Various Model Types

Tony Hürlimann
info@matmod.ch

July 21, 2026
(First version: Oct 24, 2011)

Abstract

Mathematical models can be classified into groups using various criteria: discrete, continuous, linear, non-linear, etc. Different model types also dominate in different research domains. Linear and non-linear optimization models prevail the realm of operations research. Dynamical models such as differential equations, appear more in physics and engineering. Statistical models and models treating uncertainty (p.e. stochastic models, fuzzy models) occur in social sciences. This is not to say that physicists do not use statistics. However, the various research domain use a slightly different vocabulary in building their models.

Coming from the operations research, a lot of attention is given to the classification in this research field. Several model types are formulated in mathematical notation and in the modeling system LPL (see [10]). They are compared with each other from various point of views. Concrete model application examples are given for most model type.

A ZIP file of all models can be found HERE

Contents

1 Introduction **4**

2 Variations **6**

3 A Linear Program (examp-lp) **10**

4 A Integer Linear Program (examp-ip) **14**

5 A 0-1 Integer Program (examp-ip01) **17**

6 An LP-relaxation of the 0-1 Program (examp-ip01r) **20**

7 A Quadratic Convex Program (examp-qp) **24**

8 A 0-1-Quadratic Program (examp-qp01) **27**

9 Second-Order Cone (socp1) **30**

10 Rotated second-order Cone (socp2) **31**

11 A NQCP model (bilinear) **32**

12 Largest Empty Rectangle (quadrect) **33**

13 Hanging Chain (chain) **36**

14 Discrete Dynamic System (foxrabbit) **39**

15 A Simple Permutation Model (examp-tsp) **41**

16 Capacitated Vehicle Routing Problem (examp-cvrp) **44**

17 Binpacking (examp-binpack) **47**
17.1 Permutation Problems as Implemented in LPL 48

18 Additional Variable Types **50**
18.1 Semi-continuous Variable (semi-1) 50
18.2 Semi-integer Variable (semi-2) 50
18.3 A Multiple Choice Variable (mchoice-3) 51

19 Additional Constraint Types **52**
19.1 Sos1 Constraint (sos-1) 52
19.2 Sos2 Constraint (sos-2) 53
19.3 MPEC Model Type (compl-3) 54

20 Global Constraints **55**
20.1 Alldiff and alldiff 55
20.2 Element 56
20.3 Occurrence 57
20.4 Sequence_total 57

21 Conclusion **58**

1 Introduction

Models can be classified into different groups using various criteria:

Qualitative versus quantitative: When taking a decision or solving a problem, many trust one's gut, taking into account intuition and experiences. We all build consciously or unconsciously a model to capture the situation at hand. More often than not, the model may have the form of a feeling and a decision is taken spontaneously. On the other hand, collect all kinds of relevant data, formulate the conditions in a clear way, define the goals formally may lead to superior decision. All depends on the situation and the problem. Surely, many problems must be quantified to achieve a "good" result. For example, to planify a round-trip to deliver goods to various clients, intuition will in general be a bad advisor in finding the shortest trip. This paper only treats quantitative models.

Continuous versus discrete (integer or Boolean): Fractional values may be unacceptable. "Buy 3.5 aeroplanes" does not make sense in most context, but "on average 3.5 persons die every second due to drug consumption" may be adequate. However, there is another huge and important class of models that use discrete (Boolean) values: models to represent problems that must answer yes/no questions, for example, "should we build a factory or not?". The significance of this class is shown in another paper (see [8]). Discrete – also called combinatorial – problems are in general much more difficult to solve. On the other hand, problems dealing with physical quantities and their rates of change can be modeled as differential equations. Most of them can only be solved by discretizing them.

Zero- versus One- versus Multi-objective: We may looking for *all*, for an *arbitrary* or for a *particular* solution. For zero-objective problems, we are interested only in an arbitrary solution, it may be unique or not. For example, in model example3a any solution would be fine (it is unique in this case). In many situations, we are interested in the "best" solution, with regards to one or many objectives. These problems are called *optimization problems*. We may minimize cost, resources, redundancy, etc. or we may maximize revenue, utility, turnover, customer satisfaction, robustness, etc. Normally, we are looking for one objective. In same cases, however, several – maybe conflicting – objectives have to be considered. These problems are called multi-objective optimization problems. Such problems can be handled by various methods, for example, with *goal programming* (see below).

Unconstraint versus constrained: Within the class of optimization problems there are unconstraint or constraint problems. Unconstraint problems only contain an objective function and no constraint, example: find the minimum of a parable : $\min f(x) = x^2$. Constraint problems consist of an objective and one or several constraints.

Deterministic versus stochastic: If the data in a resultant model is not known with certainty, we have stochastic models. In many situation, the data are uncertain or unknown, for example, future demand on a market are not known or uncertain, but even data from the past are often not known or only a small sample is known. Statistical methods to estimate them are then needed. Mostly, however, we pretend that data are deterministic in order to avoid complicated models. In many cases, this may be realistic, but the modeler should be aware of that fact. There is a broad theory of stochastic models, but in this paper only deterministic once are treated.

Static versus dynamical: In many situation we are looking for an (optimal) state: finding an optimal production plan, a tournament schedule of a sport league, the shortest round trip, etc. Normally, optimization models result from these problems. These models are static. When change has to be modeled, like motion over time in physics, evolution in animal population in biology, fluctuation in monetary quantities in economy, or development of a virus in a pandemic, dynamical models are commonly use, such as discrete dynamical systems or a system of

differential equations.

Linear versus non-linear: Linear models only contain linear terms with regards to the variables. Completely different methods are used to solve linear and non-linear problems. Linear, continuous problems are solved mainly by the simplex method, a modification of this method solves also certain problems containing quadratic terms (QP, QPC, etc.) – if they are convex. Linear, discrete (combinatorial) problems use branch-and-bound, cutting plane or other reduction algorithms. Non-linear problems are in general much more difficult to solve, and a large number of algorithms have been developed. The distinction between linear and non-linear models may be arbitrary. We also may partition the models into *convex and concave*, or into *"easy" and "difficult"* to solve. "Easy" could be defined as problems in P (polynomial-solvable), the "difficult" once are NP-complete, or beyond.[1] In any case, the purpose of these partitions is to classify the models into groups of different algorithmic methods. The reason is more practical then theoretical: A modeling system – that allows to *formulate* a wide range of mathematical models, must be able to be linked to a large number of solutions algorithms – so called *solvers* – in order to solve it. The modeling system should be able to recognizes into which group(s) a model falls to select the solver automatically.

In this paper, a general overview of various mathematical (optimization) model types is presented. A general specification and formulation is given first in a mathematical notation then in the modeling language LPL (see [10]).

A mathematical model has the following general form:

$$\begin{array}{lll} \min & f(\mathbf{x}) & \\ \text{subject to} & g_i(\mathbf{x}) \leq 0 & \text{forall } i \in \{1, \ldots, m\} \\ & \mathbf{x} \in \mathbf{X} & \end{array}$$

If the first line is missing, we have a zero-objective model, if the second line is missing, we have an unconstraint model. The f and g_i are functions defined in $\mathbf{R}^n$.[2] $\mathbf{X}$ is a subset of $\mathbf{R}^n$ (or $\mathbf{N}^n$), and $\mathbf{x}$ is a vector of n components $x_1, \ldots, x_n$. The above problem has to be solved for the values of the *variables* $x_1, \ldots, x_n$ that satisfy the restrictions g_i while minimizing the function f. The function f is called *the objective function*, g_i are *the constraints*. A vector $\mathbf{x} \in \mathbf{X}$ that satisfies all constraints $g_i(\mathbf{x}) \leq 0$ is called *feasible solution*. The collection of all such points is *the feasible region*. The problem then of the mathematical model above is to find a $\mathbf{x}^o$ such that $f(\mathbf{x}) \geq f(\mathbf{x}^o)$ for each feasible point $\mathbf{x}$. Such a point $\mathbf{x}^o$ is called *an optimal solution*.

A small example is the following model (see [2], page 3):

$$\begin{array}{ll} \min & (x_1 - 3)^2 + (x_2 - 2)^2 \\ \text{subject to} & x_1^2 x_2 - 3 \leq 0 \\ & x_2 - 1 \leq 0 \\ & -x_1 \leq 0 \end{array}$$

The objective function and the 3 constraints are:

[1] It would be too involving here to explain these concepts. They are developed in the theory of complexity in computer science.

[2] The set of operators that can be used in a function decide on the expressiveness on models. Boolean operators belong to that set, if all kind of combinatorial problems are to be formulated. Note that any model containing several constraints could be expressed by a single constraint – by concatenating the constraints with the Boolean operator **and**.

$$
\begin{array}{lll}
f(x_1, x_2) & \text{is} & (x_1 - 3)^2 + (x_2 - 2)^2 \\
g_1(x_1, x_2) & \text{is} & x_1^2 x_2 - 3 \leq 0 \\
g_2(x_1, x_2) & \text{is} & x_2 - 1 \leq 0 \\
g_3(x_1, x_2) & \text{is} & -x_1 \leq 0
\end{array}
$$

Figure 1 illustrates the model geometrically in the two-dimensional *real Euclidean space*.

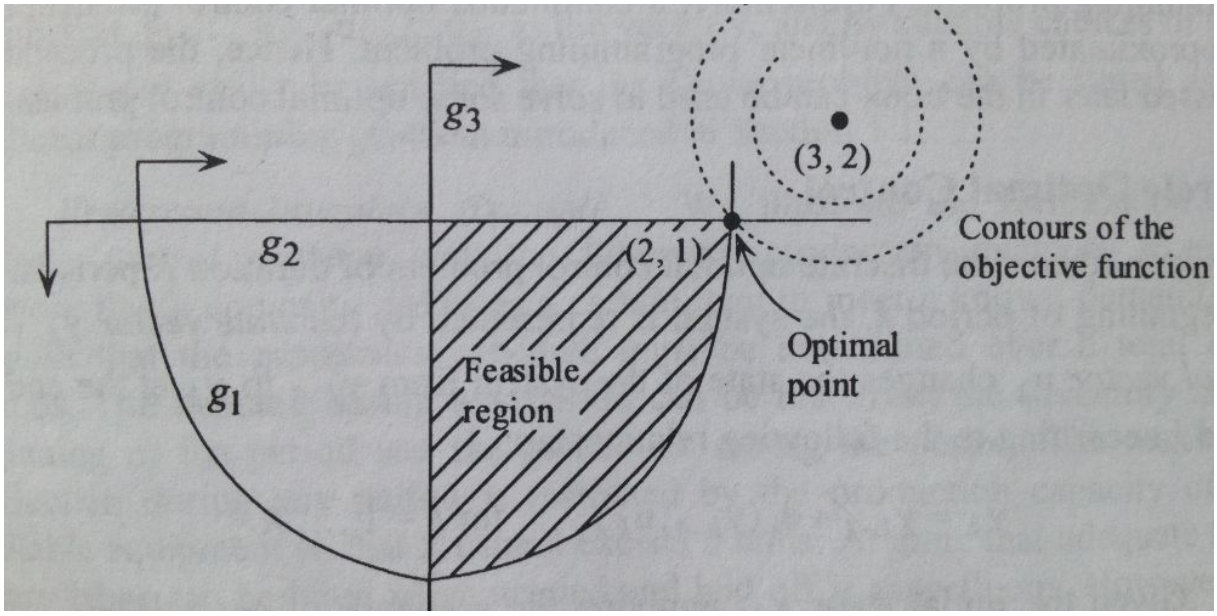

Figure 1: Geometric representation of the model

2 Variations

There are useful practical variations in notation that can be reduced to a standard model type.

1. A maximizing objective function can be transformed into a minimizing function (and vice versa):

$$\max \ f(\mathbf{x}) \Longrightarrow \min \ -f(\mathbf{x})$$

   A greater or equal constraint can be transformed into a less or equal than constraint:

$$f(\mathbf{x}) \geq 0 \Longrightarrow -f(\mathbf{x}) \leq 0$$

   A equality constraint can be transformed into a less and a greater or equal than constraint:

$$f(\mathbf{x}) = 0 \Longrightarrow \left\{ \begin{array}{l} f(\mathbf{x}) \geq 0 \\ f(\mathbf{x}) \leq 0 \end{array} \right.$$

2. A maximin (or a minimax) objective can be transformed as follows (let $I = \{1, \ldots, n\}$) (note that the functions are convexe) :

$$\max \ \left(\min_{i \in I} \ f(x_i) \right) \Longrightarrow \left\{ \begin{array}{lll} \max & z & \\ \text{s.t.} & f(\mathbf{x_i}) \leq z & \text{forall } i \in I \\ & z \geq 0 & \end{array} \right.$$

$$\min \quad \left(\max_{i \in I} \; f(x_i)\right) \Longrightarrow \left\{ \begin{array}{lll} \min & z & \\ \text{s.t.} & f(\mathbf{x_i}) \geq z & \text{forall } i \in I \\ & z \geq 0 & \end{array} \right.$$

This notation is useful for example in zero-sum games (see gameh, or in regression models and others (see model regression).

3. The following objective function is concave, hence, the transformation is more involving (M being an upper bound for x) :

$$\max \quad |x| \Longrightarrow \left\{ \begin{array}{ll} \max & z \\ \text{s.t.} & z = x_p + x_n \\ & x = x_p - x_n \\ & x_p = My \\ & x_n = M(1-y) \\ & y \in \{0,1\} \\ & z, x_p, x_n \geq 0 \end{array} \right.$$

4. An arbitrary (non-linear) objective function $f(x)$ can be replaced by a linear function by introducing an additional variable z. Then the general model is replaced by:

$$\begin{array}{lll} \min & z & \\ \text{subject to} & g_i(\mathbf{x}) \leq 0 & \text{forall } i = 1, \ldots, m \\ & \mathbf{x} \in \mathbf{X} & \\ & f(\mathbf{x}) \leq z & \end{array}$$

5. A fractional linear objective can be replaced by a linear one by introducing additional variables. The method is explained in model bill046

6. Let $f(y)$ be a non-linear function used in a constraint. Then we can approximate it by a piecewise linear function. The procedure is explained in model bill227.

7. The variables x in a standard (non-linear) model are free. If they must be positive only, then one can easily add the constraints: ($\mathbf{x} \geq 0$).

   The variables in a linear model (LP) normally are tacitly limited to be positive. If a variable x should be free, then we can make the following substitutions:

$$x = x_p + x_n, \quad \text{with} \quad x_p, x_n \geq 0$$

8. A model with k *multiple objectives*, say $f_1(\mathbf{x}), \ldots, f_k(\mathbf{x})$ can be formulated by forming a new combined objective f as follows:

$$f(\mathbf{x}) = w_1 f_1(\mathbf{x}) + \ldots + w_k f_k(\mathbf{x})$$

   where $w_1, \ldots, w_k$ are numbers (weights) that reflect the importance of the single objectives (the higher the number, the more important the objective). An illustrative example is give in the model multi1.

9. A useful variant is **goal programming** by the means of *soft constraints*. In many situation, we do not mind if a particular constraint is slightly violated. For instance, with a given budget of $1'000'000, it would certainly be acceptable to over- or undershoot it by, say, $100. On the other hand, a constraint that defines $Expenses = Budget$, enforces *exactly* the amount. To resolve (and relax) this problem, the constraint can be made "soft", that is, two positive variables p and n are added to absorb a positive or negative deviation (slacks) from the budget goal, and the constraint is modified to

$$Expenses + n - p = Budget \quad (n, p \geq 0)$$

In the objective function, a weighted sum of these slack variables can be minimized (as seen in the previous item about multiple objectives, or the maximal deviation can be minimized. Hence, the goal, namely "attaining the budget exactly" is substituted by the goal "attaining the budget approximately". Several positive and negative slacks could be added to a goal constraint, that can be penalized with an increasing weight parameters in the objective function. This method is closely related with the problem of multiple objectives: The objectives are – between others – to minimize the deviations of the different goals. A example to illustrate this technique is given in the models goalpr and in goalpr1.

10. *Preemptive goal programming* is another technique to handle soft constraints. Finding the right weights for the different goals in an multiple objective function is sometimes difficult. Hence, in this method the user orders the goals in a list of decreasing importance. The first optimization minimizes/maximizes the most important goal. Then the corresponding slack variables are fixed and the next goal is optimized, and so on, until all goals has been optimized. Formally, solve the model first with $f_1(\mathbf{x})$ (supposing f_1 is the most important goal). Let the optimal value be f_1^o, then add the constraint $g_{m+1}(\mathbf{x}) = f_1(\mathbf{x}) - f_1^o = 0$. Now continue in the same way with the second most important objective, an so on until the least important objective. This method is illustrated by the model goalpr2. A real model where this method is used can be found in the Casebook II.

In an application, the **objective function** may have various meaning. We may *maximize* profit, utility, turnover, return on investment, net present value, number of employees, customer satisfaction, probability of survival, robustness; or we may *minimize* cost, number of employees, redundancy, deviations, use of resources, etc.

