Fachseminar «Optimierung», Universität Leipzig

Vergleich mathematischer Software zur Lösung von linearen und gemischt ganzzahligen Optimierungsproblemen

Vortrag am 20. Januar 2003 von Tobias Elze

1 Einleitung

1.1 Zielstellung

- XPress: teure Spezialsoftware mit geringer Verbreitung
  - im studentischen Arbeitsalltag i. d. R. nicht verfügbar
  - in der Praxis allenfalls in Firmen vorhanden, die häufig Optimierungsprobleme lösen müssen

- Wie gut können weit verbreitete oder frei verfügbare Softwarepakete mit XPress mithalten?

- Welche sonstigen Möglichkeiten zur Optimierung bieten andere Pakete?

1.2 Warum dieser Vergleich unfair ist

- Softwarepakete mit vollkommen verschiedenen Hintergründen und Anwendungsschwerpunkten

- XPress und lp_solve: Spezialsoftware für Optimierungsprobleme

- alle anderen Programme: Optimierung entweder nur einer von vielen Schwerpunkten oder sogar nur Nebenprodukt zur Lösung anderer Probleme

- dabei Spezialprogrammen Computeralgebrasysteme und Numerikpakete gegenübergestellt

  - Vergleich von Äpfeln mit Birnen und Pflaumen

- außerdem: XPress lief aus lizenzrechtlichen Gründen auf anderer Hardware (daher sind die Zeitmessungen nur tendenziell mit den übrigen Programmen vergleichbar)

1.3 Warum dieser Vergleich dennoch sinnvoll ist

- Vergleich versucht, die Stärken der verschiedenen Programme zu berücksichtigen

- Erwartungen gemäß des Selbstanspruchs der Programme:

- ☞ Spezialprogramme zur Optimierung → hohe Berechnungsgeschwindigkeit, guter Umgang mit großen Mengen an Variablen und Bedingungen

- ☞ Computeralgebrasysteme → geeignet zum Experimentieren und Hypothesengenerieren, also Bereitstellung komfortabler Funktionen, die leicht zu bedienen sind; dafür aber erheblich langsamer und nur für kleine Problemstellungen (wenige Variablen) ausgelegt

- ☞ Numerikpakete → schneller als Computeralgebrasysteme und für größere Problemstellungen ausgerichtet als diese, aber schlechtere Leistung als Spezialsoftware zur Optimierung

2 Der Vergleich

2.1 Welche Programme wurden einbezogen?

- Spezialsoftware zur Optimierung: XPress (Version auf misun105) und lp_solve 4.0 (beta)

- Computeralgebrasysteme: Maple 7.0, Mathematica 4.2, MuPAD 2.5.2, (Reduce 3.7: disqualifiziert wegen ständiger Abstürze)

- Numerikpakete: Matlab R12,

- R 1.6.0, (Octave (Release 2002-09-11a): disqualifiziert wegen ständiger Abstürze)

- alle diese Programme sind entweder auf für Studenten frei zugänglichen Rechnern in der Universität installiert (wenn auch teilweise in anderen Versionen) und/oder frei verfügbar

- XPress konnte aus lizenzrechtlichen Gründen nur auf misun105 unter Solaris ausgeführt werden

- alle anderen Programme liefen auf einem Athlon XP 1800+ unter Windows 2000

2.2 Nach welchen Kriterien wurde verglichen?

- *quantitativ* wurden die CPU-Zeiten für verschieden große Problemstellungen gemessen

- *qualitativ* wurde der jeweilige Quelltext zur Umsetzung des Problems ausgewertet:

  - War der Quelltext einfach erstellbar?

  - Ließe sich eine Funktion (Routine) für die ausgewählte Problemstellung schreiben?

- ☞ Wie kompliziert sind leichte Änderungen in der Problemformulierung in existierenden Quelltext zu übernehmen?
- ☞ Wird die Eingabe des Problems als System von Gleichungen / Ungleichungen unterstützt (für kleine Probleme leicht zu realisieren, vermutlich Computeralgebrasysteme) vs. die Eingabe in Matrixform (geeignet zum Einlesen größerer Datenmengen, vermutlich Numerikpakete)?

3 Mathematische Problemstellung

- Als mathematisches Problem wurde das im letzten Vortrag vorgestellte *Facility Location Problem* (Standortproblem) ausgewählt.

- exemplarisch für ein gemischt ganzzahliges lineares Problem (mixed integer linear problem, MILP)

- verschiedene Herangehensweisen möglich, z. B.:

  - kombinatorisch, d. h. Ganzzahligkeit (hier: Binarität) von vornherein angenommen, geschickte Enumeration gesucht, oder

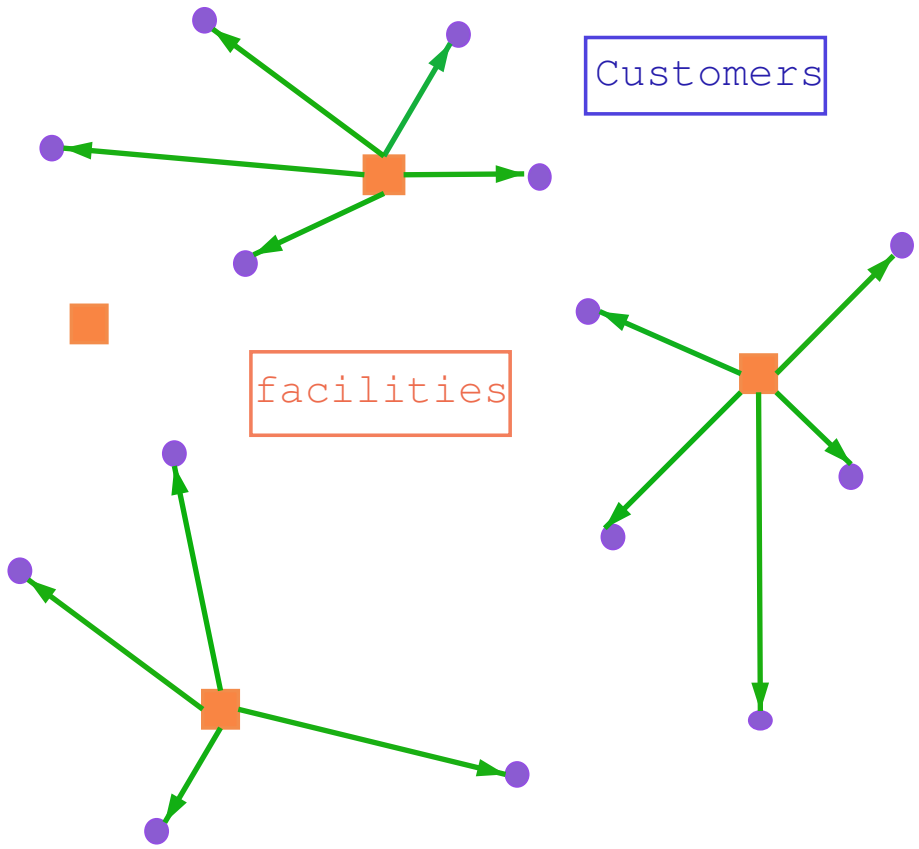
  - zuerst zu Grunde liegendes lineares Problem lösen (relaxierte Lösung), danach z. B. durch *Branch and Bound* Ganzzahligkeit herstellen

- Hier: zweite Möglichkeit gewählt

- Somit lassen sich zwei Teillösungen getrennt betrachten: das lineare Problem und der Verzweigungsalgorithmus (Branch and Bound)

3.1 Wiederholung: Facility Location Problem

⇛ Problemstellung: Eine Anzahl von *facilities* (Bedienungseinrichtungen) soll eine Anzahl von *customers* (Kunden) bedienen:


Customers
facilities

- Facility Location Problem: Kosten minimieren unter den Fragestellungen:
  - Welche Bedienungseinrichtungen werden aktiv?
  - Welche Einrichtung bedient welchen Kunden, so dass jeder Kunde von genau einer Einrichtung bedient wird?

Gegeben:

- d_j: Bedarf von Kunde j,
- $r_{i,j}$: Kosten, um Bedarf von Kunde j durch Einrichtung i zu decken,
- f_i: Fixkosten (Betriebskosten) der Einrichtung i,
- c_1: minimale Auslastung von Einrichtung i (damit deren Betrieb sich lohnt),
- c_2: maximale Kapazität von Einrichtung i.

Wir führen ein:

Entscheidungsvektor y:

$$y_i \triangleq \begin{cases} 1, & \text{falls facility i geöffnet ist} \\ 0, & \text{sonst} \end{cases}$$

$$\forall i = 1, .., n_f$$

Binäre Matrix x:

$$x_{i,j} \triangleq \begin{cases} 1, & \text{falls facility i Kunden j bedient,} \\ 0, & \text{sonst} \end{cases}$$

$$\forall i = 1, .., n_f, \forall j = 1, .., m_z$$

Somit können wir ein Facility Location Problem als *integer linear program* formulieren:

$$\min \sum_{i=1}^{n_f} \sum_{j=1}^{m_z} r_{i,j} d_j x_{i,j} + \sum_{i=1}^{n_f} f_i y_i \quad \text{Gesamtkosten}$$

s. t.

$$\sum_i x_{i,j} = 1, \quad \forall j \quad \text{1 Kunde} \rightarrow \text{1 Facility}$$

$$\sum_j d_j x_{i,j} \geq c_1 y_i, \forall i \quad \text{minimale Auslastung}$$

$$\sum_j d_j x_{i,j} \leq c_2 y_i, \forall i \quad \text{maximale Kapazität}$$

3.2 Teillösung 1: Lineare Programmierung (relaxierte Lösung)

- obige Problemstellung ohne die Ganzzahligkeitsbedingung $\Rightarrow$ x und y können nichtganzzahlige Elemente erhalten

- Betrachtung dieser Teillösung ist fruchtbar für unseren Vergleich, denn *die meisten der hier betrachteten Programme können lediglich Teillösung 1 berechnen*

- Anwendung eines Algorithmus der Linearen Programmierung (i. d. R. eine Form des *Simplex-Algorithmus*) (XPress bietet hier drei verschiedene Algorithmen zur Auswahl)

Betrachten wir dazu das folgende Beispiel (entnommen aus dem vorangegangenen Vortrag):

$$R = \begin{pmatrix} 0.7 & 0.8 & 1.3 & 2.2 \\ 1.3 & 1.1 & 0.8 & 0.8 \end{pmatrix}$$

$$d = (50, 325, 475, 200)$$

$$f = (1600, 2000)$$

$$c = (400, 600)$$

Die Lösung des zugehörigen linearen Problems (*linear relaxation*) lautet:

⇛ minimale Kosten: 4005.83

$$X = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$y = (1.41667, 0.333333)$$

Offensichtlich ist y nicht ganzzahlig – folglich gehen wir über zu Teillösung 2.

