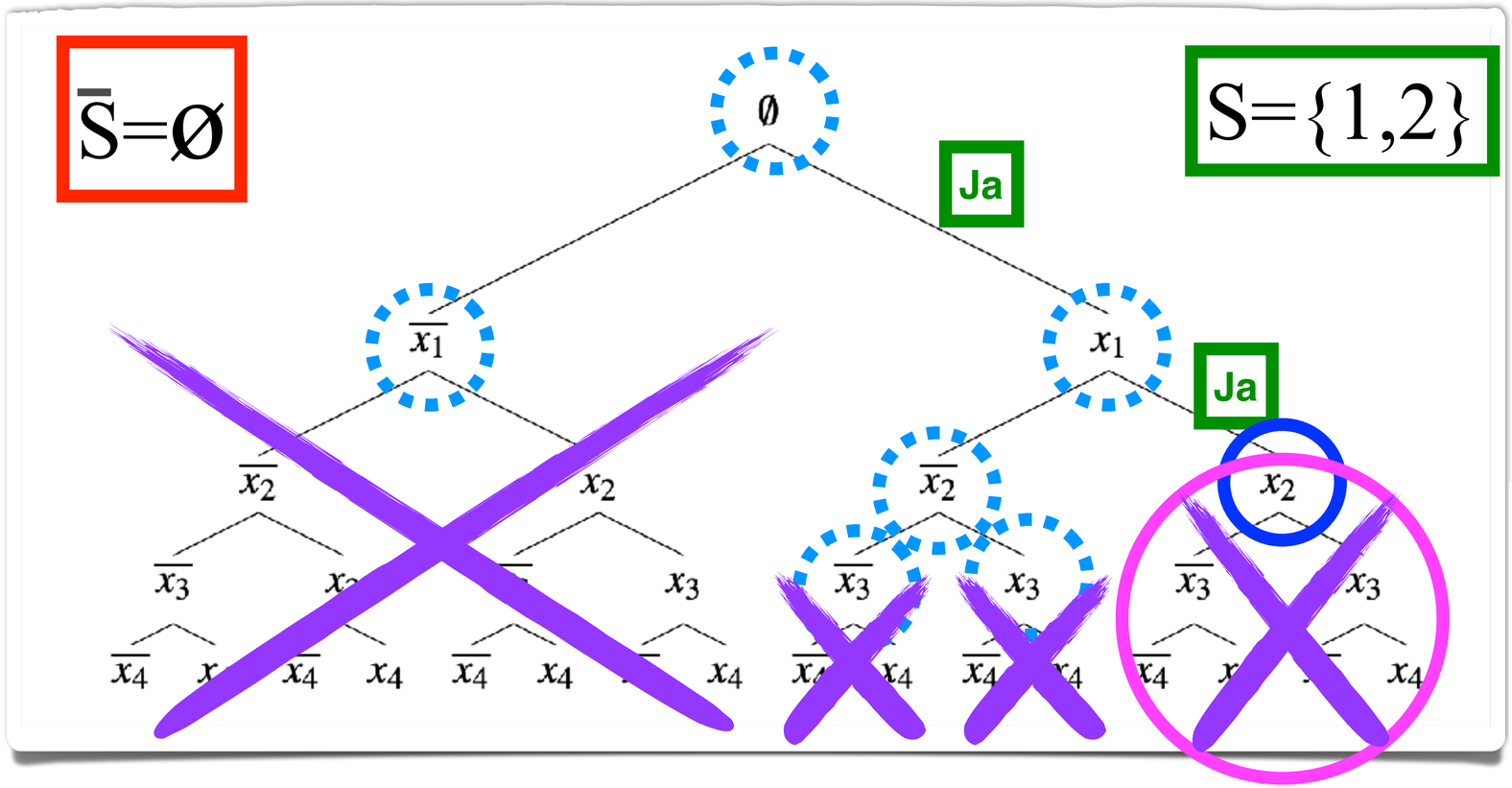
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

UB=14

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

LB=15

Gesamtbeispiel



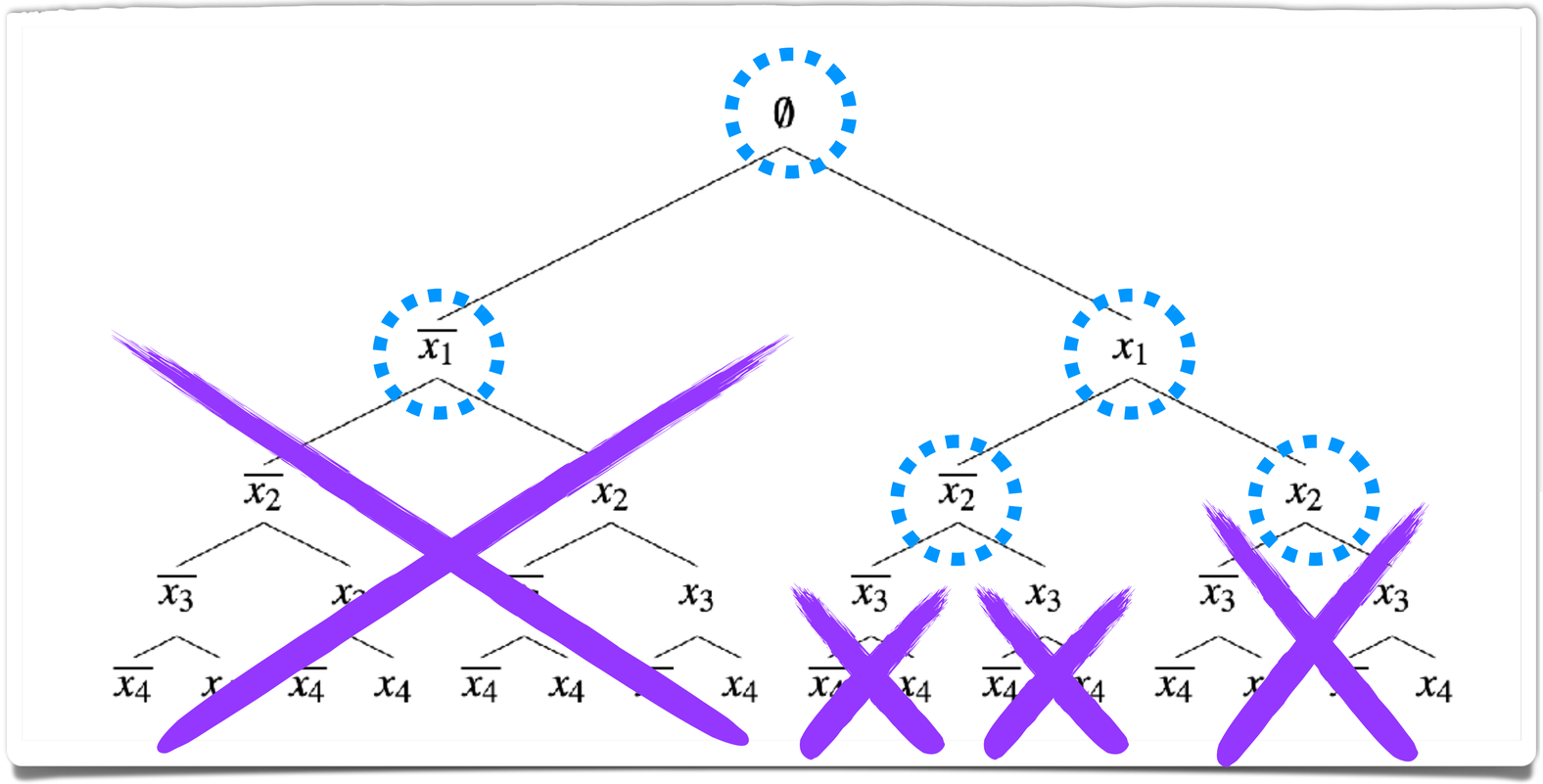
i	1	2	3	4	5	6	7
z_i	2	3	6	7	5	9	4
p_i	6	5	8	9	6	7	3

