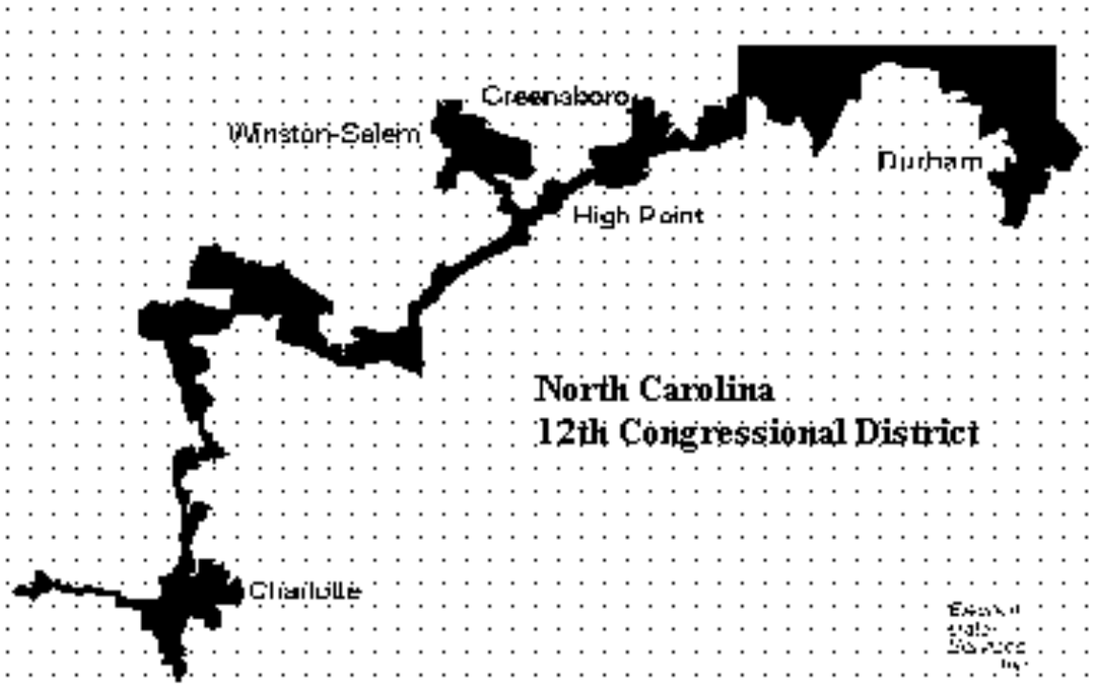
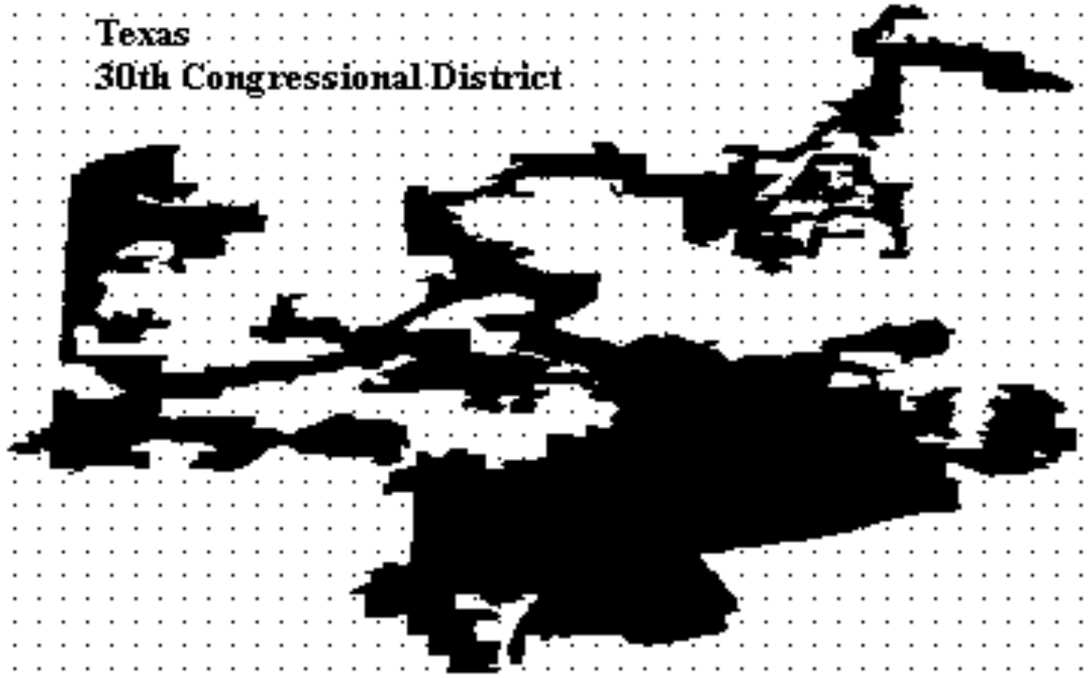