The constraints reflect a large variation of requirements in a concrete application. In a production context there may be *capacity*, *material availability marketing limitation*, *material balance* constraints; in a resource scheduling we may have *due date*, *job sequencing*, *space limitation* constraints, etc.

In the following sections various model types are presented and most of them implemented:

1. *Linear programs (LP)* consist of linear constraints and a linear objective function and real variables.

2. *Integer (linear) programs (IP)* consist of linear constraints and a linear objective function and integer variables.

3. *0-1 (linear) programs* consist of linear constraints and a linear objective function and binary variables (integer variables with only values of 0 and 1). This type is a special case of the previous one, where variables are integer but can only take the values 0 or 1. From the formulation point of view, the difference between the three types is very small.

However – as we shall see – the difficulty to solve integer problems is much higher. Linear programs (LP) can be solved in polynomial time, while IP and 0-1 IP problems are mostly NP-complete. The application field of these three model types is very large. We shall point to examples.

4. *Quadratic problems (QP)* which consist of linear constraints and a quadratic convex objective function.

5. *0-1 quadratic problems (0-1-QP)* which consist of linear constraints and a quadratic convex objective function and contain 0-1 variables. Both types have interesting applications in portfolio theory.

6. *Quadratic constraint problems (QCP)* which consist of linear and quadratic convex constraints and a quadratic convex objective function.

7. *Second order cone problems (SOCP)*, which have many applications in physics. All of these previous model classes are convex problems – and much easier to solve.

8. *Non-convex quadratic problems (NCQP)*, an interesting application is given below. All of these previous model classes can today be solved by commercial solvers like Gurobi or Cplex.

9. A large class of models is the *non-linear optimization model class (NLP)*. Many solver exist for specific subclasses of this class. Also some general commercial solvers exist such as Knitro or Hexaly solver. Version 11.0 of Gurobi also can solve some non-linear models.

10. Many application in physics, biology, or economy use dynamic models which are basically discrete *dynamic system* or systems of differential equations.

11. A particular class is the *permutation model class*. Many routing, scheduling, or assignment problems can be formulated in this way, a concrete example is given, more of them can be found in the paper [9].

For most of the model types, a concrete application and an implementation in LPL is given. We shall show the gap in difficulty to solve integer problems compared to non-integer linear problems.

Of course, one can also build combined models: LP model part mixed with IP model leads to *mixed integer programs (MIP)* containing real *and* integer variables, etc.

3 A Linear Program (examp-lp)

Problem: The general **linear programming** model – called **LP** – consists of a linear objective function $f(\mathbf{x})$ and m linear constraints $g_i(\mathbf{x})$ and n variables. It can be compactly formulated as follows (see [12]):

$$\begin{array}{lll} \min & \sum_{j \in J} c_j x_j & \\ \text{subject to} & \sum_{j \in J} a_{i,j} x_j \geq b_i & \text{forall } i \in I \\ & x_j \geq 0 & \text{forall } j \in J \\ & I = \{1 \dots m\},\ J = \{1 \dots n\},\ m, n \geq 0 & \end{array}$$

An more compact formulation using matrix notation for the model is as follows:

$$\begin{array}{ll} \min & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array}$$

The objective function f the constraints g_i, and $\mathbf{X}$ (in the general format) are as follows:

$$\begin{array}{lll} f(x_1, \dots, x_n) = & c_1 x_1 + \dots + c_n x_n & \\ g_i(x_1, \dots, x_n) = & b_i - (a_{i,1} x_1 + \dots + a_{i,n} x_n) \leq 0 & \text{forall } i = 1, \dots, m \\ & \mathbf{x} \in \mathbf{R}^n & \end{array}$$

Note that in addition to the g_i constraints we also have the non-negativity conditions on the variables $\mathbf{x}$ in the standard formulation. If we need negative variables, we must explicitly replace $\mathbf{x}$ by $\mathbf{x_1} - \mathbf{x_2}$, where $\mathbf{x_1}, \mathbf{x_2} \geq 0$ are two positive vectors $\in \mathbf{R}^n$.

In the following, a concrete problem with random data for the matrix $\mathbf{A}$, and the two vectors $\mathbf{b}$ and $\mathbf{c}$ is specified (with $n = 15$ and $m = 15$):

$$\mathbf{c} = \begin{pmatrix} 16 & 73 & 39 & 81 & 68 & 60 & 90 & 11 & 33 & 89 & 71 & 12 & 24 & 23 & 47 \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} 87 & 34 & 0 & 0 & 43 & 0 & 52 & 85 & 36 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 39 & 0 & 84 & 0 & 0 & 88 & 0 & 0 & 0 & 0 & 0 & 0 & 22 \\ 55 & 63 & 79 & 0 & 0 & 0 & 0 & 71 & 0 & 0 & 0 & 0 & 70 & 0 & 79 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 73 & 0 & 0 & 0 & 22 & 0 & 0 & 0 \\ 0 & 0 & 0 & 66 & 0 & 0 & 34 & 0 & 0 & 24 & 0 & 61 & 0 & 0 & 64 \\ 0 & 0 & 16 & 19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 \\ 15 & 70 & 61 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 52 & 0 & 14 & 0 & 0 & 92 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 98 & 0 & 19 & 0 & 0 \\ 0 & 70 & 37 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 86 & 0 & 93 & 0 \\ 90 & 0 & 82 & 0 & 66 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 \\ 77 & 30 & 0 & 0 & 0 & 0 & 13 & 10 & 0 & 0 & 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 72 & 0 & 0 & 0 & 79 & 0 & 0 & 0 & 0 & 0 & 11 & 90 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 32 & 55 & 0 & 23 & 0 & 0 & 0 & 0 & 0 & 36 \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 58 \\ 44 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 119 \end{pmatrix}$$

With these data, the model is specified by the following explicit linear program:

$$
\begin{aligned}
\text{min} \quad & 16x_1 + 73x_2 + 39x_3 + 81x_4 + 68x_5 + 60x_6 + 90x_7 + 11x_8 + 33x_9 \\
& \quad +89x_{10} + 71x_{11} + 12x_{12} + 24x_{13} + 23x_{14} + 47x_{15} \\
\text{subject to} \quad & 87x_1 + 34x_2 + 43x_5 + 52x_7 + 85x_8 + 36x_9 \geq 0 \\
& 39x_3 + 84x_5 + 88x_8 + 22x_{15} \geq 0 \\
& 55x_1 + 63x_2 + 79x_3 + 71x_8 + 70x_{13} + 79x_{15} \geq 0 \\
& 73x_8 + 22x_{12} \geq 0 \\
& 66x_4 + 34x_7 + 24x_{10} + 61x_{12} + 64x_{15} \geq 0 \\
& 16x_3 + 19x_4 + 18x_{15} \geq 0 \\
& 15x_1 + 70x_2 + 61x_3 \geq 0 \\
& 52x_9 + 14x_{11} + 92x_{14} \geq 58 \\
& 30x_8 + 98x_{11} + 19x_{13} \geq 44 \\
& 70x_2 + 37x_3 + 86x_{12} + 93x_{14} \geq 0 \\
& 90x_1 + 82x_3 + 66x_5 + 26x_{10} \geq 0 \\
& 77x_1 + 30x_2 + 13x_7 + 10x_8 + 50x_{12} \geq 0 \\
& 72x_2 + 79x_6 + 11x_{12} + 90x_{13} \geq 0 \\
& 32x_6 + 55x_7 + 23x_9 + 36x_{15} \geq 119 \\
& x_j \geq 0 \quad \text{forall } j \in \{1 \dots 15\}
\end{aligned}
$$

Modeling Steps: The formulation of the model in the LPL modeling language is straightforward and the notation is close to the mathematical notation using indices: First, define the two sets `i` and `j`. Then declare and assign the data as parameters `A`, `c`, and `b`. The variable vector `x` is declared, and finally the constraints `R` and the minimizing objective function `obj` are written.

Note that the data matrices **A**, **b**, and **c** are generated using LPL's own random generator. (To generate each time the same data, the code can also use the function `SetRandomSeed(a)` where `a` is an integer.)

Listing 1: The Complete Model implemented in LPL [10]

```
model Lp15 "A Linear Program (math)";
  set i := [1..15];  j := [1..15];
  parameter A{i,j}:= Trunc(if(Rnd(0,1)<.25,Rnd(10,100)));
            c{j}  := Trunc(Rnd(10,100));
            b{i}  := Trunc(if(Rnd(0,1)<.15,Rnd(0,120)));
  variable  x{j};
  constraint R{i} : sum{j} A*x >= b;
  minimize obj    : sum{j} c*x;
  Writep(obj,x);
end
```

Today, models with $n, m > 10000$ and much larger are solved on a regular base. LP models with millions of variables can be solved today. A linear programming model with 2000 variables and 1000 linear constraints can be downloaded and solved at: lp2000.

Solution: The small model defined above has the following solution:

$$x = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.4667 & 1.1154 & 0 & 0 & 0 & 0 & 0 & 2.5929 \end{pmatrix}$$

The optimal value of the objective function is:

$$obj = 174.8096$$

Further Comments: The linear programming (LP) model has many applications in quantitative decision making. LP is used for capacity planning, resource (raw material) allocation, manpower planning, blending, transportation, network flow, network design, portfolio selection, optimal marketing mix, multiperiod product mix, and many others.

A small example in multi period production planning is given in the model: product. A historical example of a the so-calledmin-cut problem is modeled by Tolstoi1930. An application example for transportation is trans or ship3.

A linear model implementing three common regression methods is given in regression.

Further Notes: As already seen in the model *regression* certain non-linear functions can be transformed in a way that the model becomes an linear one. We mention three of them (see [3], Chap. 13):

1. The **absolute value**. Suppose the (minimizing) objective function has the form:

$$f(\mathbf{x}) = |\mathbf{c} \cdot \mathbf{x}|$$

One can add a single variable y and modify the LP as follows:

$$\begin{array}{lll} \min & y & \\ \text{subject to} & \mathbf{A} \cdot \mathbf{x} & \geq \mathbf{b} \\ & \mathbf{c} \cdot \mathbf{x} & \leq y \\ & -c \cdot \mathbf{x} & \leq y \\ & \mathbf{x} & \geq 0 \\ & y & \geq 0 \end{array}$$

This can be generalized. If the objective function is:

$$f(\mathbf{x}) = \sum_{i \in I} |\mathbf{c}_i \cdot \mathbf{x}|$$

One can add a vector of variables y_i (with $i \in I$) and modify the LP as follows:

$$\begin{array}{lll} \min & \sum_{i \in I} y_i & \\ \text{subject to} & \mathbf{A} \cdot \mathbf{x} & \geq \mathbf{b} \\ & \mathbf{c}_i \cdot \mathbf{x} & \leq y_i \quad , \text{ forall } i \in I \\ & -c_i \cdot \mathbf{x} & \leq y_i \quad , \text{ forall } i \in I \\ & \mathbf{x} & \geq 0 \\ & y_i & \geq 0 \quad , \text{ forall } i \in I \end{array}$$

2. The **maximal value**. Suppose the objective function has the form:

$$f(\mathbf{x}) = \max \ \mathbf{c} \cdot \mathbf{x}$$

One can add a single variable y and modify the LP as follows:

$$\begin{array}{lll} \min & y & \\ \text{subject to} & \mathbf{A} \cdot \mathbf{x} & \geq \mathbf{b} \\ & \mathbf{c} \cdot \mathbf{x} & \leq y \\ & \mathbf{x} & \geq 0 \\ & y & \geq 0 \end{array}$$

In a similar way, if the objective function is:

$$f(\mathbf{x}) = \max_{i \in I} \left| \mathbf{c}_i \cdot \mathbf{x} \right|$$

One can add a single variable y and modify the LP as follows:

$$\begin{array}{lllll} \min & \sum_{i \in I} y_i & & & \\ \text{subject to} & \mathbf{A} \cdot \mathbf{x} & \geq \mathbf{b} & & \\ & \mathbf{c}_i \cdot \mathbf{x} & \leq y & , \text{ forall } i \in I \\ & -c_i \cdot \mathbf{x} & \leq y & , \text{ forall } i \in I \\ & \mathbf{x} & \geq 0 & & \\ & y_i & \geq 0 & , \text{ forall } i \in I \end{array}$$

3. The **fractional LP**. Suppose the objective function consists of maximizing the ratio of two linear functions (where $\mathbf{p}$ and $\mathbf{q}$ are two data vectors and α and β are two scalars):

$$f(\mathbf{x}) = \frac{\mathbf{p} \cdot \mathbf{x} + \alpha}{\mathbf{q} \cdot \mathbf{x} + \beta}$$

One can convert this into a LP model. If the feasible set $\{\mathbf{x} | \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}, \mathbf{x} \leq 0\}$ is nonempty and bounded and if $\mathbf{q} \cdot \mathbf{x} + \beta > 0$, using the following transformations:

$$z = \frac{1}{\mathbf{q} \cdot \mathbf{x} + \beta} \qquad , \qquad \mathbf{y} = z\mathbf{x}$$

we obtain the following LP:

$$\begin{array}{lll} \min & \mathbf{p} \cdot \mathbf{x} + \alpha & \\ \text{subject to} & \mathbf{A} \cdot \mathbf{y} - \mathbf{b}z & \leq 0 \\ & \mathbf{q} \cdot \mathbf{y} + \beta z & = 1 \\ & \mathbf{y} & \geq 0 \\ & z & \geq 0 \end{array}$$

A small example is given in bill046.

4 A Integer Linear Program (examp-ip)

Problem: The general **linear integer programming** model – called **IP** – contains a linear objective function $f(\mathbf{x})$, m linear constraints $g_i(\mathbf{x})$, and n integer variables. It can be compactly formulated as follows (see [12]):

$$\begin{array}{lll} \min & \sum_{j \in J} c_j x_j & \\ \text{subject to} & \sum_{j \in J} a_{i,j} x_j \geq b_i & \text{forall } i \in I \\ & x_j \in \mathbb{N}^+ & \text{forall } j \in J \\ & I = \{1 \ldots m\},\ J = \{1 \ldots n\},\ m, n \geq 0 & \end{array}$$

A more compact formulation using matrix notation for the model is :

$$\begin{array}{ll} \min & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \in \mathbf{N}^+ \end{array}$$

The objective function f, the constraints g_i, and $\mathbf{x}$ (in the general model format) are as follows:

$$\begin{array}{lll} f(x_1, \ldots, x_n) = & c_1 x_1 + \ldots + c_n x_n & \\ g_i(x_1, \ldots, x_n) = & b_i - (a_{i,1} x_1 + \ldots + a_{i,n} x_n) \leq 0 & \text{forall } i = 1, \ldots, m \\ & \mathbf{x} \in \mathbf{N^n} & \end{array}$$

The IP model has the same notation as the LP model, the only difference is that the variables are integer values. However, IP model are much more difficult to solve in general. (To get an idea solve the following model: lp2000. Then try to solve ip2000.)

In the following a problem with random data is specified (with $n = 15$ and $m = 15$):

$$c = \begin{pmatrix} 16 & 73 & 39 & 81 & 68 & 60 & 90 & 11 & 33 & 89 & 71 & 12 & 24 & 23 & 47 \end{pmatrix}$$

$$A = \begin{pmatrix} 87 & 34 & 0 & 0 & 43 & 0 & 52 & 85 & 36 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 39 & 0 & 84 & 0 & 0 & 88 & 0 & 0 & 0 & 0 & 0 & 0 & 22 \\ 55 & 63 & 79 & 0 & 0 & 0 & 0 & 71 & 0 & 0 & 0 & 0 & 70 & 0 & 79 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 73 & 0 & 0 & 0 & 22 & 0 & 0 & 0 \\ 0 & 0 & 0 & 66 & 0 & 0 & 34 & 0 & 0 & 24 & 0 & 61 & 0 & 0 & 64 \\ 0 & 0 & 16 & 19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 \\ 15 & 70 & 61 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 52 & 0 & 14 & 0 & 0 & 92 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 98 & 0 & 19 & 0 & 0 \\ 0 & 70 & 37 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 86 & 0 & 93 & 0 \\ 90 & 0 & 82 & 0 & 66 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 \\ 77 & 30 & 0 & 0 & 0 & 0 & 13 & 10 & 0 & 0 & 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 72 & 0 & 0 & 0 & 79 & 0 & 0 & 0 & 0 & 0 & 11 & 90 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 32 & 55 & 0 & 23 & 0 & 0 & 0 & 0 & 0 & 36 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 58 \\ 44 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 119 \end{pmatrix}$$

The data given above specify the following explicit linear program:

$$
\begin{array}{ll}
\min & 16x_1 + 73x_2 + 39x_3 + 81x_4 + 68x_5 + 60x_6 + 90x_7 + 11x_8 + 33x_9 \\
& \quad +89x_{10} + 71x_{11} + 12x_{12} + 24x_{13} + 23x_{14} + 47x_{15} \\
\text{subject to} & 87x_1 + 34x_2 + 43x_5 + 52x_7 + 85x_8 + 36x_9 \geq 0 \\
& 39x_3 + 84x_5 + 88x_8 + 22x_{15} \geq 0 \\
& 55x_1 + 63x_2 + 79x_3 + 71x_8 + 70x_{13} + 79x_{15} \geq 0 \\
& 73x_8 + 22x_{12} \geq 0 \\
& 66x_4 + 34x_7 + 24x_{10} + 61x_{12} + 64x_{15} \geq 0 \\
& 16x_3 + 19x_4 + 18x_{15} \geq 0 \\
& 15x_1 + 70x_2 + 61x_3 \geq 0 \\
& 52x_9 + 14x_{11} + 92x_{14} \geq 58 \\
& 30x_8 + 98x_{11} + 19x_{13} \geq 44 \\
& 70x_2 + 37x_3 + 86x_{12} + 93x_{14} \geq 0 \\
& 90x_1 + 82x_3 + 66x_5 + 26x_{10} \geq 0 \\
& 77x_1 + 30x_2 + 13x_7 + 10x_8 + 50x_{12} \geq 0 \\
& 72x_2 + 79x_6 + 11x_{12} + 90x_{13} \geq 0 \\
& 32x_6 + 55x_7 + 23x_9 + 36x_{15} \geq 119 \\
& x_j \in \mathbf{N}^+ \quad \text{forall } j \in \{1 \ldots 15\}
\end{array}
$$

Modeling Steps: The formulation of the model in LPL modeling language is straightforward and the notation is close to the mathematical notation using indices: First we define the two sets `i` and `j`. Then we declare and assign the data as parameters `A`, `c`, and `b`. The variable vector `x` is declared with the keyword `integer` (the only difference to the LP model), and finally the constraints `R` and the minimizing objective function `obj` are written.