UB=14

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

LB=15

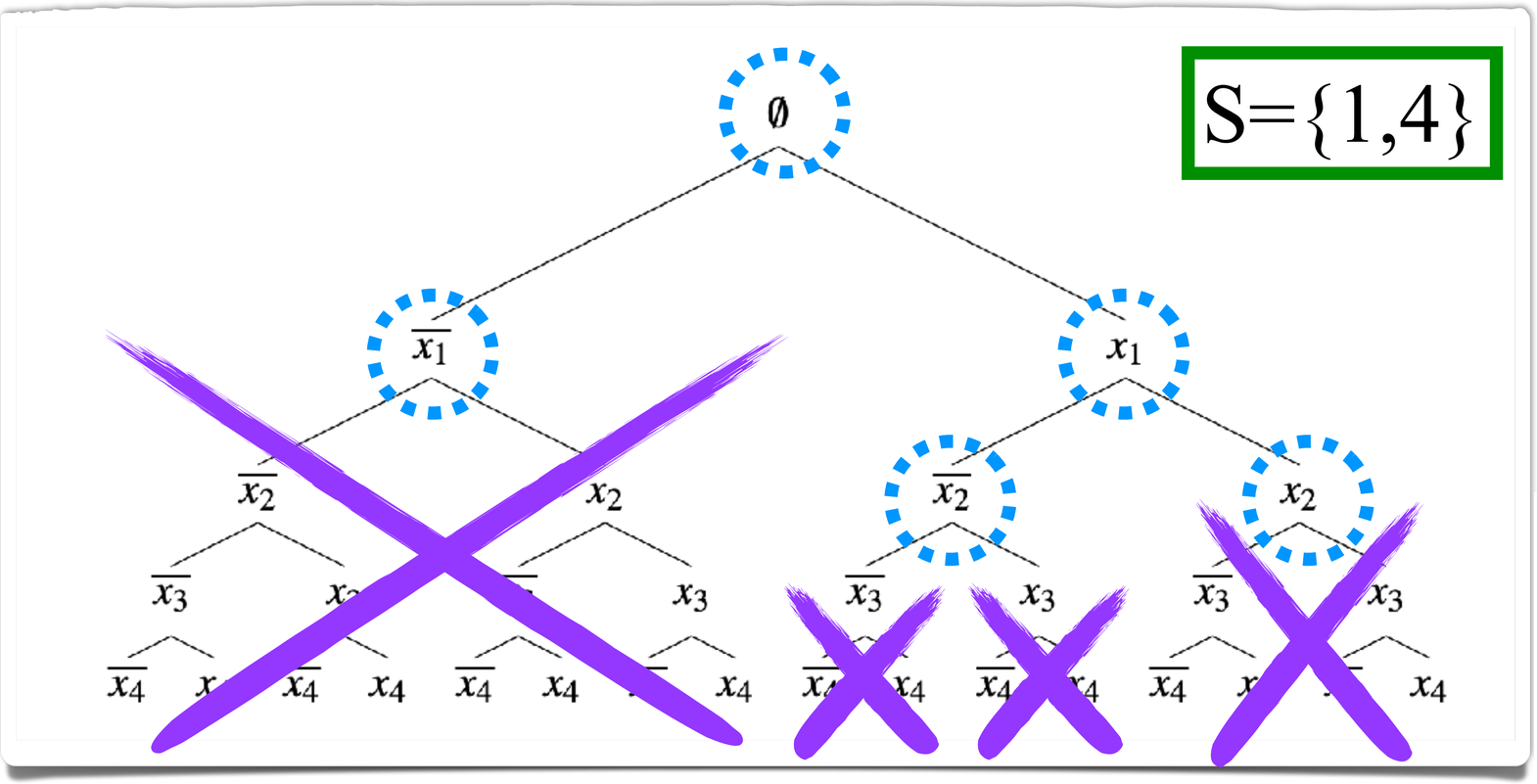
Gesamtbeispiel



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

LB=15

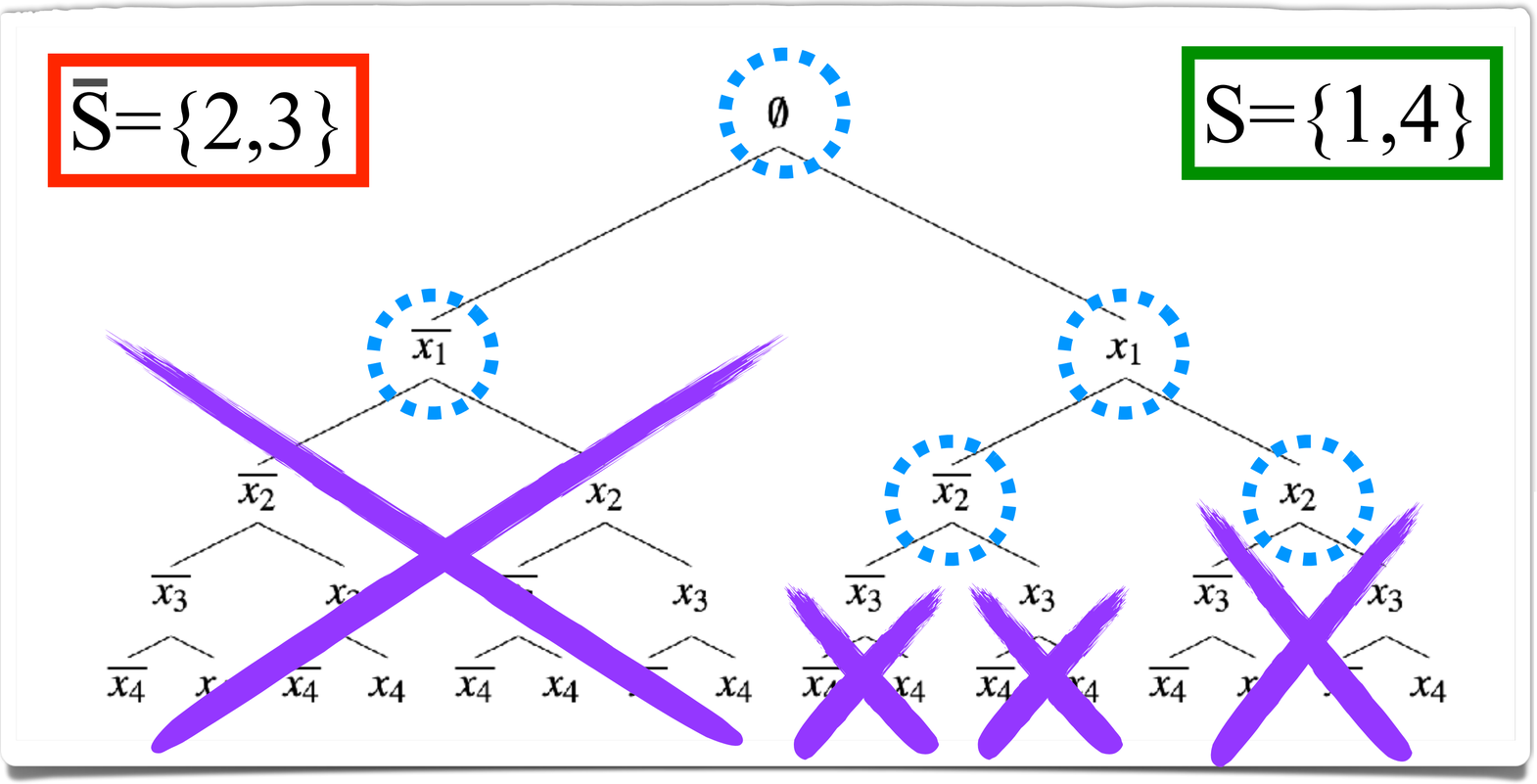
Gesamtbeispiel



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

LB=15

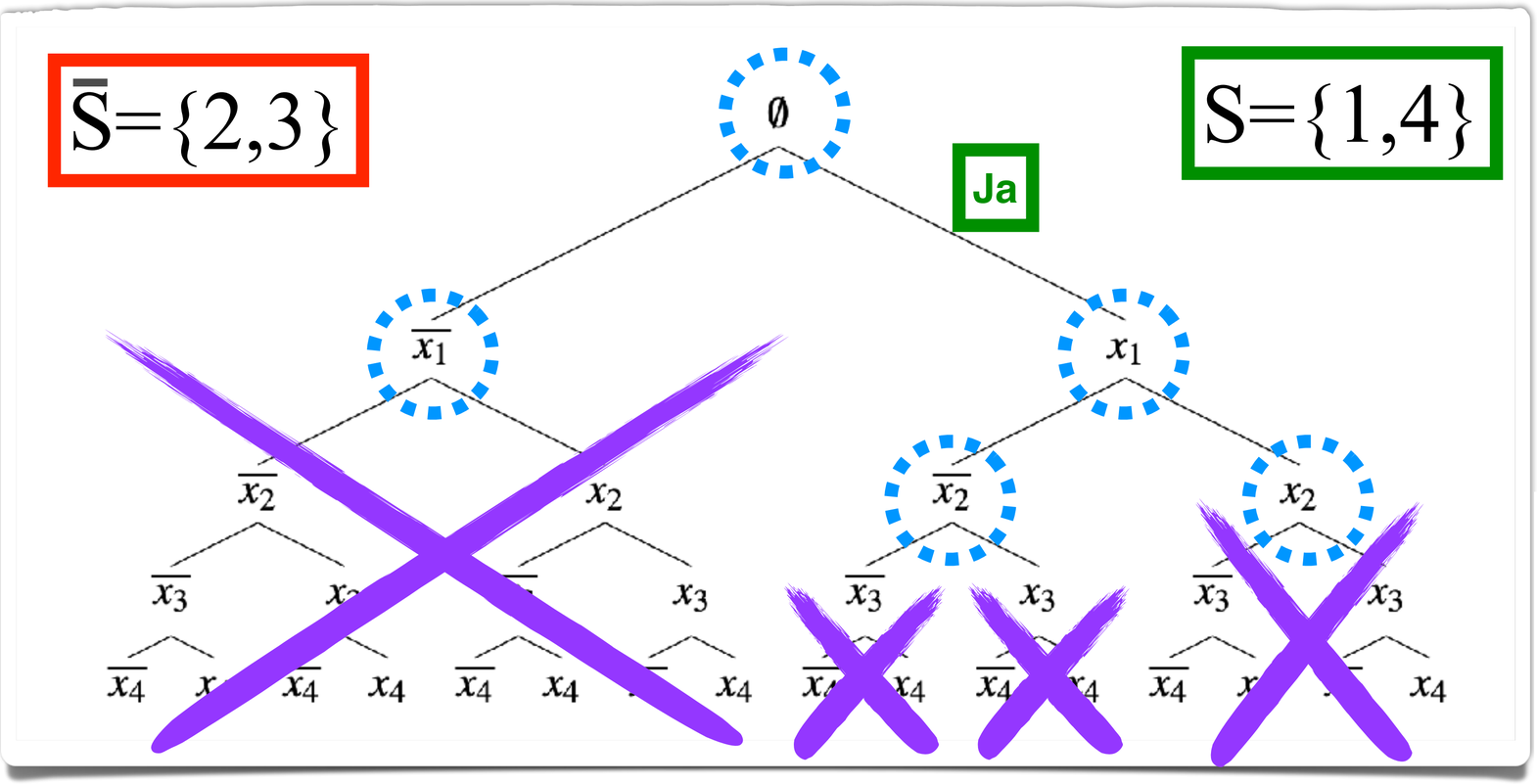
Gesamtbeispiel



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

LB=15

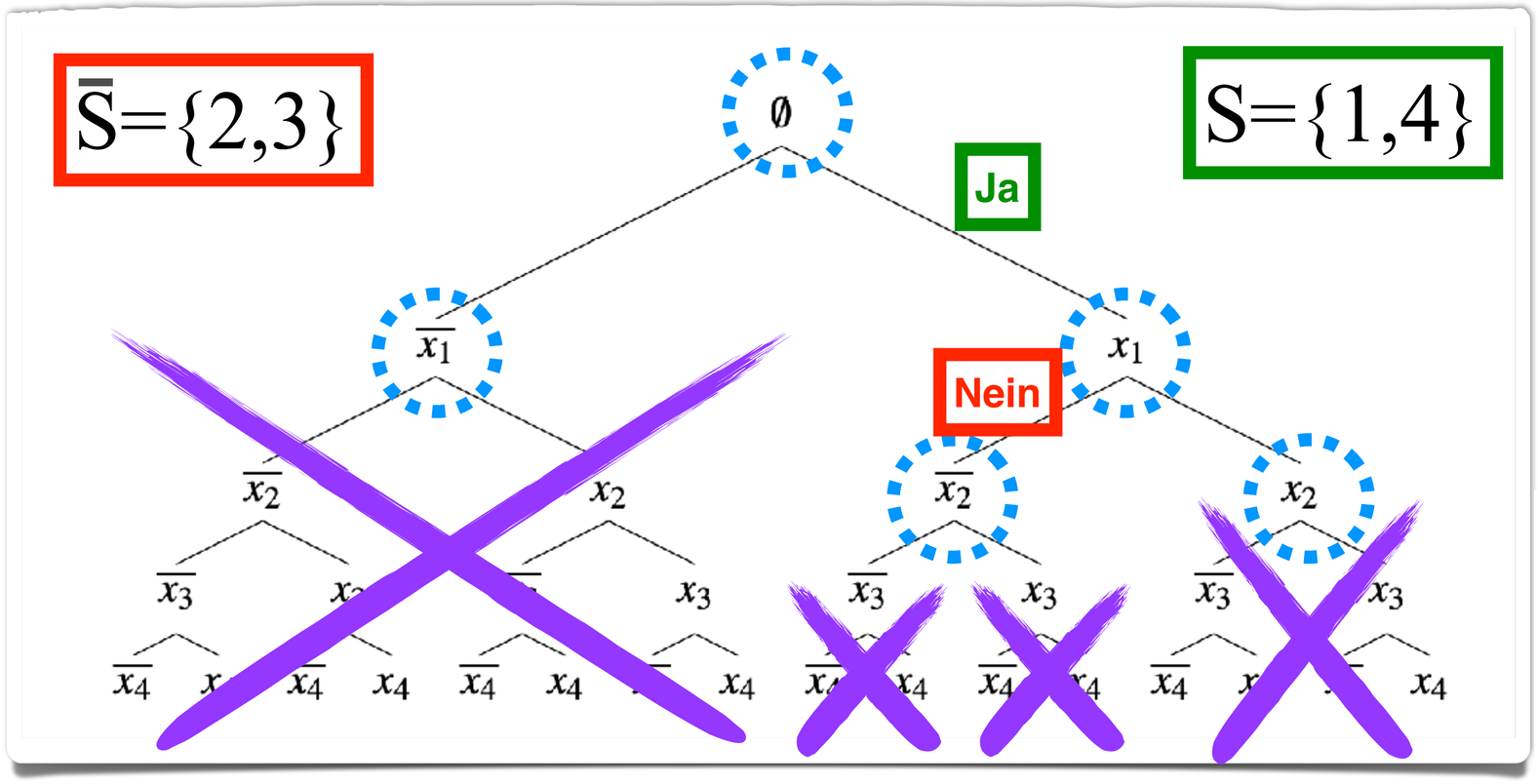
Gesamtbeispiel



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

LB=15

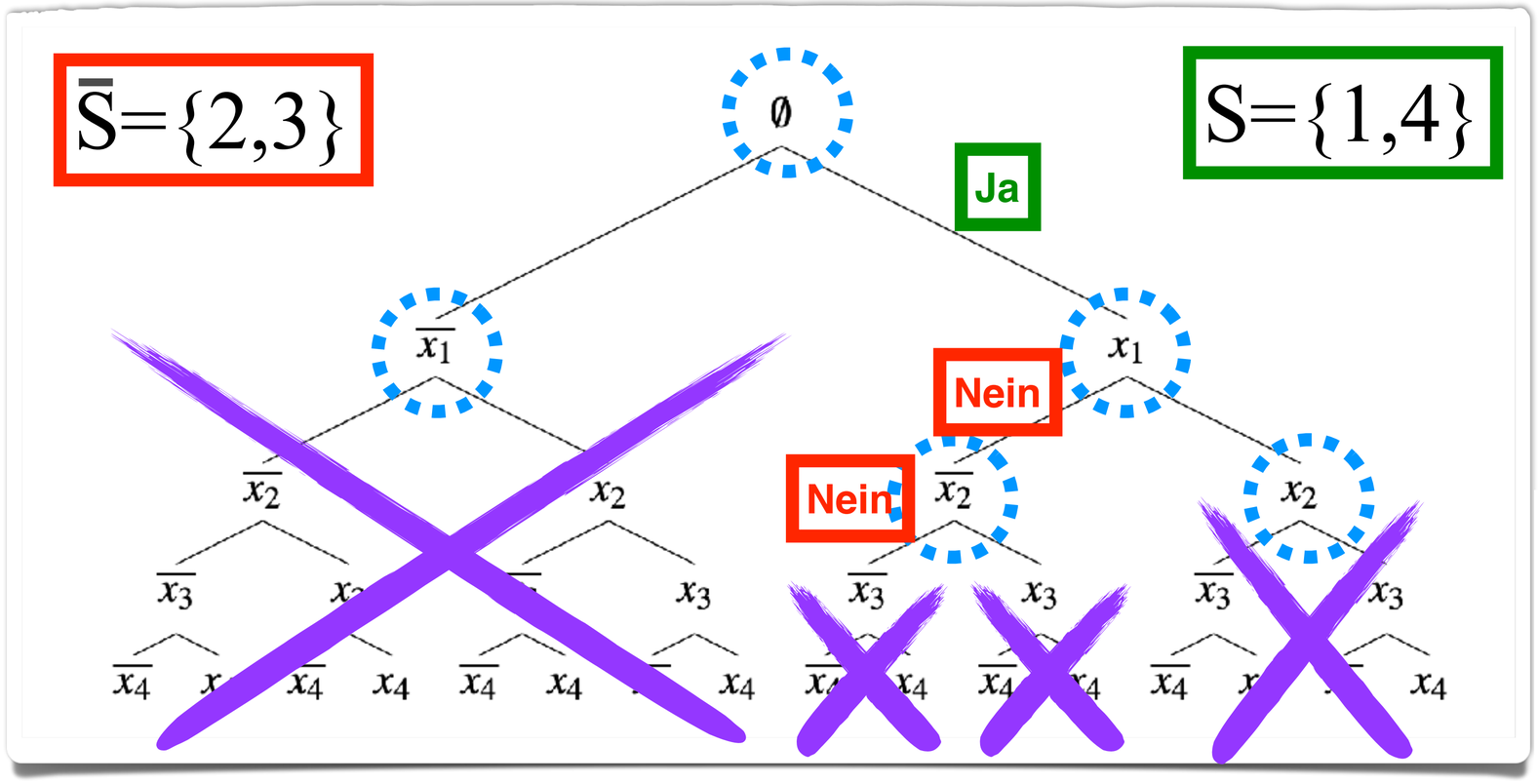
Gesamtbeispiel



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

LB=15

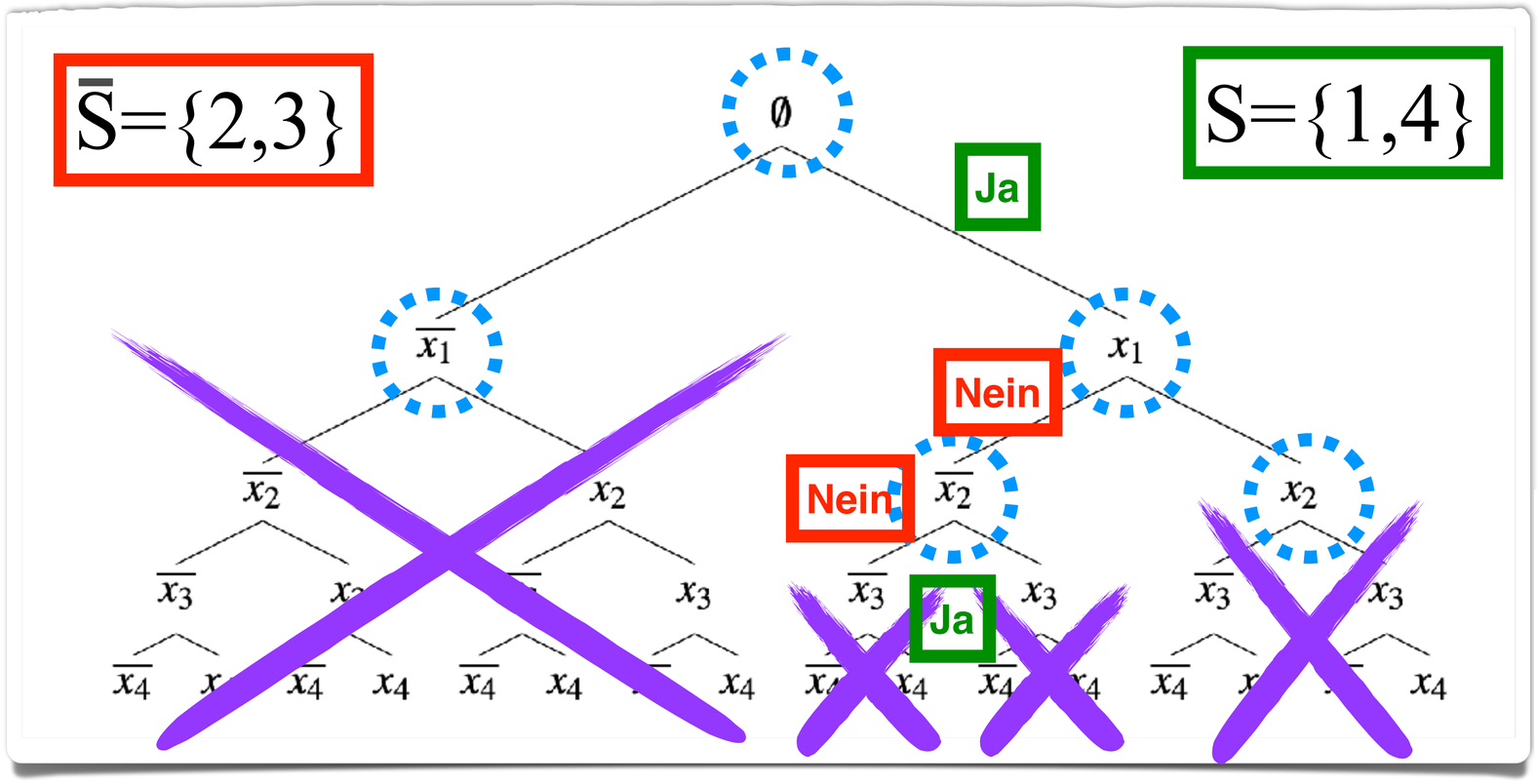
Gesamtbeispiel



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

LB=15

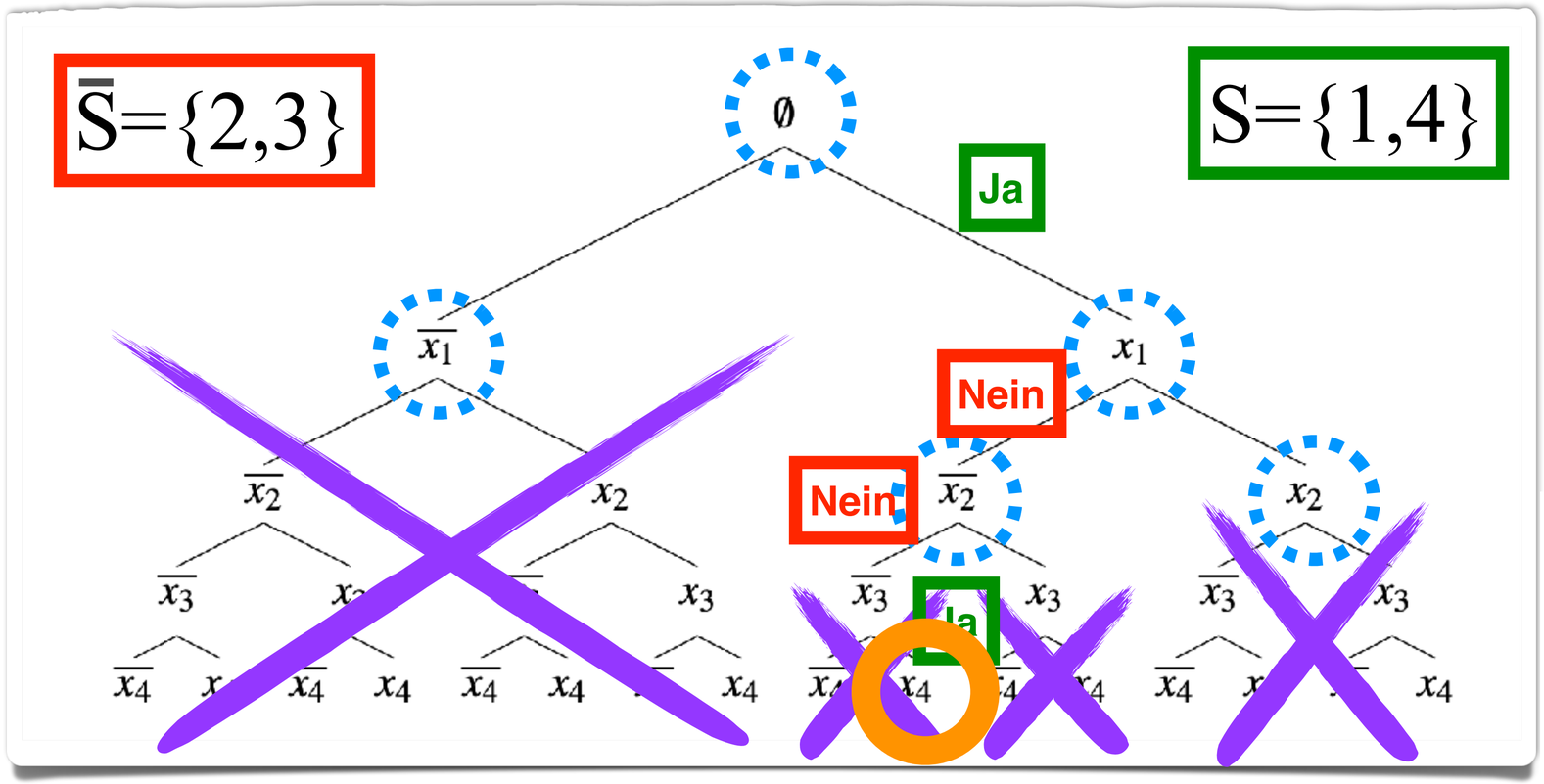
Gesamtbeispiel



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

LB=15

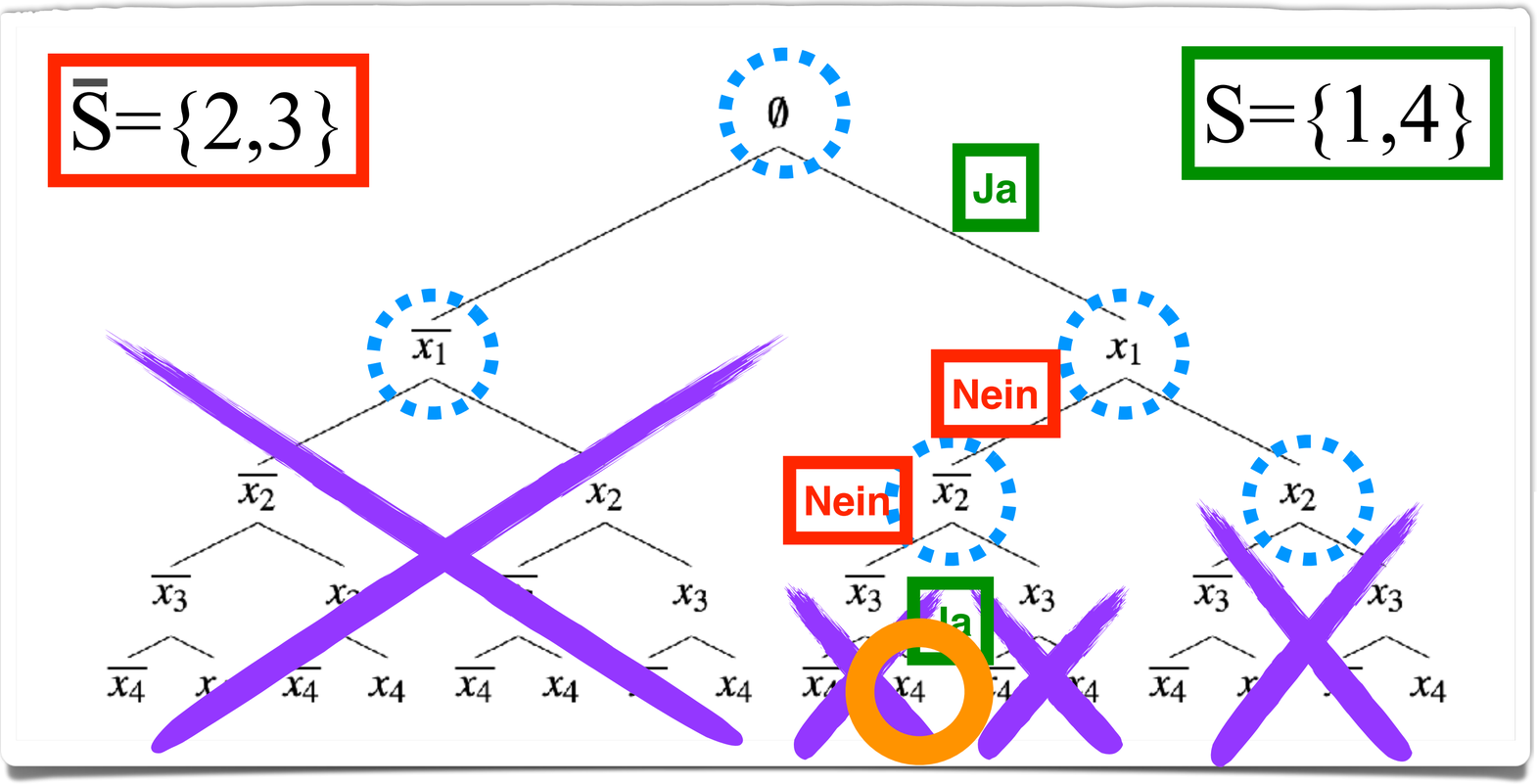
Gesamtbeispiel



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

LB=15

Gesamtbeispiel



Also: Das Optimum ist 15 mit $S = \{1, 4\}$

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

LB=15

Algorithmus

Algorithmus

Algorithmus 3.1

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

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

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

Eingabe: $z_1, ..., z_n, Z, p_1, ...p_n$ *(global: Kosten/Nutzenwerte, Kostenschranke)*
P *(bester bekannter Lösungswert)*

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

Eingabe:	$z_1, ..., z_n, Z, p_1, ...p_n$	*(global: Kosten/Nutzenwerte, Kostenschranke)*
	P	*(bester bekannter Lösungswert)*
	ℓ	*(nächster Index, über den verzweigt werden soll)*

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

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

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

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

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

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

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

P *(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*

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

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

P *(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)

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

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

P *(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*** $(\sum_{j=1}^{\ell-1} b_j z_j > Z)$ ***then return*** ▷ *unzulässig*

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