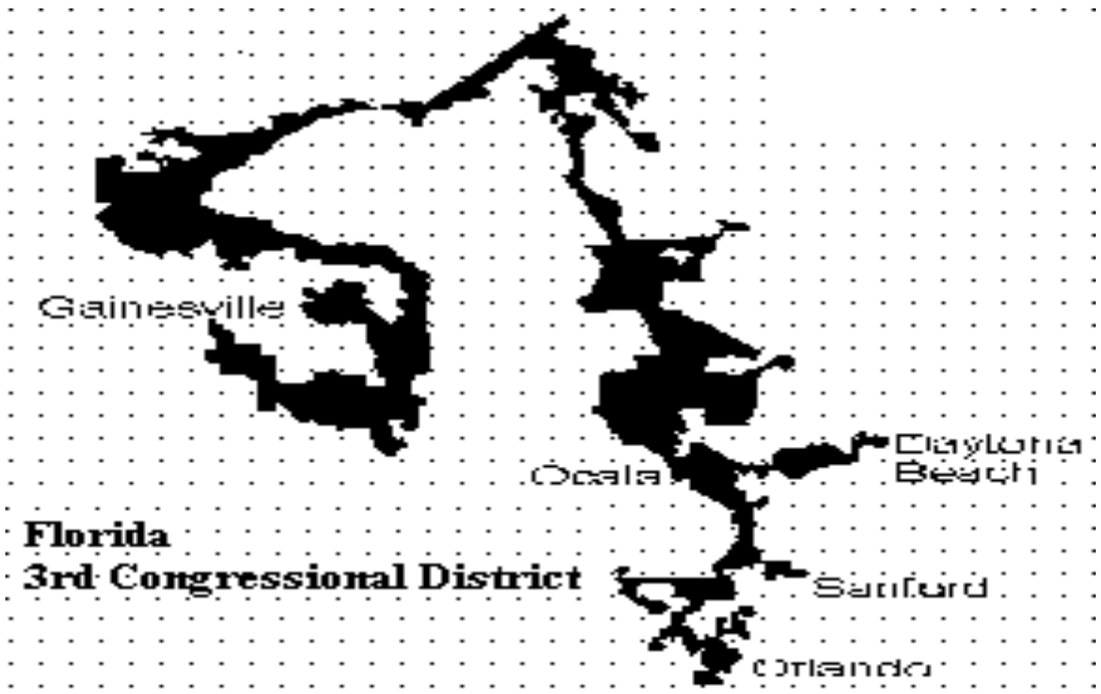
Gerrymandering

