AMPL Tutorial

v. 2.2

Adam Kasperski
Department of Operations Research and Business Intelligence
Wrocław University of Science and Technology

2023

adam.kasperski@pwr.edu.pl

List of Examples

1 AMPL language

AMPL (*A Mathematical Programming Language*) is an algebraic modeling language that allows us to model various optimization problems, in particular linear, mixed integer and convex quadratic problems. AMPL supports many commercial and non-commercial solvers. A detailed description of AMPL can be found at: https://ampl.com. Free chapters of the book devoted to AMPL can be found at: https://ampl.com/resources/the-ampl-book/chapter-downloads/.

AMPL can be downloaded at: https://portal.ampl.com. There is a version for all popular operating systems, including MS Windows, MacOS, and Linux. The free version includes AMPL language, some popular free solvers (for example, HiGHS) and commercial solvers (for example, CPLEX), and a simple but quite functional IDE. The number of variables and constraints in models that use commercial solvers may be limited. You can also use AMPL with Python by downloading the *amplpy* package (https://ampl.com/api/nightly/python/getting-started.html).

Exercise 1.1. Download and install AMPL. Look at the free chapters of the book devoted to AMPL (https://ampl.com/resources/the-ampl-book/chapter-downloads/.) Read the Introduction chapter.

2 Mixed integer programming problem

A *mixed integer linear programming problem* is of the following form:

$$\begin{array}{lll} \min(\max) & \sum_{j=1}^{n} c_j x_j & \text{[linear objective function]} \\ & \sum_{j=1}^{n} a_{ij} x_j = (\leq, \geq) b_i \quad i = 1, \ldots, m & \text{[linear constraints]} \\ & x_j \in \boldsymbol{D}_j \qquad\qquad\qquad\quad j = 1, \ldots, n & \text{[domains of variables]} \end{array}$$

The problem consists of:

1. *Decision variables* $x_1, \ldots, x_n$. Each variable x_j has domain $\boldsymbol{D}_j$, where $\boldsymbol{D}_j$ can be the set of reals, integers or binary values.
2. An *objective function*, which has to be minimized or maximized.
3. A set of m *linear constraints*, which can take the form of equalities or inequalities.
4. *Input data*, that is all the constants c_j, a_{ij} and b_i.

Together, the decision variables, objective function, and constraints define a *model*. The model can be solved, and an optimal solution can be reported after providing some input data. In modern optimization languages, the model can be separated from the input data. So, we can use the same model for many different instances of input data, which is illustrated in Figure 1

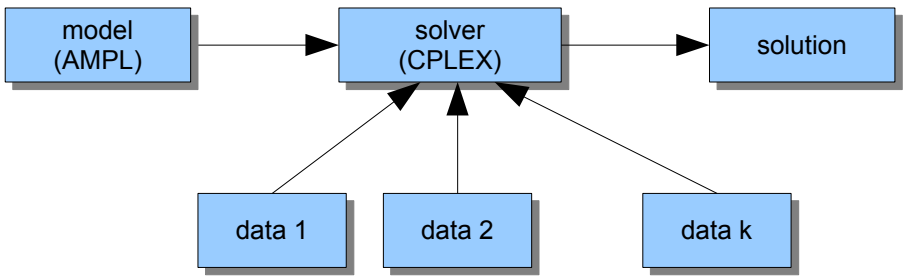


Figure 1: Separation of the model from input data.

3 Basic models

Example 3.1 (Basic example) Solve the following mixed integer programming problem:

$$\begin{array}{ll} \min & 3x_1 + 2x_2 + 3x_3 \\ & 2x_1 + 4.5x_2 + 2x_3 \leq 120 \\ & 2x_1 + 3x_2 - x_3 \geq 100 \\ & x_1, x_3 \geq 0 \\ & x_2 \geq 0 \text{ integer} \end{array}$$

We can input this model directly in the following way:

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
#decision variables
var x1 >=0;
var x2 >=0, integer;
var x3 >=0;
#objective function
minimize obj:  3*x1+2*x2+3*x3;
#constraints
subject to
c1:  2*x1+4.5*x2+2*x3<=120;
c2:  2*x1+3*x2-x3>=100;
#solve the model and display the results
solve;
display x1, x2, x3, obj;
end;
```

Exercise 3.1 Write and solve the model using AMPL IDE.

Solution:

1. Write the model in the IDE editor.
2. Save the file with the extension '.mod' (press Ctrl-S (Widows), or ⌘-S (Mac)).
3. Choose the command Send to AMPL (press Ctrk-R (Windows) or ⌘ -R (Mac)).

If you make any changes to your model you must save it (Ctrl-S, ⌘-S) before sending to AMPL. Otherwise, your changes will not be reflected.

Example 3.2 (Apex TV) The Apex Television Company has to decide on the number of 27 and 20-inch sets to be produced at one of its factories. Market research indicates that at most 20 of the 27-inch sets and 40 of the 20-inch sets can be sold per month. The maximum number of work hours available is 500 per month. A 27-inch set requires 20 work-hours and a 20-inch set requires 10 work-hours. Each 27-inch set sold gives a profit of $120 and each 20-inch set gives a profit of $80. Build and solve a linear programming model for this problem.

Solution 1. Let x_1 and x_2 be the number of 27-inch and 20-inch sets produced, respectively. The model for the problem is the following:

$$\begin{aligned} \max \quad & 120x_1 + 80x_2 \\ & x_1 \leq 20 \\ & x_2 \leq 40 \\ & 20x_1 + 10x_2 \leq 500 \\ & x_1, x_2 \geq 0 \end{aligned}$$

You can write it directly in AMPL:

```
#choose solver
option solver cplex;
```

```
#start new model (clear AMPL memory)
reset;
#decision variables
var x1 >=0; # 27-ich sets
var x2 >=0; # 20-ich sets
#objective function
maximize profit:  120*x1+80*x2;
#constraints
subject to
c1:  x1<=20;
c2:  x2<=40;
c3:  20*x1+10*x2<=500;
#solve the model and display the results
solve;
display x1, x2, profit;
end;
```

The constraints `c1:  x1<=20;` and `c2:  x1<=40;` define simple bounds on variables. They can be alternatively specified in the declaration of variables: **var** `x1>=0, x1<=20;` and **var** `x2>=0, x2<=40;`.

Solution 2. Consider a more general case, in which the company can produce more than two types of TV sets. Namely, let n be the number of possible TV set types. Demand, unit profit, and unit work hours for the ith type are equal to d_i, p_i and h_i, respectively. The number of available working hours is equal to h_limit. The model becomes then

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p_i x_i & \\ & x_i \le d_i & i = 1, \ldots, n \\ & \sum_{i=1}^{n} h_i x_i \le h\_limit & \\ & x_i \ge 0 & i = 1, \ldots, n \end{aligned}$$

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
#declaration of parameters
param n>0, integer;
param p{1..n};
param d{1..n};
param h{1..n};
param h_limit;
#decision variables with bounds
var x{i in 1..n} >=0,<=d[i];
#objective function
maximize profit:  sum{i in 1..n} p[i]*x[i];
#constraints
subject to
```

```
workload:  sum{i in 1..n} h[i]*x[i]<=h_limit;
#provide data
data;
param n:=2;
param p:= [1] 120 [2] 80;
param d:= [1] 20 [2] 40;
param h:= [1] 20 [2] 10;
param h_limit:=500;
#solve the model and display the results
solve;
display x, profit;
end;
```

The model is now more general. If your data changes, then it is enough to modify the data section of your model. Always separate model from data. Then you will be able to solve the model for various data sets.

Solution 3. AMPL language allows us to represent the model in more clear way, using sets. Assume that all possible types of TV sets are contained in the set $\boldsymbol{T}$. Then the model can be rewritten as follows:

$$\begin{aligned} \max \quad & \sum_{i \in \boldsymbol{T}} p_i x_i & \\ & x_i \leq d_i & i \in \boldsymbol{T} \\ & \sum_{i \in \boldsymbol{T}} h_i x_i \leq h\_limit & \\ & x_i \geq 0 & i \in \boldsymbol{T} \end{aligned}$$

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
#declaration of parameters
set T;
param p{T};
param d{T};
param h{T};
param h_limit;
#decision variables with bounds
var x{i in T} >=0, <=d[i];
#objective function
maximize profit:  sum{i in T} p[i]*x[i];
#constraints
subject to
workload:  sum{i in T} h[i]*x[i]<=h_limit;
#provide data
data;
param:  T: p, d, h:= '27-inch' 120 20 20
                     '20-ich' 80 40 10;
param h_limit:=500;
```

```
#solve the model and display the results
solve;
display x, profit;
end;
```

Example 3.3 (Knapsack problem) Joe C. wants to pack his knapsack, which has capacity C. He prepared a list of n items each of which has a weight w_i and a value p_i. The Joe's problem is to select a subset of the items so that the knapsack is not overloaded and the total value of the selected items is maximal.

Let us define the binary variable $x_i \in \{0, 1\}$ for each item $i = 1, \ldots, n$, where $x_i = 1$ if and only if the ith item is chosen. The model has the following form:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p_i x_i \\ & \sum_{i=1}^{n} w_i x_i \leq C \\ & x_i \in \{0, 1\} \qquad i = 1, \ldots, n \end{aligned}$$

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
#input data
param n; #number of items
param C; #knapsack cpacity
param p{1..n}; #item profitd
param w{1..n}; #item weights
#decision variables
var x{1..n} binary;
#objective function
maximize value:sum{i in 1..n} p[i]*x[i];
#constraints
subject to
knapsack:  sum{i in 1..n} w[i]*x[i]<=C;
data;
#provide data
#solve the model and display results
solve;
display x, value;
end;
```

If you want to present the solution in a more readable way, use the following code (instead of the command `display x;`):

```
print "Joe should take items:  ";
for{i in 1..n}
    if x[i]==1 then print i;
```

Exercise 3.2 Solve the model for $n = 10$, $p = [2, 1, 4, 5, 3, 2, 8, 9, 1, 1]$, $w = [3, 1, 6, 8, 1, 13, 8, 10, 1, 1]$ and $C = 25$.

Solution. Add the following code to your model:

```
data;
param n:=10;
param C:=25;
param p:= [1] 2 [2] 1 [3] 4 [4] 5 [5] 3 [6] 2 [7] 8 [8] 9 [10] 1
          [11] 1;
param w:= [1] 3 [2] 1 [3] 6 [4] 8 [5] 1 [6] 13 [7] 8 [8] 10 [10] 1
          [11] 1;
```

! If you want to see the model for your input data use the command: **expand** value, c1; You can write this command just before the **dispay** command in your model.

! If you want to write the results to a text file, say 'results.txt' use the command: **display** x, value>'results.txt' or **print**{i in **in** 1..n} x[i] , value>'results.txt'. Check the differences between these two commands.

Example 3.4 (Diet problem) Consider a problem of buying foods to meet some nutritional requirements. We have a set of different types of food $\boldsymbol{F}$. The price per unit of the jth food is p_j. We also have a set of nutrients $\boldsymbol{N}$. The nutritional requirement for the nutrient $i \in \boldsymbol{N}$ is at least b_i. Let a_{ij} be the amount of the ith nutrient in one unit of the jth food. We have to decide how many units of each food to buy in order to satisfy the nutritional requirements

Suppose x_j units of the jth food are bought, where $j \in \boldsymbol{F}$. The model has the following form:

$$\begin{aligned} \min \quad & \sum_{j \in \boldsymbol{F}} p_j x_j & \\ & \sum_{j \in \boldsymbol{F}} a_{ij} x_j \geq b_i & i \in \boldsymbol{N} \\ & x_j \geq 0 & j \in \boldsymbol{F} \end{aligned}$$

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
#declaration of parameters
set F;
set N;
param p{F} >=0;
param b{N} >=0;
param a{N,F} >=0;
#decision variables
var x{F}>=0;
#objective function
minimize cost:sum{j in F} p[j]*x[j];
#constraints
subject to
Nutrient{i in N}:  sum{j in F} a[i,j]*x[j]>=b[i];
data;
#provide data
```

```
#solve the model and display results
solve;
display x,cost;
end;
```

Consider the following data for the problem:

	Bread	Milk	Cheese	Yoghurt	
Cost per unit	1.0	2.5	3.0	4.0	content
Sugar, g.	0.5	1	0.2	4	10
Fat, g.	0	5.0	9.0	7.0	6
Proteins, g.	4.0	11.7	10.0	17.0	30
Calories	90	120	106	110	300

The sample input data can be added as follows:

```
data;
param:  F: p:= 'Bread' 1, 'Milk' 2.5, 'Cheese' 3, 'Yogurth' 4;
param:  N: b:= 'Sugar' 10, 'Fat' 6, 'Proteins' 30, 'Calories' 300;
param a:     'Bread' 'Milk' 'Cheese' 'Yogurth' :=
'Sugar'      0.50  1.00  0.20  4.00
'Fat'        0.00  5.00   9.00  7.00
'Proteins'   4.00  11.7   10.0  17.0
'Calories'   90    120   106   110;
```

Example 3.5 (Transportation problem) There are m factories which must deliver a good to n shops. Factory i has a supply s_i and shop j has a demand d_j. The cost of sending 1 unit of the good from factory i to shop j equals c_{ij}. Find a cheapest transportation plan of the good from the factories to the shops.

Let $x_{ij} \geq 0$ be the number of units of the good sent from the i th factory to the j th shop. The model takes the following form:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\
& \sum_{j=1}^{n} x_{ij} \leq s_i \qquad i = 1,\ldots,m \\
& \sum_{i=1}^{m} x_{ij} \geq d_j \qquad j = 1,\ldots,n \\
& x_{ij} \geq 0 \qquad\quad i = 1,\ldots,m, j = 1,\ldots,n
\end{aligned}$$

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
# declaration of parameters
param m;
param n;
param s{1..m};
param d{1..n};
```

```
param c{1..m, 1..n};
#decision variables
var x{1..m, 1..n} >=0;
#objective function
minimize cost:sum{i in 1..m, j in 1..n} c[i,j]*x[i,j];
#constraints
subject to
factory{i in 1..m}:  sum{j in 1..n} x[i,j]<=s[i];
shop{j in 1..n}:  sum{i in 1..m} x[i,j]>=d[j];
data;
#provide data
#solve the model and display results
solve;
display x, cost;
end;
```

The effect of **display** x can be difficult to read. AMPL allows us to present the solution in a more readable way.

The following code can be used to show the optimal solution to the problem:

```
for{i in 1..m, j in 1..n}
    if x[i,j]>0 then
        print "Factory", i, "to shop",j, "-", x[i,j], "units";
```

Exercise 3.3 Solve the model for sample data.

Exercise 3.4 Implement the model using sets.

Exercise 3.5 Extend the model by introducing fixed setup costs in factories. Namely, when any positive amount is sent from factory i, then a fixed cost q_i must be paid.

Example 3.6 (Bin packing) A company wants to pack n items into bins. The item i has size $a_i \leq C$. All available bins have the same capacity equal to C. The goal is to pack the items into the bins using the minimum number of bins.