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

P *(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*** $(\sum_{j=1}^{\ell-1} b_j z_j > Z)$ ***then return*** ▷ *unzulässig*

3: *Compute* $L := LB(b_1, \ldots, b_{\ell-1})$

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

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

P *(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*** $(\sum_{j=1}^{\ell-1} b_j z_j > Z)$ ***then return*** ▷ *unzulässig*

3: *Compute* $L := LB(b_1, \ldots, b_{\ell-1})$

4: ***if*** $L > P$ ***then*** $P := L$ ▷ *Lösungswert verbessert*

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

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

P *(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*** $(\sum_{j=1}^{\ell-1} b_j z_j > Z)$ ***then return*** ▷ *unzulässig*

3: *Compute* $L := LB(b_1, \ldots, b_{\ell-1})$

4: ***if*** $L > P$ ***then*** $P := L$ ▷ *Lösungswert verbessert*

5: ***if*** $(\ell > n)$ ***then return*** ▷ *Blatt im Baum erreicht*

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

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

P *(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*** $(\sum_{j=1}^{\ell-1} b_j z_j > Z)$ ***then return*** ▷ *unzulässig*

3: *Compute* $L := LB(b_1, \ldots, b_{\ell-1})$

4: ***if*** $L > P$ ***then*** $P := L$ ▷ *Lösungswert verbessert*

5: ***if*** $(\ell > n)$ ***then return*** ▷ *Blatt im Baum erreicht*

6: $U := UB(b_1, \ldots, b_{\ell-1})$ ▷ *(Obere Schranke berechnen)*

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

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

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

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

1: ***procedure*** BRANCH-AND-BOUND(ℓ)

2: ***if*** $(\sum_{j=1}^{\ell-1} b_j z_j > Z)$ ***then return*** ▷ *unzulässig*

3: *Compute* $L := LB(b_1, \ldots, b_{\ell-1})$

4: ***if*** $L > P$ ***then*** $P := L$ ▷ *Lösungswert verbessert*

5: ***if*** $(\ell > n)$ ***then return*** ▷ *Blatt im Baum erreicht*

6: $U := UB(b_1, \ldots, b_{\ell-1})$ ▷ *(Obere Schranke berechnen)*

7: ***if*** $(U > P)$ ***then***

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

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

P *(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);
```

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

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

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

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);
```

Algorithmus

Algorithmus 3.1 (Branch-And-Bound als Unterroutine).