Note that the data matrices **A**, **b**, and **c** are generated using LPL's own random generator. (To generate the same data each time, the code can also use the function `SetRandomSeed(a)` where `a` is an integer.)

Listing 2: The Complete Model implemented in LPL [10]

```
model Ip15 "A Integer Linear Program (math)";
  set i := [1..15];  j := [1..15];
  parameter A{i,j} := Trunc(if(Rnd(0,1)<0.25, Rnd(10,100)));
            c{j}   := Trunc(Rnd(10,100));
            b{i}   := Trunc(if(Rnd(0,1)<0.15, Rnd(0,120)));
  integer variable  x{j};
  constraint R{i} : sum{j} A*x >= b;
  minimize obj    : sum{j} c*x;
  Writep(obj,x);
end
```

Solution: The small model defined above has the following solution :

$$x = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 4 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The optimal value of the objective function is:

$$obj = 201$$

If we compare this solution with the corresponding LP model in examp-lp, we notice that the objective value is much larger (201 compared to 174.8096). Furthermore, the vector x is not simply the "round-down" of the LP solution, as one may expect. To understand why we cannot simply round up or down to get an integer solution from a corresponding LP model, open the model willi155 and read the modeling text to understand why.

Further Comments: IP problems are *much* more difficult to solve than LP problems in general. However, there is a surprisingly rich application field for integer programs. Obvious applications are problems where we have indivisible objects, such as number of persons, machines, etc. However, these are not typical applications. If we have the number of drivers in a small company then we may model this as an integer quantity, however if we have the population in a country then we may approximate it as a continuous variable. This obvious aspect, however, does *not* reveal the real power of integer programming. We really need integer programming in the four following contexts in this order of importance from a practical point of view:

1. To model problems with logical conditions. In this case, the integrality is even reduced to variables that only take the value 0 or 1 (see the next model examp-ip01) for more information about them and how to specify logical constraints).

2. To model combinatorial problems, such as *sequencing problems*, *(job-shop) scheduling*, and many others. Many of them are again transformed to integer problems with $0-1$ variables. A small example can be found here: knapsack.

3. To model non-linear problems. There exist techniques that translate a non-linear problem into a integer (linear) problem. (For more information see, for example bill320).

4. To model problems where we need discrete (integer) numbers for various entities. An example for this last category is given by the following problem: magics another is mobile1.

Various linear problems with a special structure (the matrix $\mathbf{A}$ must be uni-modular) such as the *transportation problem* have integer solutions without explicitly formulating them as integer programs. They can be solved as LP progams.

Models with general integer variables with an upper bound can also be transformed into models that only contain $0-1$ integer variables. The transformation is as follows. Let $x \in \{0, 1, \ldots, u\}$ be an general upper (and lower) bounded integer variable. Then substitute the variable $0 \leq x \leq u$ by the expression:

$$\delta_0 + 2\delta_1 + 4\delta_2 + \ldots + 2^r\delta_r$$

where $\delta_0, \ldots, \delta_r$ are $0-1$ integer variables, and r is the smallest number, such that $u \leq 2^r$.

Furthermore, problems which contain general integer variables when modelling them in a straightforward way are sometimes preferably modeled with $0-1$ variables. An example is the Sudoku game (see my Puzzlebook that has plenty of such models).

5 A 0-1 Integer Program (examp-ip01)

Problem: The general **linear 0-1 integer programming** model – called **0-1-IP** – consists of a linear objective function $f(\mathbf{x})$, m linear constraints $g_i(\mathbf{x})$, and n $0-1$ integer variables. It can be compactly formulated as follows (see [12]):

$$\begin{array}{lll} \min & \sum_{j \in J} c_j x_j & \\ \text{subject to} & \sum_{j \in J} a_{i,j} x_j \geq b_i & \text{forall } i \in I \\ & x_j \in \{0,1\} & \text{forall } j \in J \\ & I = \{1 \ldots m\},\ J = \{1 \ldots n\},\ m, n \geq 0 & \end{array}$$

A more compact formulation using matrix notation for the model is:

$$\begin{array}{ll} \min & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \in \{0,1\} \end{array}$$

The objective function f the constraints g_i, and $\mathbf{X}$ (in the general model format) are as follows:

$$\begin{array}{lll} f(x_1, \ldots, x_n) = & c_1 x_1 + \ldots + c_n x_n & \\ g_i(x_1, \ldots, x_n) = & b_i - (a_{i,1} x_1 + \ldots + a_{i,n} x_n) \leq 0 & \text{forall } i = 1, \ldots, m \\ & \mathbf{x} \in \{0,1\} & \end{array}$$

The 0-1-IP model has the same notation as the LP (and the IP) model, the only difference is that the variables are $0-1$ integer values. The 0-1-IP model is also difficult to solve in general. (To get an idea solve the following model: lp2000. Then try to solve ip01-2000.)

In the following a problem with random data is specified (with $n = 15$ and $m = 15$):

$$c = \begin{pmatrix} 16 & 73 & 39 & 81 & 68 & 60 & 90 & 11 & 33 & 89 & 71 & 12 & 24 & 23 & 47 \end{pmatrix}$$

$$A = \begin{pmatrix} 87 & 34 & 0 & 0 & 43 & 0 & 52 & 85 & 36 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 39 & 0 & 84 & 0 & 0 & 88 & 0 & 0 & 0 & 0 & 0 & 0 & 22 \\ 55 & 63 & 79 & 0 & 0 & 0 & 0 & 71 & 0 & 0 & 0 & 0 & 70 & 0 & 79 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 73 & 0 & 0 & 0 & 22 & 0 & 0 & 0 \\ 0 & 0 & 0 & 66 & 0 & 0 & 34 & 0 & 0 & 24 & 0 & 61 & 0 & 0 & 64 \\ 0 & 0 & 16 & 19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 \\ 15 & 70 & 61 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 52 & 0 & 14 & 0 & 0 & 92 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 98 & 0 & 19 & 0 & 0 \\ 0 & 70 & 37 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 86 & 0 & 93 & 0 \\ 90 & 0 & 82 & 0 & 66 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 \\ 77 & 30 & 0 & 0 & 0 & 0 & 13 & 10 & 0 & 0 & 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 72 & 0 & 0 & 0 & 79 & 0 & 0 & 0 & 0 & 0 & 11 & 90 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 32 & 55 & 0 & 23 & 0 & 0 & 0 & 0 & 0 & 36 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 58 \\ 44 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 119 \end{pmatrix}$$

The data given above specify the following explicit linear program:

$$
\begin{array}{ll}
\min & 16x_1 + 73x_2 + 39x_3 + 81x_4 + 68x_5 + 60x_6 + 90x_7 + 11x_8 + 33x_9 \\
 & \quad +89x_{10} + 71x_{11} + 12x_{12} + 24x_{13} + 23x_{14} + 47x_{15} \\
\text{subject to} & 87x_1 + 34x_2 + 43x_5 + 52x_7 + 85x_8 + 36x_9 \geq 0 \\
 & 39x_3 + 84x_5 + 88x_8 + 22x_{15} \geq 0 \\
 & 55x_1 + 63x_2 + 79x_3 + 71x_8 + 70x_{13} + 79x_{15} \geq 0 \\
 & 73x_8 + 22x_{12} \geq 0 \\
 & 66x_4 + 34x_7 + 24x_{10} + 61x_{12} + 64x_{15} \geq 0 \\
 & 16x_3 + 19x_4 + 18x_{15} \geq 0 \\
 & 15x_1 + 70x_2 + 61x_3 \geq 0 \\
 & 52x_9 + 14x_{11} + 92x_{14} \geq 58 \\
 & 30x_8 + 98x_{11} + 19x_{13} \geq 44 \\
 & 70x_2 + 37x_3 + 86x_{12} + 93x_{14} \geq 0 \\
 & 90x_1 + 82x_3 + 66x_5 + 26x_{10} \geq 0 \\
 & 77x_1 + 30x_2 + 13x_7 + 10x_8 + 50x_{12} \geq 0 \\
 & 72x_2 + 79x_6 + 11x_{12} + 90x_{13} \geq 0 \\
 & 32x_6 + 55x_7 + 23x_9 + 36x_{15} \geq 119 \\
 & x_j \in [0,1] \quad \text{forall } j \in \{1 \ldots 15\}
\end{array}
$$

Modeling Steps: The formulation of the model in LPL modeling language is straightforward and the notation is close to the mathematical notation using indices: First we define the two sets `i` and `j`. Then we declare and assign the data as parameters `A`, `c`, and `b`. The variable vector `x` is declared with the keyword `binary` (the only difference to the LP model), and finally the constraints `R` and the minimizing objective function `obj`.

Note that the data matrices **A**, **b**, and **c** are generated using LPL's own random generator. (To generate each time the same data, the code can also use the function `SetRandomSeed(a)` where `a` is an integer.) Note also that only difference in the LPL formulation compared with the LP model is the word `binary` added to the variable declaration.

Listing 3: The Complete Model implemented in LPL [10]

```
model Ip1501 "A 0-1 Integer Program (math)";
  set i := [1..15];  j := [1..15];
  parameter A{i,j} := Trunc(if(Rnd(0,1)<0.25, Rnd(10,100)));
            c{j}   := Trunc(Rnd(10,100));
            b{i}   := Trunc(if(Rnd(0,1)<0.15, Rnd(0,120)));
  binary variable  x{j};
  constraint R{i} : sum{j} A*x >= b;
  minimize obj    : sum{j} c*x;
  Writep(obj,x);
end
```

Solution: The small model defined above has the following solution :

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

The optimal value of the objective function is:

$$obj = 255$$

Comparing the optimal solution of the three problems (1) examp-lp, (2) examp-ip, and (3) this one, we have the following optimal values: 174.8096 for the LP, 201 for the IP and 255 for the 0-1-IP model.

Do the increasing optimal values for LP, IP and 0-1-IP make sense? Of course, the IP model is "more" restricted, it only can have integer values, hence the IP optimal value can never be smaller than the LP optimal value. The same is true for the 0-1-IP optimum compared with the IP optimum.

One may take the bait to solve the 0-1-IP problem using the following procedure :

1. Replace the requirement that the variables are $0-1$ integer variables by the constraint $0 \leq \mathbf{x} \leq 1$, then solve this LP problem. (This LP problem is called the **LP relaxation** of the 0-1 integer problem.)

2. All solution values for $\mathbf{x}$ are in the interval $[0 \ldots 1]$.

3. Finally, round their values up to 1 or down to 0, depending of whether the value is closer to 1 or to 0.

Voilà! We can show with a tiny example (see willi155z), that this apparently reasonable approach is completely erroneous: while the LP relaxation of this tiny example has a feasible solution, the corresponding 0-1 problem is infeasible.

It seems difficult to derive the integer solution from the continuous LP problem. For a systematic procedure – called **cutting plane method** – that starts with the continuous LP problem to find an integer solution of the 0-1-IP problem, see some explanation in the model example examp-ip01r.

Further Comments: There is a surprisingly rich application field for 0-1-integer programming, as there is for integer programming in general. 0-1 integer programming is used in the following context:

1. To model problems with logical conditions, Boolean constraints or expressing some kind of "dichotomy".

2. To model combinatorial problems, such as *sequencing problems* and others.

3. To model non-linear problems. They can often be translated into 0-1 integer (linear) problems.

For a short guide to 0-1 integer model formulation and how logical conditions can be integrated into a mathematical model see the paper [8].

6 An LP-relaxation of the 0-1 Program (examp-ip01r)

Problem: This model is the same as the model examp-ip01 with the important difference that the variables are continuous and bounded by the interval [0..1]. Hence, this model is a LP program with continuous variable. It is called the **LP relaxation** of the corresponding 0-1 integer program.
A compact formulation using matrix notation for the model is:

$$\begin{array}{ll} \min & c \cdot x \\ \text{subject to} & A \cdot x \geq b \\ & 0 \leq x \leq 1 \end{array}$$

With the same data as in model examp-ip01, we get the following model:

$$\begin{array}{ll} \min & 16x_1 + 73x_2 + 39x_3 + 81x_4 + 68x_5 + 60x_6 + 90x_7 + 11x_8 + 33x_9 \\ & \quad +89x_{10} + 71x_{11} + 12x_{12} + 24x_{13} + 23x_{14} + 47x_{15} \\ \text{subject to} & 87x_1 + 34x_2 + 43x_5 + 52x_7 + 85x_8 + 36x_9 \geq 0 \\ & 39x_3 + 84x_5 + 88x_8 + 22x_{15} \geq 0 \\ & 55x_1 + 63x_2 + 79x_3 + 71x_8 + 70x_{13} + 79x_{15} \geq 0 \\ & 73x_8 + 22x_{12} \geq 0 \\ & 66x_4 + 34x_7 + 24x_{10} + 61x_{12} + 64x_{15} \geq 0 \\ & 16x_3 + 19x_4 + 18x_{15} \geq 0 \\ & 15x_1 + 70x_2 + 61x_3 \geq 0 \\ & 52x_9 + 14x_{11} + 92x_{14} \geq 58 \\ & 30x_8 + 98x_{11} + 19x_{13} \geq 44 \\ & 70x_2 + 37x_3 + 86x_{12} + 93x_{14} \geq 0 \\ & 90x_1 + 82x_3 + 66x_5 + 26x_{10} \geq 0 \\ & 77x_1 + 30x_2 + 13x_7 + 10x_8 + 50x_{12} \geq 0 \\ & 72x_2 + 79x_6 + 11x_{12} + 90x_{13} \geq 0 \\ & 32x_6 + 55x_7 + 23x_9 + 36x_{15} \geq 119 \\ & 0 \leq x_j \leq 1 \quad \text{forall } j \in \{1 \ldots 15\} \end{array}$$

Modeling Steps: The formulation of the model in LPL modeling language is straightforward and the notation is close to the mathematical notation using indices: First we define the two sets `i` and `j`. Then we declare and assign the data as parameters `A`, `c`, and `b`. The variable vector `x` is declared, and finally the constraints `R` and the minimizing objective function `obj`. Note that the only difference between this model and the model examp-ip01 is the variable declaration. The keyword `binary` has been removed and a lower and upper bound value for the variable `[0..1]` has been added.

Note that the data matrices **A**, **b**, and **c** are generated using LPL's own random generator. (To generate each time the same data, the code can also use the function `SetRandomSeed(a)` where `a` is an integer.)