Let $x_{ij} \in \{0,1\}$ be a binary variable, where $x_{ij} = 1$ if item i is packed into the bin j and let $y_j \in \{0,1\}$ be a binary variable, where $y_j = 1$ if the bin j is used (is nonempty). Index $i = 1,\ldots,n$ and index $j = 1,\ldots,m$, where m should be an upper bound on the number of required bins. We can fix a trivial bound $m = n$, because n bins is always enough. However, this can significalntly increase the number of variables in the model. A better bound can be computed by using the following first-fit heuristic: consider the items in order $1,\ldots,n$ and n initially empty bins, numbered by $1,\ldots,n$; find the first bin into which the item i can fit; if such a bin is found, the item i is placed inside it; return the number of bins used. In typical case m will be significantly smaller than n. The first-fit heuristic

can be implemented in AMPL before the models is solved. The model takes the following form:

$$\min \sum_{j=1}^{m} y_j \tag{1}$$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad i = 1, \ldots, n \tag{2}$$

$$\sum_{i=1}^{n} a_i x_{ij} \leq C \qquad i = 1, \ldots, m \tag{3}$$

$$n y_j \geq \sum_{i=1}^{n} x_{ij} \qquad j = 1, \ldots, m \tag{4}$$

$$x_{ij}, y_j \in \{0, 1\} \qquad i = 1, \ldots, n, j = 1, \ldots, m \tag{5}$$

The objective function (1) minimizes the number of used bins. The constraints (2) mean that each item is packed to exactly one bin. The constraints (3) indicate that no bin is overloaded. Finally, the constraint (4) mean that bin j is used ($y_j = 1$) if at least one item is packed into j (i.e. $\sum_{i=1}^{n} x_{ij} > 0$).

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
# declaration of parameters
param n; #number of items
param m; #upper bound on the number of bins (trivially m=n)
param a{1..n}; #item sizes
param C; #capacity of the bins
#used in greedy algorithm
param freeCap{i in 1..n} default C; #free space in n available bins
#decision variables
var x{1..n, 1..m} binary; #x[i,j]=1 if item i is placed in bin j
var y{1..m} binary; #y[j]=1 if bin j is used
#objective function
minimize nb:  sum{i in 1..m} y[i];
#constraints
item{i in 1..n}:sum{j in 1..m} x[i,j]=1;
bincap{j in 1..m}:sum{i in 1..n} a[i]*x[i,j]<=C;
usedbin{j in 1..m}:n*y[j]>=sum{i in 1..n} x[i,j];
data;
#provide data:  n, a, C
#greedy algorithm
for{i in 1..n}{
      for{j in 1..n}{
        #if item i fits in bin j then plac i in j
        if a[i]<=freeCap[j] then {let freeCap[j]:=freeCap[j]-a[i];break;}
        }
     }
#count the number of used bins
let m:=0;
for{i in 1..n}
      if freeCap[i]<C then let m:=m+1;
```

```
#solve te problem to optimality
solve;
#show the results
print "heuristic number of bins = ", m;
print "optimal number of bins = ", nb;
#the optimal load
for{j in 1..m}{
      if y[j]==1 then print "Bin", j, " ";
      for{i in 1..n}
         if x[i,j]==1 then print i, " ";}
```

4 Financial models

Example 4.1 (Investment problem) A company must determine an optimal investment strategy for the next K months. At the beginning of each month the company receives G_i dollars in cash. At the beginning of each month the available cash can be invested in n possible investment options. The cash flows for the investments are given in matrix $\boldsymbol{Q} = [q_{ij}]$, where $i = 1, \ldots, n$ and $j = 1, \ldots, K$. Consider the following sample matrix $\boldsymbol{Q}$ for $n = 5$ and $K = 4$:

$$\boldsymbol{Q} = \begin{bmatrix} -1 & 0.5 & 1 & 0 \\ 0 & -1 & 0.5 & 1 \\ -1 & 1.2 & 0 & 0 \\ -1 & 0 & 0 & 1.9 \\ 0 & 0 & -1 & 1.5 \end{bmatrix}$$

So, \$1 invested in investment 1 at the beginning of month 1 yields \$0.5 at the beginning of mont 2 and \$1 at the beginning of mont 3; \$1 invested in investment 2 at the beginning of month 2 yields \$0.5 at the beginning of month 3 and \$1 at the beginning of month 4, etc. To ensure that the portfolio is diversified, the company requires that at most L can be invested in any single investment. Furthermore, the company can earn interest at $p\%$ per month by keeping uninvested cash in a bank. Returns from investments can be immediately reinvested. Determine an optimal investment strategy, which maximizes the cash at the beginning of the Kth month.

Let x_i, $i = 1, \ldots, n$, be the cash allocated to investment i, and let y_j, $j = 0, \ldots, K-1$ be the uninvested cash kept in bank at the beginning of the jth month. Furthermore, y_0 is the initial cash kept in the bank (we assume that $y_0 = 0$). The model is the following:

$$\max \sum_{i=1}^{n} q_{iK} x_i + (1+p) y_{K-1} + G_K \tag{6}$$

$$y_0 = 0 \tag{7}$$

$$G_i + \sum_{i=1}^{n} q_{ij} x_i + (1+p) y_{j-1} = y_j \qquad j = 1, \ldots, K-1 \tag{8}$$

$$0 \le x_i \le L \qquad i = 1, \ldots, n \tag{9}$$

$$y_j \ge 0 \qquad j = 0, \ldots, K-1 \tag{10}$$

Observe first that the quantity:

$$G_i + \sum_{i=1}^{n} q_{ij} x_i + (1+p) y_{j-1} \tag{11}$$

is the available (uninvested) cash at the beginning of the jth month. The objective function (6) expresses the cash at the beginning of the Kth month. The constraints (8) mean that uninvested cash should be kept in the bank. Because $y_j \geq 0$, the quantity (11) cannot be negative and we always have a sufficient amount of money for investments.

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
# declaration of parameters
param n; #number of investments
param K;#number of months
param p;#bank interest
param G{1..K}; #cash received at the begining od each month
param L;#maximal cash allocated to investment
param q{1..n,1..K};#cash flow matrix
# declaration of decision variables
var x{1..n}>=0,<=L;#cash allocated to investments
var y{0..K-1}>=0;#cash kept in bank
#objective function
maximize cash:  sum{i in 1..n} q[i,K]*x[i]+(1+p)*y[K-1];
#constraints
subject to
c0:  y[0]=0;
c1{j in 1..K-1}:  G[j]+sum{i in 1..n} q[i,j]*x[i]+(1+p)*y[j-1]=y[j]
data;
#provide data
#Solve the model
solve;
display cash, x
```

Exercise 4.1 Solve the model for $n = 5$, $K = 4$, $G = [100\ 000, 0, 0, 0]$\$, $L = 75\ 000$, $p = 0.08$ and the sample cash flow matrix $\boldsymbol{Q}$.

5 Network flow models

Example 5.1 (Minimum cost flow problem.) Let $G = (V, A)$ be a directed network. Each arc (i, j) has an associated cost c_{ij} and capacity $u_{ij} \geq 0$. Each node $i \in V$ has an integer number b_i representing its supply/demand. If $b_i > 0$, node i is a supply node; if $b_i < 0$, node i is a demand node and if $b_i = 0$, node i is a transshipment node. The goal is to find a cheapest transportation plan from the supply to the demand nodes. A sample instance of the problem, together with an optimal solution, is shown in Figure 2.

Let $x_{ij} \geq$ be the decision variable representing the flow (the number of units transported) along

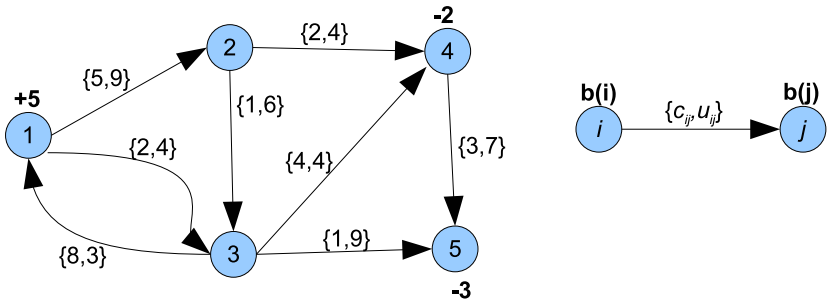


Figure 2: A sample instance of the minimum cost flow problem.

the arc $(i, j) \in A$. The model takes the following form:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{12}$$

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b_i \qquad i \in V \tag{13}$$

$$0 \le x_{ij} \le u_{ij} \qquad (i,j) \in A$$

The objective function (12) expresses the cost of the flow. The constraints (13) mean that the outflow minus inflow for each node i must be equal to $b(i)$.

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
#the set of nodes and the set of arcs
set V; #set of nodes
set A within V cross V;#set of arcs
param c{A}; #arc costs
param u{A} >=0; #arc capacities
param b{V}; #node supply/demand
#decision variables (arc flows)
var x{(i,j )in A} >=0,<=u[i,j];
#objective function
minimize cost:  sum{(i,j) in A} c[i,j]*x[i,j];
#constraints
subject to
balance{i in V}:  sum{(i,j) in A} x[i,j]-sum{(j,i) in A} x[j,i]=b[i];
data;
param:  V:=1 5 2 0 3 0 4 -2 5 -3;
param:  A: c u := 1,2 5 9;
                  1,3 2 4
                  2,3 1 6
                  2,4 2 4
                  3,1 8 3
                  3,4 4 4
                  3,5 1 9
                  4,5 3 7;
#Solve the model
solve;
```

```
display x;
```

Exercise 5.1 Modify the model to compute a shortest path from a given node s to a given node t.

Exercise 5.2 Modify the model to compute the maximum flow from a source node s to a sink node t.

The next two examples are applications of the minimum cost flow problem. In both, a particular network is constructed first.

Example 5.2 (Optimal plane loading) A company uses a plane with a capacity to carry at most p passengers. The plane visits the cities $1, 2, \ldots, n$ in this fixed order. The plane can pick up passengers at any city and drop the off at any other subsequent city. The number of passengers available at city i who want to travel to city j is b_{ij}. Let f_{ij} be the fare per passengers from city i to city j. The company wants to determine the number of passengers that the plane should carry between the various cities in order to maximize the total fare.

The problem can be represented as the minimum cost flow problem shown in Figure 3. The nodes $i - j$ for $i, j > 0$ represent the number of passengers who want to fly from i to j. The passengers taken by the plane are represented by solid arcs, while the remaining passengers are represented by dashed arcs. The nodes $i - 0$ represent the cities $i = 1, \ldots, n$.

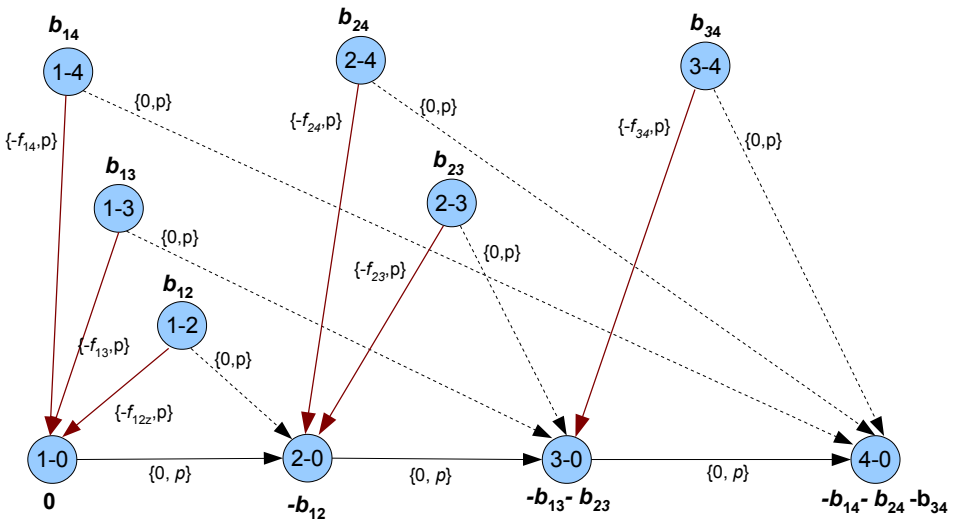


Figure 3: The optimal plan loading problem.

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
#Input data (must be provided)
param n;
param p;
set ROUTES within 1..n cross 1..n;
param passengers{ROUTES};
param fare{ROUTES};
#Define the underlying network
```

```
set V:=ROUTES union setof{i in 1..n} (i,0);
set A:=setof{(i,j) in ROUTES} (i,j,i,0)
      union setof{(i,j) in ROUTES} (i,j,j,0)
      union setof{i in 1..n-1} (i,0,i+1,0);
param c{(i,j,k,l) in A}:=if i==k then -fare[i,j] else 0;
param b{(i,j) in V}:=if j==0 then -sum{k in 1..i-1} passengers[k,i]
                     else passengers[i,j];
#Decision variables
var x{A}>=0,<=p;
#The minimum cost flow model
minimize total_fare:  sum{(i,j,k,l) in A} c[i,j,k,l]*x[i,j,k,l];
subject to
balance{(k,l) in V}:  sum{(k,l,i,j) in A} x[k,l,i,j]-
                      sum{(i,j,k,l) in A} x[i,j,k,l]=b[k,l];
#Provide input data
#Solve the model
solve;
```

In order to show the solution clearly, one can use the following code:

```
for{(i,j,k,l) in A}
     if i==k then
          print "Load from city", i, "to city", j, "is", x[i,j,k,l];
print "The total fare is", -total_fare;
```

Exercise 5.3 Solve the problem for the following input data, where $(\text{passengers}_i, \text{fare}_j)$ are the number of passengers and the fare for the connection from i to j, respectively. Fix the capacity of the plane $p = 100$.

	1	2	3	4
1		(80,100)	(100,400)	(80,600)
2			(90,200)	(90,500)
3				(60,300)

Example 5.3 (Labyrinth) We are given a labyrinth of size $n \times n$. On each square of the labyrinth a corridor or a wall is located. We can move to the right, left, up or down using only corridor boxes. The entrance to the labyrinth is located in square $[n-1, 1]$ and the exit is located in square $[2, n]$. We would like to find a shortest path from the entrance to the exit of this labyrinth. A sample labyrinth of size 11×11 together with a shortest path from the entrance to the exit is shown in Figure 4.

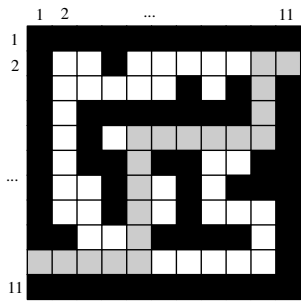


Figure 4: A sample labyrinth.

A labyrinth can be encoded as an $n \times n$ binary matrix lab, where $lab_{ij} = 0$ if there is a corridor on the square $[i, j]$ and $lab_{ij} = 1$ if there is a wall on the square $[i, j]$. Having defined the matrix lab,

we build a directed network $G = (V, A)$ whose nodes V are corridor boxes of the labyrinth and the arcs A represent all possible movements between the corridor boxes. The idea of the construction of G is shown in Figure 5. The node (i, j) is added to V and the arcs $(i, j, i-1, j)$, $(i, j, i, j+1)$ and $(i, j, i+1, j)$ are added to A.