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

P *(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
```

Satz

Satz

Satz 3.2 *Algorithmus 2.10 (als rekursiv arbeitende Unterroutine) berechnet eine optimale Lösung für das Knapsackproblem in einer Worst-Case-Laufzeit $O(2^n f(n))$, wobei $f(n)$ die Zeit für die Berechnung der oberen Schranke ist.*

Satz

Satz 3.2 *Algorithmus 2.10 (als rekursiv arbeitende Unterroutine) berechnet eine optimale Lösung für das Knapsackproblem in einer Worst-Case-Laufzeit $O(2^n f(n))$, wobei $f(n)$ die Zeit für die Berechnung der oberen Schranke ist.*

Beweis.

Satz

Satz 3.2 *Algorithmus 2.10 (als rekursiv arbeitende Unterroutine) berechnet eine optimale Lösung für das Knapsackproblem in einer Worst-Case-Laufzeit $O(2^n f(n))$, wobei $f(n)$ die Zeit für die Berechnung der oberen Schranke ist.*

Beweis. Es werden systematisch alle Teillösungen durchprobiert.

Satz

Satz 3.2 *Algorithmus 2.10 (als rekursiv arbeitende Unterroutine) berechnet eine optimale Lösung für das Knapsackproblem in einer Worst-Case-Laufzeit $O(2^n f(n))$, wobei $f(n)$ die Zeit für die Berechnung der oberen Schranke ist.*

Beweis. Es werden systematisch alle Teillösungen durchprobiert. (Dabei werden Teilmengen nur dann ausgelassen,

Satz

Satz 3.2 *Algorithmus 2.10 (als rekursiv arbeitende Unterroutine) berechnet eine optimale Lösung für das Knapsackproblem in einer Worst-Case-Laufzeit $O(2^n f(n))$, wobei $f(n)$ die Zeit für die Berechnung der oberen Schranke ist.*

Beweis. Es werden systematisch alle Teillösungen durchprobiert. (Dabei werden Teilmengen nur dann ausgelassen, wenn sie erwiesenermaßen unzulässig sind (Zeile 2)

Satz

Satz 3.2 *Algorithmus 2.10 (als rekursiv arbeitende Unterroutine) berechnet eine optimale Lösung für das Knapsackproblem in einer Worst-Case-Laufzeit $O(2^n f(n))$, wobei $f(n)$ die Zeit für die Berechnung der oberen Schranke ist.*

Beweis. Es werden systematisch alle Teillösungen durchprobiert. (Dabei werden Teilmengen nur dann ausgelassen, wenn sie erwiesenermaßen unzulässig sind (Zeile 2) oder keine Verbesserung bringen (Zeile 4)

Satz

Satz 3.2 *Algorithmus 2.10 (als rekursiv arbeitende Unterroutine) berechnet eine optimale Lösung für das Knapsackproblem in einer Worst-Case-Laufzeit $O(2^n f(n))$, wobei $f(n)$ die Zeit für die Berechnung der oberen Schranke ist.*

Beweis. Es werden systematisch alle Teillösungen durchprobiert. (Dabei werden Teilmengen nur dann ausgelassen, wenn sie erwiesenermaßen unzulässig sind (Zeile 2) oder keine Verbesserung bringen (Zeile 4)
Die Zahl der Rekursionsaufrufe (Schritt 9 und 11) ist insgesamt 2^n,

Satz

Satz 3.2 *Algorithmus 2.10 (als rekursiv arbeitende Unterroutine) berechnet eine optimale Lösung für das Knapsackproblem in einer Worst-Case-Laufzeit $O(2^n f(n))$, wobei $f(n)$ die Zeit für die Berechnung der oberen Schranke ist.*

Beweis. Es werden systematisch alle Teillösungen durchprobiert. (Dabei werden Teilmengen nur dann ausgelassen, wenn sie erwiesenermaßen unzulässig sind (Zeile 2) oder keine Verbesserung bringen (Zeile 4)
Die Zahl der Rekursionsaufrufe (Schritt 9 und 11) ist insgesamt 2^n, die sonstigen Berechnungen benötigen jeweils $f(n)$.

Satz

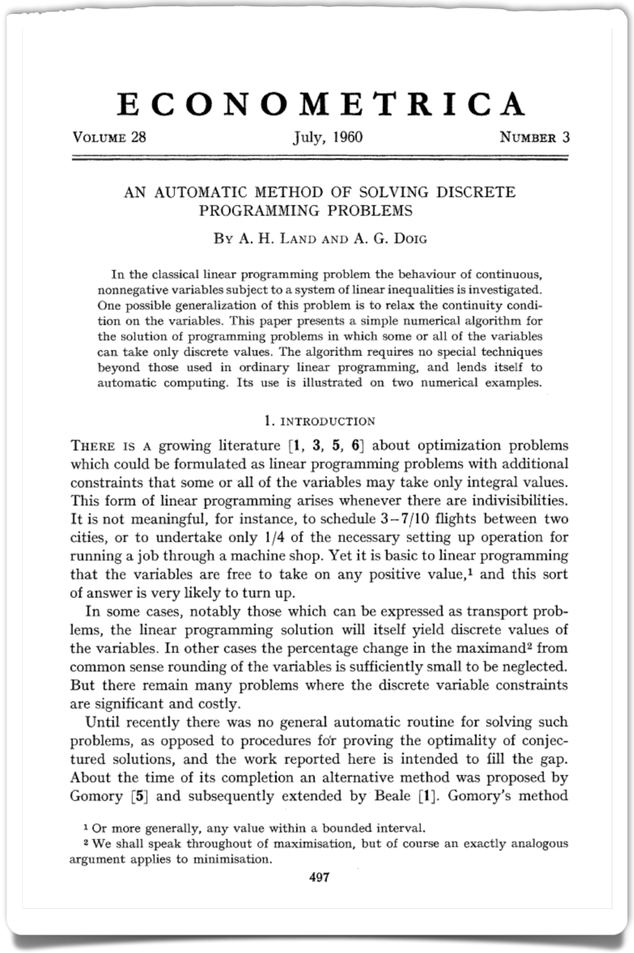
Satz 3.2 *Algorithmus 2.10 (als rekursiv arbeitende Unterroutine) berechnet eine optimale Lösung für das Knapsackproblem in einer Worst-Case-Laufzeit $O(2^n f(n))$, wobei $f(n)$ die Zeit für die Berechnung der oberen Schranke ist.*

Beweis. Es werden systematisch alle Teillösungen durchprobiert. (Dabei werden Teilmengen nur dann ausgelassen, wenn sie erwiesenermaßen unzulässig sind (Zeile 2) oder keine Verbesserung bringen (Zeile 4)
Die Zahl der Rekursionsaufrufe (Schritt 9 und 11) ist insgesamt 2^n, die sonstigen Berechnungen benötigen jeweils $f(n)$. □

3.4 Ausblicke

Historie

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 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.

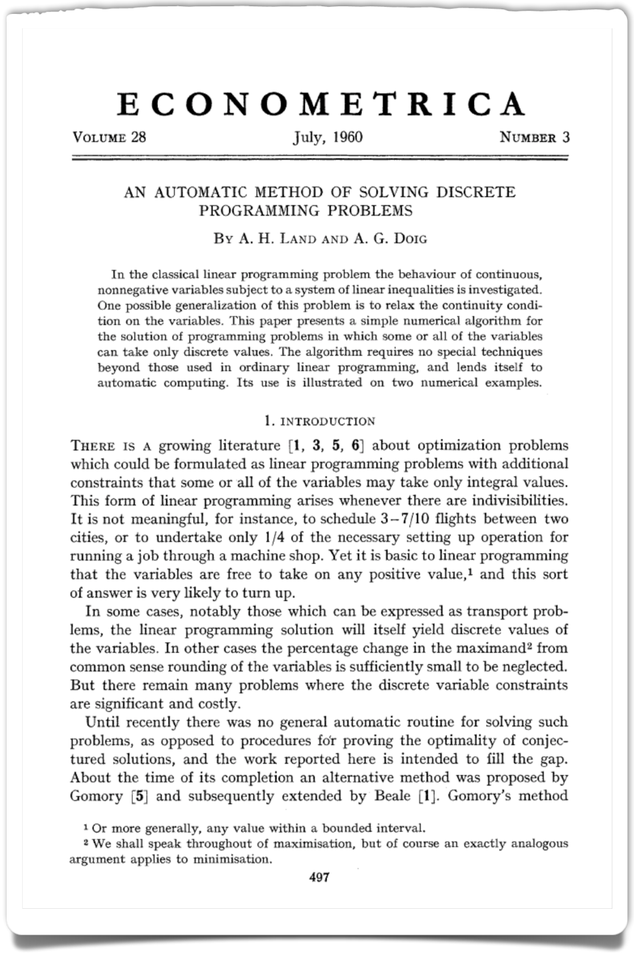
Until recently there was no general automatic routine for solving such problems, as opposed to procedures for proving the optimality of conjectured solutions, and the work reported here is intended to fill the gap. About the time of its completion an alternative method was proposed by Gomory [**5**]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



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

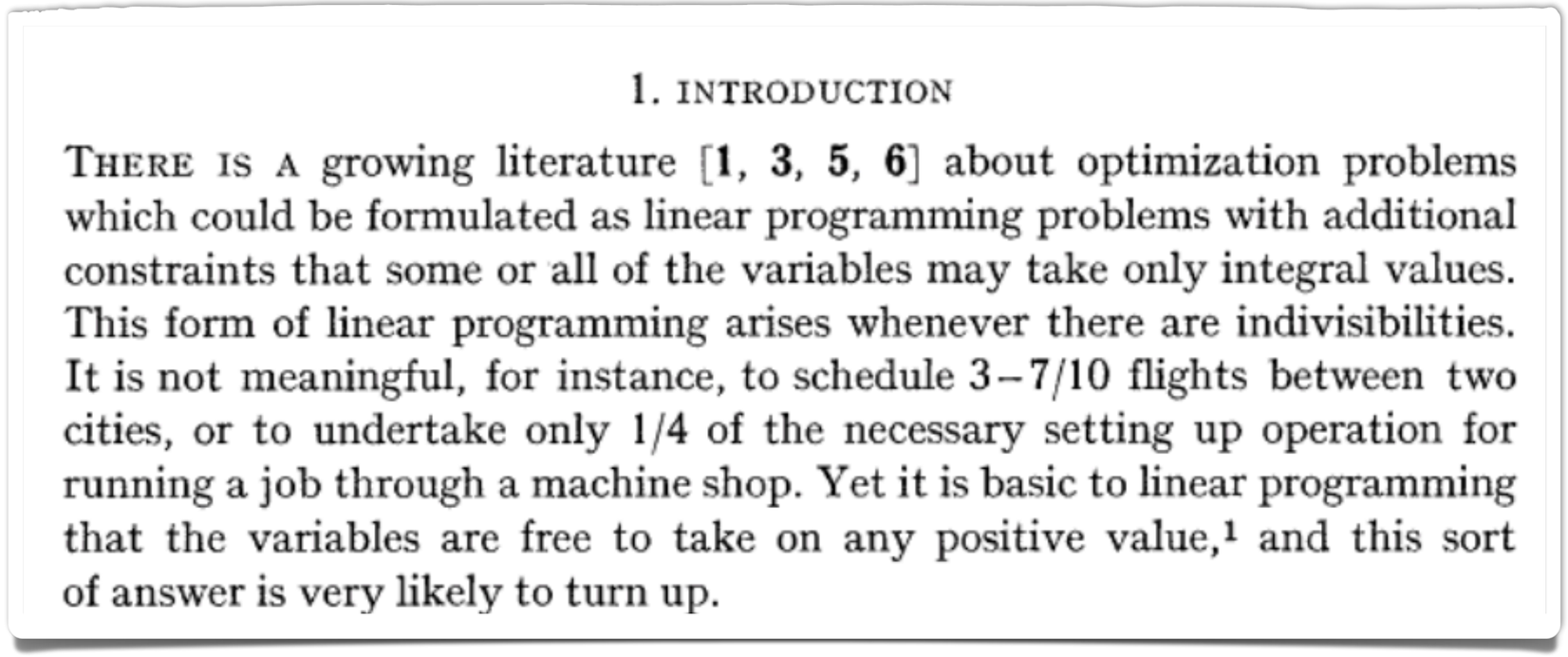