3.3 Teillösung 2: Branch and Bound

- zur Herstellung der Ganzzahligkeit: verschiedene Möglichkeiten (vgl. entsprechender Vortrag)

- hier: Branch and Bound ausgewählt (alle Programme, die Ganzzahligkeit unterstützen, geben an, mit Branch and Bound zu arbeiten)

- Was ist Branch and Bound?
  - kein Algorithmus, sondern eine Verfahrensweise → kann unterschiedlich algorithmisch implementiert werden
  - Idee: Verzweigung → Problem wird in Teilprobleme unterteilt, auf diese Teilprobleme wird rekursiv der Algorithmus angwewandt

- bei Minimierung arbeiten wir mit *unteren Schranken*
  - globale untere Schranke sind die Kosten der relaxierten Lösung (offensichtlich kann keine Integer-Lösung besser werden als diese)

Mögliche Implementierung eines Branch and Bound Algorithmus für unsere Problemstellung:

- Sei z der jeweilige Funktionswert der Zielfunktion.

- Unterprobleme: jeweils ein Variablenwert, der noch nicht binär ist, wird 0 bzw. 1 gesetzt (zusätzlicher Constraint beim LP), solange, bis alle Werte binär sind

- Auswahl des Unterproblems:

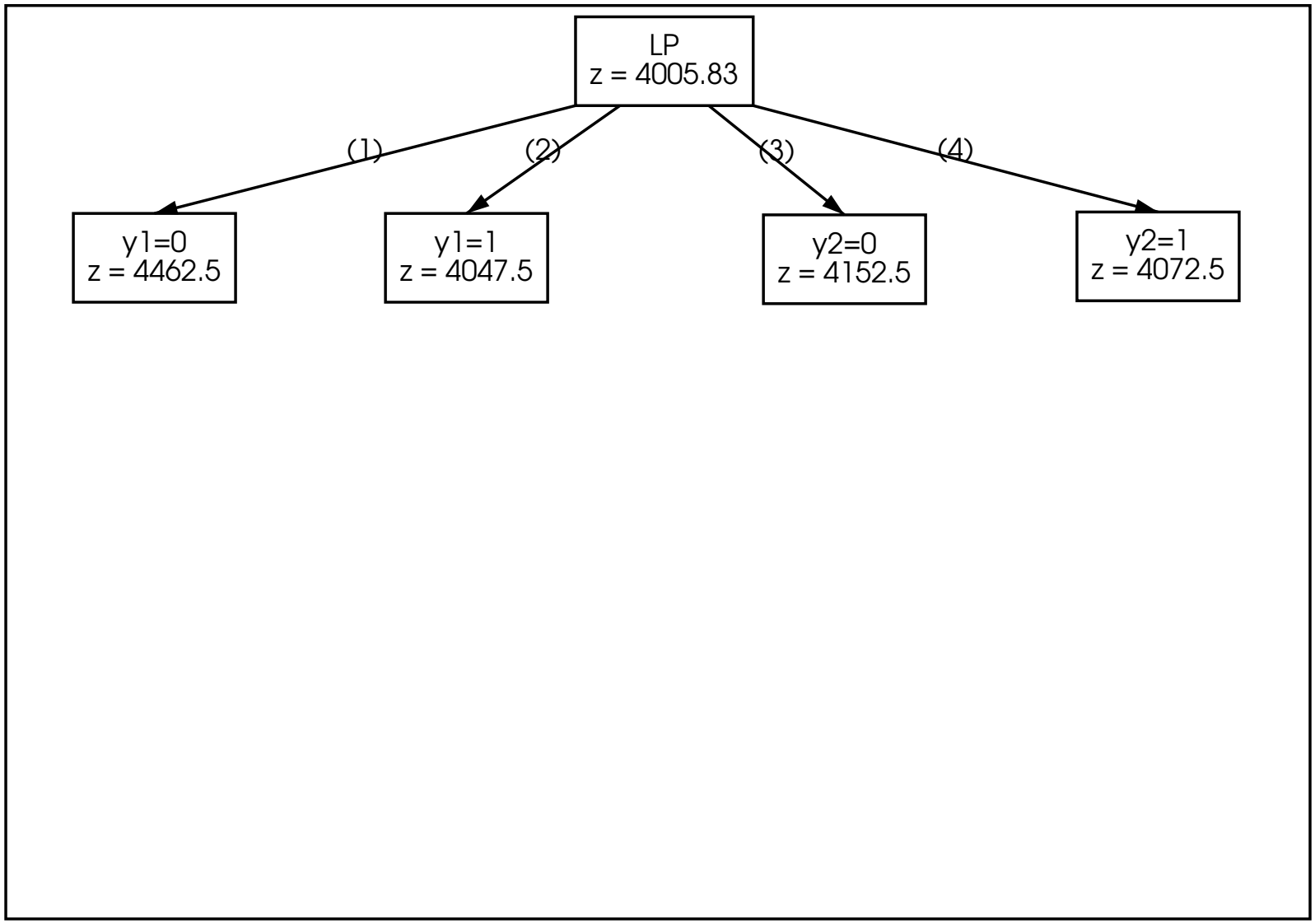
  - Auswahl eines *aktiven* Unterproblems (also eines, das zuvor noch nicht ausgewählt wurde)

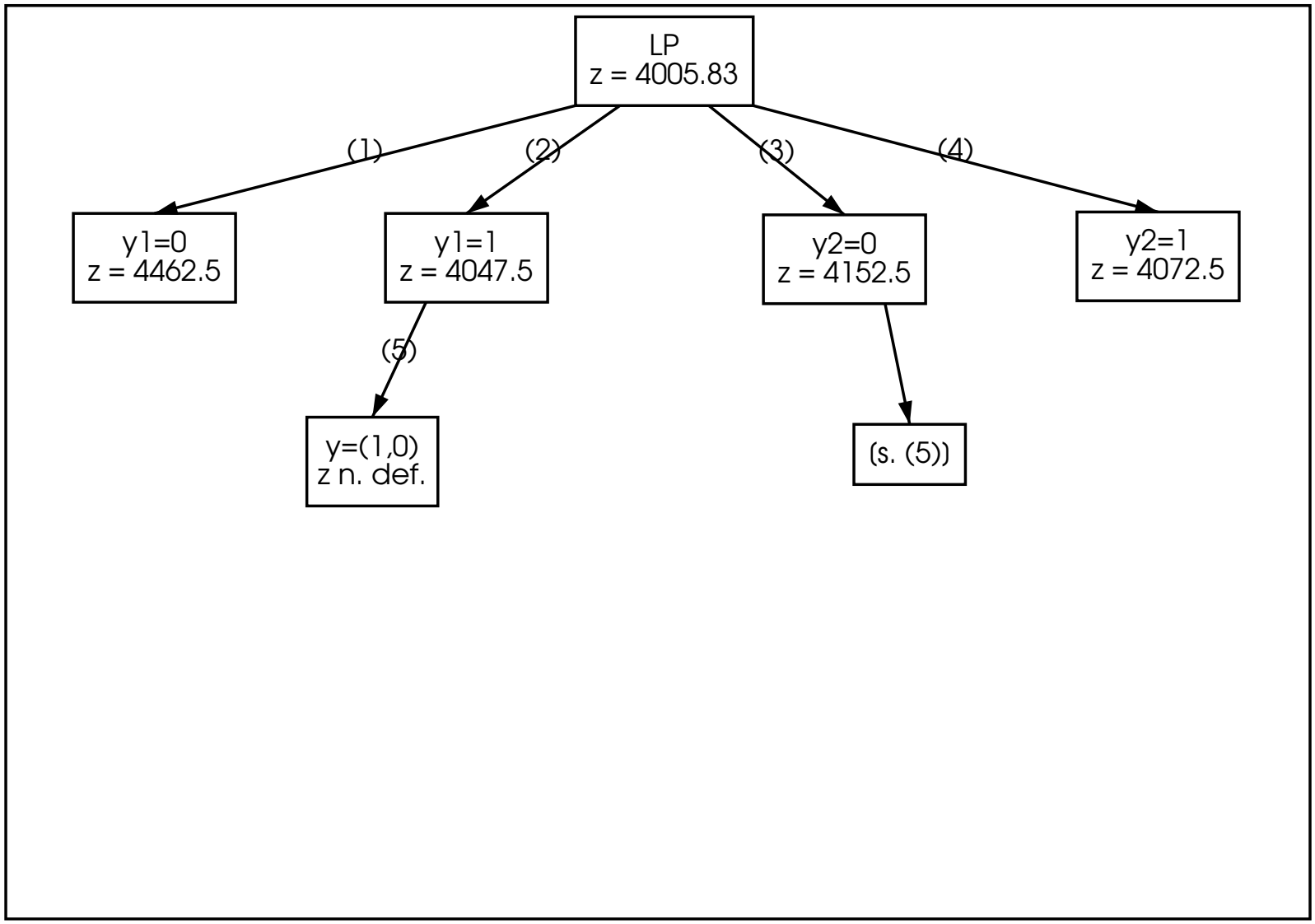
☞ Auswahl des Unterproblems mit dem niedrigsten z