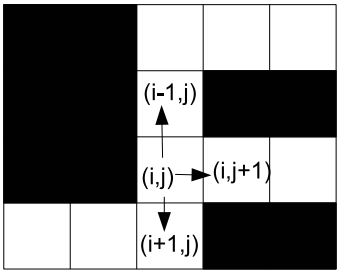


Figure 5: Encoding the labyrinth as a network.

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
#Input data
param n >0;
param lab{1..n,1..n};
set entry:={(n-1,1)};
set exit:= {(2,n)};
#The underlying network
set V:={i in 1..n, j in 1..n:  lab[i,j]==0};
set S{(i,j) in N }:={(k,l) in N: i==k and l==j+1} union
                    {(k,l) in N: i==k and l==j-1} union
                    {(k,l) in N: j==l and i==k+1} union
                    {(k,l) in N: j==l and i==k-1};
set A:={(i,j) in N,(k,l) in S[i,j]};
param c{A}:=1;
#The model is the same as in the Optimal plane loading (fix p = 1)
```

The following code can be used to visualize the path:

```
#Printing the path
print "The length of the path is", cost;
for{i in 1..n}{
    for{j in 1..n} printf "%s", if (i,j) not in N then " X "
    else if exists{(k,l) in S[i,j]} x[i,j,k,l]==1 or (i,j) in exit then
     " P " else " . ";
    printf "\n";}
```

After solving the sample instance in Figure 4, we get the following result:

```
X X X X X X X X X X X
X . X . . . . . . P P
X . . . . . X . X P X
X . X X X X X X X P X
X . X . P P P P P P X
X . X X P X X . . X X
X . . X P . X . X X X
X . . X P . X . . . X
X X P P P X X X X . X
P P P . . . . . . . X
X X X X X X X X X X X
```

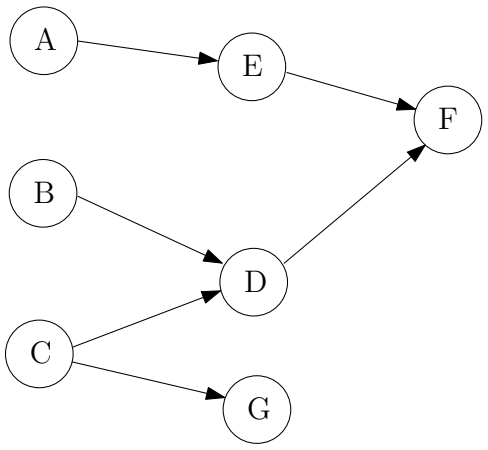
Exercise 5.4 Solve the model for the sample labyrinth shown in Figure 4.

Exercise 5.5 Assume that for each corridor a nonnegative cost of passing it is specified. We now seek a path in the labyrinth of the minimum total cost. Modify the model to take this additional requirement into account.

Exercise 5.6 Suppose that some traps are located in the labyrinth which work as follows. If explorer enters a corridor (i, j) then some other corridor (k, l) is blocked and cannot be entered. Therefore the traps are defined as quadruples (i, j, k, l) and explorer cannot use (i, j) and (k, l) simultaneously. Add the traps to the model.

6 Project scheduling

Example 6.1 (Project crashing) We are given a project consisting of a set $\boldsymbol{N}$ of tasks. For each task $i \in \boldsymbol{N}$ a set of tasks which directly precede it is specified. Additionally, for each activity $i \in \boldsymbol{N}$ a nominal duration time t_i, a unit cost of shortening its duration time c_i and a minimum duration time $\underline{t}_i$ are specified (a sample project is shown in Figure 6). Compute the minimum cost of completing the project in time T.



Activity i	Direct pred.	t_i	c_i	$\underline{t}_i$
A	-	2	1	1
B	-	4	2	2
C	-	5	1	3
D	B, C	6	2	1
E	A	3	5	1
F	E, D	4	4	2
G	C	4	1	3

Figure 6: A sample project.

Let $prec \subseteq \boldsymbol{N} \times \boldsymbol{N}$ be the precedence relation in $\boldsymbol{N}$, that is, $(i, j) \in prec$ if i is a direct predecessor of j. Let TS_i be the starting time of task $i \in \boldsymbol{N}$ and let x_i be the amount of shortening of task $i \in \boldsymbol{N}$. The model takes the following form:

$$
\begin{array}{ll}
\min & \sum_{i \in \boldsymbol{N}} c_i x_i \\
& TS_j \geq TS_i + t_i - x_i \quad (i,j) \in prec \\
& TS_i + t_i - x_i \leq T \quad i \in \boldsymbol{N} \\
& t_i - x_i \geq \underline{t}_i \quad i \in \boldsymbol{N} \\
& x_i, TS_i \geq 0 \quad i \in \boldsymbol{N}
\end{array}
$$

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
# declaration of parameters
set N; #set of tasks
param t{N}; #task duration times
param tmin{N}; #minimum task duration times
param c{N}; #unit shortening costs
param T; #project completion time
set prec within N cross N; #precedence relation
# decision variables
var TS{N}>=0; #starting times of tasks
var x{N}>=0; #shortening amounts of taska
# objective function
minimize cost:  sum{i in N} c[i]*x[i];
# constraints
subject to
c1{(i,j) in prec}:  TS[j]>=TS[i]+t[i]-x[i];
c2{i in N}:  TS[i]+t[i]-x[i]<=T;
c3{i in N}:  t[i]-x[i]>=tmin[i];
data;
#provide data
#solve the model and display results
solve;
display cost, x;
end;
```

The costs of shortening the project for each integer $T \in [T_{\min}, T_{\max}]$ can be computed and stored in file "results.txt" (notice how the value of $T_{\min}$ is computed - this is the smallest value for which the model is feasible). The following AMPL code can be used:

```
let T:=sum{i in N} t[i]; #estimation of the value of T_max
repeat
{
      solve;
      if solve_result = "infeasible" then break;
      print T, cost>"results.txt";
      let T:=T-1;
}
```

Exercise 6.1 Solve the model for the sample project shown in Figure 6. Draw a figure (use, for example, Excel) for the solutions stored in the file "results.txt".

7 Production planning models

Example 7.1 (Single item lot sizing model) A factory produces a product in periods $t = 1, \ldots, T$. The demand for the product in period t equals d_t and must be satisfied. The production cost, storage cost and setup cost in period t equal c_t, h_t and q_t, respectively. Once production in period t starts its quantity must belong to $[\underline{l}_t, \bar{l}_t]$, where the bounds of the interval denote the minimum and the maximum lot size in t. The storage capacity is equal to S. The factory wants to minimize the total cost and has to meet all demands on time. Determine an optimal production plan.

Let x_t be the amount of production in period $t = 1, \ldots, T$ and let s_t be the storage in period $t = 0, \ldots, T$, where s_0 is the initial storage (which is assumed to be 0). Let $y_t \in \{0, 1\}$, where $y_t = 1$ if production in period t is started (the setup cost q_t is then incurred). The model is the following:

$$\min \sum_{t=1}^{T} (c_t x_t + h_t s_t + q_t y_t) \tag{14}$$

$$s_0 = 0$$

$$s_{t-1} + x_t = d_t + s_t \qquad t = 1, \ldots, T \tag{15}$$

$$\underline{l}_t y_t \leq x_t \leq \bar{l}_t y_t \qquad t = 1, \ldots, T \tag{16}$$

$$0 \leq s_t \leq S \qquad t = 1, \ldots, T$$

The objective function (14) expresses the cost of the production plan. Constraints (15) are balance constraints, that is, in every period the starting inventory level plus the production amount is equal to the demand plus the end inventory level. Constraints (16) express the fact that once production is started ($y_t = 1$) its amount must be within $[\underline{l}_t, \bar{l}_t]$.

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
# declaration of parameters
set T; #set of periods 1..T
param c{T}; #production costs in periods
param d{T}; #demands in periods
param h{T}; #storage costs in periods
param q{T}; #setup costs in periods
param ld{T}; #minimum lot size in periods
param lu{T}; #maximum lot size in periods
param S; #storage capacity
#decision variables
var x{T}>=0; #production amount in periods
var s{T union {0}}>=0; #inventory levels in periods
var y{T} binary; #production stats/does not start
#objective function
minimize cost:  sum{t in T} (c[t]*x[t]+h[t]*s[t]+q[t]*y[t]);
#constraints
initial_storage:  s[0]=0;
inv{t in T}:  s[t-1]+x[t]=d[t]+s[t];
lotsizel{t in T}:  ld[t]*y[t]<=x[t];
lotsizeu{t in T}:  x[t]<=lu[t]*y[t];
storage{t in T}:  s[t]<=S;
```

```
data;
#provide data
#solve the model
solve;
display x,s,y, cost;
```

Exercise 7.1 Solve the model for 6 periods, $S = 80$, and the following data:

t	$\underline{l}_t$	$\bar{l}_t$	d_t	c_t	h_t	q_t
1	20	100	55	5	2	200
2	20	100	50	5	2	300
3	20	100	50	6	2	500
4	20	100	55	7	3	500
5	30	100	70	8	5	600
6	30	100	100	10	5	600

Example 7.2 (MRP II model) Let $\boldsymbol{S}$ be the set of *Stock Keeping Units*, containing items that are sold as well as components which are used to produce the items. For each item $i \in \boldsymbol{S}$ we are given: a lead time τ^i which is an estimate of the time between the release of an order to the shop floor or to a supplier and the receipt of the item; a minimum order/production quantity $\underline{l}^i$; initial inventory level I_0^i; a number of work-hours per period w^i required to produce one unit of the item; per period unit holding cost h^i and a total setup (changover) cost q^i. Let r^{ij} be the number of items i required to produce one unit of item j and d_t^i be the external demand for item i in period $t = 1, \ldots, T$. This demand must be satisfied. Finally L is the total numer of work-hours available in each period. We seek a production plan in periods $1, \ldots, T$ minimizing the total inventory and setup costs.

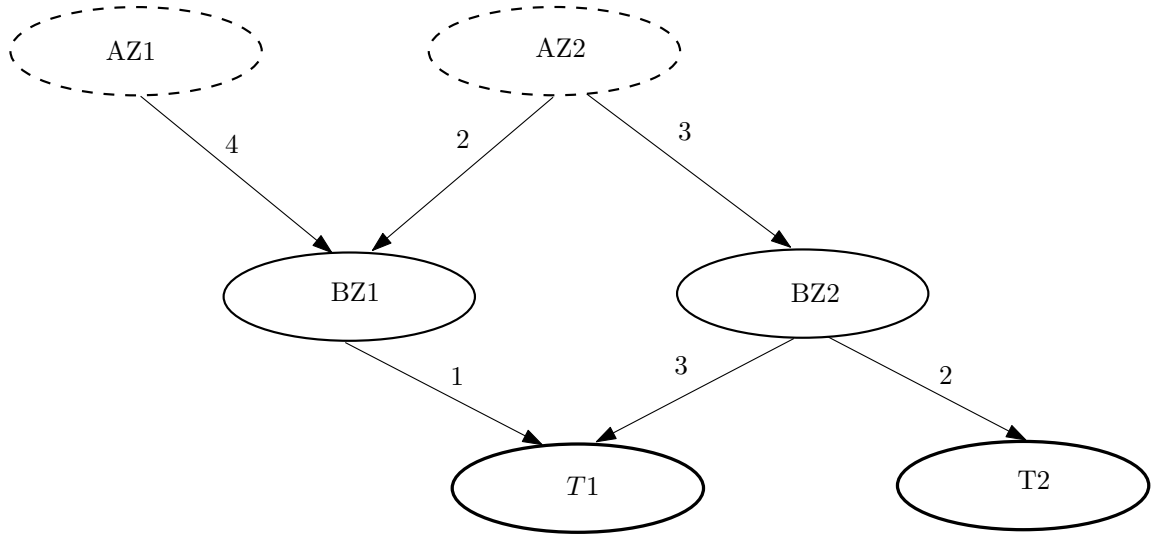


Figure 7: A sample instance of the MRP II problem.

A sample instance of the problem is shown in Figure 7. The set $\boldsymbol{S}$ contains 6 items, where AZ1, AZ2 are ordered items and BZ1, BZ2, T1, T2 are produced items. Items T1 and T2 are final products for which some external demand in periods $t = 1, \ldots, T$ should be specified. External demands for AZ1, AZ2, BZ1, BZ2 are equal to 0 in all periods. The arcs show the dependencies among the items. For example, producing one item BZ1 requires 3 items AZ1 and 2 items AZ2, thus $r_{AZ1,BZ1} = 3$ and $r_{AZ2,BZ1} = 2$. Let $x_t^i \geq 0$ be the quantity of $i \in \boldsymbol{S}$ to start/order in period t, $s_t^i \geq 0$ be the level of

inventory of $i \in \boldsymbol{S}$ in period t and $y_t^i \in \{0,1\}$, where $y_t^i = 1$ if $i \in \boldsymbol{S}$ is started/ordered in period t. The model takes the following form:

$$\min z = \sum_{t=1}^{T} \sum_{i \in \boldsymbol{S}} (h^i s_t^i + q^i y_t^i) \tag{17}$$

$$s_0^i = I_0^i \qquad i \in \boldsymbol{S}$$

$$s_{t-1}^i + x_{t-\tau^i}^i = d_t^i + \sum_{j \in \boldsymbol{S}} r^{ij} x_t^j + s_t^i \qquad i \in \boldsymbol{S}, t = 1, \dots T \tag{18}$$

$$\sum_{i \in \boldsymbol{S}} w^i x_t^i \leq L \qquad t = 1, \dots, T \tag{19}$$

$$\underline{l}_i y_t^i \leq x_t^i \leq M y_t^i \qquad i \in \boldsymbol{S}, t = 1, \dots T \tag{20}$$

$$x_t^i, s_t^i \geq 0 \qquad i \in \boldsymbol{S}, t = 1, \dots T$$

$$y_t^i \in \{0,1\} \qquad i \in \boldsymbol{S}, t = 1, \dots T$$

where M is a large constant. The objective function (17) is the total cost of storage and setup of the production plan. Constraints (18) are balance constraints. The quantity $s_{t-1}^i + x_{t-\tau^i}^i$ is the amount of item i received in period t. This amount comes from the initial inventory s_{t-1}^i and the production/order in period $t - \tau_i$. This amount of the items must be equal to the external demand d_t^i plus the internal demand $\sum_{j \in \boldsymbol{S}} r^{ij} x_t^j$ plus the end inventory level s_t^i. Constraints (19) mean that in each period the total number of required work hours cannot exceed L. Finally, constraints (20) mean that once production is started ($y_t^i = 1$), then its amount must be at least $\underline{l}_i$.

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
# declaration of parameters
set S; #Set of Stock Keeping Units
set T; #Set of periods
param L; #Work hours available in each period
param tau{S}; #Lead times of items
param I0{S}; #Initial inventory of items
param ld{S}; #Minimum order/production quantity of items
param h{S}; #Holding cost of items
param q{S}; #Setup cost of items
param w{S}; #Number of work-hours required per item and period
param r{S,S}; #Item dependencies
param d{S,T}; #External demands for items
param M:=1000000; #Big constant
# declaration of variables
var x{S,T}>=0 integer; #Quantity of items started in periods
var s{S,T}>=0; #Inventory levels for items in periods
var y{S,T} binary; #Is item started in period t
# objective function
minimize cost:  sum{t in T, i in S} (h[i]*s[i,t]+q[i]*y[i,t]);
# constraints
inv0{i in S}:  s[i,0]=I0[i];
inv{i in S, t in T}:  s[i,t-1]+if t>tau[i] then x[i,t-tau[i]]=
```

```
                    d[i,t]+sum{j in S} r[i,j]*x[j,t]+s[i,t];
workl{t in T}:  sum{i in S} x[i,t]*w[i]<=L;
setup1{i in S, t in T}:  x[i,t]<=M*y[i,t];
setup2{i in S, t in T}:  ld[i]*y[i,t]<=x[i,t];
data;
#Provide data
solve;
display x,cost;
```

Exercise 7.2 Solve the model for the instance shown in Figure 7 and the following data for each item:

i	τ^i	I_0^i	$\underline{l}^i$	w^i	h^i	c^i
AZ1	1	100	500	0	1	0
AZ2	1	200	400	0	1	0
BZ1	2	10	200	1	2	130
BZ2	2	0	200	4	3	120
T1	3	0	150	3	4	400
T2	3	0	100	2	5	500

Assume that $T = 10$, $L = 5000$ and that the external demands for T1 are $[0, 0, 0, 0, 0, 0, 180, 220, 230, 180]$ and the external demands for T2 are $[0, 0, 0, 0, 0, 0, 220, 180, 200, 220]$.