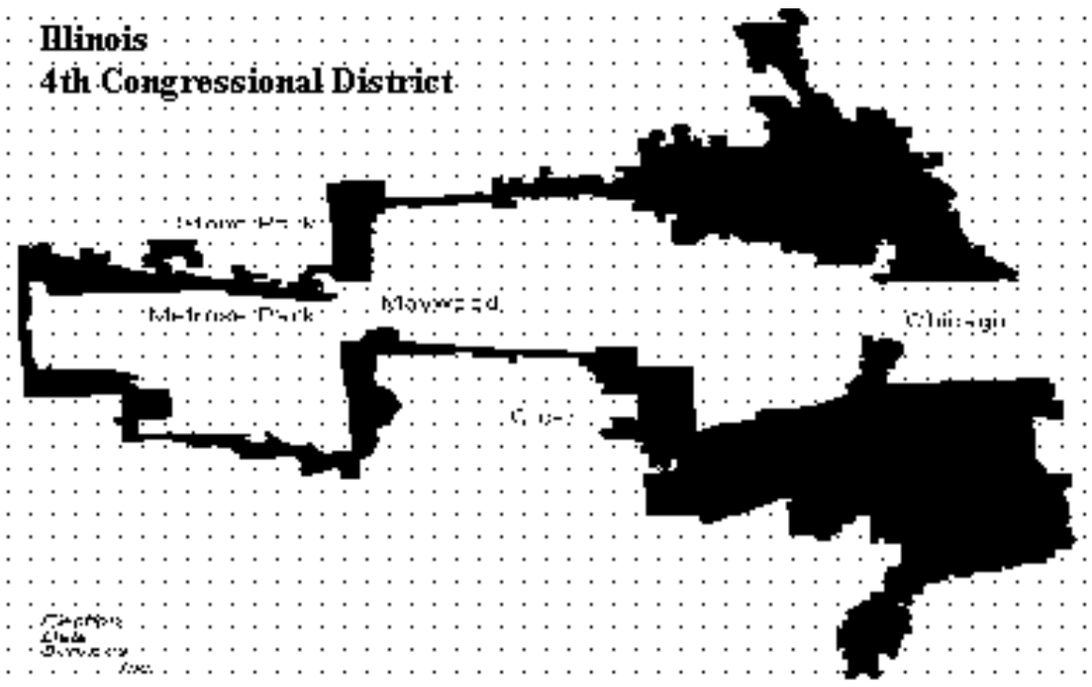
- Die Kunst der Wahlkreisformung

Gerry's gallery









ZIMPL

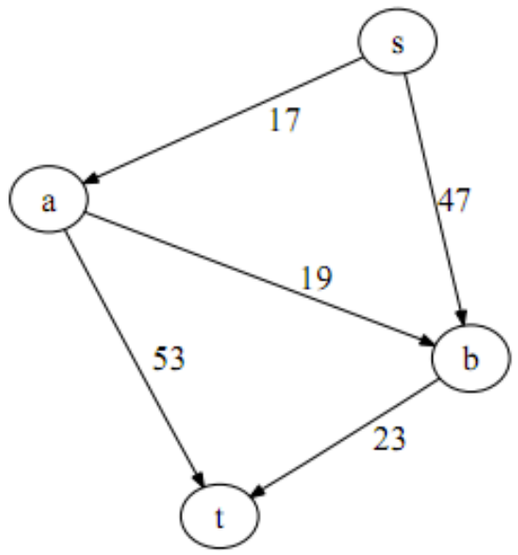
(Zuse Institute Mathematical Programming Language)

Generelle Struktur eines ZIMPL scripts

- Modellparameter
- Parameter Werte
- Variablen
- Zielfunktion
- Nebenbedingung

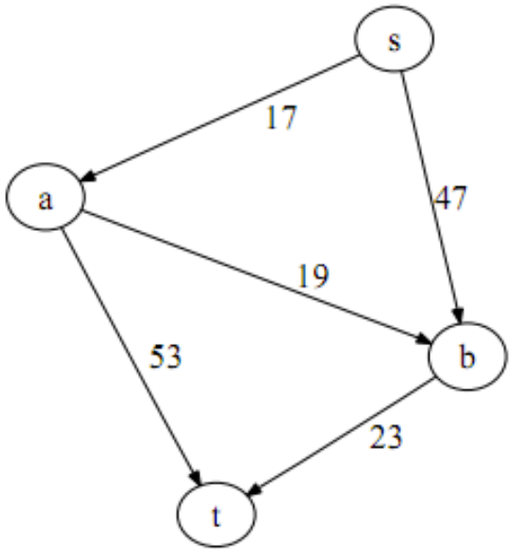
Beispielproblem

- Kürzeste Wege Problem in einem gerichteten Graph



$$
\begin{aligned}
\min \quad & 17x_{sa} + 47x_{sb} + 19x_{ab} + 53x_{at} + 23x_{bt} \\
\text{subject to} \quad & x_{sa} = x_{ab} + x_{at} \\
& x_{sb} + x_{ab} = x_{bt} \\
& x_{ij} \in \{0, 1\}, \text{for all i,j}
\end{aligned}
$$