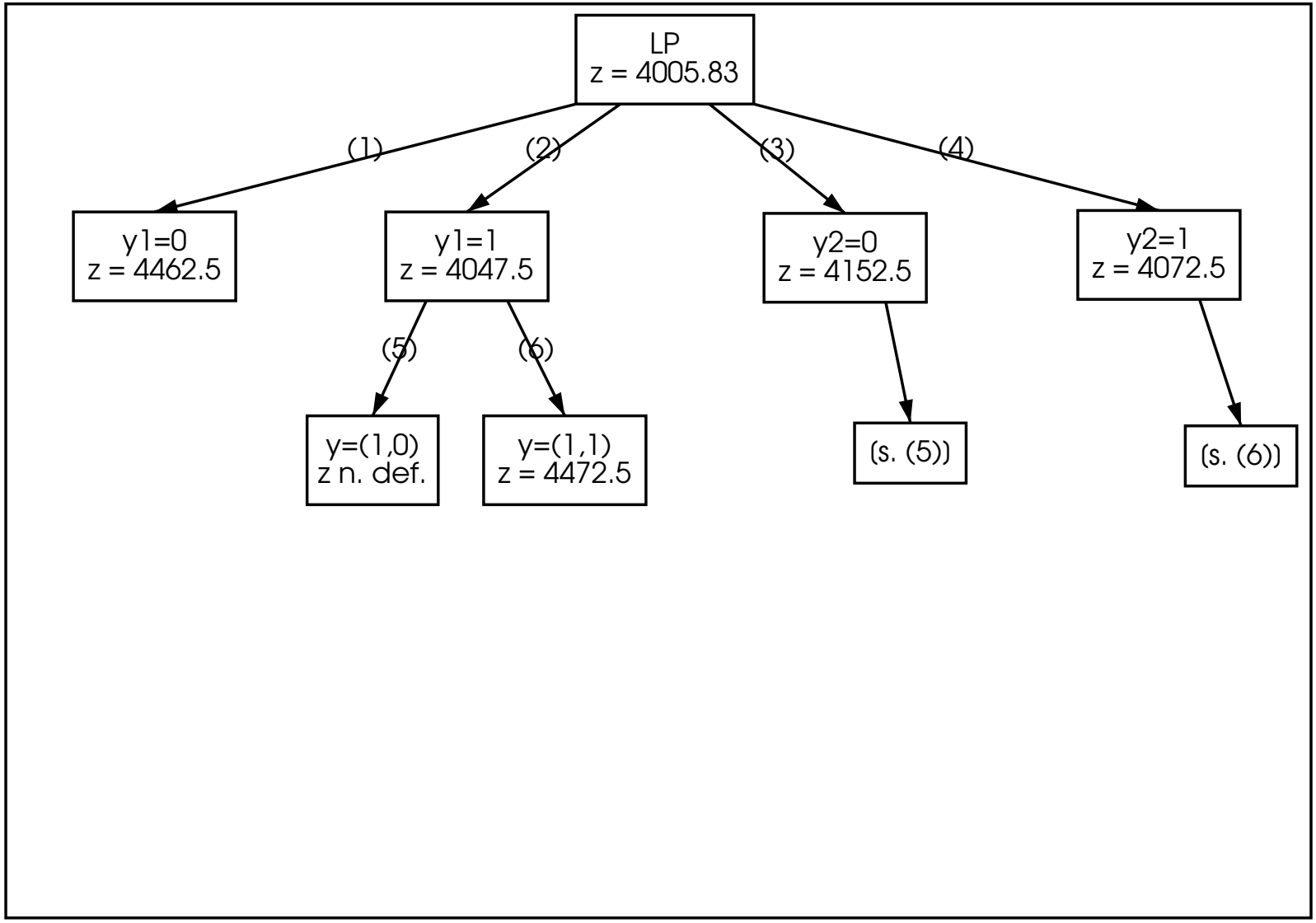
Zurück zu unserem Beispiel:

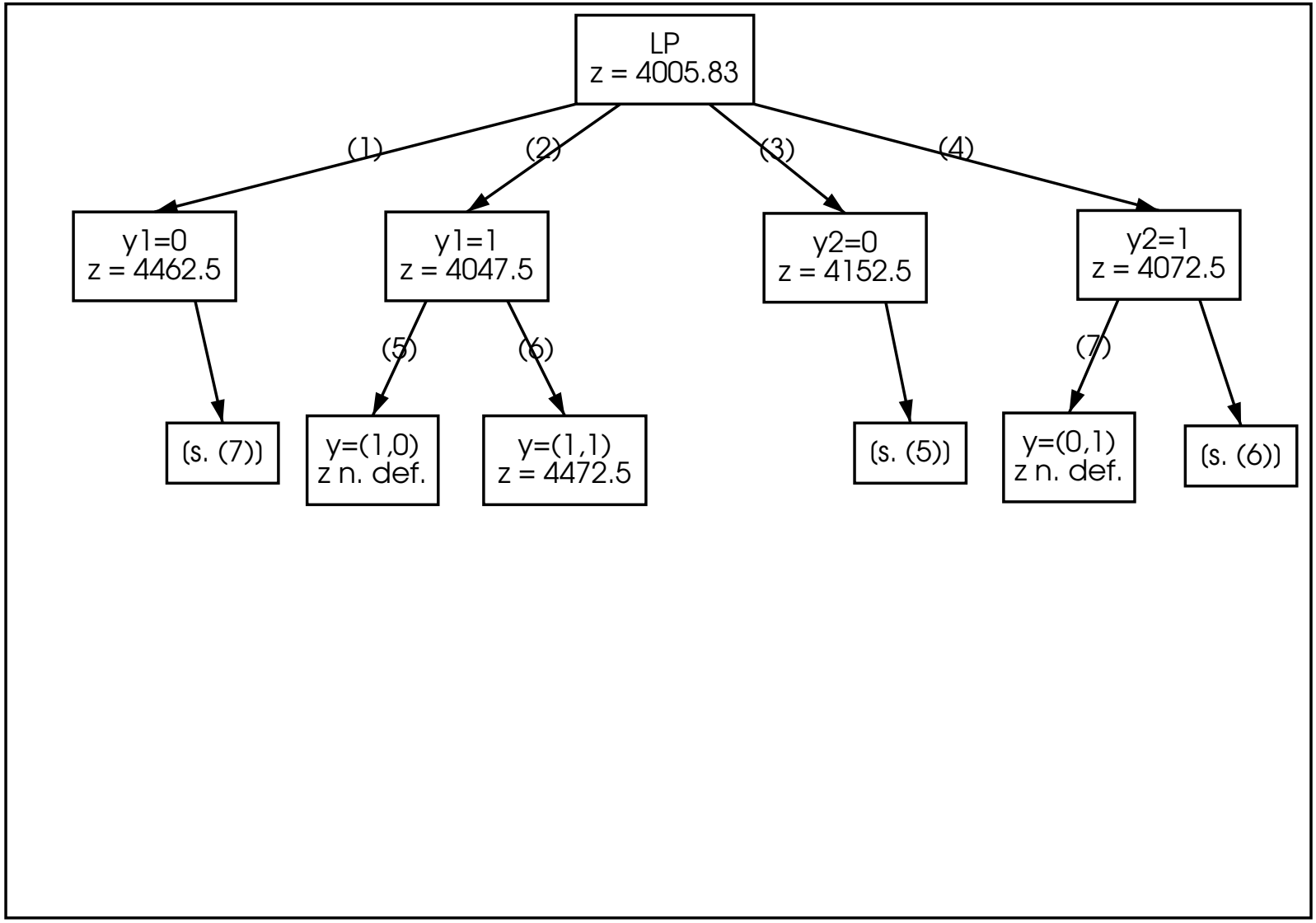
$$z = 4005.83, \quad X = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad y = (1.41667, 0.333333).$$

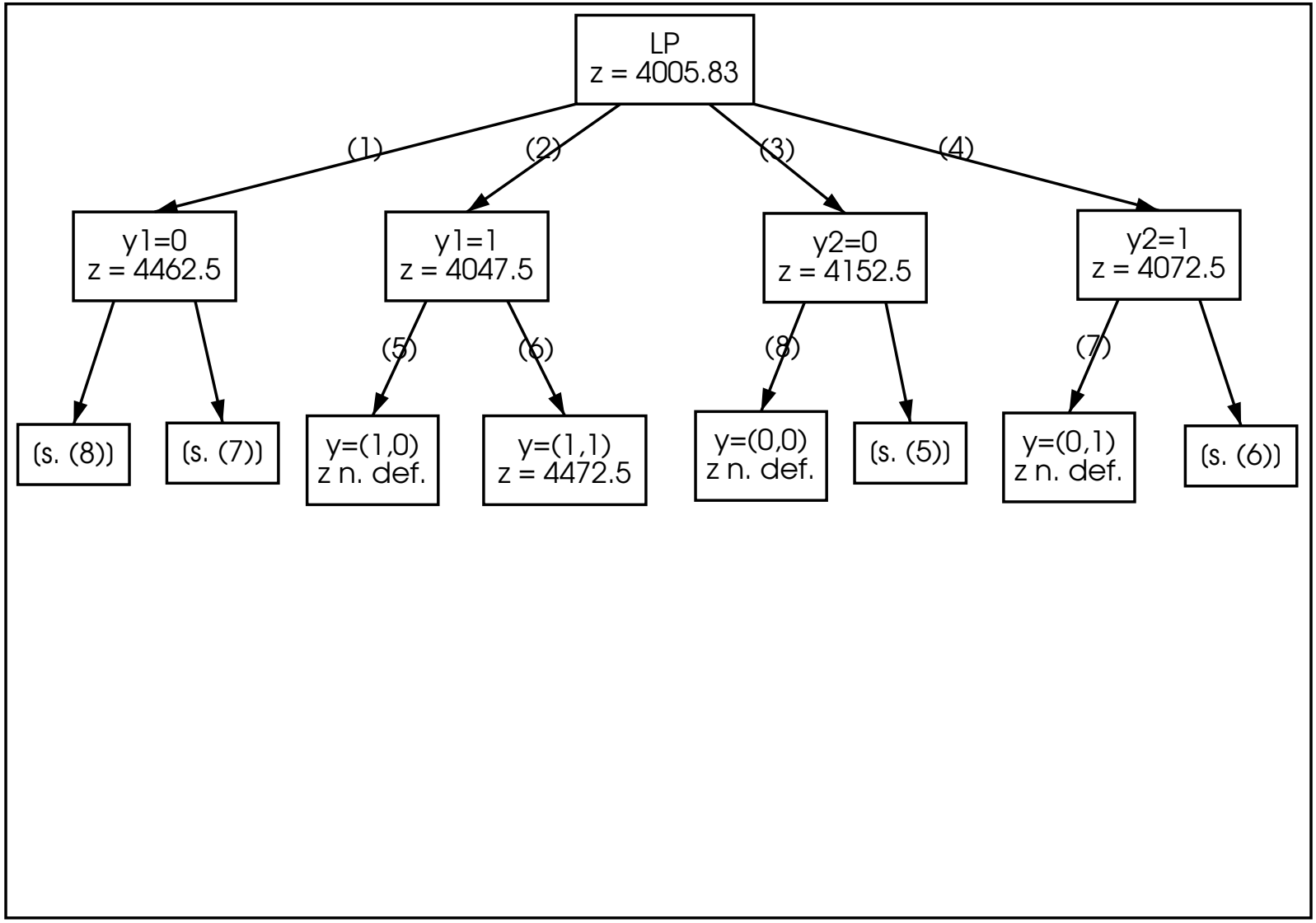
Wenn wir sukzessive verzweigen, erhalten wir Folgendes (wenn nicht anders gekennzeichnet, existieren noch Lösungen, die keine Binärzahlen sind; oben jeweils unsere neu eingeführten Constraints, unten der Funktionswert, sofern das Problem lösbar war):