Listing 4: The Complete Model implemented in LPL [10]

```
model Ip15_01r "An LP-relaxation of the 0-1 Program (math)";
  set i := [1..15];  j := [1..15];
  parameter A{i,j} := Trunc(if(Rnd(0,1)<0.25, Rnd(10,100)));
```

```
            c{j}    := Trunc(Rnd(10,100));
            b{i}    := Trunc(if(Rnd(0,1)<0.15, Rnd(0,120)));
            X{j}    := [0 0 0 0 0 1 1 1 0 0 0 0 1 1 1];
                 // X = 0-1 solution of the examp-ip01.lpl model
  variable  x{j} [0..1];
  constraint R{i} : sum{j} A*x >= b;
    --ADDED_a1: x[11]+x[14] >= 1; //add constraints
    --ADDED_a2: x[9] +x[14] >= 1;
    --ADDED_b1: x[13]+x[11] >= 1;
    --ADDED_b2: x[8] +x[11] >= 1;
    --ADDED_c:  x[6]  = 1;
    --ADDED_d:  x[7]  = 1;
    --ADDED_e:  x[15] = 1;
  minimize obj    : sum{j} c*x;
  Write('The optimal solution is as follows:\n
        Obj value = %8.4f , %9d , %9d
                        rounded    true 0-1 value\n',
         obj, sum{j} c*Round(x), sum{j} c*X);
  Write{j}('        x(%2s) = %8.4f %5d  %10d\n', j,x,Round(x),X);
end
```

Solution: The model has the following solution:

$$x = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0.16 & 1 & 1 & 1 & 0 \\ 0.14 & 0 & 0 & 0.04 & 1 & & & & & \end{pmatrix}$$

The optimal value of the objective function is:

$$obj = 201.52$$

In the following listing, we compare the three solutions: (1) the LP relaxation, (2) the rounded solution of the LP relaxation, and (3) the 0-1-IP solution. The LP relaxation has the optimal solution of 201.5179, the rounded problem has a solution of 181, which is far away from the true 0-1 solution which is 255. The rounded solution has no merit for the true integer solution.

```
  Obj value = 201.5179 ,     181 ,       255

                          rounded    true 0-1 value
        x( 1) =   0.0000     0           0
        x( 2) =   0.0000     0           0
        x( 3) =   0.0000     0           0
        x( 4) =   0.0000     0           0
        x( 5) =   0.0000     0           0
        x( 6) =   0.1563     0           1
        x( 7) =   1.0000     1           1
        x( 8) =   1.0000     1           1
        x( 9) =   1.0000     1           0
        x(10) =   0.0000     0           0
        x(11) =   0.1429     0           0
        x(12) =   0.0000     0           0
        x(13) =   0.0000     0           1
        x(14) =   0.0435     0           1
        x(15) =   1.0000     1           1
```

In contrast to the rounded version, the LP relaxation has an important function for the integer problem: The LP relaxation *generates a lower bound* for the integer solution. By solving the LP

relaxation, we know that the optimal value of the integer problem *cannot be below* the optimal value of the LP relaxation. In our model, the optimal solution of the integer problem must be at least 201.5179 (we know already that it is 255). This is an important fact.

But we may say more about the relation between the LP relaxation and the 0-1-IP problem. For example, we can look at a particular inequality. Let's choose just an arbitrary one, say:

$$52x_9 + 14x_{11} + 92x_{14} \geq 58$$

What can we say about that particular inequality? If x_{14} is zero then both x_9 *and* x_{11} must be one. Why? Because if $x_{14} = 0$, then the inequality reduces to:

$$52x_9 + 14x_{11} \geq 58$$

However, this can only be the case if – in the 0-1 integer problem – both x_9 and x_{11} are 1. Hence, we can *add* the two following inequalities to the LP relaxation model:

$$x_{14} + x_9 \geq 1 \quad , \quad x_{14} + x_{11} \geq 1$$

Why? These two additional constraints do *not* violate the 0-1-IP solution: if $x_{14} = 1$ then both inequalities are fulfilled, if $x_{14} = 0$ then both x_9 and x_{11} must be 1. That is exactly what the initial inequality claims if the values must be 0 or 1.

Now we solve the problem again without excluding a feasible solution of the 0-1 IP problem. What is interesting now: After having added these two constraints to the LP relaxation and solving it again, the optimal solution will be 220.2321. It has increased considerably, and again we can say that this is a lower bound for the 0-1 integer problem.

In the same way we could now look at the inquality:

$$30x_8 + 98x_{11} + 19x_{13} \geq 44$$

and we can repeat the same idea: If $x_{11} = 0$ then both x_8 and x_{13} must be 1 in the integer program. This gives rise to the additional inequalities:

$$x_{11} + x_8 \geq 1 \quad , \quad x_{11} + x_{13} \geq 1$$

Adding them too to the LP relaxation and solving the problem in $\mathbf{R^{15}}$, gives a optimal solution of 237.3750. That again rises the lower bound for the integer program considerably.

Looking at the solution, the unique value that is not integer is $x_6 = 0.1563$. In the integer problem, x_6 must be 0 or 1. So let try to set $x_6 = 0$ and add this to the previous problem. Solving the problem results in an infeasible problem. Hence, there is no integer solution where $x_6 = 0$. So let's try $x_6 = 1$ instead. We add this inequality to the previous problem. The new optimal solution is 243.8182.

Again, in the new solution we see that $x_7 = 0.5091$, the unique value that is not integer. Hence, we try the same procedure again: setting first $x_7 = 0$ and solving produces also an infeasible solution, setting $x_7 = 1$, gives an optimal solution of 249.7778.

There is still one variable that is not integer: $x_{15} = 0.8889$. Adding $x_{15} = 0$ gives an infeasible solution, but setting $x_{15} = 1$ produces an integer solution with the optimum of 255. This is identical to the 0-1-IP problem and we have found the optimal solution to the 0-1-IP problem by adding appropriate inequalities (and equalities) to the LP relaxation. In our case,

we added 4 inequalities and three equalities (by setting three variables to 1) (see the commented lines `--ADDED...` in the LPL code).

We conclude that the LP relaxation is important for the integer solution. It is the starting point of an iterative procedure that adds "valid" inequalities for the integer problem, until eventually we reach the optimal point of the integer problem. (Unfortunately, it is normally not so easy to add valid inequalities.) At least we have sketched an interesting idea on how to attack the solution of integer problems, that has great practical importance. For a more systematic approach to integer programming, see the interesting book [6]. See also two other small example in this context: alterna and unimodular.

7 A Quadratic Convex Program (examp-qp)

Problem: The general **quadratic programming** model – called **QP** – consists of m linear constraints, n variables and a quadratic convex objective function $f(\mathbf{x})$. It can be compactly formulated as follows:

$$\begin{array}{lll} \min & \displaystyle\sum_{j\in J,k\in J} x_j Q_{j,k} x_k + \sum_{j\in J} c_j x_j & \\ \text{subject to} & \displaystyle\sum_{j\in J} a_{i,j} x_j \geq b_i & \text{forall } i \in I \\ & x_j \geq 0 & \text{forall } j \in J \\ & J = \{1 \dots m\},\ I = \{1 \dots n\},\ m, n \geq 0 & \end{array}$$

An more compact formulation using matrix notation for the model is as follows:

$$\begin{array}{ll} \min & \mathbf{x}\mathbf{Q}\mathbf{x}' + \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array}$$

The objective function f the constraints g_i, and $\mathbf{X}$ (in the general format) are as follows:

$$\begin{array}{lll} f(x_1, \dots, x_n) = & x_1 Q_{1,1} x_1 + x_1 Q_{1,2} x_2 + \dots + x_n Q_{n,n-1} x_{n-1} + x_n Q_{n,n} x_n & \\ g_i(x_1, \dots, x_n) = & b_i - (a_{i,1} x_1 + \dots + a_{i,n} x_n) \leq 0 & \text{forall } i = 1, \dots, m \\ & \mathbf{x} \in \mathbf{R}^n & \end{array}$$

Note that the matrix $\mathbf{Q}$ must be *semi-definite positive (SDP)*, (that is: there exist a vector $\mathbf{x}$ such that $\mathbf{x}\mathbf{Q}\mathbf{x}' \geq 0$). In many applications the matrix $\mathbf{Q}$ is also symmetric ($\mathbf{Q} = \mathbf{Q}^{\mathbf{T}}$).

If the matrix $\mathbf{Q}$ is not semi-definite positive then it cannot be solved as a convex problem and it must be considered as an non-linear problem.

In the following, a concrete problem with random data for the two matrices $\mathbf{A}$, $\mathbf{Q}$ and the two vectors $\mathbf{b}$ and $\mathbf{c}$ is specified (with $n = 15$ and $m = 15$):

$$c = \begin{pmatrix} 16 & 73 & 39 & 81 & 68 & 60 & 90 & 11 & 33 & 89 & 71 & 12 & 24 & 23 & 47 \end{pmatrix}$$

$$A = \begin{pmatrix} 87 & 34 & 0 & 0 & 43 & 0 & 52 & 85 & 36 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 39 & 0 & 84 & 0 & 0 & 88 & 0 & 0 & 0 & 0 & 0 & 0 & 22 \\ 55 & 63 & 79 & 0 & 0 & 0 & 0 & 71 & 0 & 0 & 0 & 0 & 70 & 0 & 79 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 73 & 0 & 0 & 0 & 22 & 0 & 0 & 0 \\ 0 & 0 & 0 & 66 & 0 & 0 & 34 & 0 & 0 & 24 & 0 & 61 & 0 & 0 & 64 \\ 0 & 0 & 16 & 19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 \\ 15 & 70 & 61 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 52 & 0 & 14 & 0 & 0 & 92 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 98 & 0 & 19 & 0 & 0 \\ 0 & 70 & 37 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 86 & 0 & 93 & 0 \\ 90 & 0 & 82 & 0 & 66 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 \\ 77 & 30 & 0 & 0 & 0 & 0 & 13 & 10 & 0 & 0 & 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 72 & 0 & 0 & 0 & 79 & 0 & 0 & 0 & 0 & 0 & 11 & 90 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 32 & 55 & 0 & 23 & 0 & 0 & 0 & 0 & 0 & 36 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 58 \\ 44 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 119 \end{pmatrix}$$

$$diag(Q) = \begin{pmatrix} 7 & 5 & 5 & 6 & 17 & 19 & 6 & 8 & 16 & 19 & 12 & 5 & 13 & 17 & 18 \end{pmatrix}$$

Note that this matrix **Q** consisting of positive diagonal entries and zero otherwise is semidefinite positive.

The data given above specify the following explicit linear program:

$$\begin{aligned}
\text{min} \quad & 7x_1^2 + 5x_2^2 + 5x_3^2 + 6x_4^2 + 17x_5^2 + 19x_6^2 + 6x_7^2 + 8x_8^2 \\
& +16x_9^2 + 19x_{10}^2 + 12x_{11}^2 + 5x_{12}^2 + 13x_{13}^2 + 17x_{14}^2 + 18x_{15}^2 \\
& +16x_1 + 73x_2 + 39x_3 + 81x_4 + 68x_5 + 60x_6 + 90x_7 + 11x_8 \\
& +33x_9 + 89x_{10} + 71x_{11} + 12x_{12} + 24x_{13} + 23x_{14} + 47x_{15} \\
\text{subject to} \quad & 87x_1 + 34x_2 + 43x_5 + 52x_7 + 85x_8 + 36x_9 \geq 0 \\
& 39x_3 + 84x_5 + 88x_8 + 22x_{15} \geq 0 \\
& 55x_1 + 63x_2 + 79x_3 + 71x_8 + 70x_{13} + 79x_{15} \geq 0 \\
& 73x_8 + 22x_{12} \geq 0 \\
& 66x_4 + 34x_7 + 24x_{10} + 61x_{12} + 64x_{15} \geq 0 \\
& 16x_3 + 19x_4 + 18x_{15} \geq 0 \\
& 15x_1 + 70x_2 + 61x_3 \geq 0 \\
& 52x_9 + 14x_{11} + 92x_{14} \geq 58 \\
& 30x_8 + 98x_{11} + 19x_{13} \geq 44 \\
& 70x_2 + 37x_3 + 86x_{12} + 93x_{14} \geq 0 \\
& 90x_1 + 82x_3 + 66x_5 + 26x_{10} \geq 0 \\
& 77x_1 + 30x_2 + 13x_7 + 10x_8 + 50x_{12} \geq 0 \\
& 72x_2 + 79x_6 + 11x_{12} + 90x_{13} \geq 0 \\
& 32x_6 + 55x_7 + 23x_9 + 36x_{15} \geq 119 \\
& x_j \geq 0 \quad \text{forall } j \in \{1 \dots 15\}
\end{aligned}$$

Modeling Steps: The formulation of the model in LPL modeling language is straightforward and the notation is close to the mathematical notation using indices: First, the two sets `i` and `j` are defined. Then the data are declare and assigned as parameters `A`, `c`, `b`, and a semidefinite positive matrix `Q`. The variable vector `x` is declared, and finally the constraints `R` and the minimizing objective function `obj` are written.

Note that the data matrices A, b, c, and Q are generated using LPL's own random generator. (To generate each time the same data, the code can also use the function `SetRandomSeed(a)` where `a` is an integer.)

Listing 5: The Complete Model implemented in LPL [10]

```
model Qp15 "A Quadratic Convex Program (math)";
  set i := [1..15];  j,k := [1..15];
  parameter A{i,j}:= Trunc(if(Rnd(0,1)<.25,Rnd(10,100)));
            c{j}  := Trunc(Rnd(10,100));
            b{i}  := Trunc(if(Rnd(0,1)<.15, Rnd(0,120)));
            Q{j,k}:= Trunc(if(j=k, Rnd(5,20))); //SDP
  variable  x{j};
  constraint R{i} : sum{j} A*x >= b;
  minimize obj    : sum{j} c*x + sum{j,k} x[j]*Q*x[k];
  Writep(obj,x);
end
```

Solution: The small model defined above has the following optimal solution:

$$x = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0.051 & 1.37 & 0.7 & 0.839 & 0 & 0.23 & 0 & 0 & 0.121 & 0.63 \end{pmatrix}$$

The optimal value of the objective function is:

$$obj = 245.4168$$

Further Comments: There are interesting applications in *Portfolio Theory* for the quadratic convex problems. Especially, the Markowitz approach in portfolio models can be formulated as a QP model. For several implemented models see my book Casebook Studies I. Other applications are robust optimization and chance constraint optimization, and much more.

Note that the constraints can also be quadratic, we then have a QPC model where the Q matrix must be semi-definite positive:

$$\begin{array}{lll} \min & \displaystyle\sum_{j \in J} c_j x_j & \\ \text{subject to} & \displaystyle\sum_{j \in J, k \in J} x_j Q_{j,k} x_k + \sum_{j \in J} a_{i,j} x_j \geq b_i & \text{forall } i \in I \\ & x_j \geq 0 & \text{forall } j \in J \\ & J = \{1 \ldots m\},\ I = \{1 \ldots n\},\ m, n \geq 0 & \end{array}$$

8 A 0-1-Quadratic Program (examp-qp01)

Problem: The general **0-1-quadratic (convex) programming** model – called **0-1-QP** – contains n linear constraints and m binary variables and a quadratic convex objective function. It can be compactly formulated as follows (see [12]):

$$\begin{array}{lll} \min & \displaystyle\sum_{j\in J,k\in J} x_j Q_{j,k} x_k + \sum_{j\in J} c_j x_j & \\ \text{subject to} & \displaystyle\sum_{j\in J} A_{i,j}\cdot x_j \geq b_i & \text{forall } i \in I \\ & x_j \in \{0,1\} & \text{forall } j \in J \\ & J = \{1\ldots m\},\ I = \{1\ldots n\},\ m,n \geq 0 & \end{array}$$

An even more compact formulation using matrix notation for the model is :

$$\begin{array}{ll} \min & x\cdot Q\cdot x' + c\cdot x \\ \text{subject to} & A\cdot x \geq b \\ & x \in \{0,1\} \end{array}$$

Solve this problem – x are unknowns, A, b, and c are given – with $n = 15$ and $m = 15$ with the following data:

$$\mathbf{c} = \begin{pmatrix} 16 & 73 & 39 & 81 & 68 & 60 & 90 & 11 & 33 & 89 & 71 & 12 & 24 & 23 & 47 \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} 87 & 34 & 0 & 0 & 43 & 0 & 52 & 85 & 36 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 39 & 0 & 84 & 0 & 0 & 88 & 0 & 0 & 0 & 0 & 0 & 0 & 22 \\ 55 & 63 & 79 & 0 & 0 & 0 & 0 & 71 & 0 & 0 & 0 & 0 & 70 & 0 & 79 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 73 & 0 & 0 & 0 & 22 & 0 & 0 & 0 \\ 0 & 0 & 0 & 66 & 0 & 0 & 34 & 0 & 0 & 24 & 0 & 61 & 0 & 0 & 64 \\ 0 & 0 & 16 & 19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 \\ 15 & 70 & 61 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 52 & 0 & 14 & 0 & 0 & 92 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 98 & 0 & 19 & 0 & 0 \\ 0 & 70 & 37 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 86 & 0 & 93 & 0 \\ 90 & 0 & 82 & 0 & 66 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 \\ 77 & 30 & 0 & 0 & 0 & 0 & 13 & 10 & 0 & 0 & 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 72 & 0 & 0 & 0 & 79 & 0 & 0 & 0 & 0 & 0 & 11 & 90 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 32 & 55 & 0 & 23 & 0 & 0 & 0 & 0 & 0 & 36 \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 58 \\ 44 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 119 \end{pmatrix}$$

$$diag(Q) = \begin{pmatrix} 7 & 5 & 5 & 6 & 17 & 19 & 6 & 8 & 16 & 19 & 12 & 5 & 13 & 17 & 18 \end{pmatrix}$$

The data given above specify the following explicit linear program:

$$
\begin{aligned}
\min \quad & 7x_1x_1 + 5x_2x_2 + 5x_3x_3 + 6x_4x_4 + 17x_5x_5 + 19x_6x_6 \\
& +6x_7x_7 + 8x_8x_8 + 16x_9x_9 + 19x_{10}x_{10} + 12x_{11}x_{11} \\
& +5x_{12}x_{12} + 13x_{13}x_{13} + 17x_{14}x_{14} + 18x_{15}x_{15} \\
& +16x_1 + 73x_2 + 39x_3 + 81x_4 + 68x_5 + 60x_6 + 90x_7 + 11x_8 \\
& +33x_9 + 89x_{10} + 71x_{11} + 12x_{12} + 24x_{13} + 23x_{14} + 47x_{15} \\
\text{subject to} \quad & 87x_1 + 34x_2 + 43x_5 + 52x_7 + 85x_8 + 36x_9 \geq 0 \\
& 39x_3 + 84x_5 + 88x_8 + 22x_{15} \geq 0 \\
& 55x_1 + 63x_2 + 79x_3 + 71x_8 + 70x_{13} + 79x_{15} \geq 0 \\
& 73x_8 + 22x_{12} \geq 0 \\
& 66x_4 + 34x_7 + 24x_{10} + 61x_{12} + 64x_{15} \geq 0 \\
& 16x_3 + 19x_4 + 18x_{15} \geq 0 \\
& 15x_1 + 70x_2 + 61x_3 \geq 0 \\
& 52x_9 + 14x_{11} + 92x_{14} \geq 58 \\
& 30x_8 + 98x_{11} + 19x_{13} \geq 44 \\
& 70x_2 + 37x_3 + 86x_{12} + 93x_{14} \geq 0 \\
& 90x_1 + 82x_3 + 66x_5 + 26x_{10} \geq 0 \\
& 77x_1 + 30x_2 + 13x_7 + 10x_8 + 50x_{12} \geq 0 \\
& 72x_2 + 79x_6 + 11x_{12} + 90x_{13} \geq 0 \\
& 32x_6 + 55x_7 + 23x_9 + 36x_{15} \geq 119 \\
& x_j \in \{0, 1\} \quad \text{forall } j \in \{1 \ldots 15\}
\end{aligned}
$$

Modeling Steps: A 0-1-quadratic program (0-1-QP) is a mathematical model that consists of a number ($n \geq 0$) of linear inequalities in a number ($m \geq 0$) of binary variables. Furthermore, it defines an quadratic convex objective function that is to be minimized or maximized. The 0-1-QP model has many applications in quantitative decision making. The formulation of the model in LPL modeling language is straightforward and the notation is close to the mathematical notation using indices: First we define the two sets `i` and `j`. Then we declare and assign the data as parameters `A`, `c`, `b`, and a semidefinite positive (SDP) matrix `Q`. The variable vector `x` is declared, and finally the constraints `R` and the minimizing objective function `obj`.