THERE IS A growing literature [1, 3, 5, 6] about optimization problems which could be formulated as linear programming problems with additional constraints that some or all of the variables may take only integral values. This form of linear programming arises whenever there are indivisibilities. It is 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.

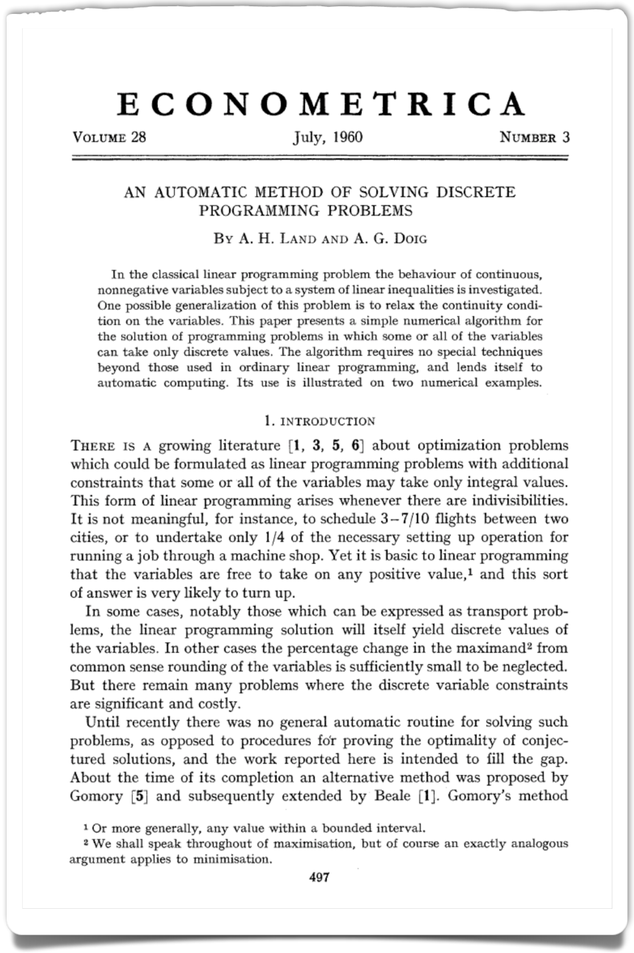
497



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



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.

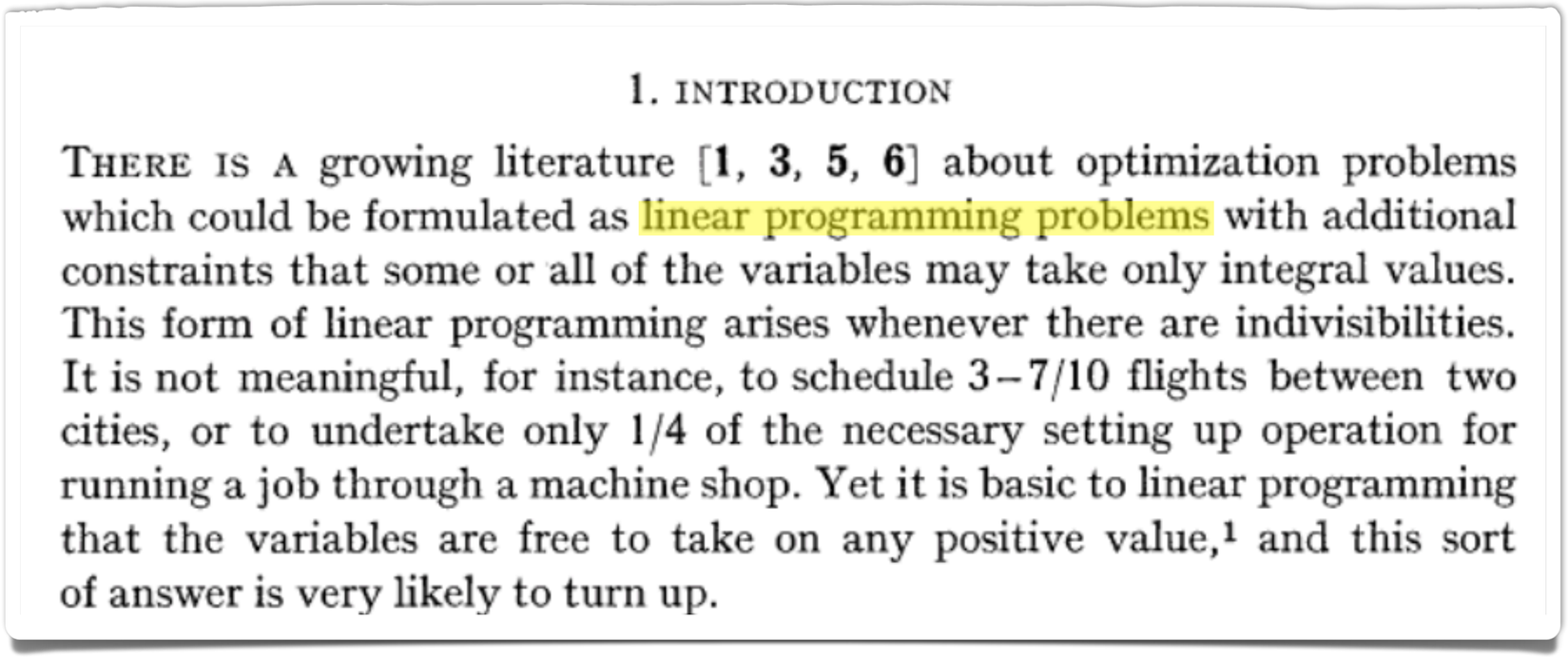
In some cases, notably those which can be expressed as transport problems, the linear programming solution will itself yield discrete values of the variables. In other cases the percentage change in the maximand[2] from common sense rounding of the variables is sufficiently small to be neglected. But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

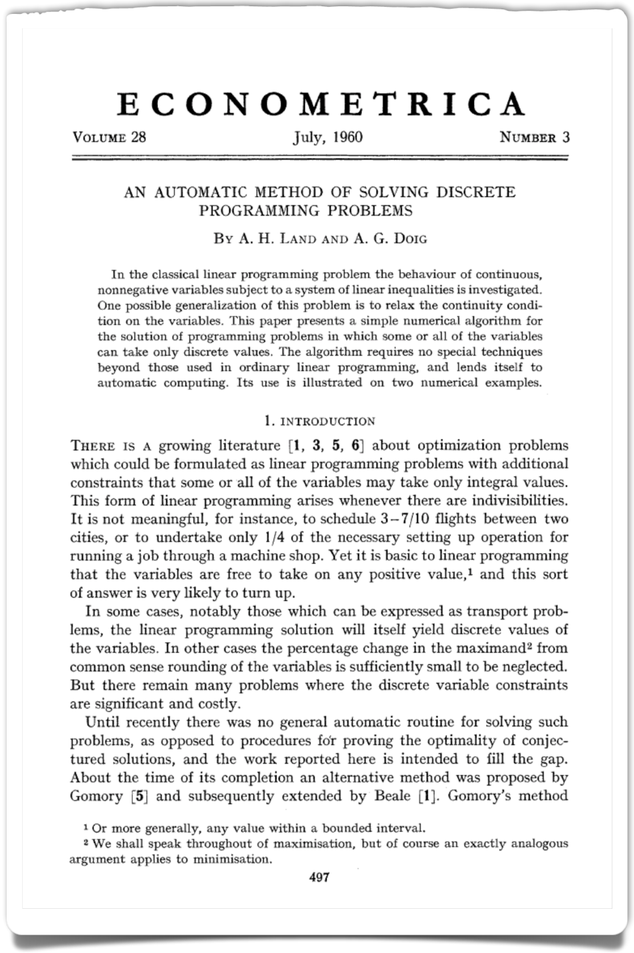
497



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



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

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

Knapsack-Probleme

0-1-Knapsack

Knapsack-Probleme

0-1-Knapsack

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

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

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

Knapsack-Probleme

0-1-Knapsack

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

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

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

Fraktionales Knapsack

Knapsack-Probleme

0-1-Knapsack

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

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

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

Fraktionales Knapsack

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

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

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

Knapsack-Probleme

0-1-Knapsack

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

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

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

Fraktionales Knapsack

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

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

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

Knapsack-Probleme

0-1-Knapsack

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

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

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

Fraktionales Knapsack

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

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

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

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

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

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

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

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

Ganzzahlige und lineare Optimierungsprobleme

Ganzzahliges Optimierungsproblem

Ganzzahlige und lineare Optimierungsprobleme

Ganzzahliges Optimierungsproblem

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

Ganzzahlige und lineare Optimierungsprobleme

Ganzzahliges Optimierungsproblem

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

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

Ganzzahlige und lineare Optimierungsprobleme

Ganzzahliges Optimierungsproblem

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

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

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

Ganzzahlige und lineare Optimierungsprobleme

Ganzzahliges Optimierungsproblem

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

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

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

Lineares Optimierungsproblem

Ganzzahlige und lineare Optimierungsprobleme

Ganzzahliges Optimierungsproblem

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

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

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

Lineares Optimierungsproblem

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

Ganzzahlige und lineare Optimierungsprobleme

Ganzzahliges Optimierungsproblem

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

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

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

Lineares Optimierungsprobleme

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

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

Ganzzahlige und lineare Optimierungsprobleme