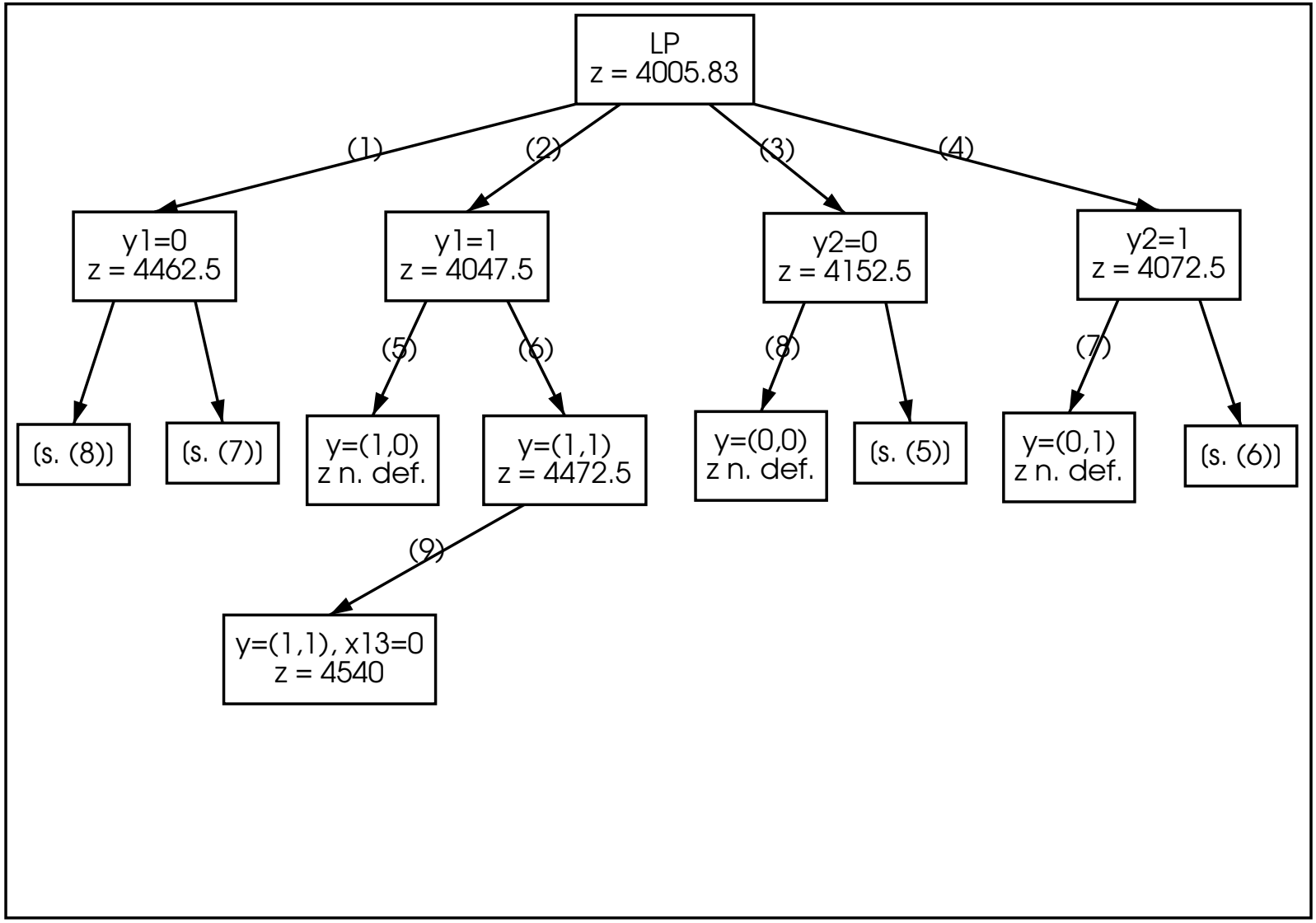

LP
z = 4005.83
(1)
(2)
(3)
(4)
y1=0
z = 4462.5
y1=1
z = 4047.5
y2=0
z = 4152.5
y2=1
z = 4072.5

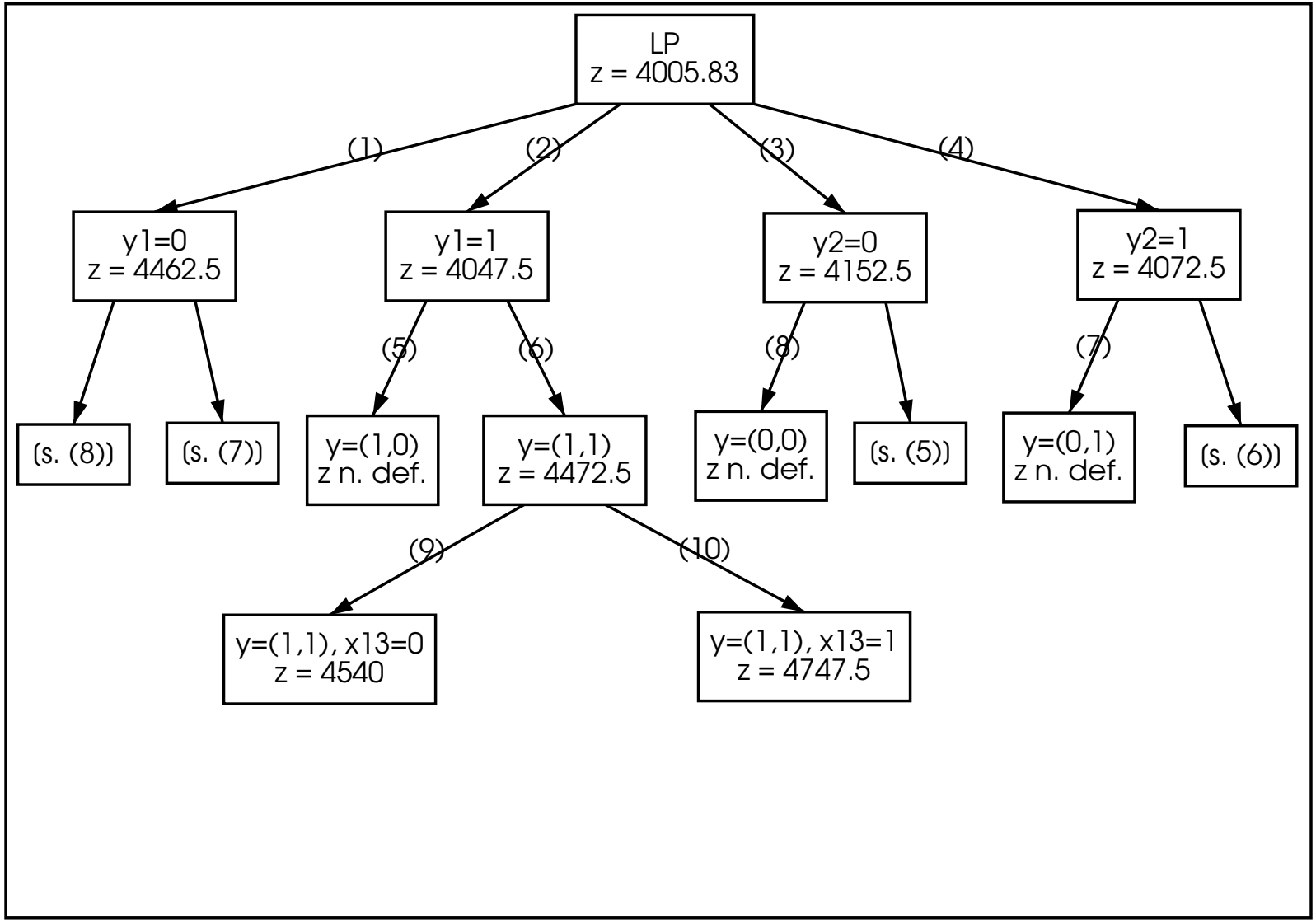

LP
z = 4005.83
(1)
(2)
(3)
(4)
y1=0
z = 4462.5
y1=1
z = 4047.5
y2=0
z = 4152.5
y2=1
z = 4072.5
(5)
y=(1,0)
z n. def.
(s. (5))