Note that the data matrices A, b, c, and Q are generated using LPL's own random generator. (To generate each time the same data, the code can also use the function `SetRandomSeed(a)` where `a` is an integer.) Note that the unique difference within this model and the QP model (see examp-qp) consists of the word `binary` in declaration of the variables.

Listing 6: The Complete Model implemented in LPL [10]

```
model Qp15_01 "A 0-1-Quadratic Program (math)";
  set i := [1..15];  j,k := [1..15];
  parameter A{i,j}:= Trunc(if(Rnd(0,1)<.25,Rnd(10,100)));
            c{j}  := Trunc(Rnd(10,100));
            b{i}  := Trunc(if(Rnd(0,1)<.15,Rnd(0,120)));
            Q{j,k}:= Trunc(if(j=k, Rnd(5,20))); //SDP
  binary variable  x{j};
  constraint R{i} : sum{j} A*x >= b;
  minimize obj    : sum{j} c*x + sum{j,k} x[j]*Q*x[k];
  Writep(obj,x);
end
```

Solution: The model has the following solution:

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

The optimal value of the objective function is:

$$obj = 336$$

Further Comments: Interesting applications for the iQP model come from portfolio theory. Especially, if we want to limit the number of assets in a portfolio we must use 0-1 variables. An applications come from clustering problems. An example is bill035.

9 Second-Order Cone (socp1)

Problem: A second-order cone model contains quadratic constrainst of the form

$$\sum_{i \in I} x_i^2 \leq w^2$$

where x_i and w are variables. These constraints are convex and they are automatically recognized and solved by Gurobi and Cplex, for example.

A small model example is given here.

Listing 7: The Complete Model implemented in LPL [10]

```
model socp1 "Second-Order Cone (math)";
  set i:=[1..3];   j:=[1..5];
  parameter a{i,j} :=
    [19  0 -17 21  0 , 12 21 0  0  0 , 0 12  0  0 16];
  variable w; x{j};
  constraint
    A{i}: sum{j} a*x = 1;
    B: sum{j} x^2 <= w^2;
  minimize obj: w;
  Write('  w = %8.5f\n', w);
  Write{j}(' x%1s = %8.5f \n',j,x);
end
```

10 Rotated second-order Cone (socp2)

Problem: A rotated second-order cone model contains quadratic constrainst of the form

$$\sum_{i \in I} x_i^2 \leq wv$$

where x_i, v, and w are variables. These constraints are convex and they are automatically recognized and solved by Gurobi and Cplex, for example.

A small model example is given here.

Listing 8: The Complete Model implemented in LPL [10]

```
model socp2 "Rotated second-order Cone (math)";
  set i:=1..3;
  parameter c{i} := [11 7 9];
            b{i} := [ 5 6 8];
  variable x{i} [0..100]; r{i} [0..100]; k{i};
  constraint
    A: sum{i} 2*r <= 1;
    B{i}: k = Sqrt(b);
    C{i}: k^2 <= 2*x*r;
  minimize obj: sum{i} c*x;
  Writep(obj,x,r,k);
end
```

11 A NQCP model (bilinear)

Problem: All models with quadratic constraints seen so far are convex: QP, QPC, SOCP.

Models with only linear and quadratic terms which are non-convex are called NQCP models in LPL. They are much harder to solve, but Gurobi 12.0 can solve them (the parameter "nonconvex" must be set to 2). (LPL automatically sets this parameter when calling Gurobi.)

A small example is this model from the Gurobi model library

Listing 9: The Complete Model implemented in LPL [10]

```
model bilinear "A NQCP model (math)";
   /*integer*/ variable x; y; z; --(x integral in second version)
  maximize obj: x;
  constraint
    A: x + y + z <= 10;
    B: x * y <= 2;          --(bilinear inequality)
    C: x * z + y * z = 1;   --(bilinear equality)
    D: x>=0 and y>=0 and z>=0;
  Writep(obj,x,y,z);
end
```

12 Largest Empty Rectangle (quadrect)

Problem: Find the largest empty rectangle in a unit square filled with random points, see Figure 2. This problem is from Erwin Kalvelagen, see also the Blog of Hexaly.
The problem is interesting for checking the solvability of a *non-convex quadratic problem* using the commercial solvers Gurobi , Cplex, and Hexaly.
A extension to higher dimension of this model is given in quadrect1.

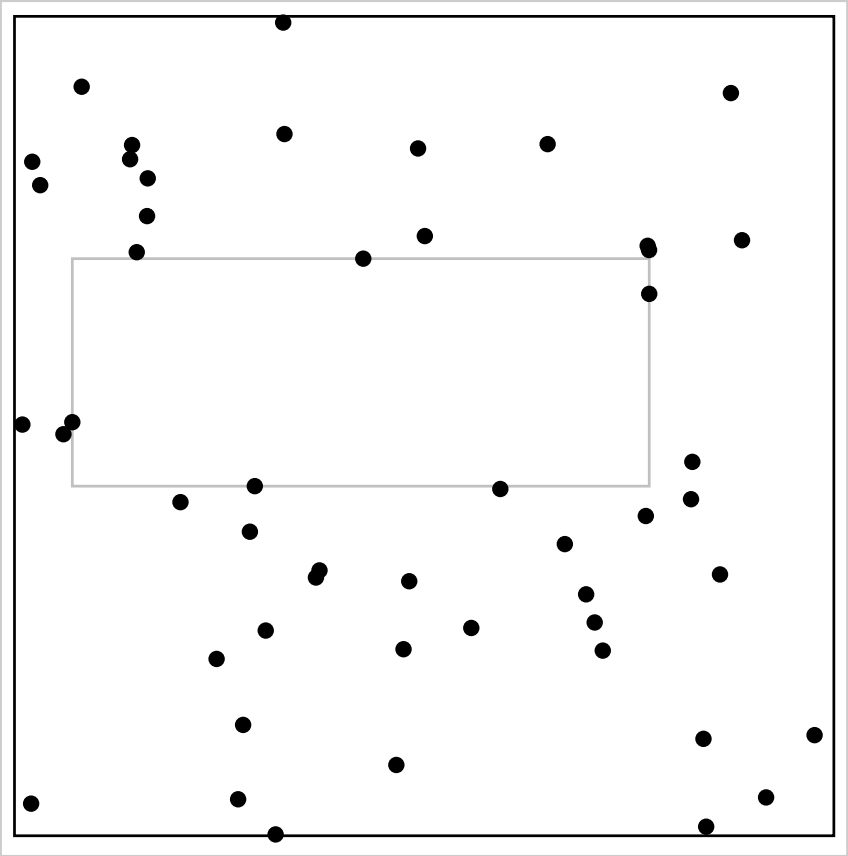

Figure 2: Largest Empty Rectangle with 50 points

Modeling Steps: Let us generate n points (x_i, y_i) with $i \in \{1, \ldots, n\}$, that are uniformly distributed in a unit square. The unknown empty rectangle within the unit square can be defined by two corner points (top/left – bottom/right) (x^a, y^a) and (x^b, y^b). These are the variables of our problem.
The model then can be formulated as follows[3] :

$$
\begin{array}{lll}
\max & (x^b - x^a) \cdot (y^b - y^a) & \\
\text{s.t.} & x^a \le x^b \ \wedge \ y^a \le y^b & \\
& x^a \ge x_i \ \vee \ x^b \le x_i \ \vee \ y^a \ge y_i \ \vee \ y^b \le y_i & \text{forall } i \in \{1, \ldots, n\} \\
& 0 \le x^a, y^a, x^b, y^b \le 1 & \\
& n > 0 &
\end{array}
$$

The objective function defines the size of the empty rectangle (width $\times$ height) and is a non-convex quadratic function. The first constraint requires that the point (x^a, y^a) is smaller than

[3]Note that the two operators $\wedge$ and $\vee$ are the Boolean operators "and" and "or"

point (x^b, y^b). The second constraint requires that at least one of the four disjunctive terms must be true for each i. Note, this model is a non-convex quadratic model. Recent versions of Gurobi and Cplex allow to solve such problems.

Listing 10: The Complete Model implemented in LPL [10]

```
model quadrect "Largest Empty Rectangle (puzzle)";
  --SetSolver(Hexaly,'TimeLimit=100');
  --SetSolver(cplexLSol);
  set i := 1..400   "a set of points";
  parameter x{i}:=Rnd(0,1); y{i}:=Rnd(0,1);
  variable xa [0..1]; ya [0..1];
           xb [0..1]; yb [0..1];
  constraint
    A: xa<=xb and ya<=yb;
    B{i}: xa>=x or xb<=x or ya>=y or yb<=y;
  maximize obj: (xb-xa)*(yb-ya);
  Draw.Scale(300,300);
  Draw.Rect(0,0,1,1,1,0);
  Draw.Rect(xa,ya,xb-xa,yb-ya,1,2);
  {i} Draw.Circle(x,y,.01);
end
```

Further Comments: If the problem is solved with Gurobi or Cplex, LPL translates the problem automatically to the following model :

$$
\begin{array}{lll}
\max & (x^b - x^a) \cdot (y^b - y^a) & \\
\text{s.t.} & x^a \leq x^b & \\
 & y^a \leq y^b & \\
 & x^a \geq x_i \cdot \delta_i^1 & \text{forall } i \in \{1, \ldots, n\} \\
 & y^a \geq y_i \cdot \delta_i^3 & \text{forall } i \in \{1, \ldots, n\} \\
 & x^b \leq 1 - (1 - x_i) \cdot \delta_i^2 & \text{forall } i \in \{1, \ldots, n\} \\
 & x^a \leq 1 - (1 - y_i) \cdot \delta_i^4 & \text{forall } i \in \{1, \ldots, n\} \\
 & \delta_i^1 + \delta_i^2 + \delta_i^3 + \delta_i^4 \geq 1 & \text{forall } i \in \{1, \ldots, n\} \\
 & 0 \leq x^a, y^a, x^b, y^b \leq 1 & \\
 & \delta_i^1,\ \delta_i^2,\ \delta_i^3,\ \delta_i^4 \in [0, 1] & \text{forall } i \in \{1, \ldots, n\} \\
 & n > 0 &
\end{array}
$$

Furthermore, LPL detects that this problem is an iNCQP (integer Non-Convex Quadratic Problem) and it adds automatically "NonConvex=2" (for the Gurobi solver) and "OptimalityTarget 3" (for the Cplex solver). If the solver Hexaly is used, none of these transformations are needed and LPL can pass the model as is.

Solution: The solution for $n = 50$ is shown in Figure 2 above.
With $n = 200$ (200 points), Gurobi 11.0 solves the problem within 8secs optimally, Cplex 20.0 has still a gap of 40% after 10mins and seems no to advance substantially. The Hexaly takes less than 2secs to solve it to optimality.
With $n = 400$, Gurobi found the optimum after 10mins, and the proof of optimality was found after 20mins. Hexaly found the optimum after 10secs and the proof of optimality after 20secs. This result confirms the claim at the Hexaly Blog.

Further Notes: In LPL, the second constraint could also be replaced by:

```
binary variable d1{i}; d2{i}; d3{i}; d4{i};
constraint
  B1{i}: d1 -> x <= xa;
  B2{i}: d2 -> y <= ya;
  B3{i}: d3 -> x >= xb;
  B4{i}: d4 -> y >= yb;
  B5{i}: d1+d2+d3+d4 >= 1;
```

Still another formulation is the linearization as given above (this is also the version to which LPL translates the model in order to solve it with Gurobi) :

```
binary variable d1{i}; d2{i}; d3{i}; d4{i};
constraint
  D1{i}: xa >= x*d1;
  D2{i}: ya >= y*d2;
  D3{i}: xb <= 1-(1-x)*d3;
  D4{i}: yb <= 1-(1-y)*d4;
  D5{i}: d1+d2+d3+d4 >= 1;
```

Questions

1. Modify the model for any dimension
2. Find the 5 smallest non-overlapping rectangles.

Answers

1. The model is implemented in quadrect1.
2. An implementation is given in model quadrect2.

13 Hanging Chain (chain)

The class of non-linear models is too diverse to be covered, too diverse in the sense of solution methods. Only one example is given here. Implemented in LPL, it is send to the solver Knitro to be solved. The following is the problem of a hanging chain.

Problem: Find the function $y = f(x)$ of a hanging chain (of uniform density) of length L suspended between the two points (x_a, y_a) and (x_b, y_b) where ($x_a < x_b$ and $L > x_b - x_a$) with minimal potential energy. See [4] and [5].

In physics and geometry, the curve of an idealized hanging chain or cable under its own weight when supported only at its ends is called a *catenary*. Galileo mistakenly conjectured that the curve was a parabola. Later Bernoulli and others proved that it is a hyperbolic cosine. Catenaries are used in a variety of application: An inverse catenary is the ideal shape of a freestanding arch of constant thickness. (See also in Wikipedia under "Catenary".)