Ganzzahliges Optimierungsproblem

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

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

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

Lineares Optimierungsproblem

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

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

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

Ganzzahlige und lineare Optimierungsprobleme

Ganzzahliges Optimierungsproblem

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

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

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

Lineares Optimierungsprobleme

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

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

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

Beweisbar schwer zu lösen

Ganzzahlige und lineare Optimierungsprobleme

Ganzzahliges Optimierungsproblem

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

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

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

Beweisbar schwer zu lösen

Lineares Optimierungsproblem

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

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

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

Beweisbar effizient lösbar

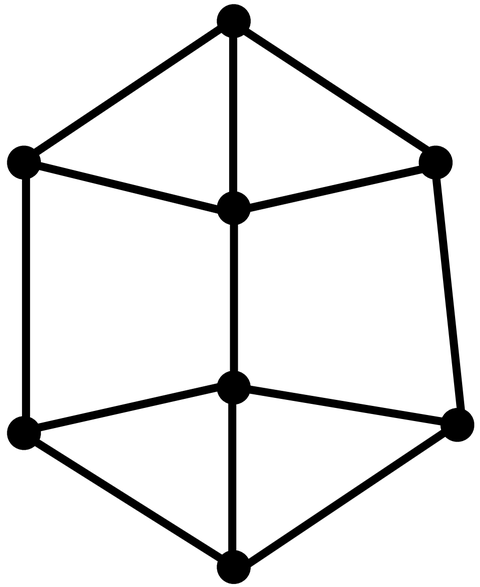
Ganzzahlige Optimierungsprobleme: Matching

Ganzzahlige Optimierungsprobleme: Matching

Gegeben:

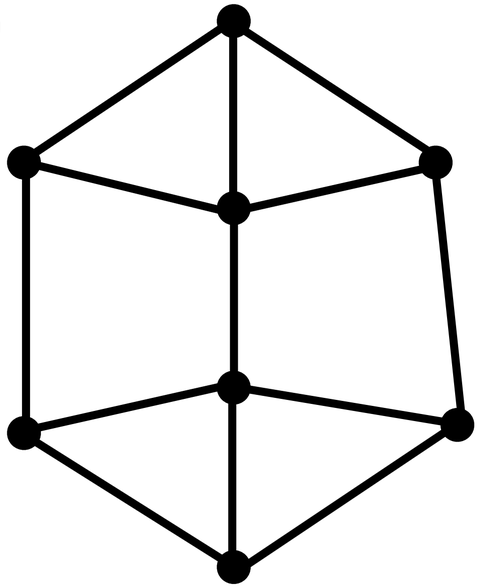
Ganzzahlige Optimierungsprobleme: Matching

Gegeben:



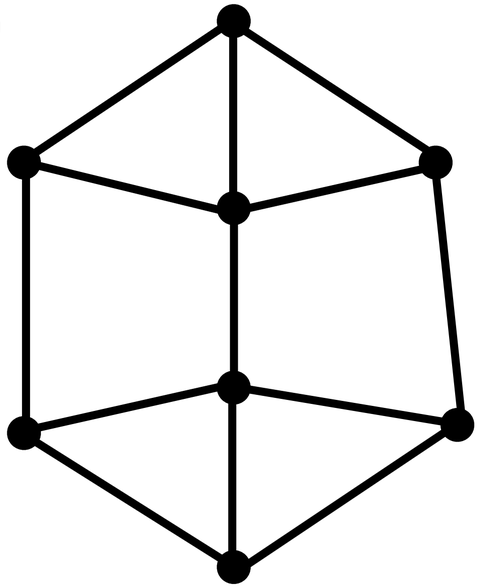
Ganzzahlige Optimierungsprobleme: Matching

Gegeben: Ein Graph G=(V,E)



Ganzzahlige Optimierungsprobleme: Matching

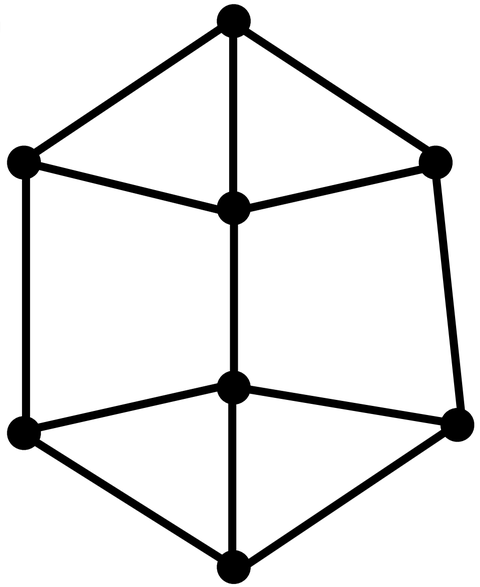
Gegeben: Ein Graph G=(V,E)



Gesucht:

Ganzzahlige Optimierungsprobleme: Matching

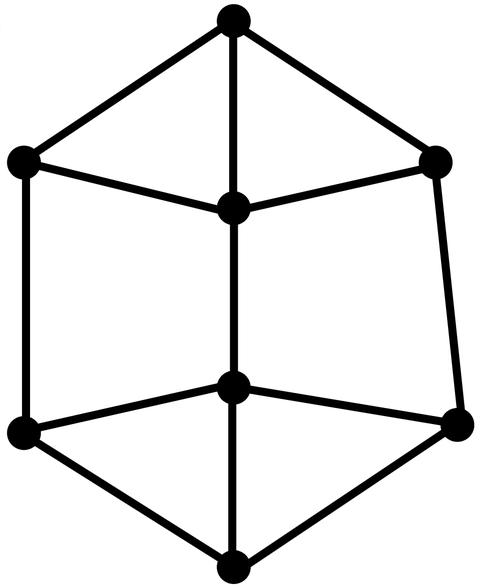
Gegeben: Ein Graph G=(V,E)



Gesucht: Ein größtmögliches **Matching** M in G:

Ganzzahlige Optimierungsprobleme: Matching

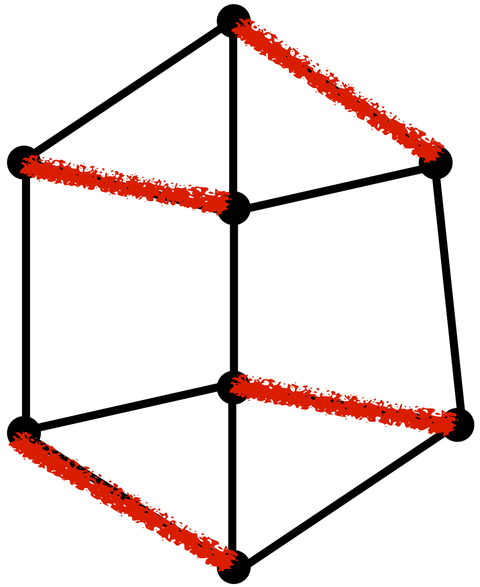
Gegeben: Ein Graph G=(V,E)



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

Ganzzahlige Optimierungsprobleme: Matching

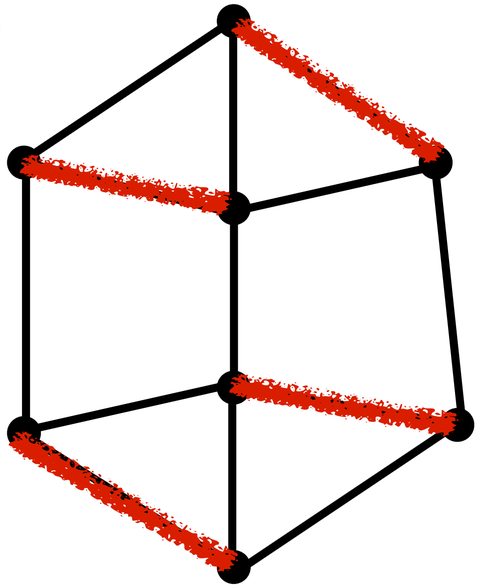
Gegeben: Ein Graph G=(V,E)



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

Ganzzahlige Optimierungsprobleme: Matching

Gegeben: Ein Graph G=(V,E)



$$\max \sum_{e \in E} x_e$$

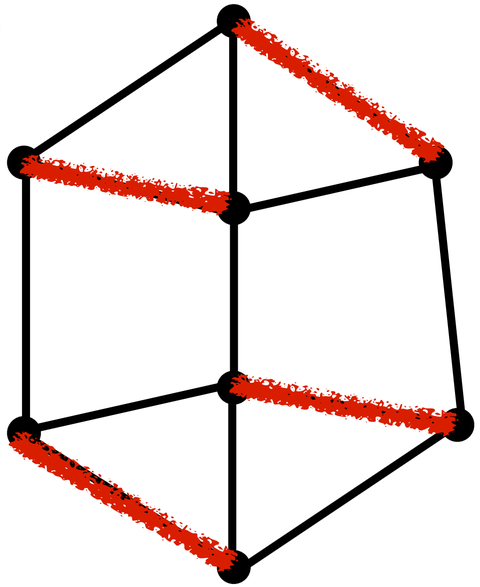
$$\forall v \in V : \sum_{e \in \delta(v)} x_e \leq 1$$

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

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

Ganzzahlige Optimierungsprobleme: Matching

Gegeben: Ein Graph G=(V,E)



$$\max \sum_{e \in E} x_e$$

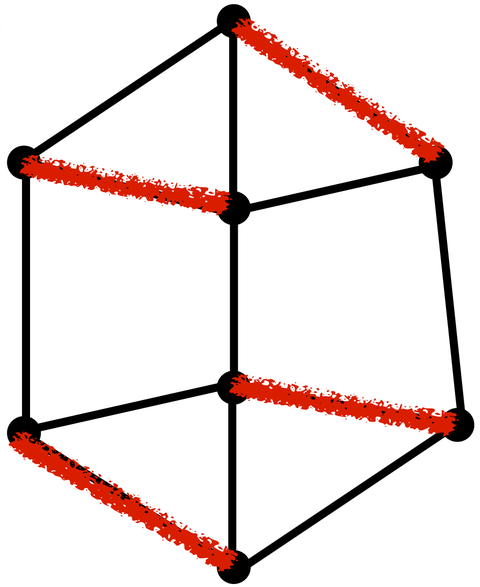
$$\forall v \in V : \sum_{e \in \delta(v)} x_e \leq 1$$

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

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

Ganzzahlige Optimierungsprobleme: Matching

Gegeben: Ein Graph G=(V,E)



$$\max \sum_{e \in E} x_e$$

$$\forall v \in V : \sum_{e \in \delta(v)} x_e \leq 1$$

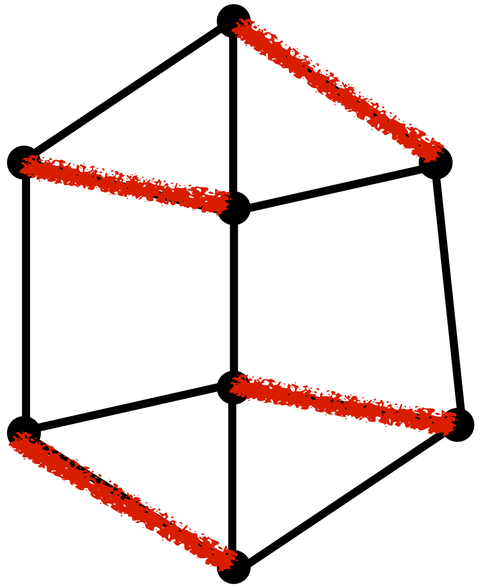
$$x_e \in \{0,1\}$$

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

Ganzzahlige Optimierungsprobleme: Matching

Gegeben: Ein Graph G=(V,E)

$x_e = 1$



$$\max \sum_{e \in E} x_e$$

$$\forall v \in V : \sum_{e \in \delta(v)} x_e \leq 1$$

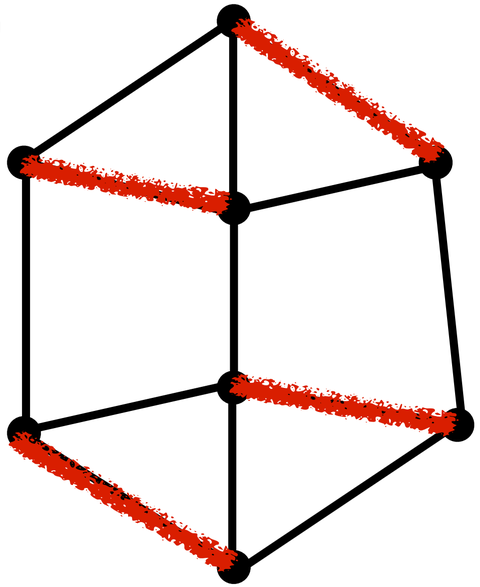
$$x_e \in \{0,1\}$$

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

Ganzzahlige Optimierungsprobleme: Matching

Gegeben: Ein Graph G=(V,E)

$x_e = 1$



$$\max \sum_{e \in E} x_e$$

$$\forall v \in V : \sum_{e \in \delta(v)} x_e \leq 1$$

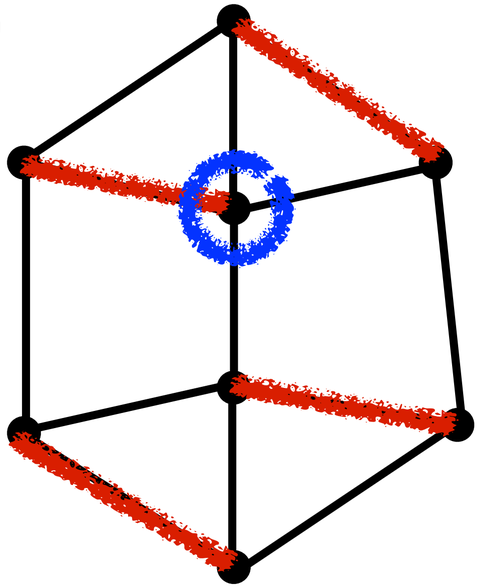
$$x_e \in \{0,1\}$$

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

Ganzzahlige Optimierungsprobleme: Matching

Gegeben: Ein Graph G=(V,E)

$x_e = 1$



$$\max \sum_{e \in E} x_e$$

$$\forall v \in V : \sum_{e \in \delta(v)} x_e \leq 1$$

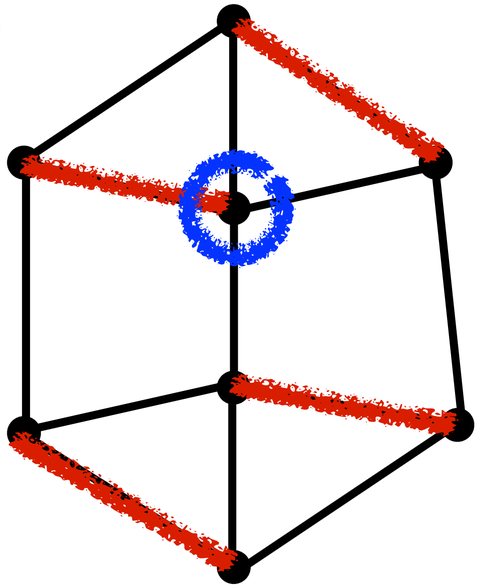
$$x_e \in \{0,1\}$$

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

Ganzzahlige Optimierungsprobleme: Matching

Gegeben: Ein Graph G=(V,E)

$x_e = 1$



$$\max \sum_{e \in E} x_e$$

$$\forall v \in V : \sum_{e \in \delta(v)} x_e \leq 1$$

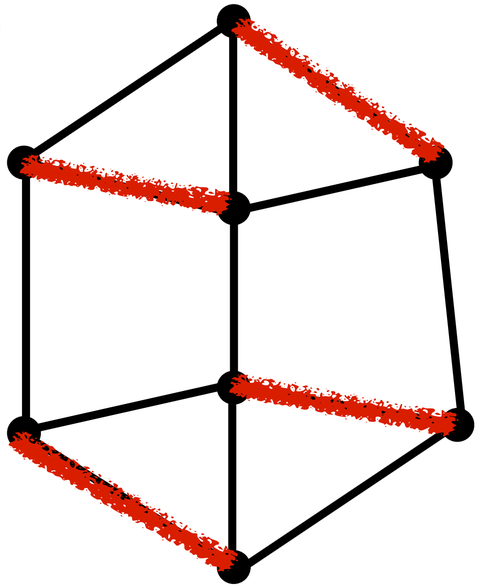
$$x_e \in \{0, 1\}$$

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

Ganzzahlige Optimierungsprobleme: Matching

Gegeben: Ein Graph G=(V,E)

$x_e = 1$



$$\max \sum_{e \in E} x_e$$

$$\forall v \in V : \sum_{e \in \delta(v)} x_e \leq 1$$