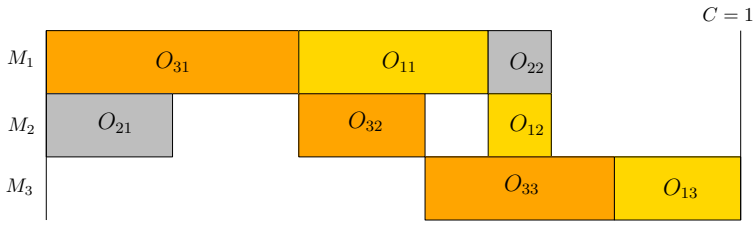
8 Job scheduling

Example 8.1 (Job shop problem) We are given a set of jobs $J = \{J_1, \ldots, J_n\}$. Job J_i consists of operations $O_{i1}, \ldots O_{in_i}$, which must be executed in order:

$$O_{i1} \to O_{i2} \to \cdots \to O_{in_i}.$$

Each operation O_{ij} must be executed on a fixed machine from the set $\{M_1, \ldots, M_m\}$ and its processing time equals p_{ij}. We seek a schedule whose makespan (the completion time of the last job) is minimal.

A sample instance of the problem together with an optimal schedule is shown in Figure 8. The optimal schedule has makespan 11.



Operation	Processing time	Machine
O_{11}	3	M_1
O_{12}	2	M_2
O_{13}	2	M_3
O_{21}	2	M_2
O_{22}	1	M_1
O_{31}	4	M_1
O_{32}	2	M_2
O_{33}	3	M_3

Figure 8: The optimal schedule for sample data.

Let S_{ij} be the starting time of operation O_{ij} on a suitable machine, and let C be the schedule makespan. The model is as follows:

$$
\begin{aligned}
&\min C \\
&S_{ij} \geq S_{ij-1} + p_{ij-1} && \forall O_{ij}, j > 1 && (21)\\
&(S_{ij} \geq S_{kl} + p_{kl}) \vee (S_{kl} \geq S_{ij} + p_{ij}) && \forall O_{ij}, O_{kl} \text{ using the same machine} && (22)\\
&C \geq S_{ij} + p_{ij} && \forall O_{ij} && (23)\\
&S_{ij} \geq 0 && \forall O_{ij} \\
&C \geq 0
\end{aligned}
$$

The first group of constraints (21) mean that operation O_{ij} must be processed after the operation O_{ij-1}. The second group of constraints (22) mean that for any two operations O_{ij} and O_{kl}, processed on the same machine, either O_{ij} is processed before O_{kl} or O_{kl} is processed before O_{ij}. The last group of constraints (23) defines the makespan C of the computed schedule. The constrains (22) are disjunctive constraints, that is, exactly one of $S_{ij} \geq S_{kl} + p_{kl}$ or $S_{kl} \geq S_{ij} + p_{ij}$ is violated. In order to convert them to linear constraints, binary variables $y_{ijkl} \in \{0, 1\}$ are defined, and the model is converted to the following mixed integer programming problem:

$$
\begin{aligned}
\min\ & C \\
& S_{ij} \geq S_{ij-1} + p_{ij-1} && \forall O_{ij}, j > 1. \\
& S_{kl} + p_{kl} - S_{ij} \leq M y_{ijkl} && \forall O_{ij}, O_{kl} \text{ using the same machine} \\
& S_{ij} + p_{ij} - S_{kl} \leq M(1 - y_{ijkl}) && \forall O_{ij}, O_{kl} \text{ using the same machine} \\
& C \geq S_{ij} + p_{ij} && \forall O_{ij} \\
& S_{ij} \geq 0 && \forall O_{ij} \\
& C \geq 0 \\
& y_{ijkl} \in \{0, 1\} && \forall O_{ij}, O_{kl} \text{ using the same machine}
\end{aligned}
$$

where M is a large constant. One can set M to the sum of the processing times of all operations.

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
# declaration of parameters
param n; #number of jobs
param m; #maximal number of operations for each job
param NMach; #number of machines
set OPER within 1..n cross 1..m; #set of operations
param p{OPER}>=0; #processing times of operations
param Mach{OPER} in 1..NMach; #machine assignment to operations
#big constant - can be the total processing times of operations
param M:=sum(i,j) in OPER} p[i,j];
#set of pairs operations executed on the same machine
set OPERSM:={(i,j,k,l) in OPER cross OPER: Mach[i,j]==Mach[k,l] and i<k};
#Decision vriables
var C; #schedule makespan
var S{OPER}>=0; #starting times of operations
var y{OPERSM} binary; #auxiliary variables or disjunctive constraints
#Objective function
minimize makespan:  C;
#Constraints
c1{(i,j) in OPER: (i,j-1) in OPER}:  S[i,j]>=S[i,j-1]+p[i,j-1];
```

```
c2{(i,j) in OPER}:  C>=S[i,j]+p[i,j];
c3{(i,j,k,l) in OPERSM}:  S[k,l]+p[k,l]-S[i,j]<=M*y[i,j,k,l];
c4{(i,j,k,l) in OPERSM}:  S[i,j]+p[i,j]-S[k,l]<=M*(1-y[i,j,k,l]);
#provide data
#Solve the model
solve;
display S, p, Mach, C;
```

An optimal schedule can be easily recovered from the values of S_{ij} of operations O_{ij}. Alternatively, one can use the following AMPL code to perform this task:

```
#visualize the schedule
param schedlen;
let schedlen:=C;
param Sched{1..NMach, 0..schedlen} default 0;
for{(i,j) in OPER}
     for{k in S[i,j]..S[i,j]+p[i,j]-1} let Sched[Mach[i,j], k]:=i;
option display_transpose -1500; #force transposition
display Sched;
```

Exercise 8.1 Solve the problem for sample data shown in Figure 8.

9 Routing problems

Example 9.1 (Vehicle routing) Transportation requests consist of the distribution of goods from a single depot, denoted as point 0, to a given set of $1, \ldots, n$ customers. The amount to be delivered to the customer i is the customer's demand, which is given by $d_i \geq 0$. A fleet of K vehicles is available in the depot. It is assumed to be homogeneous, meaning that all have the same capacity $C > 0$, and operate at identical costs. A vehicle that services a subset of customers $S \subseteq \{1, \ldots, n\}$ starts at the depot, moves once to each of the customers in S, and finally returns to the depot. A vehicle moving from i to j incurs the travel cost c_{ij}. We assume that each customer is served by exactly one vehicle. A sample solution for 3 customers and 3 vehicles is shown in Figure 9. This solution is composed of three routes, one for each vehicle.

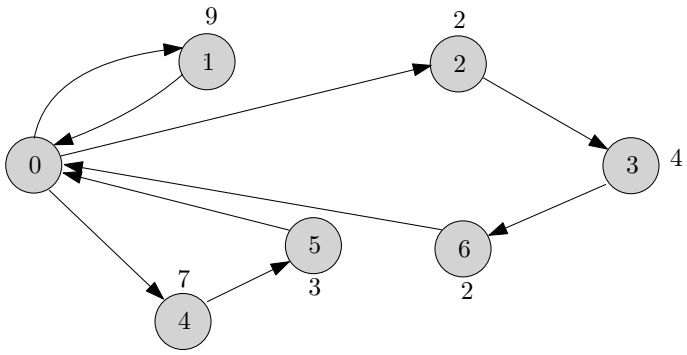


Figure 9: Three routes for $n = 6$, $K = 3$ and $C = 10$. The numbers at the nodes represent customer demands.

Let $x_{ij} \in \{0, 1\}$ for $i, j = 0, \ldots, n$, where $x_{ij} = 1$ if the connection from i to j is used by some vehicle. Furthermore, let us define variables u_i for each $i = 1, \ldots, n$, whose meaning will be described later. The model for the problem takes the following form:

$$\min \sum_{i=0}^{n} \sum_{i=0}^{n} c_{ij} x_{ij} \tag{24}$$

$$\sum_{j=1}^{n} x_{0j} = K \tag{25}$$

$$\sum_{i=1}^{n} x_{i0} = K \tag{26}$$

$$\sum_{j=0}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n \tag{27}$$

$$\sum_{i=0}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n \tag{28}$$

$$u_j - u_i + C(1 - x_{ij}) \geq d_j \qquad i, j = 1, \ldots, n \tag{29}$$

$$d_i \leq u_i \leq C \qquad i = 1, \ldots, n \tag{30}$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 0, \ldots, n$$

$$u_i \geq 0 \qquad i = 1, \ldots, n$$

The objective function (24) expresses the cost of the tours. Constraints (25) and (26) mean that exactly K vehicles enters and leaves the depot 0. The constraints (27) and (28) mean that each customer is visited exactly once by some vehicle. Finally, constraints (29) and (30) eliminate subtours (tours that do not contain the depot 0) and ensure that the demands of all visited customers are satisfied. To see this, consider a part of solution shown in Figure 10.

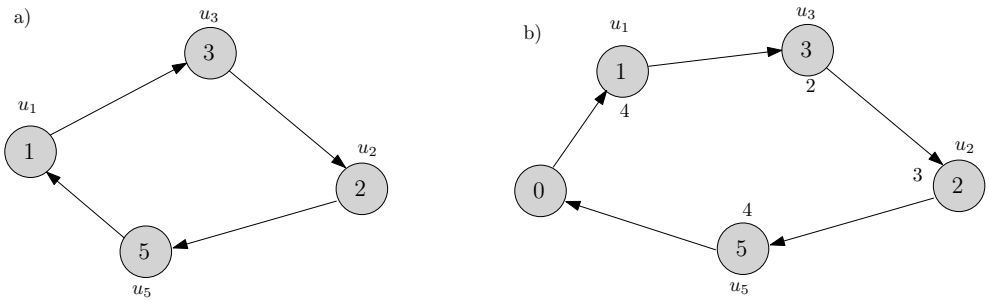


Figure 10: Infeasible solutions.

In Figure 10a an infeasible subtour which does not contain depot 0 is shown. Consider the constraints (29) for this subtour, i.e. $u_3 - u_1 \geq d_3$, $u_2 - u_3 \geq d_2$, $u_5 - u_2 \geq d_5$, $u_1 - u_5 \geq d_1$. Adding them together we get $0 \geq d_3 + d_2 + d_5 + d_1 > 0$, a contradiction.

In Figure 10b an infeasible tour for which the sum of demands exceeds the vehicle capacity $C = 10$. Consider the constraints (29) for this subtour, i.e. $u_3 - u_1 \geq 2$, $u_2 - u_3 \geq 3$, $u_5 - u_2 \geq 4$. Adding them together yields $u_5 - u_1 \geq 9$. From (30) we get $4 \leq u_5 \leq 10$ and $4 \leq u_1 \leq 10$, which is infeasible, because the largest value of $u_5 - u_1 = C - d_1$ is 6.

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
# declaration of parameters
param n; #number of customers
```

```
param K; #number of vehicles at the depot
param C; #capacity of vehicles
param d{1..n}; #demands of customers
param c{0..n,0..n}; #cost matrix
var x{0..n, 0..n} binary; #x[i,j]=1 if link (i,j) is chosen
var u{1..n}>=0; #auxiliary variables
#objective function
minimize cost:  sum{i in 0..n, j in 0..n} c[i,j]*x[i,j];
#constraints
base_start:  sum{j in 1..n} x[0,j]=K;
base_end:  sum{i in 1..n} x[i,0]=K;
point_start{i in 1..n}:  sum{j in 0..n} x[i,j]=1;
point_end{j in 1..n}:  sum{i in 0..n} x[i,j]=1;
demand{i in 1..n, j in 1..n}:  u[j]-u[i]+C*(1-x[i,j])>=d[j];
capacity{i in 1..n}:  d[i]<=u[i]<=C;
data;
#Provide data
#Solve the model and display the results
solve;
display x, cost;
```

Exercise 9.1 Solve the problem with sample data.

Example 9.2 (Traveling salesperson) A traveling salesperson starts his trip at depot 0. He must visit all customers $1, \ldots, n$ exactly once and return to depot 0. The cost of traveling from i to j equals c_{ij}. We seek a tour with the minimum total traveling cost.

This problem is a special case of the Vehicle Routing problem from Example 9.1. It is enough to set $K = 1$, $C = n$ and $d_i = 1$ for each $i = 1, \ldots, n$. In this case, the numbers u_i have the following interesting interpretation: $u_i \in \{1, \ldots, n\}$ is the position of the customer i in the computed tour. For example, if $u_3 = 5$, then customer 3 is the fifth customer visited. Using this fact, we can add more constraints to the problem. For example, if customer i must be visited before customer j, then we add the constraint $u_i \leq u_j$ to the model.

10 Sensitivity analysis

For linear programming problems, we can perform the so-called *sensitivity analysis* of the obtained optimal solution.