Modellparameter



$$
\begin{aligned}
\min \quad & 17x_{sa} + 47x_{sb} + 19x_{ab} + 53x_{at} + 23x_{bt} \\
\text{subject to} \quad & x_{sa} = x_{ab} + x_{at} \\
& x_{sb} + x_{ab} = x_{bt} \\
& x_{ij} \in \{0, 1\}, \text{for all i,j}
\end{aligned}
$$

- Typischerweise eine Menge von sets:
- set V :={"a","b","s","t"};
- set A :={<"s","a">, <"s","b">, <"a","b">, <"a","t">, <"b","t">};
- defset dminus(v) := {<i,v> in A};
- defset dplus(v) := {<v,j> in A}

Set Operatoren

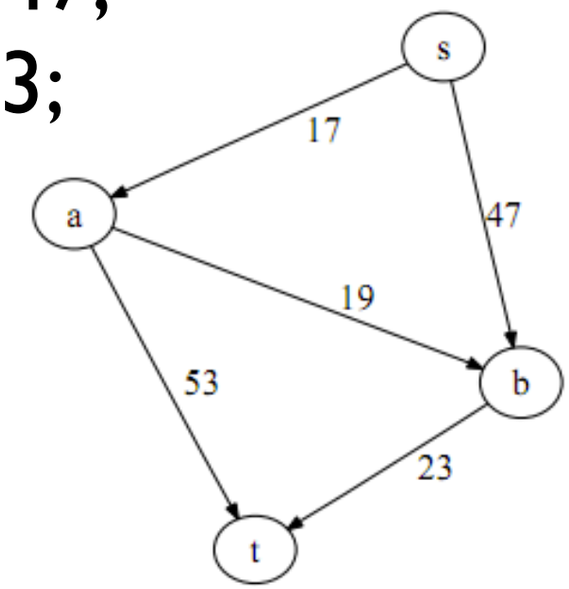
- z.B.

`A*B`, `A cross B`	cross product	$\{(x, y) \mid x \in A \wedge y \in B\}$
`A+B`, `A union B`	union	$\{x \mid x \in A \vee x \in B\}$
`union <i>` `in I: S`	ditto, of indexed sets	$\bigcup_{i \in I} S_i$
`A inter B`	intersection	$\{x \mid x \in A \wedge x \in B\}$
`inter <i>` `in I: S`	ditto, of indexed sets	$\bigcap_{i \in I} S_i$
`A\B, A-B,` `A without B`	difference	$\{x \mid x \in A \wedge x \notin B\}$

- u.v.m.

Parameterwerte

- Als Liste
  - param c[A]:= <"s","a"> 17, <"s","b"> 47, <"a","b"> 19, <"a","t"> 53,<"b","t"> 23;



- Oder als Tabelle:

```
set I := { 1 .. 10 };
set J := { "a", "b", "c", "x", "y", "z" };

param h[I*J] :=    | "a", "c", "x", "z"    |
                  |1|  12,  17, 99,     23 |
                  |3|   4,   3,-17, 66*5.5 |
                  |5| 2/3, -.4,  3, abs(-4)|
                  |9|   1,   2,  0,      3 | default -99;
```

- Oder aus Datei

Variablen

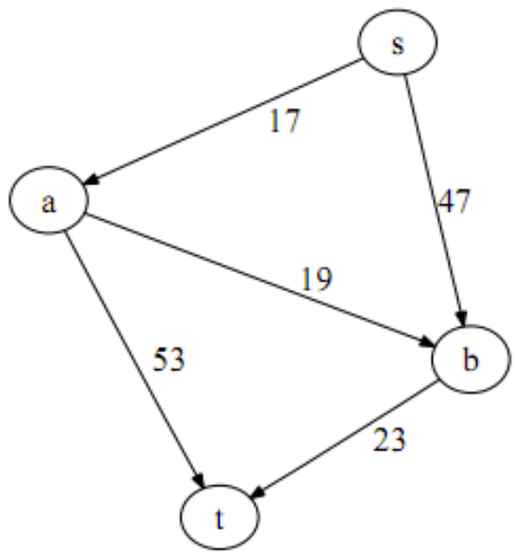
- Keywort „var“
- var x[A] binary; (real,binary,integer)