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

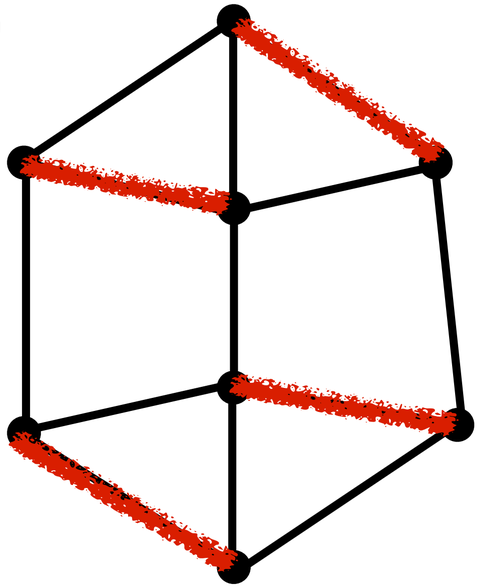
Gesucht: Ein größtmögliches **Matching** M in G:
eine möglichst große Menge disjunkter Kanten

Ganzzahlige Optimierungsprobleme: Matching

Gegeben: Ein Graph G=(V,E)

$x_e = 1$

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



$$\max \sum_{e \in E} x_e$$

$$\forall v \in V : \sum_{e \in \delta(v)} x_e \leq 1$$

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

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

Ganzzahlige Optimierungsprobleme: Vertex Cover

Ganzzahlige Optimierungsprobleme: Vertex Cover

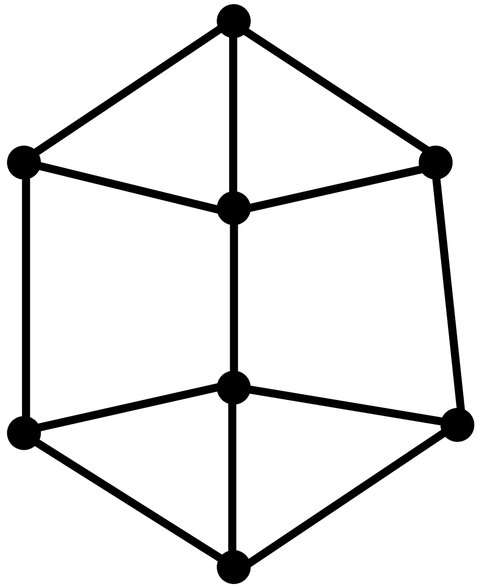
Gegeben:

Ganzzahlige Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph G=(V,E)

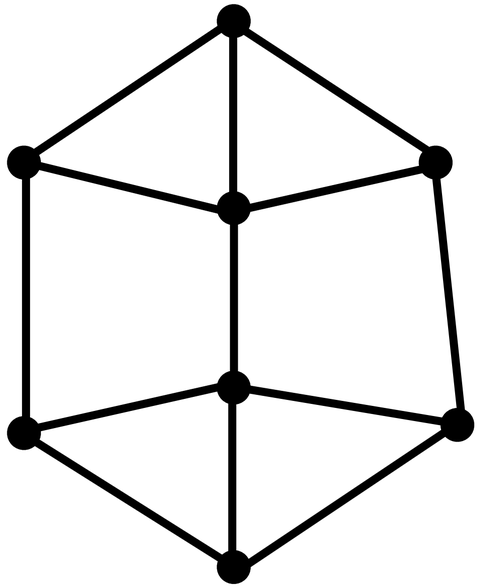
Ganzzahlige Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph G=(V,E)



Ganzzahlige Optimierungsprobleme: Vertex Cover

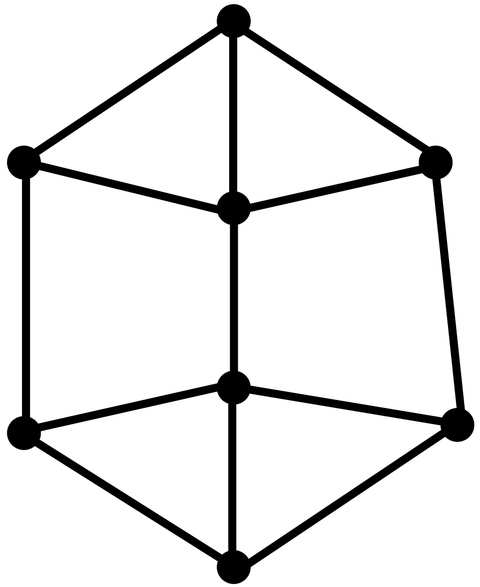
Gegeben: Ein Graph G=(V,E)



Gesucht:

Ganzzahlige Optimierungsprobleme: Vertex Cover

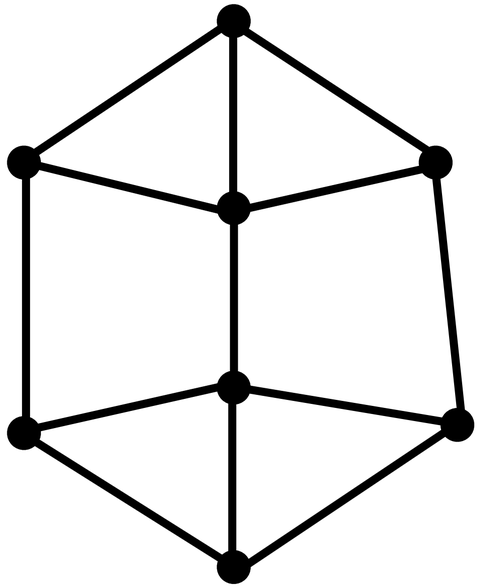
Gegeben: Ein Graph G=(V,E)



Gesucht: Ein kleinstmögliches **Vertex Cover** S in G:

Ganzzahlige Optimierungsprobleme: Vertex Cover

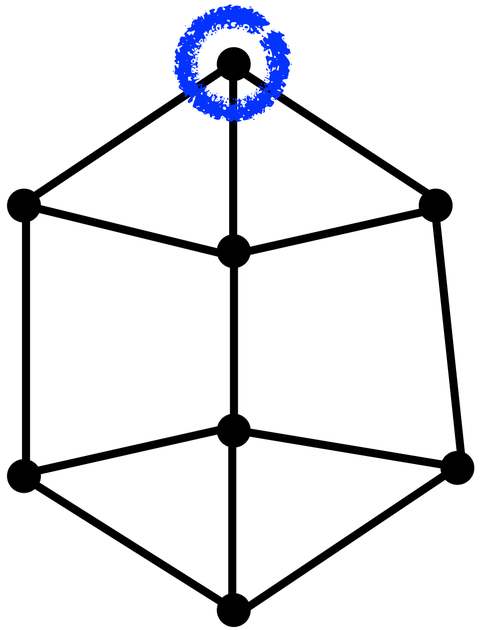
Gegeben: Ein Graph G=(V,E)



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

Ganzzahlige Optimierungsprobleme: Vertex Cover

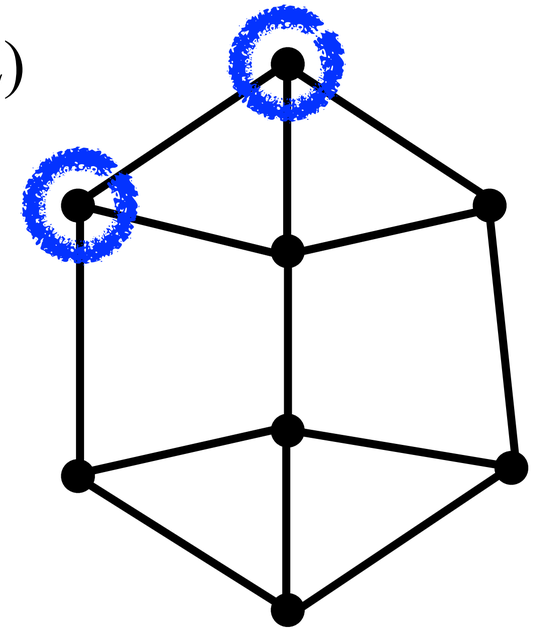
Gegeben: Ein Graph G=(V,E)



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

Ganzzahlige Optimierungsprobleme: Vertex Cover

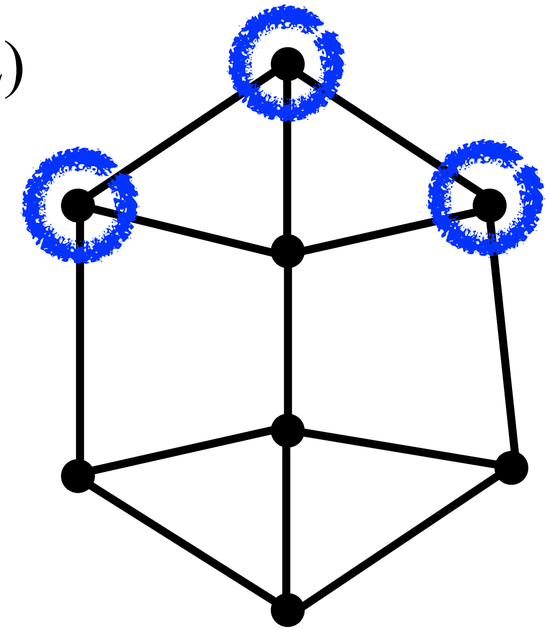
Gegeben: Ein Graph G=(V,E)



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

Ganzzahlige Optimierungsprobleme: Vertex Cover

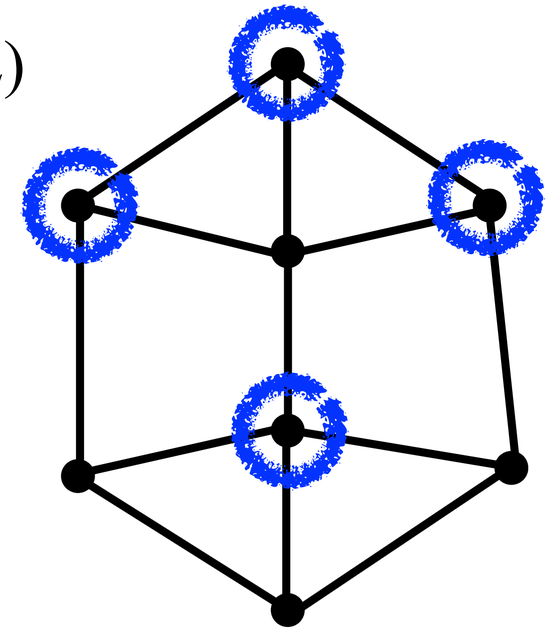
Gegeben: Ein Graph G=(V,E)



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

Ganzzahlige Optimierungsprobleme: Vertex Cover

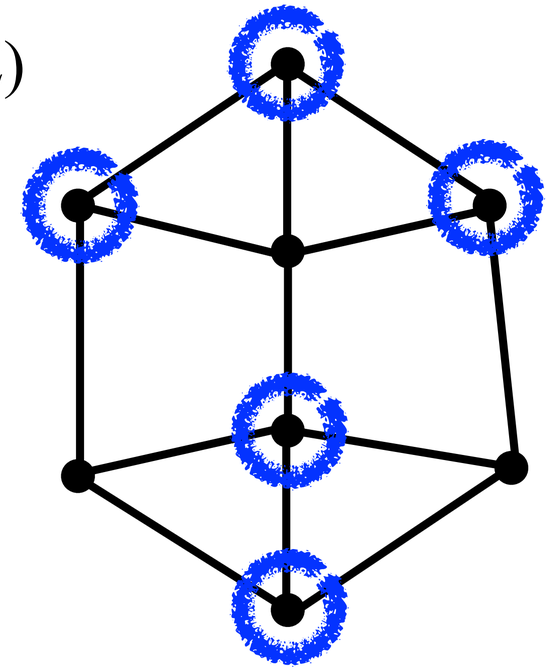
Gegeben: Ein Graph G=(V,E)



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

Ganzzahlige Optimierungsprobleme: Vertex Cover

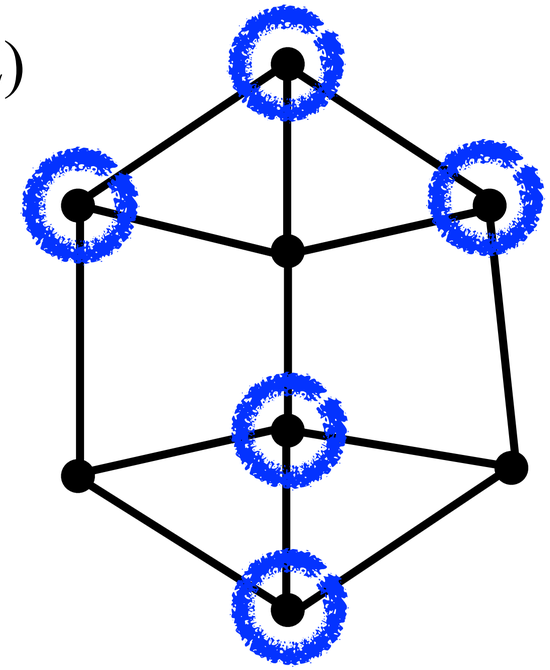
Gegeben: Ein Graph G=(V,E)



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)



$$\min \sum_{v \in V} y_e$$

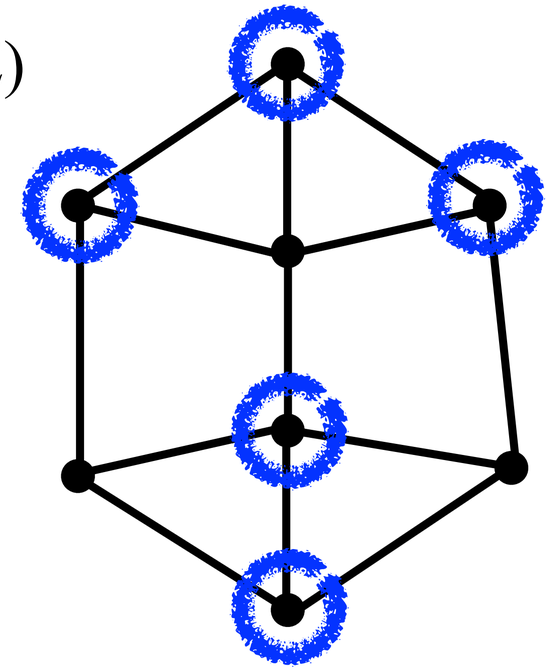
$$\forall e \in E : \sum_{e \in \delta(v)} y_v \geq 1$$

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

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

Ganzzahlige Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph G=(V,E)



$$\min \sum_{v \in V} y_e$$

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

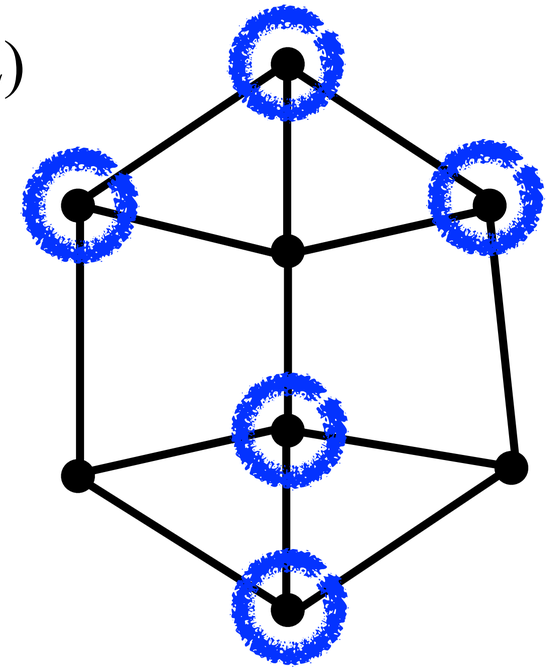
$$y_v \in \{0,1\}$$

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

Ganzzahlige Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph G=(V,E)

$y_v = 1$



$$\min \sum_{v \in V} y_e$$

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

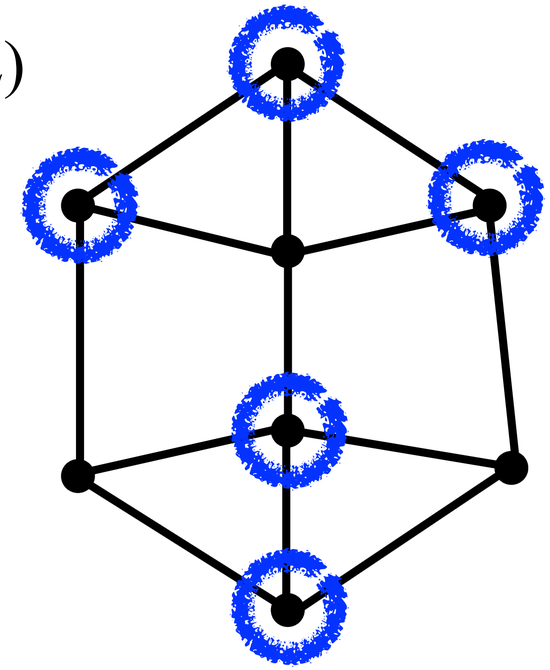
$$y_v \in \{0, 1\}$$

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

Ganzzahlige Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph G=(V,E)

$y_v = 1$



$$\min \sum_{v \in V} y_e$$

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

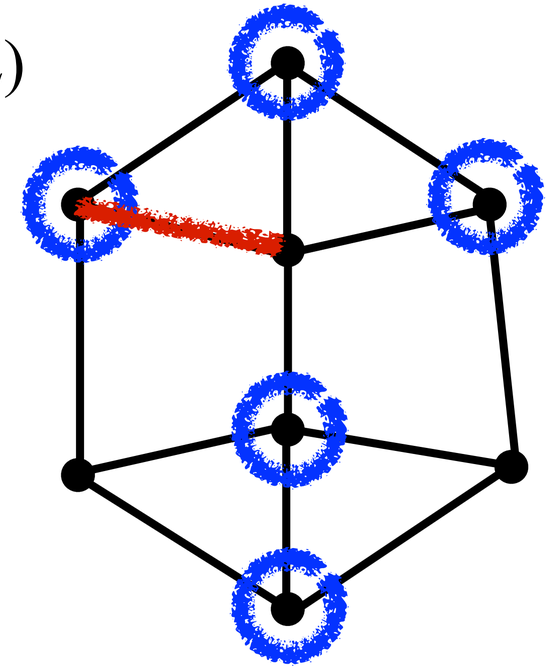
$$y_v \in \{0, 1\}$$

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

Ganzzahlige Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph G=(V,E)

$y_v = 1$



$$\min \sum_{v \in V} y_e$$

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

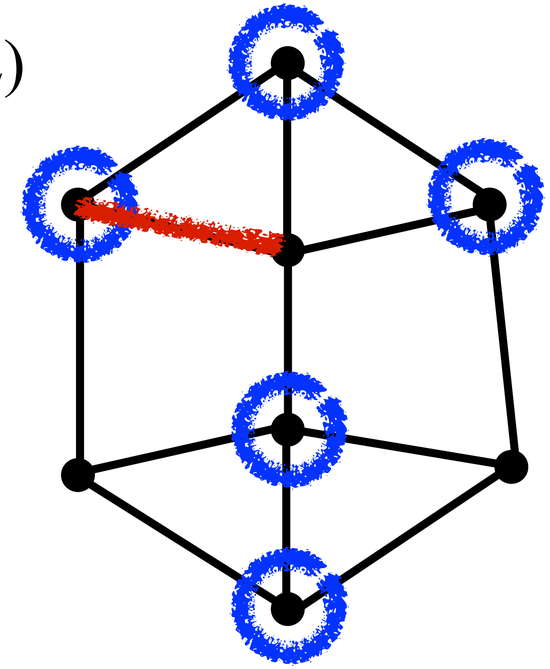
$$y_v \in \{0, 1\}$$

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

Ganzzahlige Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph G=(V,E)

$y_v = 1$



$$\min \sum_{v \in V} y_e$$

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

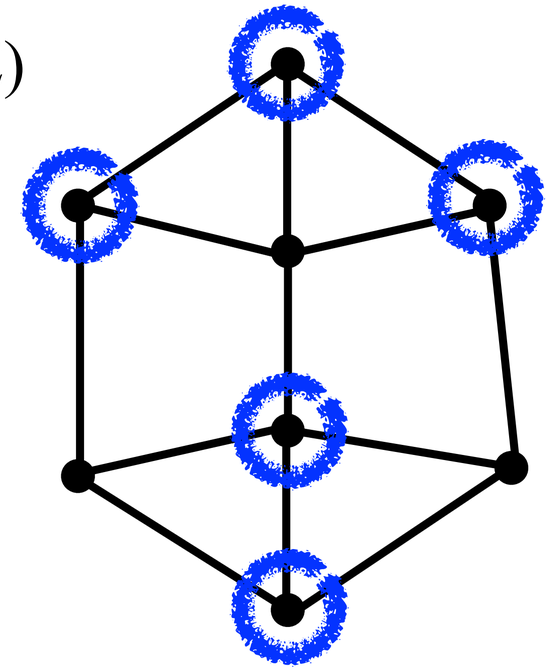
$$y_v \in \{0, 1\}$$

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

Ganzzahlige Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph G=(V,E)

$y_v = 1$



$$\min \sum_{v \in V} y_e$$

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

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

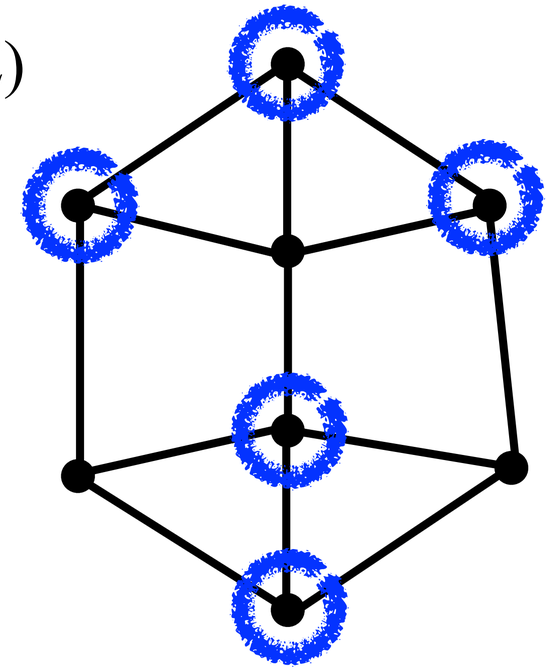
Gesucht: Ein kleinstmögliches **Vertex Cover** S in G:
eine möglichst kleine Menge von Knoten,
die alle Kanten überdecken

Ganzzahlige Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph G=(V,E)

$$y_v = 1$$

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



$$\min \sum_{v \in V} y_e$$

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

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