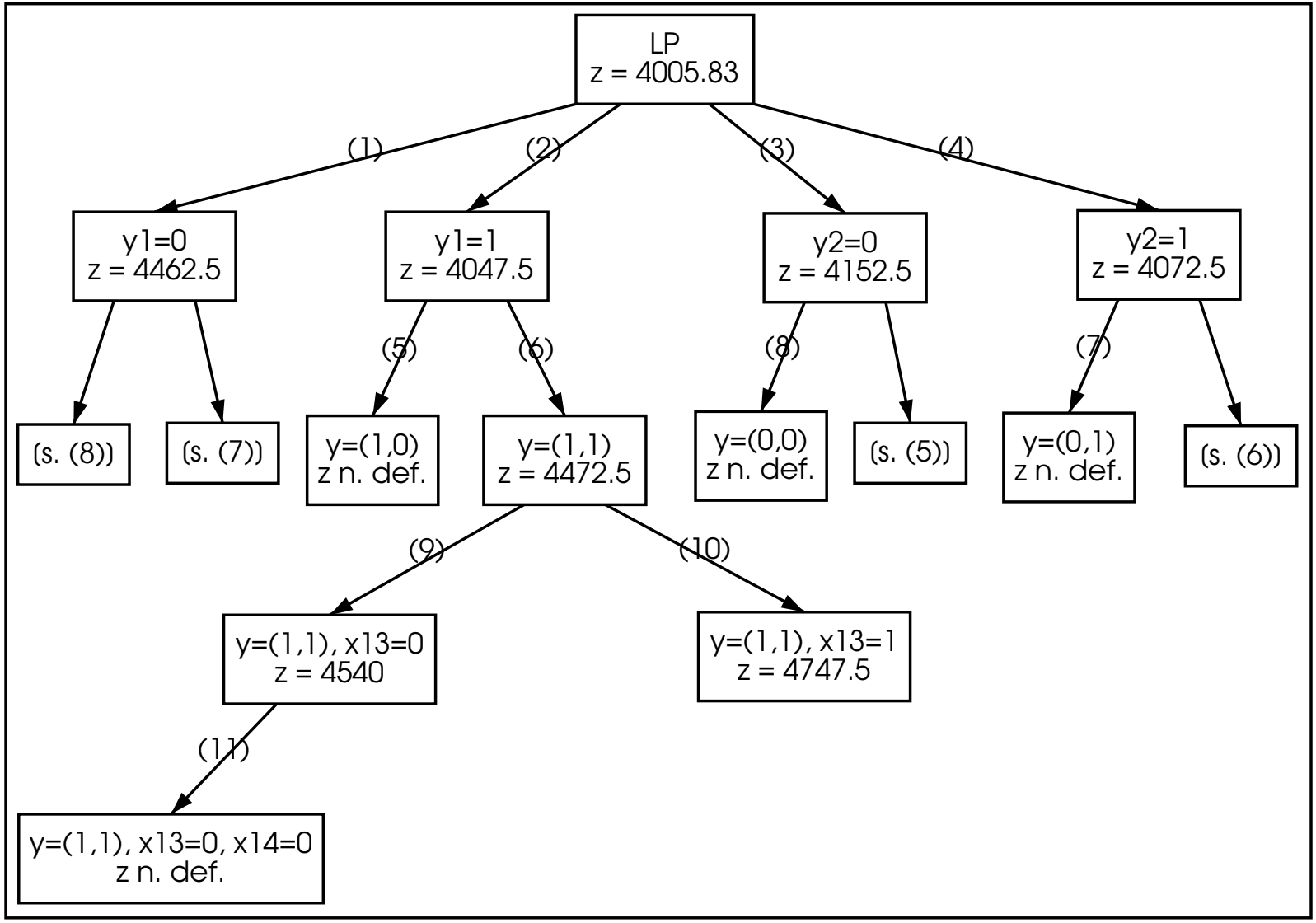

LP
z = 4005.83
(1)
(2)
(3)
(4)
y1=0
z = 4462.5
y1=1
z = 4047.5
y2=0
z = 4152.5
y2=1
z = 4072.5
(5)
(6)
y=(1,0)
z n. def.
y=(1,1)
z = 4472.5
(s. (5))
(s. (6))

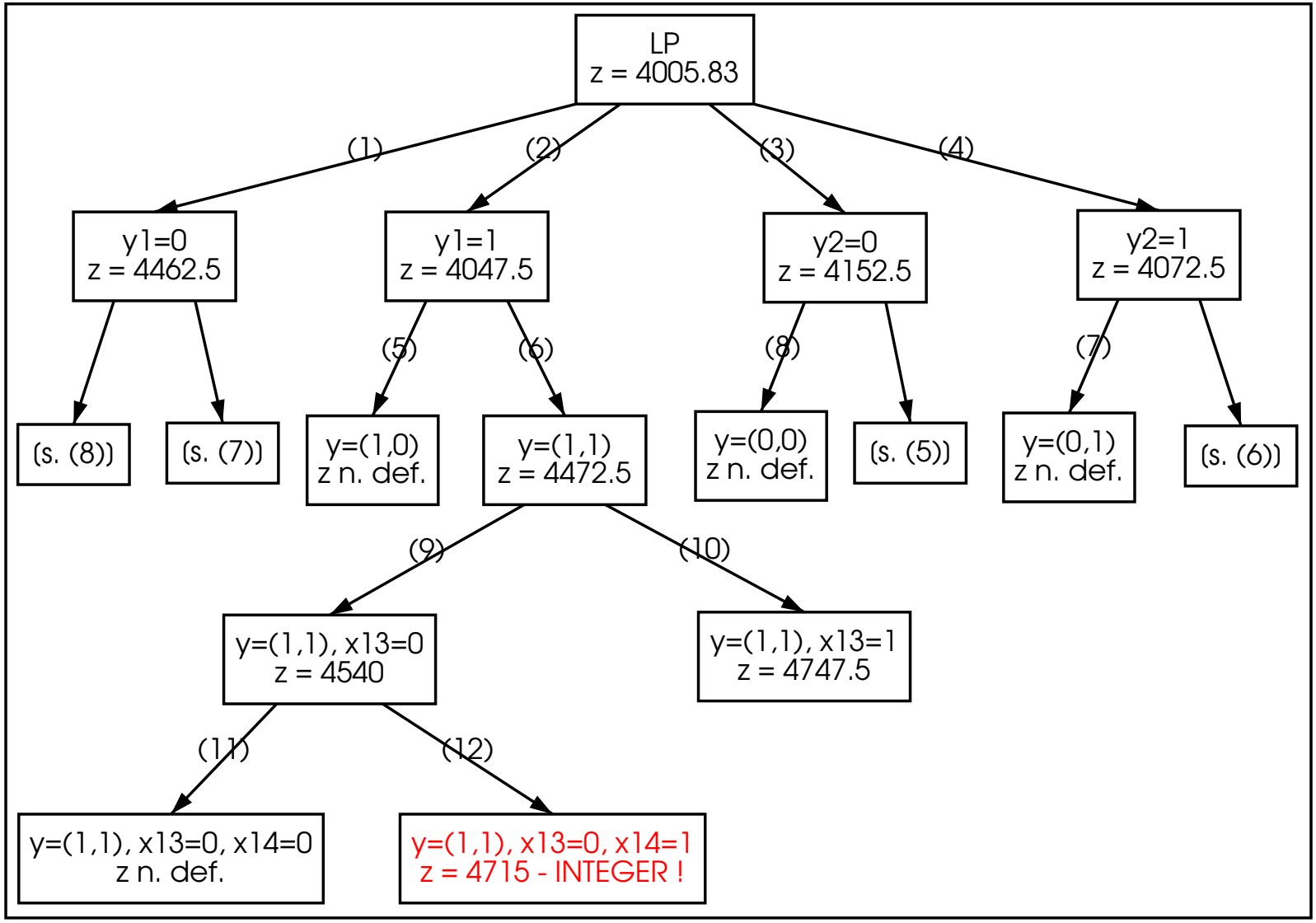

LP
z = 4005.83
(1)
(2)
(3)
(4)
y1=0
z = 4462.5
y1=1
z = 4047.5
y2=0
z = 4152.5
y2=1
z = 4072.5
(5)
(6)
(7)
(s. (7))
y=(1,0)
z n. def.
y=(1,1)
z = 4472.5
(s. (5))
y=(0,1)
z n. def.
(s. (6))


LP
z = 4005.83
(1)
(2)
(3)
(4)
y1=0
z = 4462.5
y1=1
z = 4047.5
y2=0
z = 4152.5
y2=1
z = 4072.5
(5)
(6)
(8)
(7)
(s. (8))
(s. (7))
y=(1,0)
z n. def.
y=(1,1)
z = 4472.5
y=(0,0)
z n. def.
(s. (5))
y=(0,1)
z n. def.
(s. (6))


LP
z = 4005.83
(1)
(2)
(3)
(4)
y1=0
z = 4462.5
y1=1
z = 4047.5
y2=0
z = 4152.5
y2=1
z = 4072.5
(5)
(6)
(8)
(7)
(s. (8))
(s. (7))
y=(1,0)
z n. def.
y=(1,1)
z = 4472.5
y=(0,0)
z n. def.
(s. (5))
y=(0,1)
z n. def.
(s. (6))
(9)
y=(1,1), x13=0
z = 4540


LP
z = 4005.83
(1)
(2)
(3)
(4)
y1=0
z = 4462.5
y1=1
z = 4047.5
y2=0
z = 4152.5
y2=1
z = 4072.5
(5)
(6)
(8)
(7)
(s. (8))
(s. (7))
y=(1,0)
z n. def.
y=(1,1)
z = 4472.5
y=(0,0)
z n. def.
(s. (5))
y=(0,1)
z n. def.
(s. (6))
(9)
(10)
y=(1,1), x13=0
z = 4540
y=(1,1), x13=1
z = 4747.5


LP
z = 4005.83
(1)
(2)
(3)
(4)
y1=0
z = 4462.5
y1=1
z = 4047.5
y2=0
z = 4152.5
y2=1
z = 4072.5
(5)
(6)
(8)
(7)
(s. (8))
(s. (7))
y=(1,0)
z n. def.
y=(1,1)
z = 4472.5
y=(0,0)
z n. def.
(s. (5))
y=(0,1)
z n. def.
(s. (6))
(9)
(10)
y=(1,1), x13=0
z = 4540
y=(1,1), x13=1
z = 4747.5
(11)
y=(1,1), x13=0, x14=0
z n. def.


LP
z = 4005.83
(1)
(2)
(3)
(4)
y1=0
z = 4462.5
y1=1
z = 4047.5
y2=0
z = 4152.5
y2=1
z = 4072.5
(5)
(6)
(8)
(7)
(s. (8))
(s. (7))
y=(1,0)
z n. def.
y=(1,1)
z = 4472.5
y=(0,0)
z n. def.
(s. (5))
y=(0,1)
z n. def.
(s. (6))
(9)
(10)
y=(1,1), x13=0
z = 4540
y=(1,1), x13=1
z = 4747.5
(11)
(12)
y=(1,1), x13=0, x14=0
z n. def.
y=(1,1), x13=0, x14=1
z = 4715 - INTEGER !

Fall 10 muss nicht weiter verzweigt werden, weil alle eventuell noch entstehenden Binärlösungen größer oder gleich 4747.5 sein müssen.