Zielfunktion

- (minimize/maximize) name : Lineare Funtion;
  - z.B.
    minimize cost: sum<i,j> in A: c[i,j] * x[i,j];

Nebenbedingungen

- Beginnen mit keyword „subto“
- subto fc: forall <v> in V - {"s","t"}:
  sum<i,v> in dminus(v): x[i,v] == sum<v,i> in dplus (v): x[v,i];
  (eingehende Kannte nach v erzwingt ausgehende Kante von v)
- subto uf: sum<s,i> in dplus("s"): x[s,i] == 1;
  (höchstens eine ausgehende Kante pro Knoten)



$$
\begin{aligned}
\min \quad & 17x_{sa} + 47x_{sb} + 19x_{ab} + 53x_{at} + 23x_{bt} \\
\text{subject to} \quad & x_{sa} = x_{ab} + x_{at} \\
& x_{sb} + x_{ab} = x_{bt} \\
& x_{ij} \in \{0, 1\}, \text{for all i,j}
\end{aligned}
$$

Ausdrücke

- So ziehmlich alles was das Herz begehrt:

`a^b, a**b`	a to the power of b	a^b, b must be integer
`a+b`	addition	$a + b$
`a-b`	subtraction	$a - b$
`a*b`	multiplication	$a \cdot b$
`a/b`	division	a/b
`a mod b`	modulo	$a \mod b$
`abs(a)`	absolute value	$\lvert a \rvert$
`sgn(a)`	sign	$x > 0 \Rightarrow 1, x < 0 \Rightarrow -1$, else 0
`floor(a)`	round down	$\lfloor a \rfloor$
`ceil(a)`	round up	$\lceil a \rceil$
`round(a)`	round towards zero	$\lfloor a \rceil$

- U.v.m.

Mengenoperationen

- sum index : term
  - sum<s,i> in dplus("s"): x[s,i] == 1;
- forall index : term
  - forall <n> in Nutrients:
    sum <f > in Food : data [f , n] * x [f] >=
    needed[n] ;

Beispielproblem:

Die Sache mit dem Labskaus

Ein großer Labskausproduzent hat zwei Rohstoffleiferanten und zwei Fabriken und beliefert drei Mensen. Die Transportkosten zwischen den Rohstofflieferanten und den Fabriken sowie den Fabriken und den Mensen sehen folgendermaßen aus:

	Fabrik 1	Fabrik 2
Lieferant 1	10 Euro/Tonne	15 Euro/Tonne
Lieferant 2	20 Euro/Tonne	15 Euro/Tonne

	Mensa 1	Mensa 2	Mensa 3
Fabrik 1	40 Euro/Tonne	20 Euro/Tonne	10 Euro/Tonne
Fabrik 2	30 Euro/Tonne	40 Euro/Tonne	20 Euro/Tonne