Example 10.1 (Basic sensitivity analsis) Consider the following linear programming problem:

$$\begin{aligned} \max z = \quad & 5x_1 + 4x_2 \\ & 6x_1 + 4x_2 \leq 24 \\ & x_1 + 2x_2 \leq 6 \\ & x_2 - x_1 \leq 1 \\ & x_2 \leq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

The optimal solution to this problem is $x_1 = 3$, $x_2 = 1.5$ with $z = 21$. We wish to compute the shadow prices of the constraints and the ranges for the objective function coefficients and the right-hand sides of the constraints.

```
#choose solver
option solver cplex;
option cplex_options 'sensitivity';
#start new model (clear AMPL memory)
reset;
#decision variables
var x1 >=0;
var x2 >=0,<=2;
#objective function
maximize obj:  5*x1+4*x2;
#constraints
subject to
c1:  6*x1+4*x2<=24;
c2:  x1+2*x2<=6;
c3:  x2-x1<=1;
#solve the model and display the results
solve;
display x1, x1.down, x1.up;
display x2, x2.down, x2.up;
display c1, c1.down, c1.up, c1.slack;
display c2, c2.down, c2.up, c2.slack;
display c3, c3.down, c3.up, c3.slack;
end;
```

First, we have to turn on the CPLEX option 'sensitivity'. We then have access to suffixes `down` and `up`, which allows us to obtain ranges for the objective function coefficients and constraint right-hand sides. We get `x1.down=2`, `x1.up=6`, which means that the optimal solution remains optimal for $c_1 \in [2, 6]$. For the first constraint, we get `c1=0.75`, `c1.down=20`, `c1.up=36`, `c1.slack=0` which means that the shadow (dual) price of the first constraint is 0.75, the range is $[20, 36]$ and the difference between the right and left-hand sides of the constraints is 0 (the constraint is tight at the optimum). The interpretation is the following. I we change the right-hand side of the first constraint by 1 unit, then the optimal value of the objective will change by 0.75. The shadow price 0.75 remains valid until the right-hand side belongs to the range $[20, 36]$.

The sensitivity analysis can be only performed for models with continuous variables.

Exercise 10.1 Perform sensitivity analysis for the diet problem (Section 3, Example 3.4).

Solution:
Add the line `option cplex_options 'sensitivity';` to the model and add the following lines to the AMPL code:

```
display x, x.down, x.up;
display Nutrient, Nutrient.down, Nutrient.up, Nutrient.slack
```

You will se the sensitivity report. Provide its interpretation.

Example 10.2 (Knapsack problem sensitivity) Consider the knapsack problem from Example 3.3. Suppose we want to find the optimal value of the knapsack depending on its capacity C, i.e. for

each integral value $C \in [C_{\min}, C_{\max}]$ we want to find the maximum objective function value of the problem:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p_i x_i \\ & \sum_{i=1}^{n} w_i x_i \leq C \\ & x_i \in \{0,1\} \qquad i = 1, \ldots, n \end{aligned}$$

for fixed parameters p_i, w_i, $i = 1, \ldots n$.

We cannot use dual price for the constraint as binary variables are present in the model.

To perform the sensitivity analysis, we can use simulation, i.e., solve the model for each value $C \in [C_{\min}, C_{\max}]$ and store the results in a text file.

Exercise 10.2 Consider the instance of the problem with $n = 50$ items, where the item values and weights are random integers from the interval $[1, 20]$. Compute the maximum objective value for each $C \in [0, 500]$. Store the results in the text file 'knapsack_results.txt'.

Solution:

```
#choose solver
option solver cplex;
#start new model (clear AMPL memory)
reset;
#parameters
param n; #number of items
param C; #knapsack capacity
param p{1..n}:=round(Uniform(1,20)); #random item values
param w{1..n}:=round(Uniform(1,20)); #random item weights
param Cmin; #minimal capacity
param Cmax; #maximal capacity
#decision variables
var x{1..n} binary;
#objective function
maximize value:  sum{i in 1..n} p[i]*x[i];
#constraints
cap:  sum{i in 1..n} w[i]*x[i]<=C;
#data for the problem
data;
param n:=50;
param Cmin:=0;
param Cmax:=500;
#simulation
for{Cval in Cmin..Cmax}{
              let C:=Cval;
              solve;
              print C, value >'knapsack_results.txt';
}
```

Exercise 10.3 Make a graph showing the optimal objective values of each C by using Excel.

11 Game theory

Example 11.1 (Zero-sum games) Consider two player game in which player one chooses one strategy from m possible and player two chooses one strategy from n possible. If player 1 chooses strategy i and player two chooses strategy j, then player two pays amount a_{ij} to player 1. The goal is to compute the best strategies for the players to play the game.

For example, consider the following stone, paper, and scissors game:

	stone	paper	scissors
stone	0	-1	1
paper	1	0	-1
scissors	-1	1	0

A player mixed strategy is a probability distribution in the set of his/her strategies. Player one wants to compute a mixed strategy which guarantees him the maximum expected profit. On the other hand, player two seeks a mixed strategy which guarantees him the minimum expected loss. Player one solves the following linear programming problem:

$$\begin{aligned} \max \quad & t \\ & t \le \sum_{i=1}^{m} p_i a_{ij} \qquad j = 1, \dots, n \\ & p_1 + \cdots + p_m = 1 \\ & p_i \ge 0 \qquad i = 1, \dots, m \end{aligned}$$

Player two solves the following linear programming problem:

$$\begin{aligned} \min \quad & t \\ & t \ge \sum_{j=1}^{n} q_j a_{ij} \qquad i = 1, \dots, m \\ & q_1 + \cdots + q_n = 1 \\ & q_i \ge 0 \qquad i = 1, \dots, n \end{aligned}$$

The models for player one and two form a primal-dual pair. Hence, they have the same optimal objective function values (it is called the value of the game), and the optimal solution for one player is equal to the constraint shadow prices of the second player. So, we can solve the game by solving one model.

```
#choose solver
option solver cplex;
option cplex_options 'sensitivity';
#start new model (clear AMPL memory)
reset;
#parametres
param m;
param n;
#decision variables
var p{1..m}>=0;
var t;
```

```
#objective function
maximize payoff:  t;
#constraints
subject to
q{j in 1..n}:  t<=sum{i in 1..m} p[i]*A[i,j];
c1:  sum{i in 1..m} p[i]=1;
#provide data
#solve the model and display the results
solve;
display payoff; #value of the game
display p; #mixed strategy of player one
display q; #mixed strategy of player two (the optimal dual solution)
end;
```

Exercise 11.1 Solve the stone, paper, and scissors game

12 Stochastic models

Example 12.1 (Markowitz model) Suppose an investor wants to invest some money in n stocks. The stock return rates are a random vector $\boldsymbol{r} = (r_1, \ldots, r_n)$ with unknown probability distribution. The investor has a statistical sample of past return rates $\hat{\boldsymbol{r}}_1, \ldots, \hat{\boldsymbol{r}}_K$. This sample allows us to estimate the expected rates of return $\mu_1 = \mathbf{E}[r_1], \ldots, \mu_n = \mathbf{E}[r_n]$ and the covariance matrix $\boldsymbol{\Sigma}$ of the rates of return. The investor wants to build a portfolio that ensures the desired expected profit β and minimizes the investment risk (the variance).

Based on the observations $\hat{\boldsymbol{r}}_1, \ldots, \hat{\boldsymbol{r}}_K$, we can estimate the expected return μ_j of asset j and the covariance σ_{jk} between assets j and k:

$$\mu_j = \frac{1}{K} \sum_{i=1}^{K} \hat{r}_{ij},$$

$$\sigma_{jk} = \frac{1}{K} \sum_{i=1}^{K} \hat{r}_{ij} \hat{r}_{ik} - \mu_j \mu_k.$$

A portfolio is a vector $(x_1, \ldots, x_n)$ such that $x_j \in [0, 1]$ and $x_1 + \cdots + x_n = 1$, where x_j is the share of the jth asset in the portfolio. The expected return of the portfolio is equal to $\sum_{j=1}^{n} \mu_j x_j$ and its variance is $\sum_{j=1}^{n} \sum_{k=1}^{n} \sigma_{jk} x_j x_k$. We seek a portfolio with the smallest variance that guarantees the expected return at least β. Therefore, we have to solve the following model:

$$\begin{aligned}
\min \quad & \sum_{j=1}^{n} \sum_{k=1}^{n} \sigma_{jk} x_j x_k \\
& \sum_{j=1}^{n} \mu_j x_j \geq \beta \\
& \sum_{j=1}^{n} x_j = 1 \\
& x_j \geq 0 \qquad\qquad j = 1, \ldots, n
\end{aligned}$$

The model is not linear, because there are quadratic terms $x_j x_k$ in the objective function. However, it is a convex quadratic problem, so the CPLEX solver is able to solve it.

```
#choose solver
option solver cplex;
#parametres (must be provided)
param n; # number of companies
param K; # sample size
param r{1..K, 1..n}; # sample of the rate of returns
param beta; # the assumed expected return
#compute the expected returns and covariances
param mu{j in 1..n}:=1/K*sum{i in 1..K} r[i,j];
param cov{j in 1..n, k in 1..n}:=
                  1/K*sum{i in 1..K} r[i,j]*r[i,k]-mu[j]*mu[k];
#decision variables
var x{1..n}>=0;#portfolio
minimize variance:  sum{j in 1..n, k in 1..n} cov[j,k]*x[j]*x[k];
c1:  sum{j in 1..n} mu[j]*x[j]>=beta;
c2:  sum{j in 1..n} x[j]=1;
data;
#provide data
#solve the model and display the results
solve;
display variance, x;
```

In practise, the investor wants to compute the so-called Pareto front for portfolios. Then one portfolio is ultimately chosen from the Pareto front. That is, a set of portfolios is computed minimizing the variance for various values of β, say $\beta = 0, 0.01, 0.02, \ldots, \overline{\beta}$, where $\overline{\beta} = \max\{\mu_1, \ldots, \mu_n\}$. Notice that the model will be infeasible for any $\beta > \overline{\beta}$. This can be done in AMPL in the following way. First declare a new parameter

```
param maxbeta:=max{j in 1..n} mu[j];
```

and add the following code at the end of the file:

```
for{i in 0..maxbeta*100}{
     let beta:=i/100;
     solve;
     print beta, round(variance,3)>'pareto.txt';
}
```

In the file 'pareto.txt' the points which are located on the Pareto front will be provided. This front can be visualized after importing the points, for example, into Excel.

Exercise 12.1 Compute the Pareto front for sample data.

Example 12.2 (Farmer's stochastic problem) Farmer Tom must determine how many hectares of various types of grain to plant on his h hectares field. He can sells no more than D_i tons of the ith grain (production over D_i is lost). One hectare of the ith grain yields V_i tons of this grain. One ton of the ith grain gives a profit of q_i. Suppose that the demands D_i and yields V_i are uncertain. Namely, D_i is a random variable uniformly distributed in $[a_i, b_i]$ and V_i is a random variable with normal distribution with the expected value μ_i and standard deviation σ_i. Assume that all the random variables are mutually independent. Farmer Tom wants to maximize his expected profit under the condition that the probability that the production of all grains will not exceed the demands is at lest $1 - \epsilon$.

Let G be the set of grains and $x_i \geq 0$ be the number of hectares of the ith grain sawn. The model takes the following form:

$$\begin{aligned}
\max \quad & \sum_{i \in G} q_i \mu_i x_i \\
& \sum_{i=1}^{n} x_i \leq h \\
& \Pr\{V_i x_i \leq D_i : \ i \in G\} \geq 1 - \epsilon \\
& x_i \geq 0 \qquad\qquad\qquad\qquad\qquad\qquad i \in G.
\end{aligned}$$

This model is nonlinear and complex to solve exactly. We can compute an approximate solution using a Monte Carlo simulation. Let us independently generate a random sample d_{ki} and v_{ki} for $k = 1, \ldots, K$ and $i \in G$, where d_{ki} and v_{ki} are the kth realizations of demand and yield of the ith grain, respectively. The approximate solution can be computed by solving the following mixed integer model:

$$\begin{aligned}
\max \quad & \sum_{i \in G} q_i \mu_i x_i \\
& \sum_{i=1}^{n} x_i \leq h \\
& v_{ki} x_i - M(1 - z_k) \leq d_i \quad i \in G, k = 1, \ldots, K \\
& \sum_{k=1}^{K} z_k \geq K(1 - \epsilon) \\
& x_i \geq 0 \qquad\qquad\qquad\qquad i \in G.
\end{aligned}$$

The second and third constraints ensure that the demand constraints satisfy the fraction of at least $1 - \epsilon$ demand/yield realizations.

```
#choose solver
option solver cplex;
# Set the seed of random generator
option randseed 329092;
reset;
#declare parameters
set G;
param q{G}; #price of grain
param h; #field size
param K; #numer of realizations
param a{G}; #lower bound on demand
param b{G}; #upper bound on demand
param mu{G}; #mean value for yield
param std{G}; #standard deviation for yield
param M:=100000; #big constant
param eps ; #risk level
#Monte Carlo simulation
param h{k in 1..K, i in G}:=Uniform(a[i],b[i]);
param v{k in 1..K, i in G}:=Normal(mu[i], std[i]);
#Variables
var x{G}>=0;
var z{1..K} binary;
#Model
maximize expprofit:  sum{i in G} q[i]*mu[i]*x[i];
```

```
subject to
c1:sum{i in G} x[i]<=h;
c3{k in 1..K, i in G}:  v[k,i]*x[i]-M*(1-z[k])<=d[k,i];
c4:sum{k in 1..K} z[k]>=K*(1-eps);
#provide data
#Solve the model
solve;
display x, expprofit;
```

We do not have to assume that the random variables are mutually independent. We did it to be able to perform the Monte Carlo simulation directly in the AMPL code. If the random vector $(D_1, \ldots, D_n, V_1, \ldots, V_n)$ has a joint probability distribution, we should provide a sample drawn from this distribution. This can be done using other statistical software. If the distribution is not available, past data can be used.

The time required to solve the model strongly depends on K. The greater K the harder is the model to solve, but the more exact solution is computed.

Exercise 12.2 Solve the problem for the following data (fix $\epsilon = 0.2$ and $K = 150$):

	p_i	a_i	b_i	μ_i	σ_i
Rye	30	400	500	10	3
Wheat	50	600	800	8	2
Corn	40	200	300	12	4

Exercise 12.3 Solve the problem for $\epsilon \in \{0, 0.01, 0.02, \ldots, 0.99\}$. Save the optimal objective values to a text file and prepare a figure using some external software (for example, Excel). Provide an interpretation of the results.

The solution obtained x_i, $i \in G$, is only an approximate one. This means that the true probability of satisfying all demand constraints can be different from $1 - \epsilon$. To better estimate this probability, we can perform an additional Monte Carlo simulation. Observe that for a fixed solution x_i, $i \in G$, we can generate a very large sample, say $N = 10^5$ or even larger. We can then count the number of realizations for which the constraints are satisfied. Let this number be equal to l. Then we can provide a good estimation:

$$\Pr\{V_i x_i \leq D_i : \ i \in G\} \approx \frac{l}{N}$$

In AMPL the simulation can be performed by adding the following code to the end of the file:

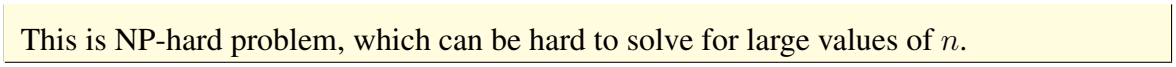
```
#Monte Carlo simulation of the optimal solution
param N:=100000; #Large sample size
var l:=0; #The number of feasible cases
for{j in 1..N}{
   if forall{i in G}
     Normal(mu[i], std[i])*x[i]<=Uniform(a[i],b[i]) then let l:=l+1}
display x;
print "Expected profit", expprofit;
print "Assumed probability of satisfying the constraints", 1-eps;
print "Simulated probability of satisfying the constraints", l/N;
```

Exercise 12.4 Estimate the true probability of the constraint violation for the data from the previous exercise.