Hier noch einmal die vollständigen Lösungen aller lösbaren Fälle:

	Constraints	z	X	y
1	y1 ... 0	4462.5	0. 0. 0. 0. 1. 1. 1. 1.	0. 1.75
2	y1 ... 1	4047.5	1. 1. 0.473684 0. 0. 0. 0.526316 1.	1. 0.75
3	y2 ... 0	4152.5	1. 1. 1. 1. 0. 0. 0. 0.	1.75 0.
4	y2 ... 1	4072.5	1. 1. 0.157895 0. 0. 0. 0.842105 1.	0.75 1.
6	y1 ... 1 y2 ... 1	4472.5	1. 1. 0.157895 0. 0. 0. 0.842105 1.	1. 1.
9	y1 ... 1 y2 ... 1 x13 ... 0	4540.	1. 1. 0. 0.375 0. 0. 1. 0.625	1. 1.
10	y1 ... 1 y2 ... 1 x13 ... 1	4747.5	1. 0.230769 1. 0. 0. 0.769231 0. 1.	1. 1.
12	y1 ... 1 y2 ... 1 x13 ... 0 x14 ... 1	4715.	1. 1. 0. 1. 0. 0. 1. 0.	1. 1.

4 Egebnisse Teillösung Lineares Programmieren

- zunächst Optimierungsprogramme, danach Computeralgebrasysteme, zuletzt Numerikprogramme getestet

- Angabe des jeweiligen Quelltextes des Beispiels

- getestet mit verschieden großen Facility Location Problemen

  - per Zufallsgenerator erstellt

- ☞ $r_{i,j}$ gleichverteilt zwischen 0 und 2.5
- ☞ d_j gleichverteilt zwischen 50 und 2050
- ☞ f_i gleichverteilt zwischen 1000 und 10000
- ☞ $c_1 = \frac{1}{20} \sum_j d$
- ☞ $c_2 = \frac{19}{20} \sum_j d$

⇒ zuerst Lineares Programming getestet, danach (falls vom Programm unterstützt) Integer Programming

4.1 XPress-MP (Mosel / Optimizer)

4.1.1 Beschreibung

XPress ist durch den intensiven Gebrauch im Seminar hinreichend bekannt – hier eine kurze Zusammenfassung:

⇛ Problembeschreibungssprache Mosel → wird kompiliert (⇒ Geschwindigkeitsgewinn)

- Lineares Programmieren: drei Algorithmen existieren im Optimizer:
  - Dual Simplex (default; -d),
  - Primal Simplex (-p),
  - Newton Barrier (-b)
- Integer Programming: In Mosel durch zusätzliche Constraints angegeben.

4.1.2 Quelltext für Beispiel, Linear Program

```
model expl
uses "mmxprs"
declarations
p = 1..2
q = 1..4
d: array(q) of real
f:  array(p)of real
r: array(p, q) of real
c: array(1..2) of real
x: array(p, q) of mpvar
y: array(p) of mpvar
end-declarations
```

```
d:= [ 50, 325, 475, 200 ]
f := [ 1600, 2000 ]
c := [ 400, 600 ]
r := [ 0.7, 0.8, 1.3, 2.2,
1.3, 1.1, 0.8, 0.8 ]
forall (j in q)
con(j) := sum(i in p) (x(i, j)) = 1
forall (i in p) do
sum(j in q) (d(j)*x(i, j)) <= c(2)*y(i)
sum(j in q) (d(j)*x(i, j)) >= c(1)*y(i)
end-do
```

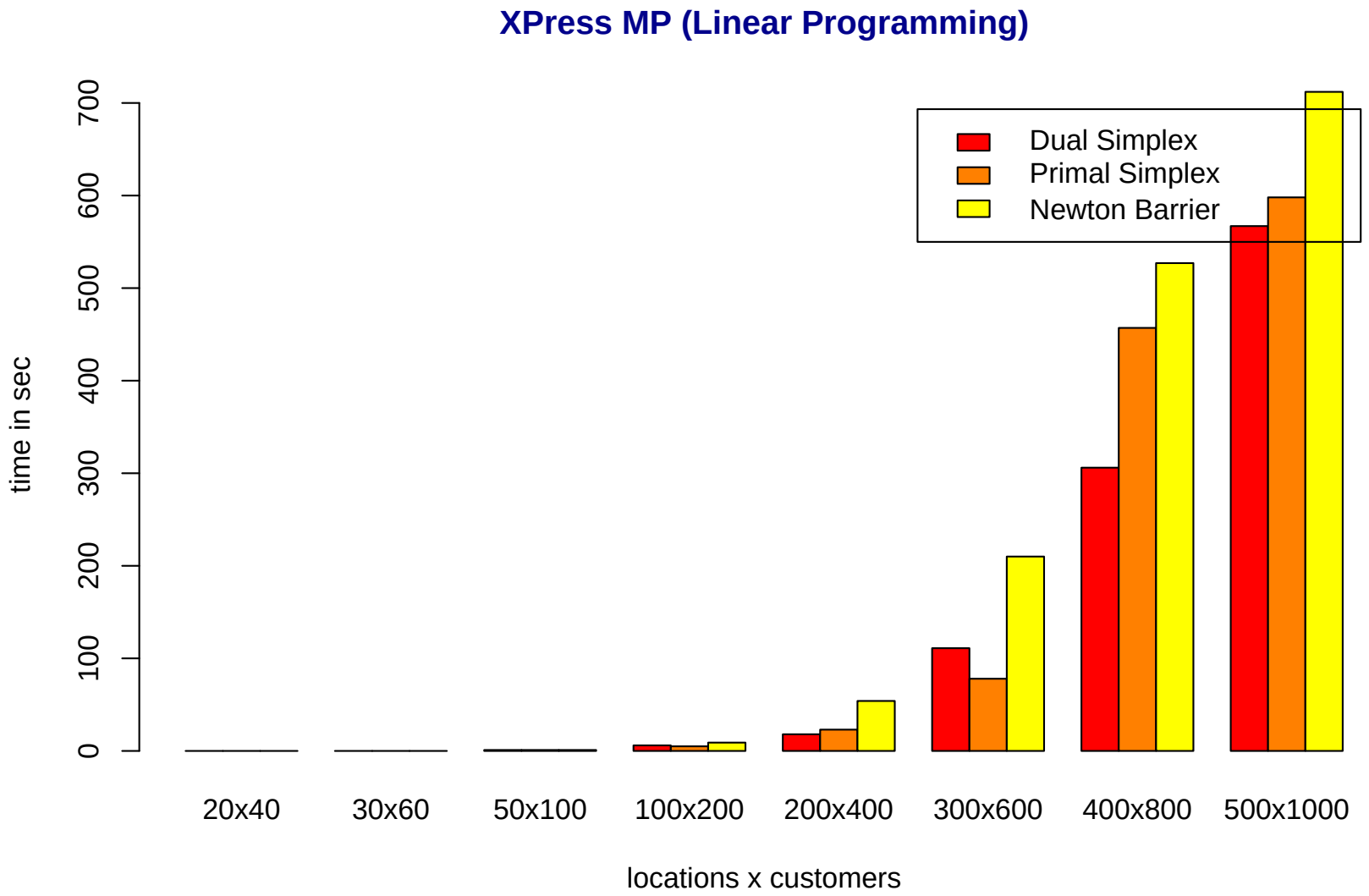
```
obj:= sum(i in p) ( sum(j in q) ( r(i, j)*d(j)*x(i, j) ) )
+ sum(i in p) ( y(i)*f(i) )
minimize(obj)

writeln("Min: ", getobjval)

forall (i in p) do
forall(j in q)
writeln("x(", i, ",", j, ") = ", getsol(x(i, j)) )
end-do
forall (i in p)
writeln("y(", i, ") = ", getsol(y(i)))

end-model
```

4.1.3 Bewertung des Quelltextes

- Mosel ideal zur Beschreibung von Optimierungsproblemen
- Facility Location Probleme können ohne Notwendigkeit von Umformulierungen direkt eingegeben werden (durch Unterstützung der Eingabe von Summen)
- Schreiben von Funktionen, denen als Parameter nur die Vektoren / Matrix übergeben werden, ist kein Problem

4.1.4 Geschwindigkeiten (in Sek.) LP

size	Dual Simpl.	Primal Simpl.	Newton Barrier
20x40	0	0	0
30x60	0	0	0
50x100	1	1	1
100x200	6	5	9
200x400	18	23	54
300x600	111	78	210
400x800	306	457	527
500x1000	567	598	712


XPress MP (Linear Programming)
Dual Simplex
Primal Simplex
Newton Barrier
time in sec
0
100
200
300
400
500
600
700
20x40
30x60
50x100
100x200
200x400
300x600
400x800
500x1000
locations x customers