Lieferant 1 hat eine Kapazität von 10 Tonnen, Lieferant 2 von 15 Tonnen. Die drei Mensen benötigen 8 Tonnen, 14 Tonnen und 3 Tonnen. Die Kapazität der Fabriken ist zunächst einmal so groß, dass sie keine Rolle spielt.

```
set LIEFER := {"l1", "l2" };
set MENSA := {"m1","m2","m3"};
set FABRIK := {"f1","f2"};

set ELF := LIEFER*FABRIK; #Product: l2|{<"l1","f1">,<"l1","f2">,<"l2","f1">,<"l2","f2">}
set EFM := FABRIK*MENSA; #Product: l2|{<"f1","m1">,<"f1","m2">,<"f1","m3">,<"f2","m1">,<"f2","m2">,<"f2","m3">}


param kap[LIEFER] := <"l1"> 10, <"l2"> 15;

param plf[LIEFER*FABRIK] :=          |"f1","f2"|
                                  |"l1"| 10 , 15 |
                                  |"l2"| 20 , 15 |;

param pfm[FABRIK*MENSA] :=             |"m1","m2","m3"|
                                    |"f1"| 40 , 20 , 10 |
                                    |"f2"| 30 , 40 , 20 |;
param dem[MENSA] := <"m1"> 8, <"m2"> 14, <"m3"> 3;


var xRes[ELF] >=0;
var xLabs[EFM] >=0;

minimize cost: (sum<i,j> in ELF: plf[i,j]*xRes[i,j]) + (sum<k,l> in EFM: pfm[k,l]*xLabs[k,l]);


subto SKAP: forall <l> in LIEFER :
(sum <f> in FABRIK: xRes[l,f]) <= kap[l];

subto SDEM: forall <m> in MENSA :
(sum <f> in FABRIK: xLabs[f,m]) == dem[m];

subto SLFM: forall <f> in FABRIK :
(sum <m> in MENSA: xLabs[f,m]) - (sum <l> in LIEFER: xRes[l,f]) <=0;
```

Aufruf:

- (/usr/local/zibopt/bin/) zimpl labs.zpl

Output: (labs.lp,labs.tbl)

*.tbl enthält Mapping zwischen Namen in .zpl File und Namen im .lp File

Aufruf:

- (/usr/local/zibopt/bin/) zimpl labs.zpl

Output: (labs.lp,labs.tbl)

*.tbl enthält Mapping zwischen Namen in .zpl File und Namen im .lp File

„labs.lp“

```
\ This file was automatically generated by Zimpl
\ set LIEFER := {"l1", "l2" };
\ set MENSA := {"m1","m2","m3"};
\ set FABRIK := {"f1","f2"};
\ set ELF := LIEFER*FABRIK;
\ set EFM := FABRIK*MENSA;
\ do print ELF;
\ do print EFM;
\ param kap[LIEFER] := <"l1"> 10, <"l2"> 15;
\ param plf[LIEFER*FABRIK] :=     |"f1","f2"|          |"l1"| 10 , 15 |          |"l2"| 20 , 15 |;
\ param pfm[FABRIK*MENSA] :=      |"m1","m2","m3"|          |"f1"| 40 , 20 , 10 |          |"f2"| 30 , 40 , 20 |;
\ param dem[MENSA] := <"m1"> 8, <"m2"> 14, <"m3"> 3;
\ var xRes[ELF] >=0;
\ var xLabs[EFM] >=0;
\ minimize cost: (sum<i,j> in ELF: plf[i,j]*xRes[i,j]) + (sum<k,l> in EFM: pfm[k,l]*xLabs[k,l]);
\ subto SKAP: forall <l> in LIEFER : (sum <f> in FABRIK: xRes[l,f]) <= kap[l];
\ subto SDEM: forall <m> in MENSA : (sum <f> in FABRIK: xLabs[f,m]) <= dem[m];
\ subto SLFM: forall <f> in FABRIK : (sum <m> in MENSA: xLabs[f,m]) - (sum <l> in LIEFER: xRes[l,f]) <=0;
\Problem name: zimpl_test.zpl
...
```

```
Minimize
 cost:  +10 xRes$l1$f1 +15 xRes$l1$f2 +20 xRes$l2$f1 +15 xRes$l2$f2
+40 xLabs$f1$m1 +20 xLabs$f1$m2
  +10 xLabs$f1$m3 +30 xLabs$f2$m1 +40 xLabs$f2$m2 +20 xLabs
$f2$m3
Subject to
 SKAP_1:  + xRes$l1$f2 + xRes$l1$f1 <= 10
 SKAP_2:  + xRes$l2$f2 + xRes$l2$f1 <= 15
 SDEM_1:  + xLabs$f2$m1 + xLabs$f1$m1 == 8
 SDEM_2:  + xLabs$f2$m2 + xLabs$f1$m2 == 14
 SDEM_3:  + xLabs$f2$m3 + xLabs$f1$m3 == 3

SLFM_1:  - xRes$l2$f1 - xRes$l1$f1
     + xLabs$f1$m3 + xLabs$f1$m2
     + xLabs$f1$m1 <= 0

 SLFM_2:  - xRes$l2$f2 - xRes$l1$f2
     + xLabs$f2$m3 + xLabs$f2$m2
     + xLabs$f2$m1 <= 0
Bounds
 0 <= xRes$l1$f1 <= +inf
 0 <= xRes$l1$f2 <= +inf
 0 <= xRes$l2$f1 <= +inf
 0 <= xRes$l2$f2 <= +inf
 0 <= xLabs$f1$m1 <= +inf
 0 <= xLabs$f1$m2 <= +inf
 0 <= xLabs$f1$m3 <= +inf
 0 <= xLabs$f2$m1 <= +inf
 0 <= xLabs$f2$m2 <= +inf
 0 <= xLabs$f2$m3 <= +inf
End
```