Modeling Steps: The formula for potential energy is $E = ghm$ where h is the height, m is the mass and g is a gravitational constant. As a function the height h is y and the mass m is proportional to the arc length of the chain. Hence, the model is to minimize the potential energy E under the condition of the chain length L:

$$\begin{aligned} \min \quad & \int_{x_a}^{x_b} y \cdot \sqrt{1 + y'^2}\, dx \\ \text{subject to} \quad & \int_{x_a}^{x_b} \sqrt{1 + y'^2}\, dx = L \end{aligned}$$

Explanation: It is easy to see that in a small horizontal interval Δx of the function $y = f(x)$ and the corresponding Δy interval, the arc length of the function is:

$$\sqrt{\Delta x^2 + \Delta y^2} = \sqrt{\frac{\Delta x^2}{\Delta x^2} + \frac{\Delta y^2}{\Delta x^2}} = \sqrt{1 + \left(\frac{\Delta y}{\Delta x}\right)^2}\,\Delta x$$

With $\Delta x \to 0$ the length of the arc of the function $y = f(x)$ between x_a and x_b is then given by:

$$\int_{x_a}^{x_b} \sqrt{1 + \left(\frac{dx}{dy}\right)^2}\, dx = \int_{x_a}^{x_b} \sqrt{1 + y'^2}\, dx$$

To implement the problem in LPL, we need to discretizise the problem in say n vertical small intervals of width $\Delta x = (x_b - x_a)/n$. Let $I = \{0, \ldots, n\}$ and let $x_i = x_a(1 - i)/n + x_b\, i/n$ be the starting x-coordinate of the interval i, and let $y_i = f(x_i)$. Then the function $y = f(x)$ in an interval i can be approximated by the mid-point in that interval :

$$\left(\frac{x_i + x_{i-1}}{2}\,,\ \frac{y_i + y_{i-1}}{2}\right) \quad \text{forall } i \in \{1, \ldots, n\}$$

The derivative y' at interval i can be approximated by :

$$ydot_i = (y_i - y_{i-1})/\Delta x \quad \text{forall } i \in \{1, \ldots, n\}$$

Finally the arc length is approximated by :

$$\sqrt{1 + ydot_i^2}\ \Delta x \quad \text{forall } i \in \{1, \ldots, n\}$$

Note that the mass m is proportionl to the arc length (since the chain has uniform density), and the gravitational constant does not influence the form of the function. So, it can be dropped. With this hints the model can be easily constructed as follows:

Listing 11: The Complete Model implemented in LPL [10]

```
model chain "Hanging Chain (math)";
  parameter n := 50;
  set i := 0..n  "number of intervals for the discretization";
  parameter
    L  := 10   "length of the suspended chain";
    xa := 0    "left x coordinate";
    ya := 0    "height (y) of the chain at left";
    xb := 5    "right x coordinate";
    yb := 1    "height (y) of the chain at right" ;
    dx   := (xb-xa)/n  "interval";
    x{i} := xa+dx*i    "x-coor of chain";
  variable
    y{i}      "hanging height";
    ydot{i}   "derivative of y";
  constraint
    YDOT{i|i>0}: y[i] - y[i-1] = dx*ydot[i];
    LE: sum{i|i>0} Sqrt(1+ydot[i]^2)*dx = L;
    ST: y[0]=ya and y[n]=yb;
  minimize energy: sum{i|i>0} (y[i]+y[i-1])/2 * Sqrt(1+ydot[i]^2)*dx;
  //
  model output;
    Draw.Scale(1,10);
    Draw.DefFont('Verdana',8);
    Draw.XY(x,y);
  end
  // output the graph
  model output1;
    set k:=0..10;
    parameter X:=1; Y:=10; --scale
      x1:=400/X; y1:=300/Y; -- draw rect (0,0) to (x1,y1)
      xmin:=0; xmax:=xb; ymin:=-5; ymax:=0;
      Xt{k}:=xmin+(xmax-xmin)/(#k-1)*k; Yt{k}:=ymin+(ymax-ymin)/(#k-1)*
          k;;
    x{i}:=x*x1/Abs(xmax-xmin); y{i}:=y*y1/-(ymax-ymin);
    Draw.Scale(X,Y);
    Draw.DefFont('Verdana',8);
    Draw.Line(0,0,0,y1);
    {k} Draw.Line(x1/10*k,y1,x1/10*k,y1+10/Y);
    {k} Draw.Text(Round(Xt,-1)&'',x1/10*k-7/X,y1+20/Y);
    Draw.Line(0,y1,x1,y1);
    {k} Draw.Line(-10/X,y1-y1/10*k,0,y1-y1/10*k);
    {k} Draw.Text(Round(Yt,-1)&'',-25/X,y1-y1/10*k+3/Y);
    {i|i>0} Draw.Line(x[i-1],y[i-1],x[i],y[i]);
  end
end
```

Solution: The hanging chain of length $L = 10$ between the points $(x_a, y_a) = (0, 0)$ and $(x_b, y_b) = (5, 0)$ is drawn in figure 3. This figure is generated by the model `output`.

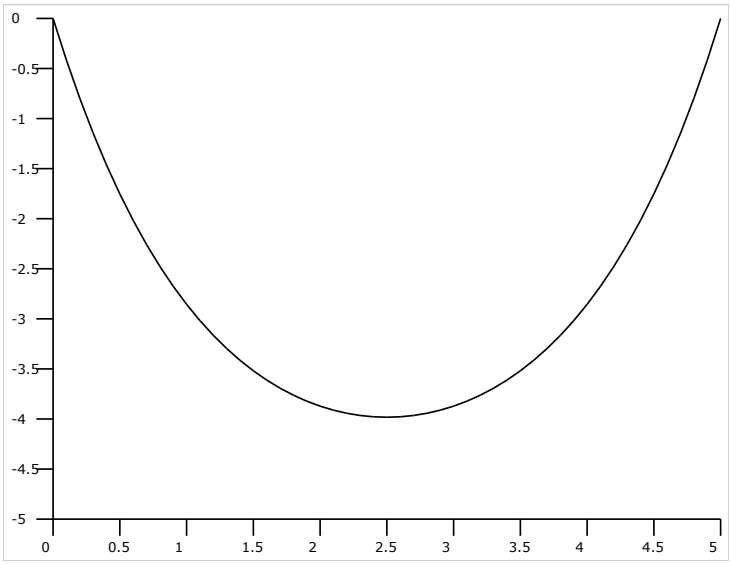

Figure 3: A Hanging Chain

14 Discrete Dynamic System (foxrabbit)

The class of discrete and continuous model system is huge. In LPL it is not possible right now to model differential systems, however an example of a discrete dynamic system is given here. It is the famous predator-prey model.

Problem: "Consider a forest containing foxes and rabbits where the foxes eat the rabbits for food. We want to examine whether the two species can survive in the long-term. A forest is a very complex ecosystem. So to simplify the model, we will use the following assumptions:

1. The only source of food for the foxes is rabbits and the only predator of the rabbits is foxes.
2. Without rabbits present, foxes would die out.
3. Without foxes present, the population of rabbits would grow.
4. The presence of rabbits increases the rate at which the population of foxes grows.
5. The presence of foxes decreases the rate at which the population of rabbits grows."

(This problem is from [1] p. 132ff).

Modeling Steps: The populations are modeled using a discrete dynamical system, that is, at each point in time – let's say each month – the size of the two populations is calculated and then mapped on a graph (for more information see [1]). We calculate the populations over a period of 500 months. Hence, let $i \in I = \{0, \ldots, 500\}$ a set of time points (first of month). Furthermore, let R_i and F_i be the population at time i. In the absense of the other specie, the populations would grow/shrink proportionally to the actual size, so (with $0 \leq \alpha, \delta \leq 1$) (The foxes would die out, while the population of rabbits grows infinitely.):

$$\begin{aligned} \Delta F_i &= F_{i+1} - F_i &&= -\alpha F_i \\ \Delta R_i &= R_{i+1} - R_i &&= \delta F_i \end{aligned}$$

Now, The presence of the other specie either increases (the foxes) its population by a rate of β, or decreases (the rabbits) its population by a rate of γ. Therefore we have:

$$\begin{aligned} \Delta F_i &= F_{i+1} - F_i &&= -\alpha F_i + \beta R_i \\ \Delta R_i &= R_{i+1} - R_i &&= -\gamma F_i + \delta R_i \end{aligned}$$

Reassigning the terms gives:

$$\begin{aligned} F_{i+1} &= (1-\alpha) F_i + \beta R_i \\ R_{i+1} &= -\gamma F_i + (1+\delta) R_i \end{aligned}$$

Given an initial population of $F_0 = 110$ and $R_0 = 900$, and the rates $\alpha = 0.12$, $\beta = 0.0001$, $\gamma = -0.0003$, and $\delta = 0.039$, it is easy to calculate the population at each time point using the simple LPL model :

Listing 12: The Complete Model implemented in LPL [10]

```
model foxrabbit "Discrete Dynamic System (math)";
  parameter n:=500;
  set i:= 0..n;
  parameter x{i}:=i; F{i}; R{i};;
  F[0]:= 110, R[0]:=900,
```

```
    {i|i<#i-1} (F[i+1]:=0.88*F[i] + 0.0001*F[i]*R[i],
                R[i+1]:=-0.0003*F[i]*R[i] + 1.039*R[i]);
  Draw.Scale(1,1);
  Draw.DefFont('Verdana',8);
  Draw.XY(x,F,R);
end
```

Solution: With the rates given, the populations ocillate as can be seen in Figure 4 (Black the population of foxes, red the population of rabbits.)

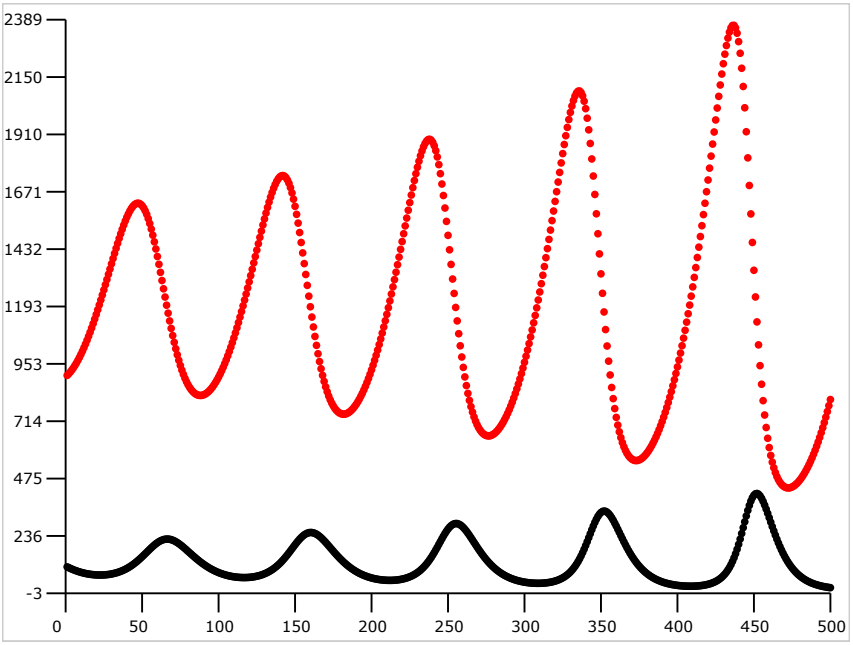

Figure 4: Fox-Rabbit Population over time

15 A Simple Permutation Model (examp-tsp)

An interesting model class are the permutation problems (PERM). There is a separate paper about permutation problems (see [9]), that explains in-depth what I mean by this problem class. Many of these problems can be formulated in linear (MIP) or non-linear discrete models too, but are inefficient in solving so. In simple permutation problem we are looking for a permuation out of all permutations that minimizes an expression. A typical example is the TSP problem (see also model tsp): The cities to be visited are numbered from 1 to n. Then any permutation of these numbers can be interpreted as "round trip" – visiting all cities in the order of the sequence of these numbers. The goal is now to find a particular permutation that minimizes the total distance. (In my book Casebook Studies I, several formulations of the TSP problem are given, for example.) In practice such problems are often solved using (meta)-heuristics. There are many problems that fall into this class: quadratic assignment (QAP), flowshop scheduling, linear ordering problem (LOP), etc. As an example, the traveling salsman problem (TSP) is an used here.

Problem: Given a complete graph $G = (V, E, c)$, with a set of nodes $i, j \in V = \{1, \ldots, n\}$ and edge list $E = V \times V$ and an edge length $c_{i,j}$ for each edge $(i, j) \in E$, find a *Hamilton cycle* of the graph, that is, a "round trip" in the graph that visits all nodes exactly once with minimal length (the sum of its edge lengths).

In a complete graph it is easy to get a Hamilton cycle: Every permutation of the nodes determines a Hamilton cycle. The difficulty arises when we ask for the shortest cycle. At the present time, we do not known a better method (in the worst case) to get the shortest one, then to enumerate *all* Hamilton cycles and pick the shortest, that means to check all $(n-1)!$ permutations and check each time its length. For a small graph with $n = 30$, we already have the gigantic number of permutations of $29! = 8.84 \cdot 10^{30}$, far to big to be enumerated even by supercomputers in reasonable time. Nevertheless clever methods and mathematical models have been invented that can solve larger problems *most of the time*.

Modeling Steps: A mathematical formulation can be derived directly from the definition: Out of all permutation, find the one that minimizes the cycle.

1. Let Π be the set of all permutations, and let π be a single permutation ($\pi \in \Pi$). For example, $\pi = [1, 2, 3, \ldots, n]$ is a single permutation. Let π_i be the i-th element in π.

2. Then the distance of a roundtrip of the permutation $\pi = [\pi_1, \pi_2, \ldots, \pi_n, \pi_1]$ can be formulated as :

$$\sum_{i=1}^{n} c_{\pi_i,\pi_j} \quad , \text{ with } j = i \mod n+1$$

   or

$$\sum_{i=1}^{n-1} c_{\pi_i,\pi_{i+1}} + c_{\pi_n,\pi_1}$$

   (j is the next element of i in the permutation π, and the next to the last is the first element in π). That is:

$$j = \begin{cases} i+1 & : \quad i < n \\ 1 & : \quad i = n \end{cases}$$

3. In other words: Let π be a permutation $\pi_1\pi_2\pi_3 \ldots \pi_{n-1}\pi_n$. Then c_{π_i,π_j} with $1 \le i, j \le n$ is the cost from the node π_i to node π_j. The summation, therefore, expresses the costs of a

round trip: $c_{\pi_1,\pi_2}+c_{\pi_2,\pi_3}+\ldots+c_{\pi_{n-1},\pi_n}+c_{\pi_n,\pi_1}$. Each possible round trip can be generated by a permutation. Minimizing this sum over all permutations means to look for the shortest round trip. Hence, the traveling salesman problem can be formulated as following:

$$\begin{aligned} \min \quad & \sum_{i=2}^{n} c_{\pi_{i-1},\pi_i} + c_{\pi_n,\pi_1} \\ \text{subject to} \quad & \pi \in \Pi \end{aligned}$$

Note the syntax characteristic for this kind of models is, that (integer) variable are used as indexes: π_i is a variable. For example, $\pi_2 = 7$ means that 7 is at the second position (from left to right) in the permutation. And c_{π_i,π_j} is the distance value of $c_{h,k}$ where $h = \pi_i$ and $k = \pi_j$. This notation extension allows the modeler to formulated the problem directly in the modeling language LPL:

Listing 13: The Main Model implemented in LPL [10]

```
model tsp "A Simple Permutation Model (tsp)";
  set i,j           "the node set";
  parameter c{i,j} "distance matrix";
  alldiff variable z{i} [1..#i] "a permutation";
  minimize obj: sum{i} c[z[if(i=1,#i,i-1)],z[i]];
end
```

Further Comments: The LPL code is an one-to-one formulation of the model above. The variable definition `alldiff` defines a permutation of numbers starting with 1. The minimization function is also close to the mathematical notation: in LPL, the construct with `if` is used instead of the modulo operation. One could also use the modulo operation to specify the objective function as follows:

```
minimize obj: sum{i} c[z[i],z[i%#i+1]];
```

This is a very compact formulation, but how is the problem solved? A model in LPL consisting of only a permutation variable and an objective function is identified by LPL as a special model type, called here as *Simple Permutation Problem* (PERM). These problems are sent to an special solver integrated in LPL which is based on an Tabu-Search metaheuristic method[4]. A much more powerful solver is the commercial Hexaly.

Solution: A random instance with 30 locations (nodes) is generated in the `data` model. So, $n = 30$ coordinates are generated for the nodes in a rectangle and calculate the distances as Euclidean distances:

Listing 14: The Data Model

```
model data;
  parameter n:=[30] "problem size";
  parameter X{i}; Y{i}; m:=Trunc(Sqrt(n));;
  i:=1..n;
  X{i}:=(i%m+1)*2+Trunc(Rnd(0,2));
  Y{i}:=(i/m+1)*2+Trunc(Rnd(0,2));
  c{i,j}:= Sqrt((X[j]-X[i])^2+(Y[j]-Y[i])^2);
end
```

[4]The solver is quite primitive an its neighborhood structure is based on a 2-opt exchange. It was implemented only to demonstrate the feasibility of connecting LPL to that kind of solver – also heuristics. For a deeper insight into Metaheuristics etc. consult the Internet.

With the problem size $n = 30$, the Tabu-Search method normally finds the optimal solution in a few seconds, which is 51.8052 in this case (see Figure 5).

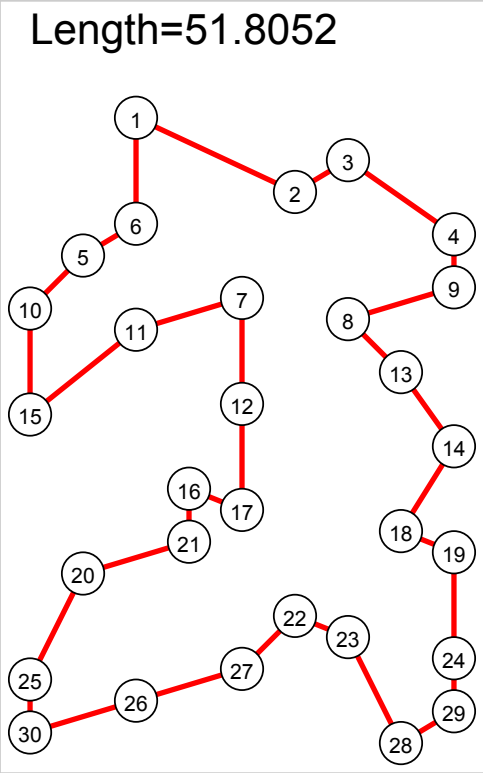

Figure 5: The Optimal Solution of a 30 Location Problem

16 Capacitated Vehicle Routing Problem (examp-cvrp)

Another example of a permutation problem is the capacitated vehicle routing problem (cvrp). In this problem, we are looking for a "partitioned permutation". A partitioned permutation is a permutation partitioned into subsequences. For example, the permutation $\{2,3,4,1,9,8,5,6,7,10\}$ could be partitioned into 3 subsequences as follows:

$$\{\{2,3,4\},\{1,9,8,5\},\{6,7,10\}\}$$

The first subsequence is $\{2,3,4\}$, the second is $\{1,9,8,5\}$, etc. this is exactly what is needed for the capacitated vehicle routing problem.