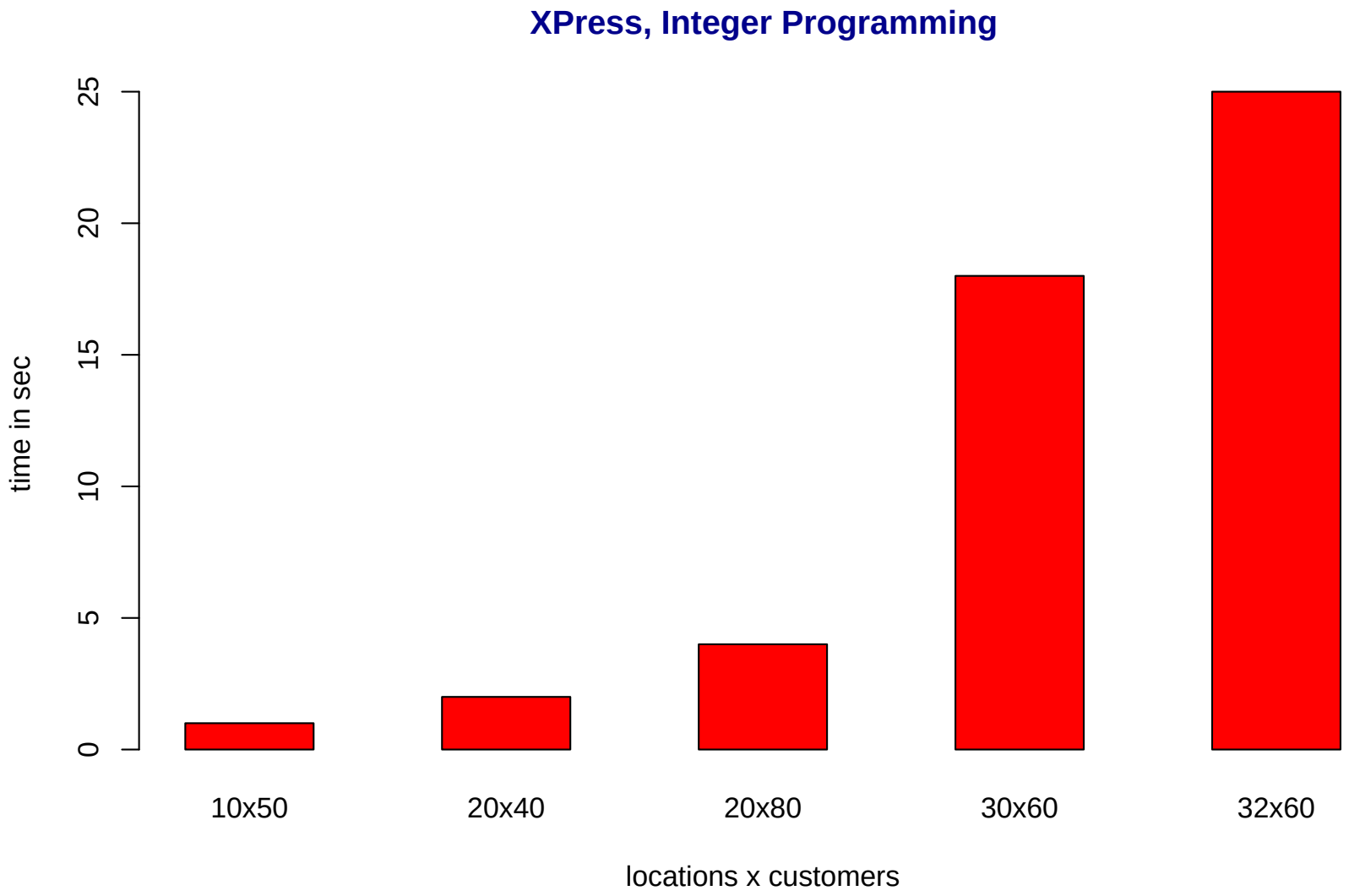
4.1.5 Quelltext für Beispiel, Integer Program

Es werden lediglich die folgenden Zeilen hinzugefügt:

```
forall (i in p) do
forall(j in q)
x(i, j) is_binary
end-do

forall (i in p)
y(i) is_binary
```

4.1.6 Geschwindigkeiten (in Sek.) IP

size	time
10x50	1
20x40	2
20x80	4
30x60	18
32x60	25


XPress, Integer Programming
time in sec
0
5
10
15
20
25
10x50
20x40
20x80
30x60
32x60
locations x customers

4.2 lp_solve 4.0 beta

4.2.1 Beschreibung

- lp_solve ist Open Source Programm (GPL), geschrieben in ANSI-C
- Quelle: ftp://ftp.es.ele.tue.nl/pub/lp_solve
- liest sowohl Eingabe von Kommandozeile als auch mps- und lp-Dateien

- ☞ in Kommandozeile: Eingabe der Matrix möglich
- ☞ lp-Dateien: lassen Eingabe der Gleichungen / Ungleichungen zu

⇒ Aber: alle Zahlen müssen direkt eingegeben werden, ohne die Möglichkeit der Manipulation durch mathematische Operationen

⇒ $\Rightarrow$ schreiben von Funktionen nicht möglich

⇒ brauchbar für Aufrufe aus anderen Programmen, ansonsten unhandlich

4.2.2 Quelltext für Beispiel, Linear Program (lp-Datei)

```
min: 35*x1c1 + 260*x1c2 + 617.5*x1c3 + 440*x1c4 + 65*x2c1 +
357.5*x2c2 + 380*x2c3 + 160*x2c4 + 1600*y1 + 2000*y2;

+x1c1+x2c1 = 1;
+x1c2+x2c2 = 1;
+x1c3+x2c3 = 1;
+x1c4+x2c4 = 1;
50*x1c1 + 325*x1c2 + 475*x1c3 + 200*x1c4 - 400*y1 >= 0;
50*x2c1 + 325*x2c2 + 475*x2c3 + 200*x2c4 - 400*y2 >= 0;
50*x1c1 + 325*x1c2 + 475*x1c3 + 200*x1c4 - 1000*y1 <= 0;
50*x2c1 + 325*x2c2 + 475*x2c3 + 200*x2c4 - 1000*y2 <= 0;
```

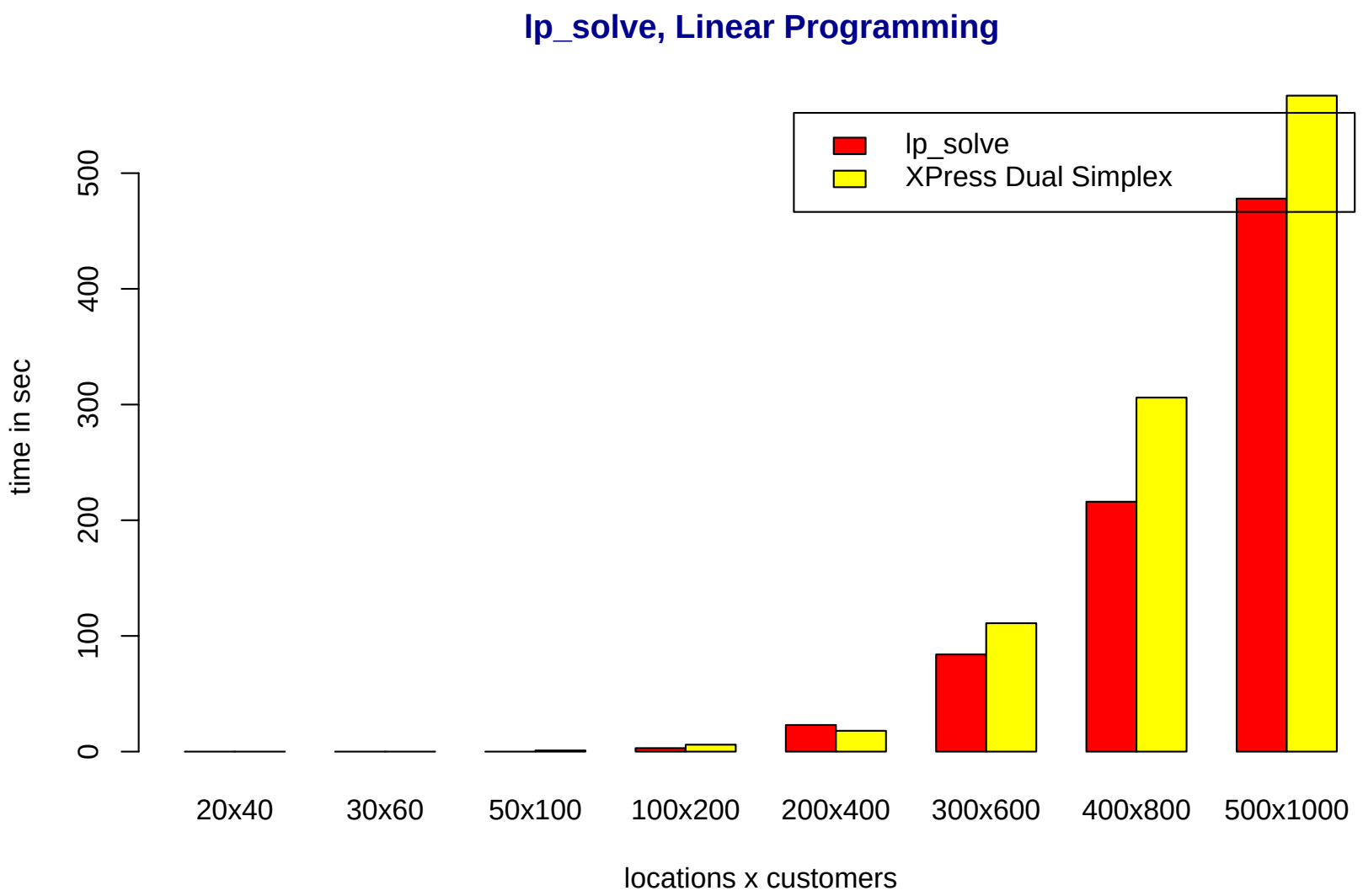
4.2.3 Bewertung des Quelltextes

- unkomfortabel für eine Eingabe per Hand

- Quelltext sollte automatisch generiert werden (z. B. Aufruf aus anderem Programm)

4.2.4 Geschwindigkeiten (in Sek.) LP

size	lp_solve	Dual Simpl. v. XPress
20x40	0	0
30x60	0	0
50x100	0	1
100x200	3	6
200x400	23	18
300x600	84	111
400x800	216	306
500x1000	478	567


lp_solve, Linear Programming
lp_solve
XPress Dual Simplex
time in sec
0
100
200
300
400
500
20x40
30x60
50x100
100x200
200x400
300x600
400x800
500x1000
locations x customers

4.2.5 Quelltext für Beispiel, Integer Program

➨ an obige Datei wird unten angefügt:

```
int x1c1, x1c2, x1c3, x1c4, y1, x2c1, x2c2, x2c3, x2c4, y2;
```

4.2.6 Geschwindigkeiten (in Sek.) IP

size	lp_solve	XPress
10x50	0	1
20x40	0	2
20x80	31	4

(Alles weitere mehr als 10 Minuten $\rightarrow$ Abbruch.)