	Fabrik 1	Fabrik 2
Lieferant 1	10 Euro/Tonne	15 Euro/Tonne
Lieferant 2	20 Euro/Tonne	15 Euro/Tonne

	Mensa 1	Mensa 2	Mensa 3
Fabrik 1	40 Euro/Tonne	20 Euro/Tonne	10 Euro/Tonne
Fabrik 2	30 Euro/Tonne	40 Euro/Tonne	20 Euro/Tonne

Lieferant 1 hat eine Kapazität von 10 Tonnen, Lieferant 2 von 15 Tonnen. Die drei Mensen benötigen 8 Tonnen, 14 Tonnen und 3 Tonnen. Die Kapazität der Fabriken ist zunächst einmal so groß, dass sie keine Rolle spielt.

(a) Es seien folgende Variablentypen definiert:

x_{ij} **Transportmenge von Lieferant** i **zu Fabrik** j

y_{jk} **Transportmenge von Fabrik** j **zu Mensa** k

Damit läßt sich das Problem folgendermaßen definieren:

$$
\begin{array}{lrcl}
\min & 10x_{11} + 15x_{12} + 20x_{21} + 15x_{22} + 40y_{11} + 20y_{12} + 10y_{13} + 30y_{21} + 40y_{22} + 20y_{23} & & \\
\text{s.t.} & x_{11} + x_{12} & \leq & 10, \\
 & x_{21} + x_{22} & \leq & 15, \\
 & y_{11} + y_{21} & = & 8, \\
 & y_{12} + y_{22} & = & 14, \\
 & y_{13} + y_{23} & = & 3, \\
 & -x_{11} - x_{21} + y_{11} + y_{12} + y_{13} & \leq & 0, \\
 & -x_{12} - x_{22} + y_{21} + y_{22} + y_{23} & \leq & 0, \\
 & x_{11},\ x_{12},\ x_{21},\ x_{22},\ y_{11},\ y_{12},\ y_{13},\ y_{21},\ y_{22},\ y_{23} & \geq & 0.
\end{array}
$$

Die ersten beiden Restriktionen spiegeln die Kapazitäten der Lieferanten wider. Die drei darauffolgenden Restriktionen sorgen dafür, daß der Bedarf jeder Mensa gedeckt wird, die letzten zwei Restriktionen verhindern, daß aus einer Fabrik mehr geliefert wird, als von den Lieferanten hereinkommt.

Lösen mit CPLEX

Lösen mit CPLEX

```
Problem 'zimpl_test.lp' read.
Read time =    0.00 sec.
CPLEX> optimize
Tried aggregator 1 time.
Aggregator did 3 substitutions.
Reduced LP has 4 rows, 7 columns, and 14 nonzeros.
Presolve time =    0.00 sec.

Iteration log . . .
Iteration:     1   Dual objective     =           720.000000

Dual simplex - Optimal:  Objective =  9.1000000000e+02
Solution time =    0.00 sec.  Iterations = 4 (0)

CPLEX> di so va -
Variable Name           Solution Value
xRes$l1$f1                 10.000000
xRes$l2$f1                  7.000000
xRes$l2$f2                  8.000000
xLabs$f1$m2                14.000000
xLabs$f1$m3                 3.000000
xLabs$f2$m1                 8.000000
All other variables in the range 1-10 are 0.
```