Problem: A fixed fleet of delivery vehicles of uniform capacity must service known customer demands for a single commodity from a common depot at minimum transit cost. An concrete application is given in cvrp. The models cvrp-1, cvrp-2, and cvrp-3 give MIP-formulations of the problem. These formulations are explained in my book Case Studies I.

To interpret this problem as a "partitioned permutation" problem is easy: Starting at a depot, the first vehicle visits the customers in the first subsequence and returns to the depot, the second vehicle, also starting from the depot, visits the customers in the second subsequence and returns to the depot, etc. Note that the depot is not part of the sequences itself.

Modeling Steps: Let $i, j \in I = \{1,\ldots,n-1\}$ be a set of customer locations (n is the number of locations including the warehouse (or the depot) and let $k \in K = \{1,\ldots,m\}$ be a set of trucks. Let the capacity of each truck be CA, and let the demand quantity to deliver to a customer i be dem_i. Furthermore, the distance between two customer i and j is $d_{i,j}$. Finally, the distance from the warehouse – the depot from where the trucks start – to each customer i is dw_i.

The customers are enumerated with integers from 1 to $n-1$. If there were one single truck (that must visits all customers), every permutation sequence of the numbers 1 to $n-1$ would define a legal tour. Since we have m trucks, to each truck a sequence of a subset of 1 to $n-1$ must be assigned. Hence, each truck starts at the depot, visiting a (disjoint) subset of customers in a given order and returns to the depot.

The variables can now be formulated as a "partitioned permutation". In LPL, we can declare these partitioned permutation with a (sparse) permutation variable $x_{k,i} \in [1 \ldots n-1]$. For example, for the example above we have :

$$x = \begin{pmatrix} 2 & 3 & 4 & - & - & - & - & - & - & - \\ 1 & 9 & 8 & 5 & - & - & - & - & - & - \\ 6 & 7 & 10 & - & - & - & - & - & - & - \end{pmatrix}$$

Note that the two-dimensional table is sparse – it only contains $n-1$ entries. The symbol "-" (dash) means: no entry. Hence, $x_{1,1} = 2$ means that the customer 2 is visited by the truck 1 right after the depot, $x_{1,2} = 3$ means that the next customer (after 2) is 3, etc.

The unique constraint that must hold for the partitioned permutation is the capacity of the trucks: each subset sequence must be chosen in such a way that the capacity of the truck is larger than the cumulated demand of the customers that it visits – note, the variables $x_{k,i}$ are used as indexes as well as in the condition of the summation:

$$\sum_{i|x_{k,i}} dem_{x_{k,i}} \leq CA \quad \text{forall } k \in K$$

We want to minimize the total travel distance of the trucks. Let cc_k be the size of the subset k (the numbers of customers that the truck k visits). (In the example above $c_k = \{3,4,3\}$.)

Of course, these numbers is variable and cannot be given in advance! Let $routeDist_k$ be the (unknown) travel distance of truck k, then we want to minimize the total distances:

$$\min \sum_k routeDist_k$$

where[5]

$$routeDist_k = \sum_{i \in 2..cc_k} d_{x_{k,i-1},x_{k,i}} + if(cc_k > 0, dw_{x_{k,1}} + dw_{x_{k,cc_k}} \quad \text{forall } k \in K$$

The term $dw_{x_{k,1}}$ denotes the distance of the truck k from the depot to the first customer, and $dw_{x_{k,cc_k}}$ is the distance of the truck tour k from the last customer to the depot, $d_{x_{k,i-1},x_{k,i}}$ is the distance from a customer to the next, and $\sum_{i \in 2..cc_k} \ldots$ sums that distances of a tour k.

As a variant we may also minimize, in a first round, the number of trucks used – this is added as a comment in the code:

Listing 15: The Main Model implemented in LPL [10]

```
model cvrp "Capacitated Vehicle Routing Problem (cvrp)";
  set i,j  "customers";
      k    "trucks";
  parameter
      d{i,j} "distances";
      dw{i}  "distance from/to the warehouse";
      dem{i} "demand";
      CA     "truck capacity";
  alldiff x{k,i} 'partition';
  expression
    cc{k}: count{i} x;
    trucksUsed{k}: cc[k] > 0;
    routeDist{k}: sum{i in 2..cc} d[x[k,i-1],x[k,i]]
       + if(cc[k]>0, dw[x[k,1]] + dw[x[k,cc[k]]]);
    nbTrucksUsed: sum{k} trucksUsed[k];
    totalDistance: sum{k} routeDist[k];
  constraint CAP{k}: sum{i|x} dem[x[k,i]] <= CA;
  --minimize obj1: nbTrucksUsed;
  minimize obj2: totalDistance;
end
```

Further Comments: The question is now what solver can solve this kind of model! I developed a simple Tabu-search solver that can deliver near optimal solution for small simple permutation problems, but not for partitioned permutation problems. However, there is a powerful commercial solver called Hexaly that can find near optimal solutions to large problems. LPL contains an (experimental) interface to that solver.

Solution: The LPL data code reads the "A-n32-k5.vrp" file from the Augerat et al. Set A instances.

Listing 16: The Data Model

```
model data;
  set h; //h is i plus 1, 1 is depot (warehouse)
```

[5]The expression $if(boolExpr, Expr)$ returns $Expr$ if $boolExpr$ is true else it returns 0 (zero).

```
    parameter de{h}; n; m; X{h}; Y{h}; string typ; dum;
    Read('A-n32-k5.vrp,%1;-1:DIMENSION',dum,dum,n);
    --Read('A-n45-k6.vrp,%1;-1:DIMENSION',dum,dum,n);
    Read('%1;-1:EDGE_WEIGHT_TYPE',dum,dum,typ);
    if typ<>'EUC_2D' then Write('Only␣EUC_2D␣is␣supported\n'); return
        0; end;
    Read('%1;-1:CAPACITY',dum,dum,CA);
    Read{h}('%1:NODE_COORD_SECTION:DEMAND_SECTION', dum,X,Y);
    Read{h}('%1:DEMAND_SECTION:DEPOT_SECTION', dum,de);
    m:=Ceil(sum{h} de/CA);
    k:=1..m;
    i:=1..n-1;
    d{i,j}:=Round(Sqrt((X[i+1]-X[j+1])^2+(Y[i+1]-Y[j+1])^2));
    dem{i}:=de[i+1];
    dw{i}:=Round(Sqrt((X[i+1]-X[1])^2+(Y[i+1]-Y[1])^2));
end
```

The Hexaly finds the optimal solution (see 6) after 25secs[6]

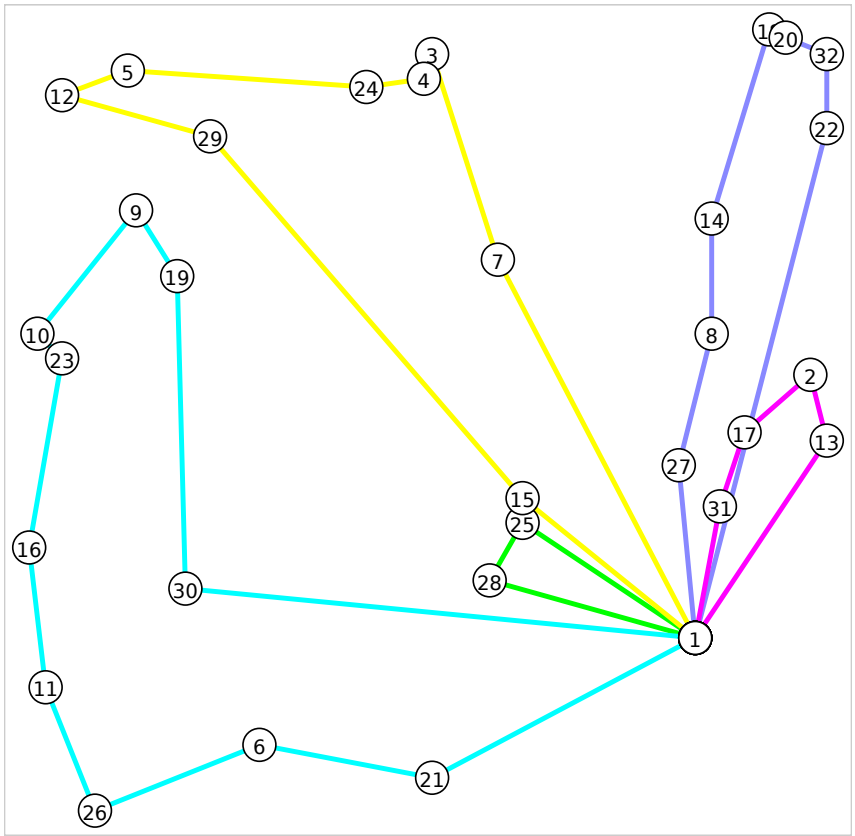

Figure 6: Optimal Solution to the "A-n32-k5.vrp" Instance

[6]Note that Hexaly is not simply a heuristic solver that finds hopefully near optimal solution. It integrates apparently many lower-bound checks that can guarantee the quality of the solution.

17 Binpacking (examp-binpack)

Problem: Another example, that can be formulated as a permutation problem is the Bin-Packing problem. In contrast the the CVRP problem (see examp-cvrp), the order of the subsets is not important: a bin just contains a subset of items, the order is not important.

The bin packing problem (BPP) is a classical model from the operations research (see also binpack for another formulation of the problem). Different items of given weights must be packed into a number of bins of given capacity. How many bins are needed to pack all items? A MIP implementation of the binpacking problem can be found in binpack.

Modeling Steps: Given a set of items $i \in I$ with a weight w_i and a set of bins $k \in K$ with capacity CA (we suppose that $w_i \leq CA$ for all $i \in I$), that is all items can be packed in a bin. We need at most $maxBins$ bins :

$$maxBins = \min(|I|, \sum_i w_i/CA)$$

So let $K = \{1, \ldots, maxBins\}$.
Let's enumerate all items from 1 to $|I|$. Then we are asked to partition these numbers into maximally $|K|$ subsets (but as few as possible), such that the cumulated weight of the items in a subset do not exceed the capacity CA. The item numbers form a permutation, that must be partitioned into subsets. We can formulated this by introducing a variable $x_{k,i} \in [0 \ldots |I|]$ with $k \in K$ and $i \in I$. For example: $x_{1,1} = 16$ means that the first item in bin 1 is the item with number 16, $x_{1,2} = 19$ means that the second item in bin 1 is the item with number 19, etc. $x_{i,k} = 0$ means that there is no k-th item in bin i.

The model then can be formulated as follows:

let	$bW_k = \sum_{i \mid x_{k,i} \neq 0} w_{x_{k,i}}$	forall $k \in K$
let	$bU_k = \underset{i}{\text{count}}\ x_{k,i} > 0$	forall $k \in K$
s.t.	$bW_k <= CA$	forall $k \in K$
min	$\sum_k bU_k$	
set partition	$x_{k,i} \in [1 \ldots \lvert I \rvert]$	forall $k \in K, i \in I$

bW_k is the (unknown) weight of a bin k. bU_k is true (1) if the bin k is not empty (it counts the items in a bin k and if there is at least 1 then it is true). The unique constraint makes sure that the capacity of a bin is not exceeded, and the objective function minimizes the number of bins used.

Listing 17: The Complete Model implemented in LPL [10]

```
model binpacking "Binpacking (binpack)";
  set i,j  "items";
      k    "bins";
  parameter CA "capacity of bin";
    weight{i}  "weight of item i";
  alldiff x{k,i} 'set␣partition';
  expression bWeight{k}: sum{i|x} weight[x];
  expression bUsed{k}: count{i} x > 0;
  constraint bCapa{k}: bWeight <= CA;
```

```
  minimize nrBins: sum{k} bUsed;
  //---
  model data;
    parameter n; m;
    Read('bin-t60-00.txt',n);
    i:=1..n;
    Read('%1;1',CA);
    Read{i}('%1;2',weight);
    parameter minBins:=Ceil(sum{i} weight/CA);
              maxBins:= min(#i, 2*minBins);;
    k:=1..maxBins;
  end
  //---
  model output;
    Write{k|bUsed}('Bin weight: %4d \
       | Items: %3d\n',bWeight,{i|x} x);
  end
end
```

Further Comments: In LPL, the permutation variables is declared as a `alldiff` variable with two dimensions and the attribute `'set partition'` in this case.

17.1 Permutation Problems as Implemented in LPL

There are various kinds of permutation problems, all are declared with an `alldiff` variable in LPL. The unique commercial solver (I know) that can solve this kind of problems is the Hexaly.

1. The variable is one-dimensional then we are looking for a simple permutation (the keyword `variable` can be omitted):

   ```
   alldiff variable x{i};
   ```

   Example problems are TSP, LOP (linear ordering problem), QAP, etc. If a single objective function is requested – without constraints (as in the TSP) then the small problems can be solved with the internal Tabu-Search solver of LPL.

2. A variant of the one-dimensional permutation is the subset variant:

   ```
   alldiff variable x{i} 'notall';
   ```

   In this case we are looking only for a subset of the permutation. An example is the price collecting TSP, a round-trip where not all location are visited.

3. The variable is 2-dimensional without any additional attribute then are looking for a partitioned permutation. The set i defines the permutation itself and the set k is the partitioning. The permutation is **repeated** over k.

   ```
   alldiff x{k,i};
   ```

   Example with defines a permutation of 4 items repeated over 3 sets.

$$x_{k,i} = \begin{pmatrix} 2 & 3 & 4 & 1 \\ 1 & 3 & 2 & 4 \\ 4 & 1 & 2 & 3 \end{pmatrix}$$

   An example is the jobshop problem, where the job sequence (permutation) is repeated over all machines (see jobshop1).

4. The variable is 2-dimensional with the additional attribute `'patition'` means that the unique permutation is partitioned over the set k.

   ```
   alldiff x{k,i} 'partition';
   ```

   This is the case for the exam-cvrp model: the set of locations defines a permutation, which is partitioned into subtours. Note that the order in the subsets (subtours) matters.

5. The variable is 2-dimensional with the additional attribute `'set patition'` means that the unique permutation is partitioned over the set k, but the order of the items in the subsets does not matter.

   ```
   alldiff x{k,i} 'set␣partition';
   ```

   An example ist the bin-packing problem: the items to place into the bins define a permutation, which is partitioned into the bins, but the order within the bins does not matter.

6. The variable is 2-dimensional with the additional attribute `'cover'` means that the unique permutation is distributed over the set k, the items in the permutation can be repeated in the different subsets.

   ```
   alldiff x{k,i} 'cover';
   ```

   A example is the Split Delivery Vehicle Routing (SDVRP) problem, the same customer (location) can be visited by more than one vehicle (can be in more then one subtour) (see sdvrp).

18 Additional Variable Types

Variables – as seen – can be in the domain *real*, *integer*, or *binary*. An additional type of variables are special integer variables: the *permutation variables*. Another type of variables are *interval* variables. LPL defines them for the Hexaly solver, a model examples is rcpsp1.

Three further variables types are *semi-continuous*, *semi-integer*, or *multiple-choice* variables. There is no need to introduce extra syntax for them, one can just extend the lower/upper bound specifications of the variable definitions. In LPL, the lower/upper bound are defined as `[lo..up]`.

18.1 Semi-continuous Variable (semi-1)

A semi-continuous variable is defined as a real variable that can be zero or between a (positive) lower and upper bound.

A semi-continuous variable x can be modeled by introducing a binary variable z and an addition constraint (where lo and up are the (positive) lower and upper bounds) defined as :

$$lo \cdot z \leq x \leq up \cdot z$$

This constraint models the two alternatives: if $z = 0$ then $x = 0$, otherwise if $z = 1$ then x must be between the lower and upper bound: $lo \leq x \leq up$.
In LPL, a semi-continuous variable can simply be defined as:

```
variable x [0,lo..up];
```

The transformations, adding a binary variable and the constraint as defined above, are done automatically by LPL

Listing 18: The Complete Model implemented in LPL [10]

```
model semi1 "Semi-continuous Variable (math)";
  parameter lo:=5; up:=7;
  variable x [0,lo..up];
  minimize obj: x;
  --maximize obj: x;
  Writep(x);
end
```

This model will be translated into the following model by LPL (the names may be different):

```
model semi1 "Semi-continuous variable";
  parameter lo:=5; up:=7;
  variable x;
  binary variable z;
  constraint A: 5*z <= x <= 7*z;
  solve;
end
```

18.2 Semi-integer Variable (semi-2)

A semi-integer variable is defined as an integer variable that can be zero or between a (positive) lower and upper integer bound.