13 Row generation

In some problems, the number of constraints is too large to list them explicitly in the model. We can then solve a sequence of models by adding the constraints one by one until an optimal solution is achieved. The AMPL language allows to perform this task. Furthermore, the model can be separated from the algorithm, which can be implemented in an additional "*.run" file.

Example 13.1 (Traveling salesperson, row generation) A traveling salesperson starts his trip at city 1. He must visit all the remaining cities $2, \ldots, n$ exactly once and return to city 1. The cost of traveling from city i to city j equals c_{ij}. We seek a tour with the minimum total traveling cost.



Let $x_{ij} \in \{0, 1\}$ be a binary variable such that $x_{ij} = 1$ if the connection from city i to city j is used by the tour. The model takes the following form:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} & \\ & \sum_{i=1}^{n} x_{ij} = 1 & j = 1, \ldots, n \\ & \sum_{j=1}^{n} x_{ij} = 1 & i = 1, \ldots, n \\ & \sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 & S \in \mathcal{S}, \end{aligned} \tag{31}$$

where

$$\mathcal{S} = \{S \subseteq \{1, \ldots, n\} : 1 \leq |S| < n\}$$

The first two groups of constraints (called assignment constraints) ensure that each city is visited exactly once. The third group of constraints ensures that there are no subtours in the solution. For example, if $n = 5$, then $2 \to 4 \to 5 \to 2$ is infeasible subtour, corresponding to $x_{24} = x_{45} = x_{52} = 1$. This subtour is eliminated by choosing $S = \{2, 4, 5\}$. The number of subtour-eliminating constraints is exponential (equal to 2^{n-1}), so it is not possible to list all of them explicitly. Suppose that we solved the model for some $\mathcal{S}' \subseteq \mathcal{S}$. If the solution computed does not contain any subtour, then it must be optimal. On the other hand, if it contains a subtour corresponding to $S \in \mathcal{S}$, then we add S to $\mathcal{S}'$ and resolve the model. The set S is called a *cut*. We can start with $\mathcal{S}' = \emptyset$ and repeat this procedure until a feasible (and thus optimal) solution is obtained or the maximum number of cuts is achieved. In this case, no optimal tour is computed, but a lower bound on the optimal solution is found. This can be done in AMPL as follows. First, the following file "tsp.mod" with the model is prepared:

```
#choose solver
option solver cplex;
#declare parameters and provide random data
param n:=21;
param c{1..n, 1..n}:=if i=j then 100000 else Uniform(1,200);
#additional parameters
param n_cuts; #current number of cuts
set CUTS{1..n_cuts} within 1..n; #cuts
#decision variables
var x{1..n, 1..n} binary;
```

```
#objective function
minimize cost:  sum{i in 1..n, j in 1..n} c[i,j]*x[i,j];
#constraints
subject to
c1{i in 1..n}:  sum{j in 1..n} x[i,j]=1;
c2{j in 1..n}:  sum{i in 1..n} x[i,j]=1;
c3{k in 1..n_cuts}:  sum{i in CUTS[k], j in CUTS[k]} x[i,j]
                                    <=card(CUTS[k])-1;
```

The row generation algorithm can be implemented in the following "tsp.run" file:

```
reset;
model tsp.mod
problem tsp:  x, cost, c1, c2, c3
param max_cut:=20; #fix the limit for the number of cuts
param tour symbolic; #encode the tour
param z; #auxiliary parameter

let n_cuts:=0;
solve tsp;
for{i in 1..max_cuts+1}{
    if i=max_cuts+1 then{
         print "Cut limit achieved.  Lower bound ", cost; break;}
    let n_cuts:=i;
    let tour:="1";
    let z:=1;
    let CUTS[i]:={1};#find a subtour starting at 1
    repeat{
         for{j in 1..n}
           if x[z,j]==1 then{
             let tour:=tour & "-" & j;
             let z:=j;
             let CUTS[i]:=CUTS[i] union {j};
             if j=1 then break;}
    }until z=1;
    if card(CUT[i])==n then{
         display "Optimal tour found";
         display tour, cost; break;}
    else print "Lower bound", cost;
    solve;}
```

At each iteration of the row generation algorithm, the objective value of the computed solution is a lower bound on the objective value of the optimal tour.

Exercise 13.1 Solve the problem for $n = 20$ and randomly generated costs. How many iterations on average are necessary to solve the problem?

Exercise 13.2 Implement in AMPL the following very fast nearest-neighbour heuristic: *starting at city 1 always go to the closest unvisited city. Return to city 1 from the last visited city*. Compare the cost of the obtained heuristic tour with the lower bounds computed by the row generation algorithm.

Example 13.2 (Minmax regret knapsack, row generation) Consider the following version of 0-1 knapsack problem:

$$\begin{array}{ll} \min & \displaystyle\sum_{j=1}^{n} c_j x_j \\ & \displaystyle\sum_{j=1}^{n} w_j x_j \geq B \\ & x_j \in \{0,1\} \qquad j = 1, \ldots, n \end{array} \tag{32}$$

Suppose that the objective function coefficients (the item costs) are uncertain, i.e. the value of c_i is only known to belong to a closed interval $[\underline{c}_i, \overline{c}_i]$. Let $\mathcal{U}$ be the Cartesian product of these intervals. The set $\mathcal{U}$ is called *unertainty set* and contains all possible realizations of the costs (called scenarios). For a given scenario $\boldsymbol{c} \in \mathcal{U}$, we use $\mathrm{opt}(\boldsymbol{c})$ to denote the cost of the optimal solution for $\boldsymbol{c}$. The value of

$$\sum_{j=1}^{n} c_j x_j - \mathrm{opt}(\boldsymbol{c})$$

is called the regret of solution $\boldsymbol{x}$ under $\boldsymbol{c}$. The maximum regret of $\boldsymbol{x}$ can be computed as follows:

$$\sum_{j=1}^{n} \overline{c}_j x_j - \mathrm{opt}(\boldsymbol{c_x}) \tag{33}$$

where $\boldsymbol{c_x}$ is a scenario in which $c_i = \overline{c}_i$ if $x_i = 1$ and $c_i = \underline{c}_i$ if $x_i = 0$. We seek a solution minimizing the maximum regret, i.e. we solve the following problem:

$$\begin{array}{ll} \min \max\limits_{\boldsymbol{c} \in \mathcal{U}} & \displaystyle\sum_{j=1}^{n} c_j x_j - \mathrm{opt}(\boldsymbol{c}) \\ & \displaystyle\sum_{j=1}^{n} w_j x_j \geq B \\ & x_j \in \{0,1\} \qquad j = 1, \ldots, n \end{array}$$

which can be rewritten as follows:

$$\begin{array}{ll} \min & t \\ & \displaystyle\sum_{j=1}^{n} c_j x_j - \mathrm{opt}(\boldsymbol{c}) \leq t \quad \forall \boldsymbol{c} \in \mathcal{U} \\ & \displaystyle\sum_{j=1}^{n} w_j x_j \geq B \\ & x_j \in \{0,1\} \qquad j = 1, \ldots, n \end{array}$$

It is not difficult to show that we can replace the infinite scenario set $\mathcal{U}$ with the so-called set of extreme scenarios $\mathcal{U}_e$ in which $c_i \in \{\underline{c}_i, \overline{c}_i\}$ for $i = 1, \ldots, n$, which leads to the following equivalent problem:

$$\begin{array}{ll} \min & t \\ & \displaystyle\sum_{j=1}^{n} c_j x_j - \mathrm{opt}(\boldsymbol{c}) \leq t \quad \forall \boldsymbol{c} \in \mathcal{U}_e \\ & \displaystyle\sum_{j=1}^{n} w_j x_j \geq B \\ & x_j \in \{0,1\} \qquad j = 1, \ldots, n \end{array} \tag{34}$$

Problem (34) has a finite number of constraints. However, the number of constraints is exponential in n (i.e. $|\mathcal{U}_e| = 2^n$) and it is not possible to list all of them for larger n. A method of solving (34) consists in generating the constraints one by one until and optimal or suboptimal solution is attained.

If we solve (34) for a subset $\mathcal{U}'_e \subset \mathcal{U}_e$ of scenarios, then we get a lower bound LB on the optimal objective value of (34). Suppose that $\boldsymbol{x}$ is an optimal solution to (34) for $\mathcal{U}'_e \subset \mathcal{U}_e$. We can compute the maximum regret of $\boldsymbol{x}$ using (33), which gives us an upper bound UB on the optimal objective value of (34). If $|UB - LB| \leq \epsilon$, for some specified small threshold ϵ, then $\boldsymbol{x}$ is an optimal solution. Otherwise, the worst scenario $\boldsymbol{c_x}$ for $\boldsymbol{x}$ (called a cut) is added to $\mathcal{U}'_e$ and problem (34) is again solved for the larger $\mathcal{U}'_e$. The algorithm starts from $\mathcal{U}'_e = \emptyset$ and iteratively adds scenarios to $\mathcal{U}'_e$ until the optimality is attained or the prescribed number of iterations is exceeded. In the latter case a suboptimal solution is obtained with a gap of $(UB - LB)/LB$.

The algorithm terminates in a finite number of steps. Indeed, if a scenario $\boldsymbol{c_x}$, already existing in $\mathcal{U}'_e$, is obtained after solving (34) for $\mathcal{U}'_e$, then

$$LB \geq \sum_{j=1}^{n} \overline{c}_j \boldsymbol{x} - \text{opt}(\boldsymbol{c_x}) = UB$$

and $\boldsymbol{x}$ is optimal. As $|\mathcal{U}_e|$ is finite, only a finite number of steps is possible.

The models used in the algorithm are implemented in the following "kanspsack.mod" file:

```
#choose solver
option solver cplex;
#problem data
param n;
param cl{1..n}; #cost lower bounds
param cu{1..n}; #cost upper bounds
param w{1..n}; #weights
param B; #knapsack capacity
param n_cuts; #number of cuts
param cut{1..n_cuts,1..n};#the cuts
param optcut{1..n_cuts}; #optimal costs for the cuts
#decision variables
var x{1..n} binary;
var y{1..n} binary;
var t>=0;
#Master problem
minimize regret:  t;
c1{i in 1..n_cuts}:  sum{j in 1..n} cut[i,j]*x[j]-optcut[i]<=t;
c2:  sum{j in 1..n} w[j]*x[j]>=B;
#Slave problem
minimize cost:  sum{j in 1..n} cut[n_cuts,j]*y[j];
c3:  sum{j in 1..n} w[j]*y[j]>=B;
```

The algorithm is implement in the following "knapsack.run" file:

```
reset;
model knapsack.mod;
data knapsack.dat;
problem knap_master:  x,t, regret, c1, c2;
problem knap_slave:  y, cost, c3;
param max_cuts:=10; #maximal number of cuts added
```

```
param LB; #lower bound
param UB; #upper bound
param regr; #regret of the current solution
param bestx{1..n}; #best solution found
let n_cuts:=0;
let LB:=0;
let UB:=10000000;
for{i in 1..max_cuts}{
    #solve the master problem
    solve knap_master;
    let LB:=t;
    let n_cuts:=n_cuts+1;
    for{j in 1..n} let cut[n_cuts,j]:=if x[j]==1 then cu[j] else cl[j];
    #solve the slave problem
    solve knap_slave;
    let optcut[n_cuts]:=cost;
    #test the optimality
    let regr:=sum{j in 1..n} cu[j]*x[j]-cost;
    if regr<UB then{
        let UB:=regr; #better solution found
        for{j in 1..n} let bestx[j]:=x[j];}
    if abs(UB-LB)<=0.01 then{
        display "Optimum found";
        display bestx, regr;
        break;}
    if n_cuts==max_cuts then{
        display "Suboptimal solution found";
        display bestx, regr, LB, UB;}
    }
```

The data for the problem should be included in a separate "knapsack.dat" file.

The method can be easily adopted to other combinatorial optimization problems. It is enough to modify the constraints c2 and c3 in the model file and add other constrains that describe the set of feasible solutions.

Exercise 13.3 Solve the problem for sample data included in "knapsack.dat" file.

Exercise 13.4 Solve the problem for some randomly generated data. Check how large problems can be solved to optimality and what is the quality of suboptimal solutions.

Exercise 13.5 Modify the file "knapsack.mod" to solve other combinatorial problems. For example, consider the multidimensional knapsack problem (i.e. the knapsack problem with more than one constraint).

14 Column generation

In some problems, the number of variables is too large to list them explicitly. We can then solve a sequence of models, generating basic variables, until the optimality conditions are satisfied.

Example 14.1 (Cutting stock, column generation) A sawmill has standard boards that are L inches wide. It receives orders for d_i boards that are $l_i \leq L$ inches wide, $i = 1, \ldots, m$. To satisfy demand, the standard board must be cut into pieces. The sawmill wants to minimize the number of standard boards cut. A jth pattern is vector $a = (a_{1j}, \ldots, a_{mj})$, where a_{ij} is the number of l_i inches board obtained from one L inches board. For example, if $L = 10$, $l_1 = 4$, $l_2 = 3$, $l_3 = 2$, then one possible pattern is $(1, 2, 0)$.

Let x_j be the number of L inches boards cut using the jth pattern, $j = 1, \ldots, n$, where n is the number of possible patterns. We have to solve the following problem:

$$\begin{array}{lll} \min z = & \displaystyle\sum_{j=1}^{n} x_j & \\ & \displaystyle\sum_{j=1}^{n} a_{ij} x_j \geq d_i & i = 1, \ldots, m \\ & x_j \geq 0, \text{ integer} & j = 1, \ldots, n \end{array} \tag{35}$$

The number of possible patterns n grows exponentially with L and m. As a consequence, n can be very large and the model cannot be implemented directly. We first relax the problem and write its dual:

$$\begin{array}{llll} & \min z = & \displaystyle\sum_{j=1}^{n} x_j & \\ a) & \displaystyle\sum_{j=1}^{n} a_{ij} x_j \geq d_i & i = 1, \ldots, m & \\ & x_j \geq 0 & j = 1, \ldots, n & \end{array} \qquad \begin{array}{llll} & \max z' = & \displaystyle\sum_{i=1}^{m} y_i d_i & \\ b) & \displaystyle\sum_{i=1}^{m} a_{ij} y_i \leq 1 & j = 1, \ldots, n & \\ & y_i \geq 0 & i = 1, \ldots, m & \end{array} \tag{36}$$

Observe that the constraint coefficients of the dual problem describe all feasible patterns. Let $\boldsymbol{x}^*$ be an optimal solution to (35) and let $\boldsymbol{x}'$ be an optimal solution to (36a). Convert $\boldsymbol{x}'$ to a feasible solution $\hat{\boldsymbol{x}}$ to (35) by rounding up all fractional components. Because $\boldsymbol{x}'$ is a basic feasible solution, the number of positive components in $\boldsymbol{x}'$ is at most m. Therefore

$$\sum_{j=1}^{n} \hat{x}_j \leq \sum_{j=1}^{n} x'_j + m \leq \sum_{j=1}^{n} x^*_j + m$$

The additive error of the approximate solution $\hat{\boldsymbol{x}}$ is at most m. Because in applications $\sum_{j=1}^{n} x^*_j >> m$, the additive term is negligible and the rounded solution $\hat{\boldsymbol{x}}$ is nearly optimal. We now show how to solve (36a). Let $\boldsymbol{x}$ be an optimal solution to (36a) for some subset of $n' \leq n$ patterns, that is, to the following problem:

$$\begin{array}{lll} \min z = & \displaystyle\sum_{j=1}^{n'} x_j & \\ & \displaystyle\sum_{j=1}^{n'} a_{ij} x_j \geq d_i & i = 1, \ldots, m \\ & x_j \geq 0 & j = 1, \ldots, n' \end{array} \tag{37}$$

Let $\boldsymbol{y}$ be the corresponding dual solution to (37). Then, by strong duality, $\boldsymbol{x}$ is optimal to (36a) if $\boldsymbol{y}$ is

feasible to (36b). To check the feasibility of $\boldsymbol{y}$ to (36b) we solve the following problem:

$$\begin{aligned} \max t = & \sum_{i=1}^{m} p_i y_i \\ & \sum_{i=1}^{m} l_i p_i \leq L \\ & p_i \geq 0, \text{integer} \end{aligned} \tag{38}$$

If $t^* \leq 1$, then $\boldsymbol{y}$ is feasible to (36b) and $\boldsymbol{x}$ is optimal to (36a). Otherwise, the optimal solution $\boldsymbol{p}$ to (38) describes a patters (a column of (36a)) which violates the optimality conditions for $\boldsymbol{x}$. The pattern $\boldsymbol{p}$ is added to (37), i.e. $n' := n' + 1$, $a_{in'} := p_i$, $i = 1, \ldots, m$, and the procedure is repeated. Initially, one can fix $n' := m$ and consider m patterns in which $a_{ii} = \lfloor L/l_i \rfloor$ for $i = 1, \ldots, m$ and $a_{ij} = 0$ if $i \neq j$.