4.3 Maple 7.0

4.3.1 Beschreibung

- Computeralgebrasystem, kommerziell
- im ilabws-Pool (Sun-Pool, 1. Etage) installiert
- keine Möglichkeiten zur Integer Programmierung

- hat aber Simplex-Paket:
  - viele verschiedene Teile des Simplex-Algorithmus lassen sich einzeln ansprechen
  - z. B. Funktionen basis, pivot, standardize, ...
  - $\Rightarrow$ geeignet zum Experimentieren
- Funktionen maximze und minimize zur Linearen Programmierung aus diesem Paket

4.3.2 Quelltext für Beispiel, (nur Linear Program)

```
with(simplex):

d:= [ 50, 325, 475, 200 ];
f := [ 1600, 2000 ];
r := [[0.7, 0.8, 1.3, 2.2],
[1.3, 1.1, 0.8, 0.8]];
c := [400, 1000];
minimize(
r[1, 1]*d[1]*x11
+ r[1, 2]*d[2]*x12
+ r[1, 3]*d[3]*x13
+ r[1, 4]*d[4]*x14
+ r[2, 1]*d[1]*x21
```

```
+ r[2, 2]*d[2]*x22
+ r[2, 3]*d[3]*x23
+ r[2, 4]*d[4]*x24
+ y1*f[1]+ y2*f[2],
{
x11 + x21 = 1,
x12 + x22 = 1,
x13 + x23 = 1,
x14 + x24 = 1,
d[1]*x11 + d[2]*x12 + d[3]*x13 + d[4]*x14 >= c[1]*y1,
d[1]*x21 + d[2]*x22 + d[3]*x23 + d[4]*x24 >= c[1]*y2,
d[1]*x11 + d[2]*x12 + d[3]*x13 + d[4]*x14 <= c[2]*y1,
d[1]*x21 + d[2]*x22 + d[3]*x23 + d[4]*x24 <= c[2]*y2
},
NONNEGATIVE
);
```

4.3.3 Bewertung des Quelltextes

- zwar lassen sich Gleichungen / Ungleichungen direkt eingeben (übersichtlich, einfach verständlich)

- aber: bei größeren Problemstellungen unflexibel (Quelltexte müssen wieder automatisch generiert werden)

- Funktion zum Facility Location Problem lässt sich nur mit großem Aufwand verwirklichen

4.3.4 Geschwindigkeiten (in Sek.)

size	time
5x10	0
10x20	4
20x40	24
30x60	70

(Alles weitere führte zum Absturz von Maple bei direkter Eingabe.)

4.4 Mathematica 4.2

4.4.1 Beschreibung

- Computeralgebrasystem, kommerziell
- im Mathe-Pool installiert
- keine Möglichkeiten zur Integer Programmierung (es existiert dazu aber ein kommerzielles Zusatzpaket)

- Lineare Programmierung: zwei Möglichkeiten
  - entweder Gleichungen / Ungleichungen eingeben (Funktionen ConstrainedMin[] / ConstrainedMax[])
  - oder Matrix direkt eingeben (Funktion LinearProgramming[])
- folglich Funktion für Facillity Location Problem einfach realisierbar

4.4.2 Quelltext für Beispiel, (nur Linear Program), Funktion für Fac. Loc.

```
facloc[d_, f_, r_, c_]:= LinearProgramming[
Flatten[{# d & /@ r, f}],
Join[
Transpose[Join[Flatten[IdentityMatrix[Length[d]] & /@
f, 1], 0 r]],
Flatten[MapThread[Join, {Flatten[d # & /@ #] & /@
IdentityMatrix[Length[f]],
-#*IdentityMatrix[Length[f]]}] & /@ c, 1]
],
Transpose[{
Join[0*d+1, 0*f, 0*f],
Join[0*d, 0*f+1, 0*f-1]}]]
```

4.4.3 Bewertung des Quelltextes

- kurz und elegant
- möglicherweise für Einsteiger verwirrend
- durch die Existenz zweier Möglichkeiten der Umsetzung hervorragend zum Experimentieren geeignet

4.4.4 Geschwindigkeiten (in Sek.)

size	time
5x10	0
10x20	0
20x40	5
30x60	15
50x100	88

4.5 MuPAD 2.5.2

4.5.1 Beschreibung

- Computeralgebrasystem
- kommerziell, aber für Universitätsangehörige kostenlos
- Quelle: http://www.mupad.com/
- im ilabws-Pool installiert

- Optimierung: Es existiert Bibliothek linopt, die Lineare Optimierung incl. Integer Programming umfasst.

- viele verschiedene Teile des Simplex-Algorithmus lassen sich einzeln ansprechen

- $\Rightarrow$ hervorragend zum Experimentieren geeignet

4.5.2 Quelltext für Beispiel

```
d:= [ 50, 325, 475, 200 ];
f := [ 1600, 2000 ];
r := array(1..2, 1..4, [[0.7, 0.8, 1.3, 2.2],
[1.3, 1.1, 0.8, 0.8]]);
c := [400, 1000];
linopt::minimize([
{
x11 + x21 = 1,
x12 + x22 = 1,
x13 + x23 = 1,
x14 + x24 = 1,
d[1]*x11 + d[2]*x12 + d[3]*x13 + d[4]*x14 >= c[1]*y1,
d[1]*x21 + d[2]*x22 + d[3]*x23 + d[4]*x24 >= c[1]*y2,
```

```
d[1]*x11 + d[2]*x12 + d[3]*x13 + d[4]*x14 <= c[2]*y1,
d[1]*x21 + d[2]*x22 + d[3]*x23 + d[4]*x24 <= c[2]*y2
},
r[1, 1]*d[1]*x11
+ r[1, 2]*d[2]*x12
+ r[1, 3]*d[3]*x13
+ r[1, 4]*d[4]*x14
+ r[2, 1]*d[1]*x21
+ r[2, 2]*d[2]*x22
+ r[2, 3]*d[3]*x23
+ r[2, 4]*d[4]*x24
+ y1*f[1]+ y2*f[2],
NonNegative
]);
```

```
// Integer Programming:

linopt::minimize([
{
x11 + x21 = 1,
x12 + x22 = 1,
x13 + x23 = 1,
x14 + x24 = 1,
d[1]*x11 + d[2]*x12 + d[3]*x13 + d[4]*x14 >= c[1]*y1,
d[1]*x21 + d[2]*x22 + d[3]*x23 + d[4]*x24 >= c[1]*y2,
d[1]*x11 + d[2]*x12 + d[3]*x13 + d[4]*x14 <= c[2]*y1,
d[1]*x21 + d[2]*x22 + d[3]*x23 + d[4]*x24 <= c[2]*y2,
y1<=1,
y2<=1
},
```

```
r[1, 1]*d[1]*x11
+ r[1, 2]*d[2]*x12
+ r[1, 3]*d[3]*x13
+ r[1, 4]*d[4]*x14
+ r[2, 1]*d[1]*x21
+ r[2, 2]*d[2]*x22
+ r[2, 3]*d[3]*x23
+ r[2, 4]*d[4]*x24
+ y1*f[1]+ y2*f[2],
NonNegative,
All
]);
```

4.5.3 Bewertung des Quelltextes

Siehe Maple. **4.5.4 Geschwindigkeiten (in Sek.), LP**

size	**time**
5x10	2
10x20	16
20x40	106
30x60	540

4.5.5 Geschwindigkeiten IP

Es ließen sich in annehmbarer Zeit (unter 10 min) nur sehr kleine Beispiele ausführen – ein Vergleich mit den beiden Optimierungsprogrammen ist daher kaum machbar.

4.6 Matlab R12

4.6.1 Beschreibung

- kommerzielles Paket für technische Berechnungen (sehr teuer)
- installiert im Pool für Computergraphik (1. Etage)
- im Praxis in Großbetrieben weit verbreitet

- ⇒ besteht aus Grundpaket + Toolboxes

- ⇒ eine Toolbox: Optimization, darin Funktion linprog, die die Eingabe in Matrix-Form unterstützt

- ⇒ keine Integer-Programmierung (es existieren aber Zusatzpakete dafür)

4.6.2 Quelltext für Beispiel

```
d = [ 50, 325, 475, 200 ];
f = [ 1600, 2000 ];
r = [0.7, 0.8, 1.3, 2.2;
1.3, 1.1, 0.8, 0.8];
c = [400, 1000];

n = length(f); % # of facilities
k = length(d); % # of zones
fob = []; % object function
Aeq = [];
for i = 1:n
fob = [ fob r(i, :).*d ];
Aeq = [ Aeq diag(diag(ones(k))) ];
```

```
end
fob = [fob f];
Aeq = [ Aeq zeros(k, n) ];
beq = ones(k,1);
A = [];
for i=1:n
A = [ A; zeros(1, (i-1)*k) -d zeros(1,
(n-i)*k) zeros(1, i-1) c(1) zeros(1, n - i) ];
end
for i=1:n
A = [ A; zeros(1, (i-1)*k) d zeros(1,
(n-i)*k) zeros(1, i-1) -c(2) zeros(1, n - i) ];
end
b = zeros(2*n, 1);
lb = zeros(n*(k+1), 1);
[x,fval,exitflag,output,lambda] = linprog(fob, A, b, Aeq, beq, lb)
```

4.6.3 Bewertung des Quelltextes

- ⇒ geeignet für die Programmierung von Funktionen

- ⇒ aber: bei Vorgabe des Problems in Form von Gleichungen / Ungleichungen unintuitiv und umständlich

4.6.4 Geschwindigkeiten (in Sek.)

size	time
5x10	0
10x20	0
50x100	3

(Höhere Werte überstiegen meinen Arbeitsspeicher und dauerten schlagartig weitaus länger)

4.7 R 1.6.0

4.7.1 Beschreibung

- Open Source (GPL) Numerik-Paket mit Schwerpunkt Statistik
- Quelle: http://www.r-project.org
- Optimierung nur Nebenprodukt, Funktion simplex()

4.7.2 Quelltext für Beispiel

```
require(boot);

d <- c( 50, 325, 475, 200 );
f <- c( 1600, 2000 );
r <- matrix(c(0.7, 0.8, 1.3, 2.2, 1.3, 1.1, 0.8, 0.8), 2, 4, TRUE);
c <- c(400, 1000);

n = length(f); # of facilities
k = length(d); # of zones
fob <- c(t(r)*d, f)
A3 <- matrix(c(rep(diag(k), n), t(0*r)), k)
b3 <- rep(1, k)
A2 <- NULL
```

```
A1 <- NULL
for (i in 1:n)
{
A2 <- c(A2, rep(0, (i-1)*k), d, rep(0, (n-i)*k+i-1),
-c[1], rep(0, n-i))
A1 <- c(A1, rep(0, (i-1)*k), d, rep(0, (n-i)*k+i-1),
-c[2], rep(0, n-i))
}
A2 <- matrix(A2, n, byrow = TRUE)
b2 <- rep(0, n)
A1 <- matrix(A1, n, byrow = TRUE)
b1 <- rep(0, n)
x <- simplex(fob, A1, b1, A2, b2, A3, b3)
print(x)
```

4.7.3 Bewertung des Quelltextes

Siehe Matlab.

4.7.4 Geschwindigkeiten (in Sek.)

size	time
5x10	0
10x20	1
20x40	10
30x60	56