A semi-integer variable x can be modeled by introducing a binary variable z and an addition constraint (where lo and up are the (positive) lower and upper bounds) defined as :

$$lo \cdot z \leq x \leq up \cdot z$$

This constraint models the two alternatives: if $z = 0$ then $x = 0$, otherwise if $z = 1$ then x must be between the lower and upper bound: $lo \leq x \leq up$.
In LPL, a semi-integer variable can simply be defined as:

```
integer variable x [0,lo..up];
```

The transformations, adding a binary variable and the constraint as defined above, are done automatically by LPL

Listing 19: The Complete Model implemented in LPL [10]

```
model semi2 "Semi-integer Variable (math)";
  parameter lo:=5; up:=7;
  integer variable x [0,lo..up];
  minimize obj: x;
  --maximize obj: x;
  Writep(x);
end
```

This model will be translated into the following model in LPL (the names may be different):

```
model semi1 "Semi-integer variable";
  parameter lo:=5; up:=7;
  integer variable x;
  binary variable z;
  constraint A: 5*z <= x <= 7*z;
  solve;
end
```

18.3 A Multiple Choice Variable (mchoice-3)

A multiple choice variable is a real or integer variable, that can be assigned to values of multiple intervals. For example, a variable x is restricted to the values 2, 4, between 6 and 8, and 11, that is :

$$x \in \{2, 4, 6 \ldots 8, 11\}$$

Note that a multiple-choice variable is a generalization of a semi-continuous/integer variable. It can be modeled as follows:

1. Introduce a set and two parameters as follows:

$$i \in I = \{1 \ldots 4\} \quad a_i = \{2, 4, 6, 11\} \quad b_i = \{2, 4, 8, 11\}$$

2. Add a binary variable

$$z_i \in \{0, 1\} \quad i \in I$$

3. Add the two constraints:

$$\sum_{i \in I} a_i z_i \leq x \leq \sum_{i \in I} b_i z_i$$
$$\sum_{i \in I} z_i = 1$$

In LPL, the multiple-choice variable can simply be defined as (the variable may also be of type `integer`):

```
variable x [2,4,6..8,11];
```

Listing 20: The Complete Model implemented in LPL [10]

```
model mchoice "A Multiple Choice Variable (math)";
  integer variable x [2,4,6..8,11];
  minimize obj: x;
  --maximize obj: x;
  Writep(x);
end
```

This model will be translated into the following model by LPL (the names may be different):

```
model mchoice "A multiple choice variable";
  variable x;
  set i:=1..4;
  parameter a{i}:=[2,4,6,11];
            b{i}:=[2,4,8,11];
  binary variable z{i};
  constraint A: sum{i} a*z <= x <= sum{i} b*z;
             B: sum{i} z = 1;
  solve;
end
```

An alternative way to formulate a multiple-choice variable is by using the logical *or* operation. Although correct, this is not recommended in general, because it will generate more binary variables. For the example above one may formulate the model as follows:

```
model mchoice "A multiple choice variable";
  variable x [2..11];
  constraint A: x=2 or x=4 or 6<=x<=8 or x=11;
  solve;
end
```

19 Additional Constraint Types

Besides of linear and non-linear constraints[7], there are many other classes in practice. Some can be transformed and formulated as linear or non-linear constraints: the *SOS1*, the *SOS2*, and the *complementarity* constraints, as explained below.

19.1 Sos1 Constraint (sos-1)

A SOS1 (special order set of type 1) constraint is a set of variables where at most one variable in the set can take a value different from zero.

Given a set of variables x_i with $i \in I = \{1, \ldots, n\}$. If the variable are binary then a SOS1 constraint can be formulated simply as :

$$\sum_i x_i \leq 1$$

[7] Since a constraint expression can include Boolean operand and Boolean operator, constraints with logical connectors are included in this definition

If the variables are real (or integer) then an additional binary variable y_i has to be introduced together with the constraints (where up_i is an upper bound on the variable x_i) :

$$\begin{aligned} \sum_i y_i &\leq 1 \\ x_i &\leq up_i \cdot y_i \quad \text{forall } i \in I \end{aligned}$$

In LPL, we can use the function `Sos1` to specify a SOS1 constraint. For example, to specify that $v = 0$ or $w = 0$ (at most one can be non-zero) we write:

```
constraint C: Sos1(v,w)
```

To specify that only in a indexed list of variable must be non-zero, one specifies:

```
constraint D: Sos1( {i} x[i] );
```

Listing 21: The Complete Model implemented in LPL [10]

```
model sos1 "Sos1 Constraint (math)";
  set i:=1..10;
  parameter up:=10.2;
  variable x{i} [0..up];
  constraint A: Sos1({i} x);
  maximize obj: sum{i} i*x;
  Writep(obj,x);
end
```

Note only Gurobi understands this syntax, otherwise the explicit formulation must be used as follows:

```
model sos1;
  set i:=1..10;
  parameter up:=10.2;
  variable x{i}  [0..up];
  binary variable y{i};
  constraint A: sum{i} y <= 1;
  constraint B{i}: x <= up*y;
  maximize obj: sum{i} i*x;
  Writep(obj,x);
end
```

19.2 Sos2 Constraint (sos-2)

A SOS2 (special order set of type 2) constraint is a set of variables where at most two adjacent variable in the set can take a value different from zero.

Given a set of variables x_i with $i \in I = \{1, \ldots, n\}$. If the variables are real (or integer) then an additional binary variable y_i has to be introduced together with the constraints (where up_i is an upper bound on the variable x_i) :

$$\begin{aligned} \sum_i y_i &\leq 2 \\ x_i &\leq up_i \cdot y_i \quad \text{forall } i \in I \\ y_i + y_j <= 1 \quad &\text{forall } i \in I, j \in \{i+2, \ldots, n\} \end{aligned}$$

In LPL, we can use the function `Sos2` to specify a SOS2 constraint. For example, to specify that only in a indexed list of variable must be non-zero, one specifies:

```
  constraint D: Sos2( {i} x[i] );
```

Listing 22: The Complete Model implemented in LPL [10]

```
model sos2 "Sos2 Constraint (math)";
  set i:=1..10;
  parameter up:=10.2;
  variable x{i} [0..up];
  constraint A: Sos2({i} x);
  maximize obj: sum{i} i*x;
  Writep(obj,x);
end
```

Note only Gurobi (and Cplex) understand this syntax from LPL, otherwise the explicit formulation must be used as follows:

```
  model sos2;
    set i,j:=1..10;
    parameter up:=10.2;
    variable x{i} [0..up];
    binary y{i};
    constraint B: sum{i} y <= 2;
    constraint C{i}: x <= up*y;
    constraint D{i,j| j>=i+2}: y[i] + y[j] <= 1;
    maximize obj: sum{i} i*x;
    Writep(obj,x);
  end
```

19.3 MPEC Model Type (compl-3)

A complementarity constraint enforces that two variables are complementary to each other, that is, the following conditions hold for scalar variables x and y:

$$x \cdot y = 0 \quad , \quad x \geq 0 \quad , \quad y \geq 0$$

In LPL, this condition can be formulated by a constraint as follows:

```
  constraint A: Complements(x,y);
```

A more general version can also be implemented in LPL, where the two scalar variables can be replaced by arbitrary expressions. LPL reformualtes this version. Hence, for example, the constraint

```
  constraint A1: Complements(2*x-1,4*y-1);
```

will be replaced by (where v and w are two new variables):

```
  variable v; w;
  constraint A1: v*w = 0;
  constraint X: v = 2*x-1;
  constraint Y: w = 4*y-1;
```

Note that Gurobi can solve these constraints if the expressions within the complementarity are linear or quadratic, otherwise a non-linear solver (such as Knitro) will do the job.

Listing 23: The Complete Model implemented in LPL [10]

```
model complement "MPEC Model Type (math)";
```

```
  variable x [0..10]; y [0..10];
  constraint
    A1: Complements(2*x-1,4*y-1);
  maximize obj: x + y;
  Writep(obj,x,y);
end
```

20 Global Constraints

Global constraints are constraints representing a specific relation on a number of variables. Some of them can be rewritten as a conjunction of algebraic constraints. Other global constraints extend the expressivity of the constraint framework. In this case, they usually capture a typical structure of combinatorial problems. Global constraints are used to simplify the modeling of constraint satisfaction problems (CSP), to extend the expressivity of constraint languages, and also to improve the constraint resolution. Many of the global constraints are referenced into an The Online Catalog, a catalog that presents a list of 423 global constraints.
Actually, LPL implements only a few to illustrate the use and the power in modeling. Depending on the solver used, these constraint must be transformed.

20.1 Alldiff and alldiff

One of the most famous global constraint is the *alldifferent* constraint. It requires that a certain number of (integer) variables must all be different from each other. A important special case are permutations. (Permutation problems have been defined and presented in the paper [9].)
In LPL, the *alldifferent* constraint come in two versions:

1. A variable vector can be declared with the keyword **alldiff**. A typical application is the TSP problem, as formulated in tsp, in which the variables are formulated as a permutation.

2. A global constraint **Alldiff()** defines lists of variables that must be different from each other. Typical models are the model sudokuM and the model vier.

The first version can be used for most permutation problems, as explained in the paper [9]: for the TSP-PC (price collecting TSP), for CVRP (Capacitated Vehicle Routing Problem), Jobshop, and many others, as explained in the paper. The syntax for a simple permutation is:

```
alldiff variable x{i} [1..#i];
```

The second version in a constraint has two forms:

```
constraint A: Alldiff(y,z,w,v);     // explicit list of variables
constraint B: Alldiff({i} x[i]);    // indexed list of variables
```

When using a MIP solver, LPL transforms both versions into linear constraints. There are several methods:
The first method (used by LPL) is as follows :

1. Add the binary variables: $y_{i,j}$ with $i, j \in I$.

2. Add the constraints:

$$1 \leq x_j - x_i + n y_{i,j} \leq n - 1 \quad \text{forall } i, j \in I, i < j$$

The second Method (avoiding big-M) is as follows :

1. Add the binary variables: $y_{i,j}$ with $i, j \in I$.
2. Add constraints:

$$\begin{aligned} \sum_j y_{i,j} &= 1 \qquad \text{forall } i \in I \\ \sum_i y_{i,j} &= 1 \qquad \text{forall } j \in I \\ x_i &= \sum_j j y_{i,j} \qquad \text{forall } i \in I \end{aligned}$$

Note the second method can be extended. The case $x_i \in \{1, \ldots, n\}$ where $i \in \{1, \ldots, n\}$ is a very special case of the more general problem where $x_i \in \{a_1, \ldots, a_n\}$ with $a_{i+1} > a_i$. This problem can be handled by the constraints:

$$\begin{aligned} \sum_j y_{i,j} &= 1 \qquad \text{forall } i \in I \\ \sum_i y_{i,j} &= 1 \qquad \text{forall } j \in I \\ x_i &= \sum_j a_j y_{i,j} \qquad \text{forall } i \in I \end{aligned}$$

Even the case with duplicates in a ($a_{i+1} \geq a_i$ (example: $a = \{1, 2, 2, 3, 3, 3\}$) can be handled with the previous constraints.
Another extension is to define a subset: consider $x_j \in \{a_1, \ldots, a_n\}$ where $j \in \{1, \ldots, m\}$ and $m < n$, for example, choose two values x_j from the set $\{1, \ldots, 3\}$.

$$\begin{aligned} \sum_j y_{i,j} &\leq 1 \qquad \text{forall } i \in I \\ \sum_i y_{i,j} &= 1 \qquad \text{forall } j \in I \\ x_i &= \sum_j a_j y_{i,j} \qquad \text{forall } i \in I \end{aligned}$$

For more information see Erwin Kalvelagen, see also [11].

20.2 Element

The element constraint requires that the (unknown) x-th entry of a data vector c_i with $i \in \{1, \ldots, k\}$ is exactly v.

$$Element(x, c_{i|i\in\{1,\ldots,k\}}, v)$$

In an explicitly mathematical version this could be formulated as :

$$v = c_x$$

Note that x is an (integer) variable and it is at an index position. v may or may not bo a variable. In LPL, the constraint can be written as follows :

```
constraint A: Element((x, {i}c, v);
```

When using a MIP solver the constraint is translated to :

$$
\begin{aligned}
& v = \sum_i c_i y_i \\
& y_i \leftrightarrow (x_i = i) \quad \text{forall } i \in I \\
& y_i \in \{0,1\} \quad \text{forall } i \in I
\end{aligned}
$$

A real application model is given in model assignCP another example is guer05-6a.

20.3 Occurrence

The occurrence constraint requires that between m and n ($m <= n$) elements of an variable vector x_i with $i \in \{1,\ldots,k\}$ must be assigned the value v.

$$Occurrence(x_{i|i\in\{1,\ldots,k\}}, v, m, n)$$

In an explicitly mathematical version this could be formulated as :

$$m \leq \sum_i (x_i = v) \leq n$$

Note that the term $(x_i = v)$ is a Boolean expression that returns 0 or 1, and hence, the *sum* counts the number of occurrences that x_i *is exactly* v. This number must be between m and n.
For example, let x be a variable vector of 10 elements (x_i with $i \in \{1,\ldots,10\}$), and let $v = 30$ and $m = 3$, $n = 5$. Then the constraint is fulfilled if $x = (1,2,5,4,30,7,30,6,30,8)$ because the value 30 occurs 3 times.
In LPL, the constraint is not yet explicitly implemented. It can be formulated in the mathematical way as follows:

```
constraint A: m <= sum{i} (x[i] = v) <= n ;
```

When calling a MIP solver, then LPL automatically translates this into linear constraints. In the following real models this constraint is used: model Kalis and model car2. In both models, the special case where $m = n$ is used, which reduces the formulation into

```
constraint A: sum{i} (x[i] = v) = n ;
```

20.4 Sequence_total

The sequence_total constraint requires that in each sequence of length $s > 0$ of a variable vector x_i with $i \in \{1,\ldots,k\}$ between m and n ($m \leq n$) elements must be assigned to v.

$$Sequence\_total(x_{i|i\in\{1,\ldots,k\}}, v, m, n, s)$$

In an explicitly mathematical version this could be formulated as :

$$m \leq \sum_{h|i\leq h\leq i+s-1} (x_i = v) \leq n \quad \text{forall } i \in \{1,\ldots,k-s+1\}$$

Note that the term $(x_i = v)$ is a Boolean expression that returns 0 or 1, and hence, the *sum* counts the number of occurrences that x_i *is exactly* v. This number must be between m and n.
For example, let x be a variable vector of 10 elements (x_i with $i \in \{1,\ldots,10\}$), and let $v = 30$ and $m = 1$, $n = 2$, $s = 4$. Then the constraint is fulfilled if $x = (1,30,5,4,30,7,30,6,30,8)$ because the value 30 occurs in each sub-sequence of 4 elements 1 or 2 times.

In LPL, the constraint is not yet explicitly implemented. It can be formulated in the mathematical way as follows:

```
constraint A{i|#i-s+1}: m <= sum{h|i<=h<=i+s-1} (x[i] = v) <= n ;
```

When calling a MIP solver, then LPL automatically translates this into linear constraints. The constraint is used in the following real model of "car assembly line sequence", see car2.

21 Conclusion

This paper gave a limited overview of some model classes. It shows a unified implementation into the LPL modeling language. LPL can analyze the model and selects automatically the right solver, based on a classification, to solve it. The actual LPL language is far from complete, it is thought as a research vision to formulate all kind of models in a unified modeling language. That language should also contain most concepts of a modern programming language – which LPL doesn't right now.

As said, LPL is a small subset of that unified modeling language, but a special model class, namely large linear MIP models, are efficiently formulated and are used in a commercial context.

References

[1] Fox W.P. Albright B. *Mathematical Modeling with Excel*. CRC Press, NewYork, 2020, second edition.

[2] Shetty C.M. Bazaraa M.S., Sherali H.D. *Nonlinear Programming, Theory and Algorithms*. Wiley, 2006, third edition.

[3] Pedregal P. García R. Alguacil N. Castillo E., Conejo A.J. *Building and Solving Mathematical Programming Models in Engineering and Science*. Wiley, 2002.

[4] More J.J. Dolan E.D. Benchmarking Optimization Software with COPS., Tech. Report. Mathematics and Computer Science Division,, 2000.

[5] Cesari L. *Optimization – Theory and Applications*. Springer Verlag, 1983.

[6] Wolsey L.A. *Integer Programming*. Wiley, 1998.

[7] MatMod. Homepage for Learning Mathematical Modeling : https://matmod.ch.

[8] Hürlimann T. Logical modeling. https://matmod.ch/lpl/doc/logical.pdf.

[9] Hürlimann T. Permutation Problems. https://matmod.ch/lpl/doc/permutation.pdf.

[10] Hürlimann T. Reference Manual for the LPL Modeling Language, most recent version. https://matmod.ch/lpl/doc/manual.pdf.

[11] Hong Y. Williams H.P. Representations of the all-different Predicate of Constraint Satisfaction in Integer Programming. *INFORMS Journal on Computing*, 13:96–103, 2001.

[12] Winston W.L. *Operations Research, Applications and Algorithms*. Duxbury, 3rd ed., 1998.