The models are implemented in the following csp.mod file:

```
#choose solver
option solver cplex;

#Provide these parameters in file csp.dat
param L;
param m;
param d{1..m};
param l{1..m};
#----------------

param n; #number of patterns
param a{1..m,1..n}; #patterns

#Cutting stock problem
var x{1..n}>=0;
minimize Number:  sum{j in 1..n} x[j];
subject to
Order{i in 1..m}:  sum{j in 1..n} a[i,j]*x[j]>=d[i];

#Violating pattern problem
param y{1..m}; #dual solution to Cutting stock problem
var p{1..m}>=0 integer; #pattern
maximize RedC: sum{i in 1..m} p[i]*y[i];
subject to
Width_limit:  sum{i in 1..m} l[i]*p[i]<=L;
```

The column generation algorithm is implemented in the following csp.run file:

```
reset;
model csp.mod;
data csp.dat;

problem Cutting_Stock:  x, Number, Order;
problem Violating_pattern:  p, RedC, Width_limit;
```

```
#Initial patterns
let n:=m;
let{i in 1..m, j in 1..n} a[i,j]:= if i==j then floor(L/l[i]) else 0;

#Column generation algorithm
repeat{
     solve Cutting_Stock;
     let{i in 1..m} y[i]:=Order[i].dual;
     solve Violating_pattern;
     if RedC>1.000001 then{
          let n:=n+1;
          let{i in 1..m} a[i,n]:=p[i];}
    else break;}

#Print the approximate solution
print "Number of standard boards used:", Number;
for{j in 1..n} {
     if x[j]>0 then {
     print ceil(x[j]), "pattern:  ";
          for{i in 1..m} print a[i,j], "pieces ", l[i];}}
```

Exercise 14.1 Solve the problem for $L = 11$, $m = 4$, $l_1 = 4$, $l_2 = 3$, $l_3 = 2$, $l_1 = 5$ and $d_1 = 100$, $d_2 = 175$, $d_3 = 300$, $d_4 = 80$.

15 Multiobjective linear programming

A multiobjective linear programming problem can be stated as follows:

$$\begin{array}{ll} \min & (\boldsymbol{c}_1^T\boldsymbol{x}, \ldots, \boldsymbol{c}_K^T\boldsymbol{x}) \\ & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{array} \tag{39}$$

where $\boldsymbol{c}_k^T\boldsymbol{x}$ is the kth linear objective function. Let us denote by

$$\mathcal{X} = \{\boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$$

the set of feasible solutions. We will assume that $\mathcal{X}$ is nonempty and bounded. Set

$$\mathcal{F} = \{(\boldsymbol{c}_1^T\boldsymbol{x}, \ldots, \boldsymbol{c}_K^T\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{X}\} \subseteq R^K$$

is called an *objective space*. Each point $(y_1, \ldots, y_K) \in \mathcal{F}$ represents possible values of objective functions. Let $\boldsymbol{x}_k^*$ be an optimal solution with respect to the kth objective. Then $\boldsymbol{y}^{\mathrm{id}} = (\boldsymbol{c}_1^T\boldsymbol{x}_1^*, \ldots, \boldsymbol{c}_K^T\boldsymbol{x}_K^*)$ is called an *ideal point*. In most cases $\boldsymbol{y}^{\mathrm{id}} \notin \mathcal{F}$, which means that there is no solution $\boldsymbol{x} \in \mathcal{X}$, which simultaneously minimizes all objective functions.

We say that solution $\boldsymbol{x}_1$ *dominates* solution $\boldsymbol{x}_2$ if $\boldsymbol{c}_k^T\boldsymbol{x}_1 \leq \boldsymbol{c}_k^T\boldsymbol{x}_2$ for each $k = 1, \ldots, K$ with $\boldsymbol{c}_k^T\boldsymbol{x}_1 < \boldsymbol{c}_k^T\boldsymbol{x}_2$ for at least one k. Hence $\boldsymbol{x}_1$ is not worse than $\boldsymbol{x}_2$ under all objectives and is better than $\boldsymbol{x}_2$ under at least one objective. A feasible solution $\boldsymbol{x} \in \mathcal{X}$ is called *Pareto optimal* if there is no feasible solution that dominates $\boldsymbol{x}$. Let $\mathcal{P} \subseteq \mathcal{X}$ be the set of all Pareto optimal solutions. In multiobjective optimization problem we seek a solution which is Pareto optimal. The main problem is that $\mathcal{P}$ typically contains a lot of solutions (possibly infinite number of solutions). Therefore, we need a method of choosing one of them.

15.1 Lexicographical approach

Assume that the objectives are ordered with respect to their importance, that is $\boldsymbol{c}_1^T\boldsymbol{x}$ is the most important, $\boldsymbol{c}_2^T\boldsymbol{x}$ is the second most important etc. We first solve the linear programming problem

$$\begin{array}{ll}\min & \boldsymbol{c}_1^T\boldsymbol{x} \\ & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}\end{array} \tag{40}$$

obtaining an optimal solution $\boldsymbol{x}_1^*$ with the objective function value opt_1. We then solve the problem

$$\begin{array}{ll}\min & \boldsymbol{c}_2^T\boldsymbol{x} \\ & \boldsymbol{c}_1^T\boldsymbol{x} \leq (1+\epsilon_1)\mathrm{opt}_1 \\ & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}\end{array} \tag{41}$$

So, we minimize the second objective under the additional constraint that the first objective cannot be greater than $(1-\epsilon_1)\mathrm{opt}_1$. The value of $\epsilon_1 \geq 0$ is given by decision maker and expresses a tolerance about violating the optimality of the first objective. If $\epsilon_1 = 0$, then the optimal solution to (41) must be optimal to (40). A positive value of ϵ_1 allows us to relax this optimality. The procedure is repeated, namely in the kth step we solve the problem

$$\begin{array}{ll}\min & \boldsymbol{c}_k^T\boldsymbol{x} \\ & \boldsymbol{c}_i^T\boldsymbol{x} \leq (1+\epsilon_i)\mathrm{opt}_i \quad i = 1,\ldots,k-1 \\ & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}\end{array} \tag{42}$$

A final solution is obtained after executing the Kth step.

We get a Pareto optimal solution if $\epsilon_i = 0$ for each $i = 1,\ldots,K$. If some tolerances are positive that the final solution need not to be Pareto optimal.

Example 15.1 (Multiobjective optimization - lexicographical approach) Solve the multiobjective problem:

$$\begin{array}{ll}\min & (2x_1+x_2+3x_3+x_4, x_1+4x_2+2x_3+3x_4, x_2+2x_3+5x_4) \\ & 4x_1+2x_2+x_3 \geq 20 \\ & x_1+3x_2+3x_3+4x_4 \geq 10 \\ & x_i \geq 0 \qquad\qquad i = 1,\ldots,4\end{array}$$

with tolerances $\epsilon_1 = 0.05$, $\epsilon_2 = 0.1$, $\epsilon_2 = 0.1$. We start with the providing the following "lexicog.mod" file:

```
#choose solver
option solver cplex;

#Provide these parameters in file lexicog.dat
param n; #Number of variables
param m; #Number of constraints
param K; #Number of objectives
param c{1..K, 1..n}; #Objectives
param A{1..m, 1..n};#Constraint coefficients
param b{1..m};#Constraint RHS
```

```
param eps{1..K};#Tollerances

#Compute this
param best_obj{1..K} default 0;#Optimal objective values
param c_obj in 1..K; #Current objective function

var x{1..n}>=0;

minimize goal:  sum{j in 1..n} c[c_obj, j]*x[j];
subject to
c1{i in 1..m}:  sum{j in 1..n} A[i,j]*x[j]>=b[i];
c2{i in 1..c_obj-1}:  sum{j in 1..n} c[i,j]*x[j]<=(1+eps[i])*best_obj[i];
```

The algorithm is implemented in the following "lexicog.run" file:

```
reset;
model lexicog.mod;
data lexicog.dat;

problem lexikog:  x, goal, c1, c2;

#Lexicographical method
for{i in 1..K}{
    let c_obj=i;#Fix the current objective function
    solve lexikog;
    let best_obj[i]=goal;}
display x, goal;
```

In the first step, we solve

$$\begin{aligned} \min \quad & 2x_1 + x_2 + 3x_3 + x_4 \\ & 4x_1 + 2x_2 + x_3 \geq 20 \\ & x_1 + 3x_2 + 3x_3 + 4x_4 \geq 10 \\ & x_i \geq 0 \qquad\qquad i = 1, \ldots, 4 \end{aligned}$$

We get $\boldsymbol{x}_1^* = (4, 2, 0, 0)$ with $\text{opt}_1 = 10$. We solve

$$\begin{aligned} \min \quad & x_1 + 4x_2 + 2x_3 + 3x_4 \\ & 2x_1 + x_2 + 3x_3 + x_4 \leq (1 + 0.05) \cdot 10 \\ & 4x_1 + 2x_2 + x_3 \geq 20 \\ & x_1 + 3x_2 + 3x_3 + 4x_4 \geq 10 \\ & x_i \geq 0 \qquad\qquad i = 1, \ldots, 4 \end{aligned}$$

We get $\boldsymbol{x}_2^* = (4.2, 1.6, 0, 0.25)$ with $\text{opt}_2 = 11.35$. We solve

$$\begin{aligned} \min \quad & x_2 + 2x_3 + 5x_4 \\ & 2x_1 + x_2 + 3x_3 + x_4 \leq (1 + 0.05) \cdot 10 \\ & x_1 + 4x_2 + 2x_3 + 3x_4 \leq (1 + 0.1) \cdot 11.35 \\ & 4x_1 + 2x_2 + x_3 \geq 20 \\ & x_1 + 3x_2 + 3x_3 + 4x_4 \geq 10 \\ & x_i \geq 0 \qquad\qquad i = 1, \ldots, 4 \end{aligned}$$

We get $\boldsymbol{x}_3^* = (4.3, 1.9, 0, 0)$ with $\text{opt}_3 = 1.9$. Hence $\boldsymbol{x}_3^*$ is the final solution to our problem.

15.2 Weighted sum approach

Let $\boldsymbol{w} = (w_1, \ldots, w_K)$ be a vector of positive weights. Weight w_k expresses an importance of the kth objective. We replace the multiobjective problem with the following one:

$$\begin{aligned} \min \quad & \sum_{k=1}^{K} w_k \boldsymbol{c}_k^T \boldsymbol{x} \\ & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned} \tag{43}$$

The objective function of (43) is linear, so (43) is linear programming problem with one objective.

> **!** If all weights are positive, then each optimal solution to (43) is Pareto optimal.

In practical applications the objectives can be expressed in different units. For example, one can express the costs $\boldsymbol{c}_i$ and $\boldsymbol{c}_j$ in different currency. Even if the units are the same, we may estimate that $\boldsymbol{c}_i^T \boldsymbol{x} \in [0, 100]$ and $\boldsymbol{c}_j^T \boldsymbol{x} \in [0, 1000]$. The second objective can then dominate the first one. It is clear that the solution to the problem should not depend on units and, sometimes, should be also independent on the scale. Therefore, a normalization of the objectives is required.