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

Lineare Optimierungsprobleme: Vertex Cover

Lineare Optimierungsprobleme: Vertex Cover

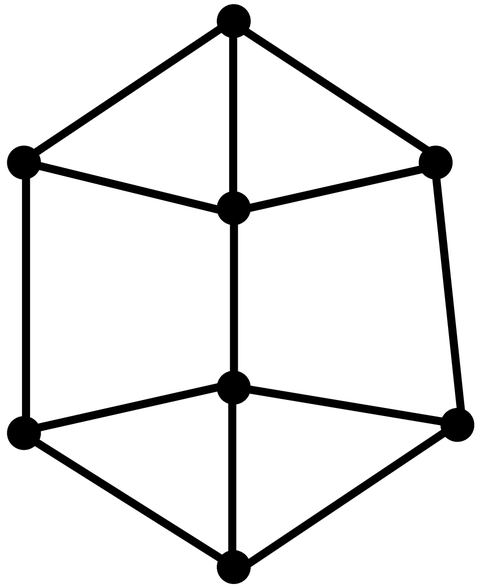
Gegeben:

Lineare Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph G=(V,E)

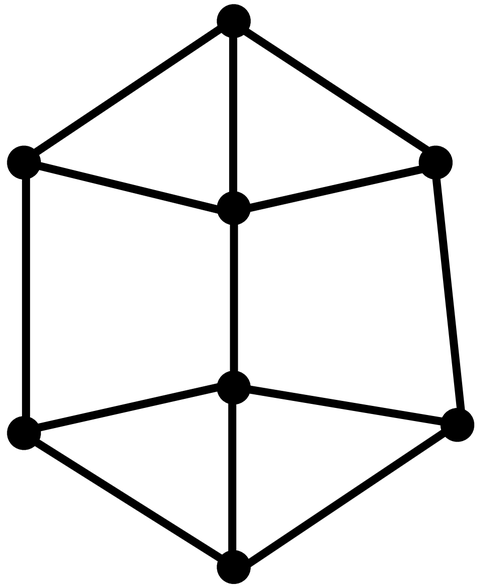
Lineare Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph G=(V,E)



Lineare Optimierungsprobleme: Vertex Cover

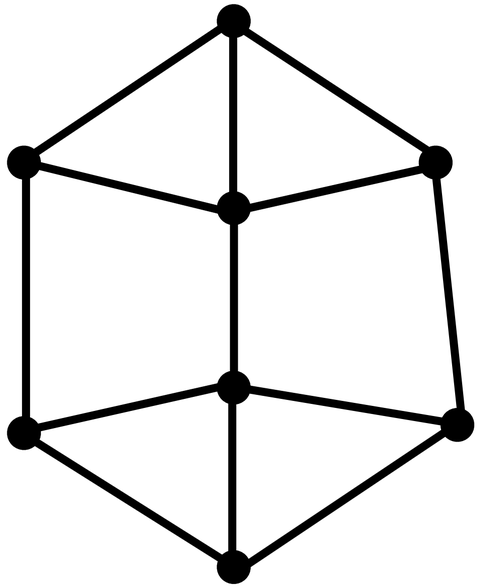
Gegeben: Ein Graph G=(V,E)



Gesucht:

Lineare Optimierungsprobleme: Vertex Cover

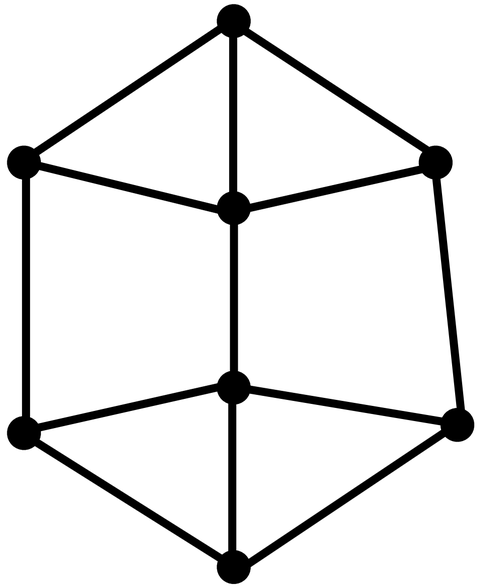
Gegeben: Ein Graph G=(V,E)



Gesucht: Ein kleinstmögliches **Vertex Cover** S in G:

Lineare Optimierungsprobleme: Vertex Cover

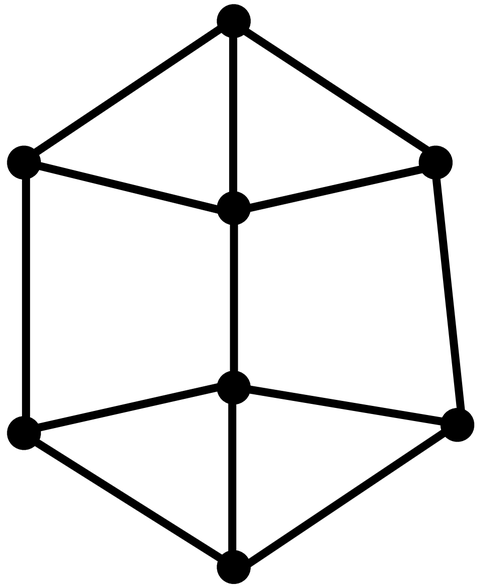
Gegeben: Ein Graph G=(V,E)



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)



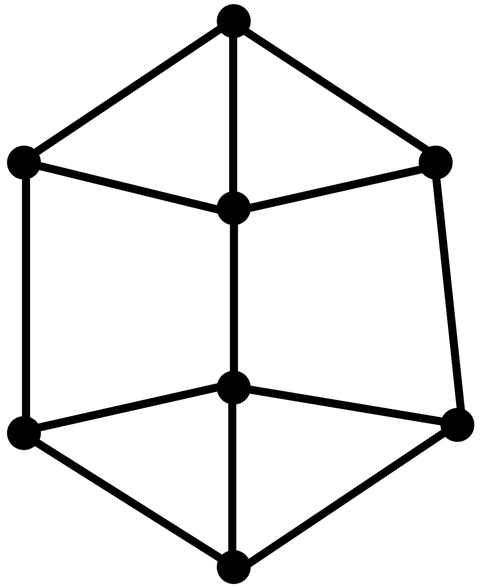
$$\min \sum_{v \in V} y_e$$

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

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)



$$\min \sum_{v \in V} y_e$$

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

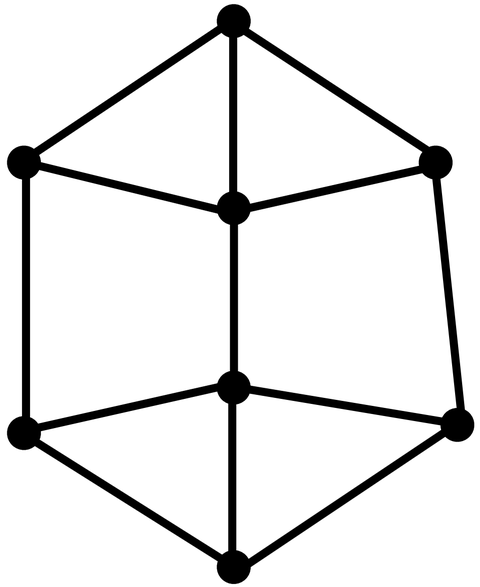
$$y_v \in [0,1]$$

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

Relaxierung

Lineare Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph G=(V,E)



$$\min \sum_{v \in V} y_e$$

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

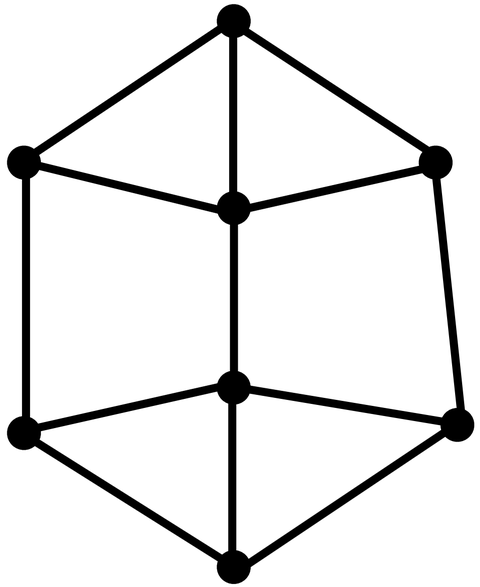
$$y_v \in [0,1]$$

Relaxierung

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)



$$\min \sum_{v \in V} y_e$$

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

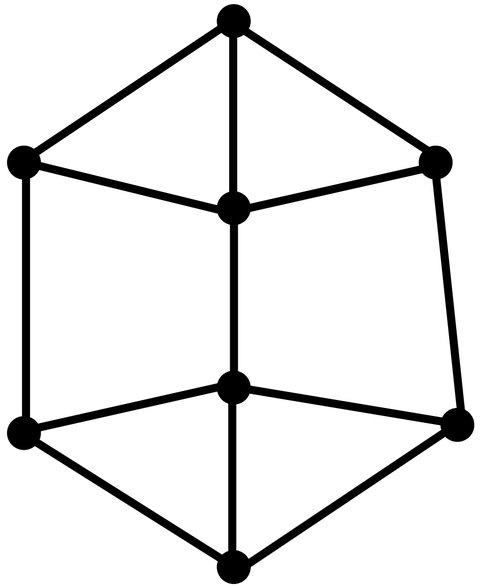
$$y_v \in [0,1]$$

Relaxierung

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)



$$\min \sum_{v \in V} y_e$$

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

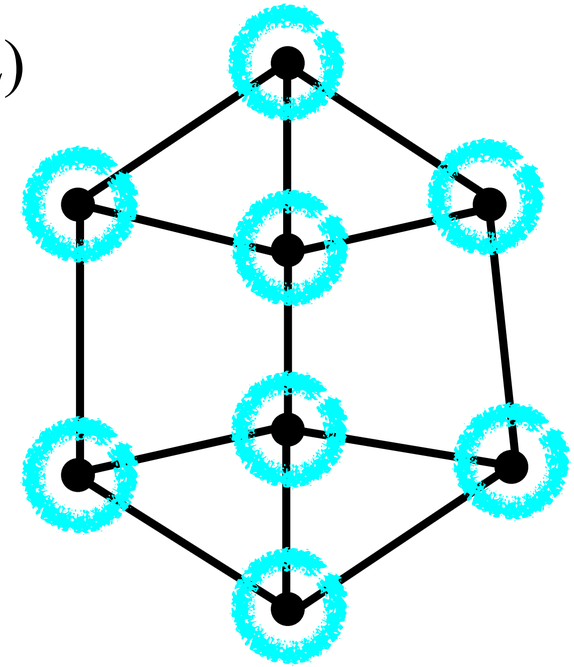
$$y_v \in [0,1]$$

Relaxierung

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)



$$\min \sum_{v \in V} y_e$$

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

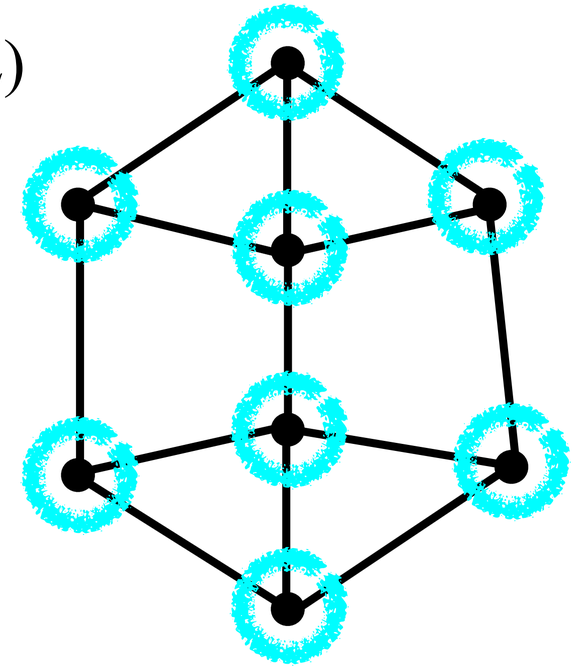
$$y_v \in [0,1]$$

Relaxierung

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)



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

$$\min \sum_{v \in V} y_e$$

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

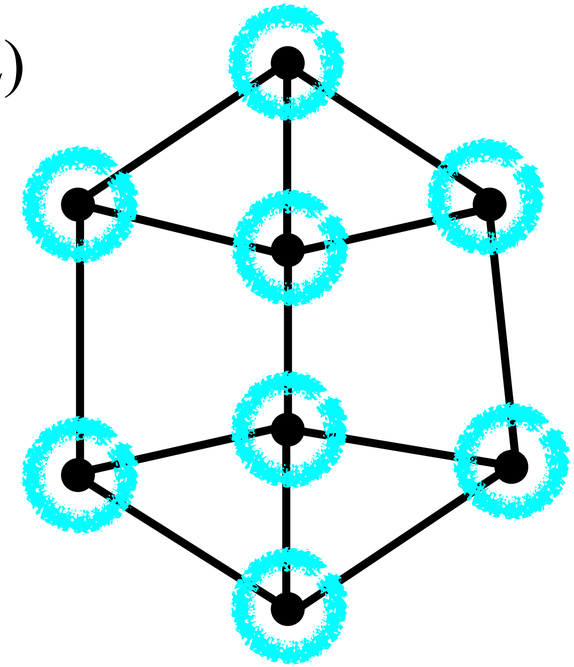
Relaxierung

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)

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



$$\forall e \in E: \quad \sum_{e \in \delta(v)} y_v \geq 1$$

$$\min \sum_{v \in V} y_e$$

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

Relaxierung

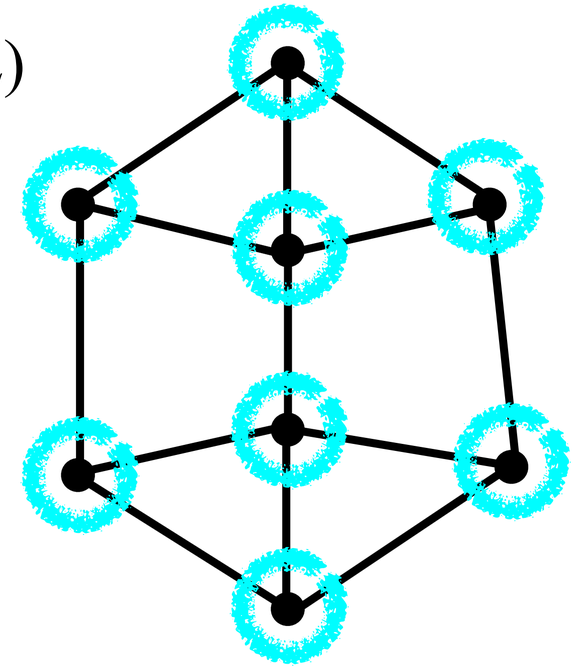
Gesucht: Ein kleinstmögliches **Vertex Cover** S in G:
eine möglichst kleine Menge von Knoten,
die alle Kanten überdecken

Lineare Optimierungsprobleme: Vertex Cover

Gegeben: Ein Graph G=(V,E)

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

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



$$\min \sum_{v \in V} y_e$$

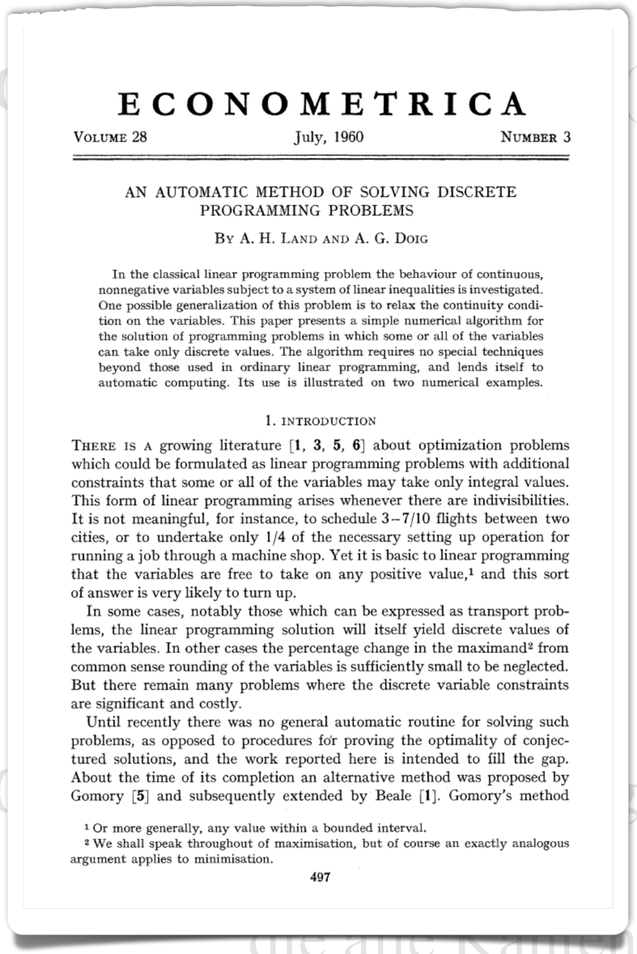
$$\forall e \in E : \sum_{e \in \delta(v)} y_v \geq 1$$

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

Relaxierung

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

Lineare Optimierungsprobleme: Vertex Cover



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

Ganzzahlig?!

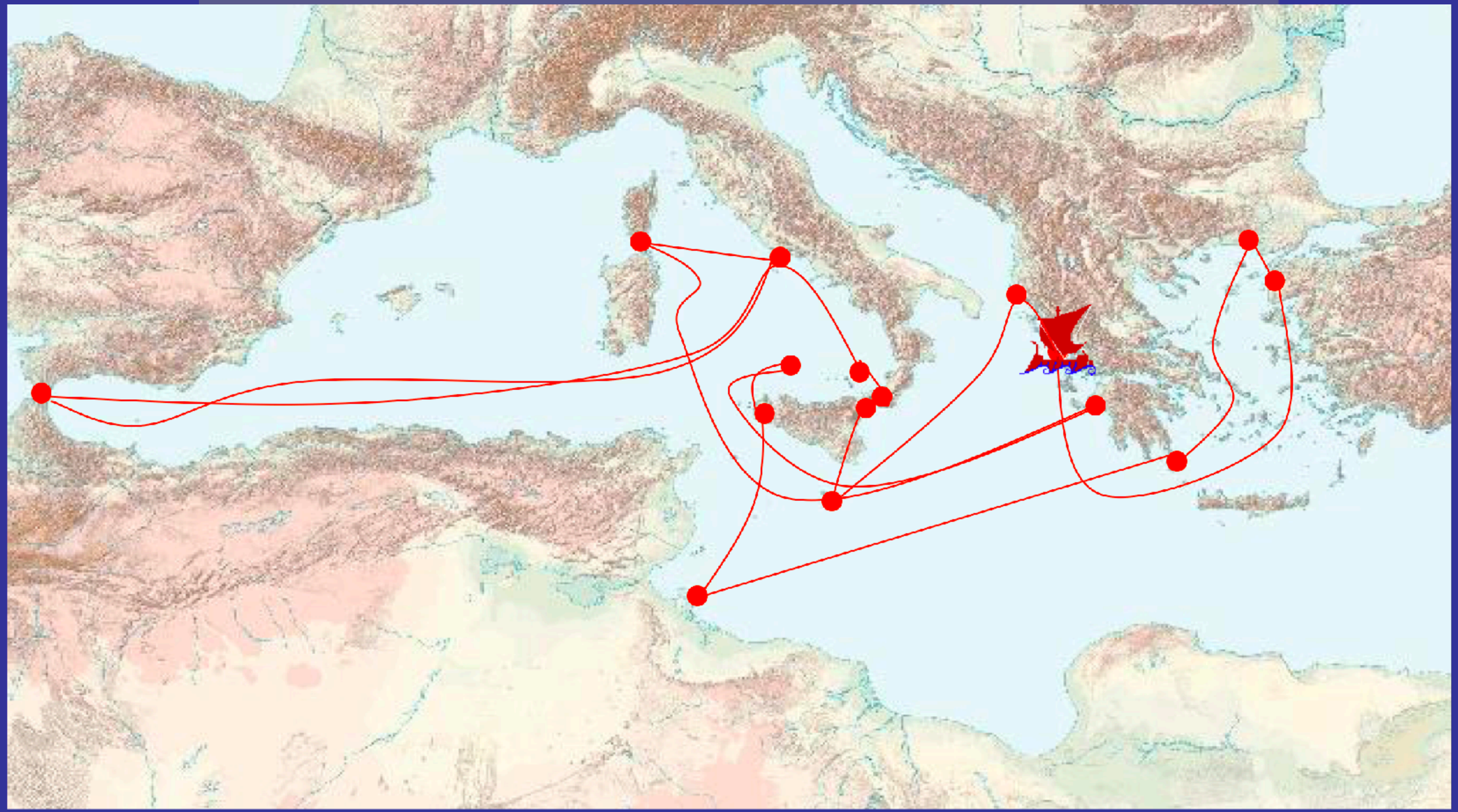
1. INTRODUCTION

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

Das Rundreiseproblem

Das Rundreiseproblem



Das Rundreiseproblem

Das Rundreiseproblem

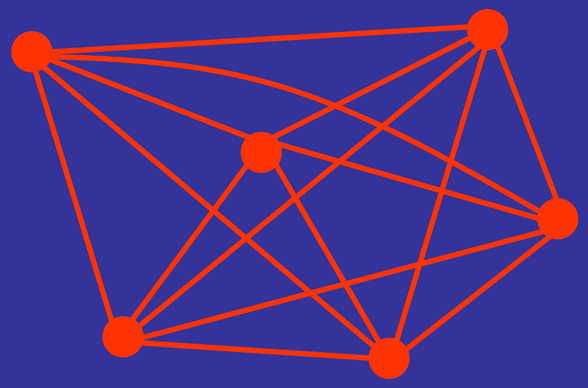
Gegeben:

Das Rundreiseproblem

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

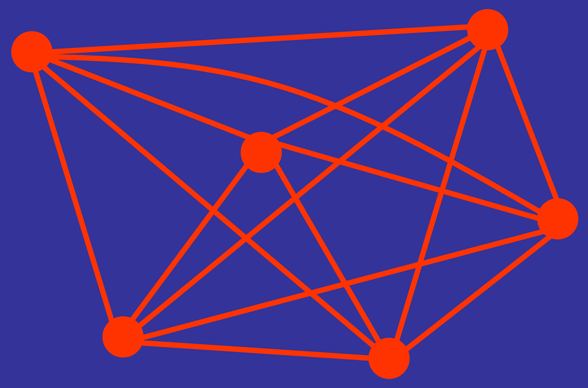
Das Rundreiseproblem

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



Das Rundreiseproblem

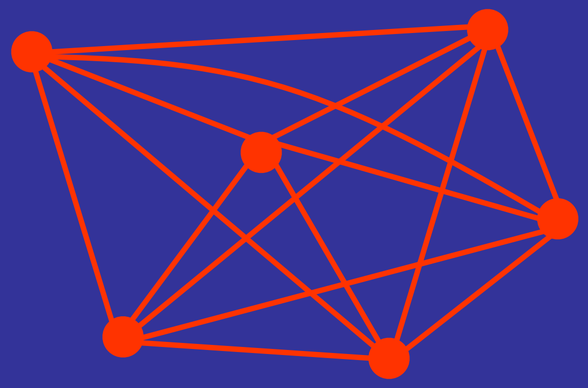
Gegeben: Ein Graph $G = (V, E)$
mit Kantenlängen w_e



Gesucht:

Das Rundreiseproblem

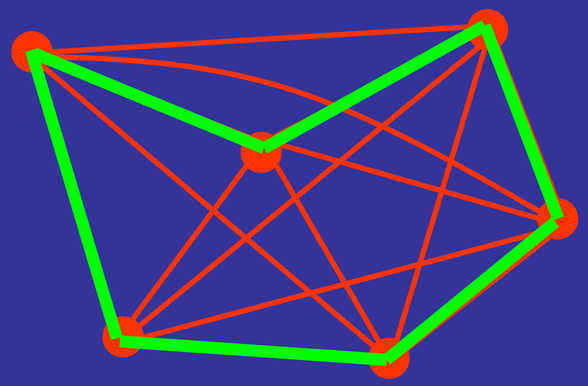
Gegeben: Ein Graph $G = (V, E)$
mit Kantenlängen w_e



Gesucht: Eine kürzeste Rundreise

Das Rundreiseproblem

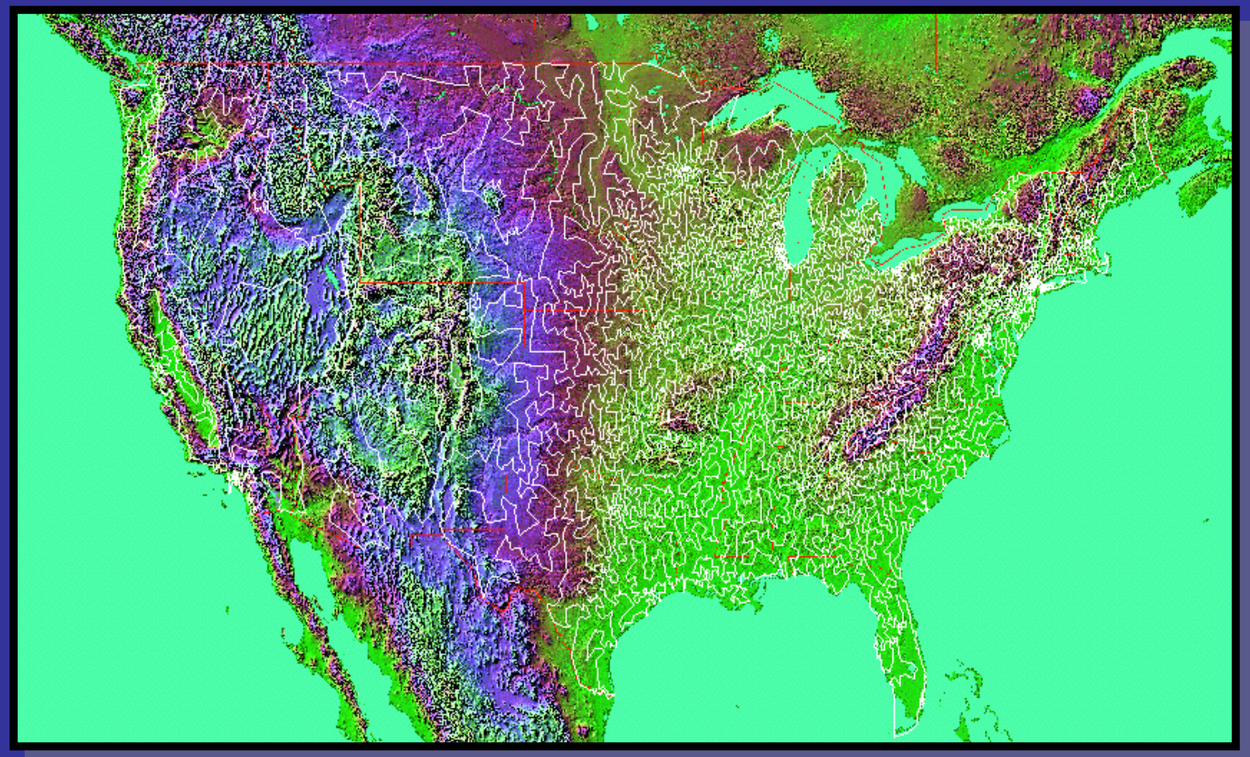
Gegeben: Ein Graph $G = (V, E)$
mit Kantenlängen w_e



Gesucht: Eine kürzeste Rundreise

Das Rundreiseproblem

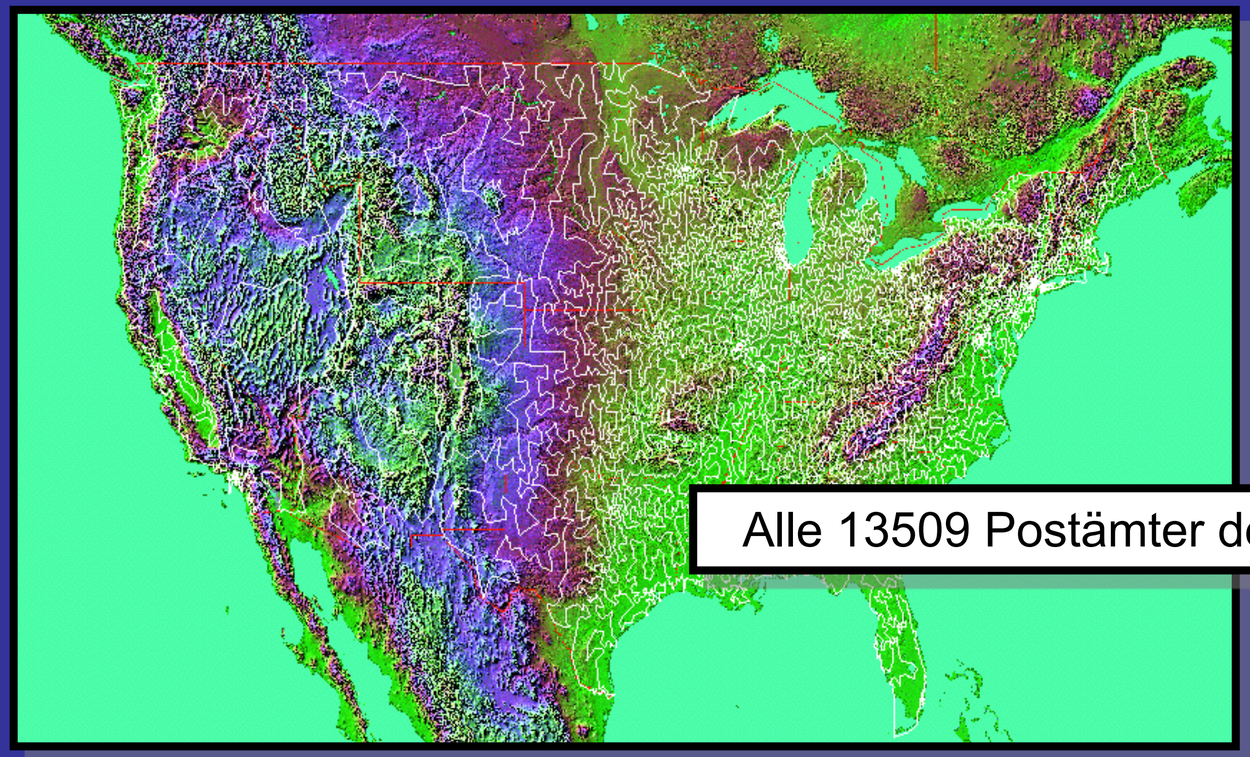
Gegeben: Ein Graph $G = (V, E)$
mit Kantenlängen w_e



Gesucht: Eine kürzeste Rundreise

Das Rundreiseproblem

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



Alle 13509 Postämter der USA

Gesucht: Eine kürzeste Rundreise

Vielen Dank!

s.fekete@tu-bs.de