Let $\boldsymbol{y}^{\mathrm{id}} = (y_1^{\mathrm{id}}, \ldots, y_K^{\mathrm{id}})$ be the ideal point, i.e. y_k^{id} is the minimum value of the objective function $\boldsymbol{c}_k^T \boldsymbol{x}$ for $\boldsymbol{x} \in \mathcal{X}$. It is easy to verify that $\boldsymbol{y}^{\mathrm{id}}$ is Pareto optimal. Let us define a *nadir point* $\boldsymbol{y}^{\mathrm{nad}} = (y_1^{\mathrm{nad}}, \ldots, y_K^{\mathrm{nad}})$, where

$$y_k^{\mathrm{nad}} = \max_{\boldsymbol{x} \in \mathcal{P}} \boldsymbol{c}_k^T \boldsymbol{x}, \tag{44}$$

so y_k^{nad} is the maximum of $\boldsymbol{c}_k^T \boldsymbol{x}$ over all $\boldsymbol{x}$ in the Pareto set $\mathcal{P}$. We get $\boldsymbol{c}_k^T \boldsymbol{x} \in [y_k^{\mathrm{id}}, y_k^{\mathrm{nad}}]$ for $k = 1, \ldots, K$. We can now use the normalized weighted objective function

$$\sum_{k=1}^{K} w_k \frac{\boldsymbol{c}_k^T \boldsymbol{x} - y_k^{\mathrm{id}}}{y_k^{\mathrm{nad}} - y_k^{\mathrm{id}}}.$$

Now $\frac{\boldsymbol{c}_k^T \boldsymbol{x} - y_k^{\mathrm{id}}}{y_k^{\mathrm{nad}} - y_k^{\mathrm{id}}} \in [0, 1]$ for each $k = 1, \ldots, K$. Unfortunately, problem (44) is difficult to solve. We can use the following approximation of the nadir point, which works well in practice:

$$\tilde{y}_k^{\mathrm{nad}} = \max_{i=1,\ldots,K} \boldsymbol{c}_k^T \boldsymbol{x}_i^*,$$

where $\boldsymbol{x}_i^*$ minimizes the objective $\boldsymbol{c}_i^T \boldsymbol{x}$. It holds $y_k^{\mathrm{id}} \leq \tilde{y}_k^{\mathrm{nad}} \leq y_k^{\mathrm{nad}}$. Finally, we solve the following linear programming problem:

$$\begin{aligned} \min \quad & \sum_{k=1}^{K} w_k \frac{\boldsymbol{c}_k^T \boldsymbol{x} - y_k^{\mathrm{id}}}{\tilde{y}_k^{\mathrm{nad}} - y_k^{\mathrm{id}}} \\ & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned} \tag{45}$$

Example 15.2 (Multiobjective optimization - weighted sum approach) Solve the multiobjective problem:

$$\begin{aligned} \min \quad & (2x_1 + x_2 + 3x_3 + x_4, x_1 + 4x_2 + 2x_3 + 3x_4, x_2 + 2x_3 + 5x_4) \\ & 4x_1 + 2x_2 + x_3 \geq 20 \\ & x_1 + 3x_2 + 3x_3 + 4x_4 \geq 10 \\ & x_i \geq 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i = 1, \ldots, 4 \end{aligned}$$

with weights $\boldsymbol{w} = (0.2, 0.6, 0.2)$. We start with the providing the following "weightedsum.mod" file:

```
#choose solver
option solver cplex;

#Provide these parameters in file weightedsum.dat
param n; #Number of variables
param m; #Number of constraints
param K; #Number of objectives
param c{1..K, 1..n}; #Objectives
param A{1..m, 1..n};#Constraint coefficients
param b{1..m};#Constraint RHS
param w{1..K};#Weights

#Compute this
param ideal{1..K} default 0;#Ideal point
param nadir{1..K} default -100000;#Approximation of nadir point
param c_obj in 1..K;

var x{1..n}>=0;

minimize obj:  sum{j in 1..n} c[c_obj, j]*x[j];
minimize weighted_obj:  sum{i in 1..K} w[i]*
(sum{j in 1..n} c[i,j]*x[j]-Ideal[i])/(Nadir[i]-Ideal[i]);
subject to
c1{i in 1..m}:  sum{j in 1..n} A[i,j]*x[j]>=b[i];
```

The algorithm is implemented in the following "weightedsum.run" file:

```
reset;
model weightedsum.mod;
data weightedsum.dat;

problem kthobj:  x, obj, c1;
problem wobj:  x, weighted_obj, c1;

#Weighted sum method
for{i in 1..K}{
     let c_obj=i;#Fix the current objective function
     solve kthobj;#Solve the problem under i-th objective
     let Ideal[i]:=obj;
     for{j in 1..K}{
         let c_obj:=j;
         let Nadir[j]:=max(Nadir[j],obj);}
    }
solve wobj;
display x, weighted_obj;
```

We first compute the ideal and approximation of the nadir point, obtaining $\boldsymbol{y}^{\text{id}} = (10, 8.18, 0)$ and

$\tilde{\boldsymbol{y}}^{\text{nad}} = (20, 12, 3.636)$. We then solve the problem (45):

$$\begin{array}{ll} \min & 0.1971x_1 + 0.7035x_2 + 0.4842x_3 + 0.864x_4 \\ & 4x_1 + 2x_2 + x_3 \geq 20 \\ & x_1 + 3x_2 + 3x_3 + 4x_4 \geq 10 \\ & x_i \geq 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i = 1, \ldots, 4 \end{array}$$

We get the solution $\boldsymbol{x} = (4.545, 0, 1.818, 0)$.

15.3 OWA minimization

Suppose that $\boldsymbol{c}_k$, $k = 1, \ldots, K$, is a realization of a random vector $\tilde{\boldsymbol{c}}$. Then $\boldsymbol{c}_k^T\boldsymbol{x}$ is the cost of $\boldsymbol{x}$ under realization $\boldsymbol{c}_k$. We have to find a criterion that aggregates the K costs $(\boldsymbol{c}_1^T\boldsymbol{x}, \ldots, \boldsymbol{c}_K^T\boldsymbol{x})$ into a single objective. This naturally leads to the multiobjective optimization problem (39). Let us introduce a vector of weights $\boldsymbol{w} = (w_1, \ldots, w_K)$ such that $w_k \geq 0$ and $w_1 + w_2 + \cdots + w_K = 1$. Given solution $\boldsymbol{x} \in \mathcal{X}$, let σ be a permutation of the set $\{1, \ldots, K\}$ such that

$$\boldsymbol{c}_{\sigma(1)}^T\boldsymbol{x} \geq \boldsymbol{c}_{\sigma(2)}^T\boldsymbol{x} \geq \cdots \geq \boldsymbol{c}_{\sigma(K)}^T\boldsymbol{x}$$

We define the *Ordere Weighted Averaging* (OWA) criterion:

$$\text{Owa}_{\boldsymbol{w}}(\boldsymbol{x}) = \sum_{i=1}^{K} w_i \boldsymbol{c}_{\sigma(i)}^T\boldsymbol{x}.$$

For example, let $\boldsymbol{c}_1 = (1, 2, 5)$, $\boldsymbol{c}_2 = (-2, 3, 1)$, $\boldsymbol{c}_2 = (0, 4, 2)$, $\boldsymbol{c}_4 = (3, 0, 1)$ be $K = 4$ realizations of random vector $\tilde{\boldsymbol{c}} = (\tilde{c}_1, \ldots, \tilde{c}_4)$. Consider solution $\boldsymbol{x} = (1, 3, 2)$, so $\boldsymbol{c}_1^T\boldsymbol{x} = 17$, $\boldsymbol{c}_2^T\boldsymbol{x} = 9$, $\boldsymbol{c}_3^T\boldsymbol{x} = 16$, $\boldsymbol{c}_4^T\boldsymbol{x} = 5$. Let $\boldsymbol{w} = (0.5, 0.2, 0.1, 0.2)$. Because $\boldsymbol{c}_1^T\boldsymbol{x} \geq \boldsymbol{c}_3^T\boldsymbol{x} \geq \boldsymbol{c}_2^T\boldsymbol{x} \geq \boldsymbol{c}_4^T\boldsymbol{x}$, we have $\sigma = (1, 3, 2, 4)$ and

$$\text{Owa}_{\boldsymbol{w}}(\boldsymbol{x}) = 0.5 \cdot \boldsymbol{c}_1^T\boldsymbol{x} + 0.2 \cdot \boldsymbol{c}_3^T\boldsymbol{x} + 0.1 \cdot \boldsymbol{c}_2^T\boldsymbol{x} + 0.2 \cdot \boldsymbol{c}_4^T\boldsymbol{x} = 13.6$$

Special cases of OWA:

1. $\boldsymbol{w} = (1, 0, \ldots, 0)$: $\text{Owa}_{\boldsymbol{w}}(\boldsymbol{x}) = \max\{\boldsymbol{c}_1^T\boldsymbol{x}, \ldots, \boldsymbol{c}_K^T\boldsymbol{x}\}$ is the maximal cost.
2. $\boldsymbol{w} = (0, 0, \ldots, 1)$: $\text{Owa}_{\boldsymbol{w}}(\boldsymbol{x}) = \min\{\boldsymbol{c}_1^T\boldsymbol{x}, \ldots, \boldsymbol{c}_K^T\boldsymbol{x}\}$ is the minimal cost.
3. $\boldsymbol{w} = (\underbrace{0, \ldots, 0}_{k-1 \text{ times}}, 1, 0, \ldots, 0)$: $\text{Owa}_{\boldsymbol{w}}(\boldsymbol{x})$ is the kth largest cost of $\boldsymbol{x}$
4. $\boldsymbol{w} = (\alpha, 0, \ldots, 0, 1 - \alpha)$: $\text{Owa}_{\boldsymbol{w}}(\boldsymbol{x})$ is the Hurwicz criterion, i.e. a convex combination of the maximal and the minimal cost of $\boldsymbol{x}$
5. $\boldsymbol{w} = (\frac{1}{K}, \ldots, \frac{1}{K})$: $\text{Owa}_{\boldsymbol{w}}(\boldsymbol{x}) = \frac{1}{K}\sum_{i=1}^{K} \boldsymbol{c}_i^T\boldsymbol{x}$ is the mean cost.
6. $\boldsymbol{w} = (\underbrace{\frac{1}{k}, \ldots, \frac{1}{k}}_{k \text{ times}}, 0, \ldots, 0)$: $\text{Owa}_{\boldsymbol{w}}(\boldsymbol{x})$ is the average of the k largest costs.

We consider the following problem:

$$\begin{array}{ll} \min & \text{Owa}_{\boldsymbol{w}}(\boldsymbol{x}) \\ & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{array} \tag{46}$$

Let y_k, $k = 1, \ldots, K$, be the kth largest cost of solution $\boldsymbol{x}$. Let $z_{ik} \in \{0, 1\}$, $i, k = 1, \ldots, K$, be binary variables, called *ordering variables*. The problem can be expressed as the following mixed integer program:

$$\begin{aligned} \min \quad & \sum_{k=1}^{K} w_k y_k \\ & y_k + M z_{ik} \geq \boldsymbol{c}_i^T \boldsymbol{x} \quad i, k = 1, \ldots, K \\ & \sum_{i=1}^{K} z_{ik} \leq k - 1 \qquad i = 1, \ldots, K \\ & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ & z_{ik} \in \{0, 1\} \qquad i, k = 1, \ldots, K \end{aligned} \tag{47}$$

where M is a big-constant. Consider the value of y_k for some fixed $k = 1, \ldots K$. Using the second constraint we can set the value equal to 1 to $k-1$ variables among z_{ik}. To achieve the minimum value of y_k we fix $y_{ik} = 1$ to $k - 1$ largest costs among $\boldsymbol{c}_i^T \boldsymbol{x}$, $i = 1, \ldots, K$. Then y_k becomes the kth largest cost among $\boldsymbol{c}_i^T \boldsymbol{x}$, $i = 1, \ldots, K$ and the objective function is the value of $\text{Owa}_{\boldsymbol{w}}(\boldsymbol{x})$.

Rearrangement inequality: for any $u_1 \geq u_2 \geq \cdots \geq u_K$, $v_1 \geq v_2 \geq \cdots \geq v_K$ and any permutation σ of $\{1, \ldots, K\}$:

$$\sum_{i=1}^{K} u_i v_i \geq \sum_{i=1}^{K} u_i v_{\sigma(i)}.$$

Consider the case when $w_1 \geq w_2 \geq \ldots w_K$. Then using the rearrangement inequality, we get

$$\text{Owa}_{\boldsymbol{w}}(\boldsymbol{x}) = \max_{\sigma \in \Pi} \sum_{i=1}^{K} w_i \boldsymbol{c}_{\sigma(i)}^T \boldsymbol{x}, \tag{48}$$

where Π is the set of all permutations of the set $\{1, \ldots, K\}$. We can now represent (48) in the following way:

$$\begin{aligned} \max \quad & \sum_{i=1}^{K} w_i \sum_{k=1}^{K} z_{ik} \boldsymbol{c}_k^T \boldsymbol{x} \\ & \sum_{i=1}^{K} z_{ik} = 1 \qquad k = 1, \ldots, K \\ & \sum_{k=1}^{K} z_{ik} = 1 \qquad i = 1, \ldots, K \\ & z_{ik} \in \{0, 1\} \qquad i, k = 1, \ldots, K \end{aligned} \tag{49}$$

where permutation σ is represented by the system of assignment constraints, i.e. $z_{ik} = 1$ if $\boldsymbol{c}_k^T \boldsymbol{x}$ is the ith largest cost. We can replace the constraints $x_{ij} \in \{0, 1\}$ with $x_{ij} \geq 0$ and write the dual to (48)

$$\begin{aligned} \min \quad & \sum_{i=1}^{K} (u_k + v_k) \\ & u_k + v_i \geq w_i \boldsymbol{c}_k^T \boldsymbol{x} \quad i, j = 1, \ldots, K \end{aligned} \tag{50}$$

By strong duality, the optimal objective values of (49) and (50) are equal. Finally, we can represent (46) with nonincreasing weights as follows:

$$\begin{array}{ll} \min & \sum_{i=1}^{K}(u_k + v_k) \\ & u_k + v_i \geq w_i \boldsymbol{c}_k^T \boldsymbol{x} \quad i,j = 1,\ldots,K \\ & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{array} \tag{51}$$

The general model (47) requires binary variables and is not convex. On the other hand, if $w_1 \geq w_2 \geq \ldots w_K$, then the problem can be represented as linear programming problem (49). In this case, the problem is convex and thus much easier to solve.

Example 15.3 (Minimizing OWA) Solve the problem:

$$\begin{array}{ll} \min & \mathrm{Owa}_{\boldsymbol{w}}(\boldsymbol{x}) \\ & 4x_1 + 2x_2 + x_3 \geq 20 \\ & x_1 + 3x_2 + 3x_3 + 4x_4 \geq 10 \\ & x_i \geq 0 \qquad\qquad\qquad\qquad i = 1,\ldots,4 \end{array}$$

for $K = 5$ cost vectors $\boldsymbol{c}_1 = (1,2,-1,0)$, $\boldsymbol{c}_2 = (0,5,2,1)$, $\boldsymbol{c}_3 = (3,-2,4,3)$, $\boldsymbol{c}_4 = (3,1,3,-2)$, $\boldsymbol{c}_5 = (1,1,1,2)$. Apply weights:

1. $\boldsymbol{w} = (0.5, 0.3, 0.2, 0, 0)$
2. $\boldsymbol{w} = (0, 0.2, 0.6, 0.2, 0)$

In point 1 the weight vector is nonincreasing, so we can apply model (51). The implementation in AMPL is as follows:

```
#choose solver
option solver cplex;
reset;

param n; #number of variables
param m; #number of constraints
param K; #number of cost realizations
param b{1..m};#constraint RHS
param c{1..K, 1..n};#cost realizations
param w{1..K};#OWA weights

var x{1..n}>=0;
var u{1..K};
var v{1..K};

minimize cost:  sum{k in 1..K}(u[k]+v[k]);
subject to;
c1{i in 1..K, k in 1..K}:  u[k]+v[i]>=w[i]*sum{j in 1..n} c[k,j]*x[j];
c2{i in 1..m}:  sum{j in 1..n} A[i,j]*x[j]>=b[i];
```

Exercise 15.1 Solve the model for the sample data and weight in point 1.

In point 2 the weight vector is not nonincreasing, so we can apply model (47). The implementation in AMPL is as follows:

```
#choose solver
option solver cplex;
reset;

param n; #number of variables
param m; #number of constraints
param K; #number of cost realizations
param b{1..m};#constraint RHS
param c{1..K, 1..n};#cost realizations
param w{1..K};#OWA weights
param M:=100000;#big constant

var x{1..n}>=0;
var y{1..K};
var z{1..K, 1..K} binary;

minimize cost:  sum{k in 1..K} w[k]*y[k];
subject to
c1{k in 1..K, i in 1..K}:  y[k]+M*z[i,k]>=sum{j in 1..n} c[i,j]*x[j];
c2{k in 1..K}:  sum{i in 1..K} z[i,k]<=k-1;
c3{i in 1..m}:  sum{j in 1..n} A[i,j]*x[j]>=b[i];
```

Exercise 15.2 Solve the model for the sample data and the weights in point 2.