Integer Linear Programming

CONTACT@ADAMFURMANEK.PL

HTTP://BLOG.ADAMFURMANEK.PL

FURMANEKADAM

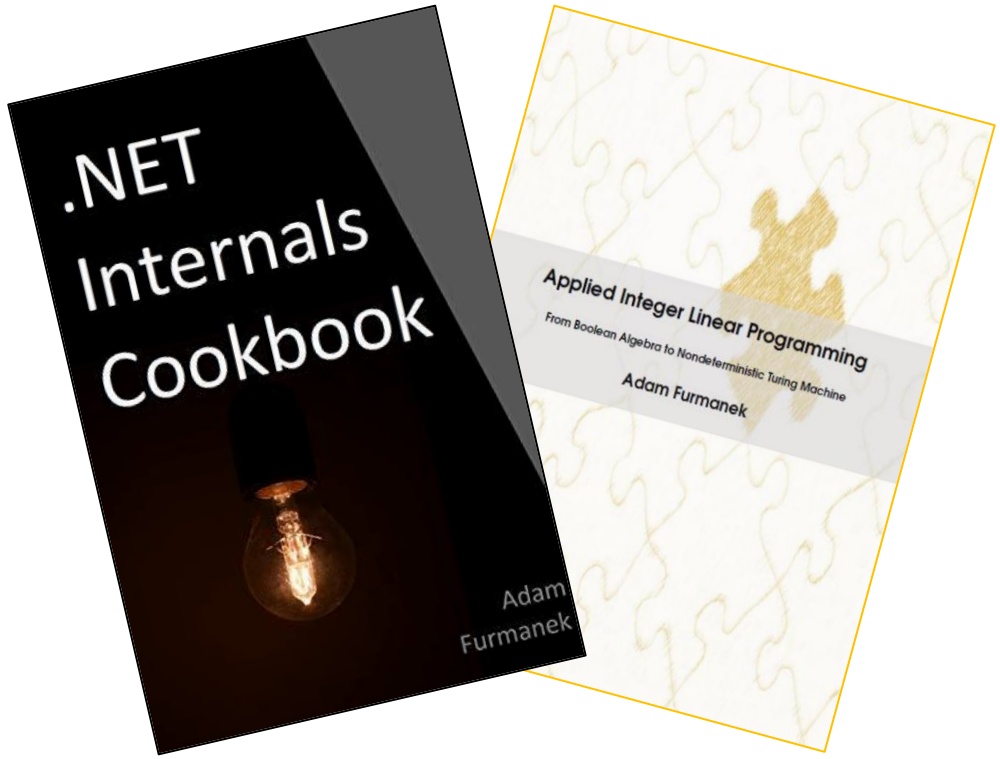
About me

Software Engineer, Blogger, Book Writer, Public Speaker.
Author of ***Applied Integer Linear Programming*** and ***.NET Internals Cookbook***.

http://blog.adamfurmanek.pl

contact@adamfurmanek.pl

furmanekadam



Random IT Utensils

IT, operating systems, maths, and more.

Agenda

Declarative programming at a glance.

Some theory.

Real life example.

Implementation consideration.

Summary.

Declarative programming

Declarative programming

Paradigm that expresses the logic of a computation without describing its control flow.

Declarative programming often considers programs as theories of a formal logic, and computations as deductions in that logic space.

A high-level program that describes what a computation should perform.

SELECT * FROM Orders

Declarative programming — why?

Decouples problem formulation from solving process.

Solving part can be replaced without modifying the formulation.

Easier to optimise algorithms for because „the goal” is understandable for the machine.

Can be used with little to no programming skills.

Sometimes we don’t know how to solve it.

Example — RTS game

We are playing RTS game.

We are allowed to hire **footmen** and **archers**.

Every footman costs 10 gold and 30 food. Every archer costs 20 gold, 25 food, and 10 wood.

Footman's attack is equal to 5, archer's attack is equal to 7.

Our population limit is set to 200 units. Every footman „costs" 1 unit, every archer „costs" 2 units.

We have 1000 gold, 1000 food, and 200 wood. We want to get strongest possible army.

Units

	Gold	Food	Wood
Footman	10	30	0
Archer	20	25	10

	Attack	Unit count
Footman	5	1
Archer	7	2

	Available gold	Available food	Available wood	Max units count
Constraint	1000	1000	200	200

Variables

Variables:

$$f - footmen$$
$$a - archers$$

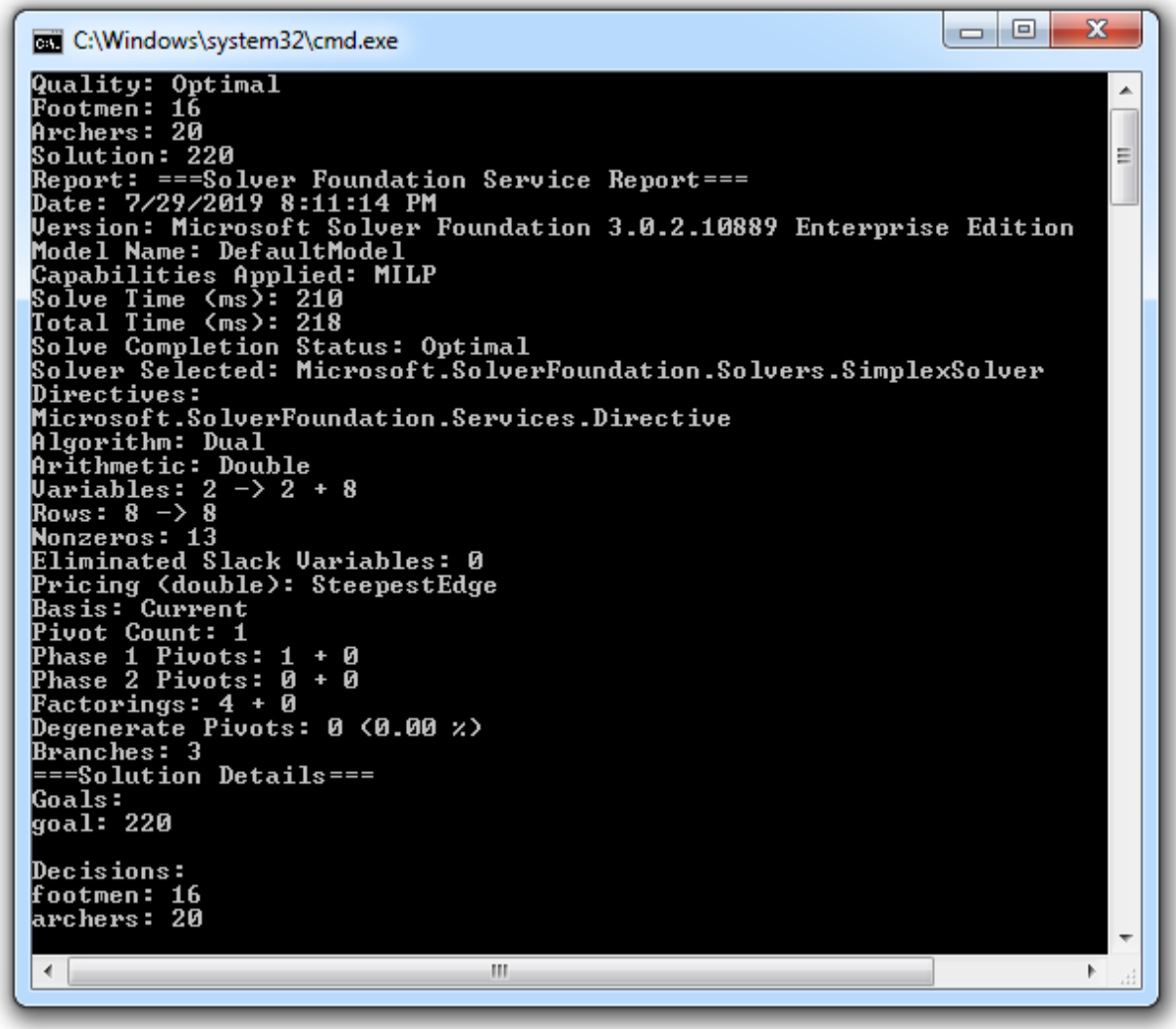
Constraints:

$$f + 2 \cdot a \leq 200 \quad \text{// population}$$
$$10 \cdot f + 20 \cdot a \leq 1000 \quad \text{// gold}$$
$$30 \cdot f + 25 \cdot a \leq 1000 \quad \text{// food}$$
$$10 \cdot a \leq 200 \quad \text{// wood}$$

Target value:

$$5 \cdot f + 7 \cdot a$$

```
static void Main()
{
    var context = SolverContext.GetContext();
    var model = context.CreateModel();

    var footmen = new Decision(Domain.Integer, "footmen");
    var archers = new Decision(Domain.Integer, "archers");
    model.AddDecisions(footmen, archers);

    model.AddConstraint("footmen_lower_bound", 0 <= footmen);
    model.AddConstraint("archers_lower_bound", 0 <= archers);

    model.AddConstraints("population", 0 <= footmen + 2 * archers, footmen + 2 * archers <= 200);

    model.AddConstraint("gold", 10 * footmen + 20 * archers <= 1000);
    model.AddConstraint("food", 30 * footmen + 25 * archers <= 1000);
    model.AddConstraint("wood", 10 * archers <= 200);

    model.AddGoal("goal", GoalKind.Maximize, 5 * footmen + 7 * archers);

    using (var writer = File.CreateText("problem.mps"))
    {
        context.SaveModel(FileFormat.FreeMPS, writer);
    }

    var solution = context.Solve();

    Console.WriteLine($"Quality: {solution.Quality}");
    Console.WriteLine($"Footmen: {footmen}");
    Console.WriteLine($"Archers: {archers}");
    Console.WriteLine($"Solution: {solution.Goals.First()}");

    Console.WriteLine($"Report: {solution.GetReport()}");
}
```



Example

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

Example

$$\begin{array}{rcccc} & 9 & 5 & 6 & 7 \\ + & 1 & 0 & 8 & 5 \\ \hline = & 1 & 0 & 6 & 5 & 2 \end{array}$$

Example

$$\begin{array}{cccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

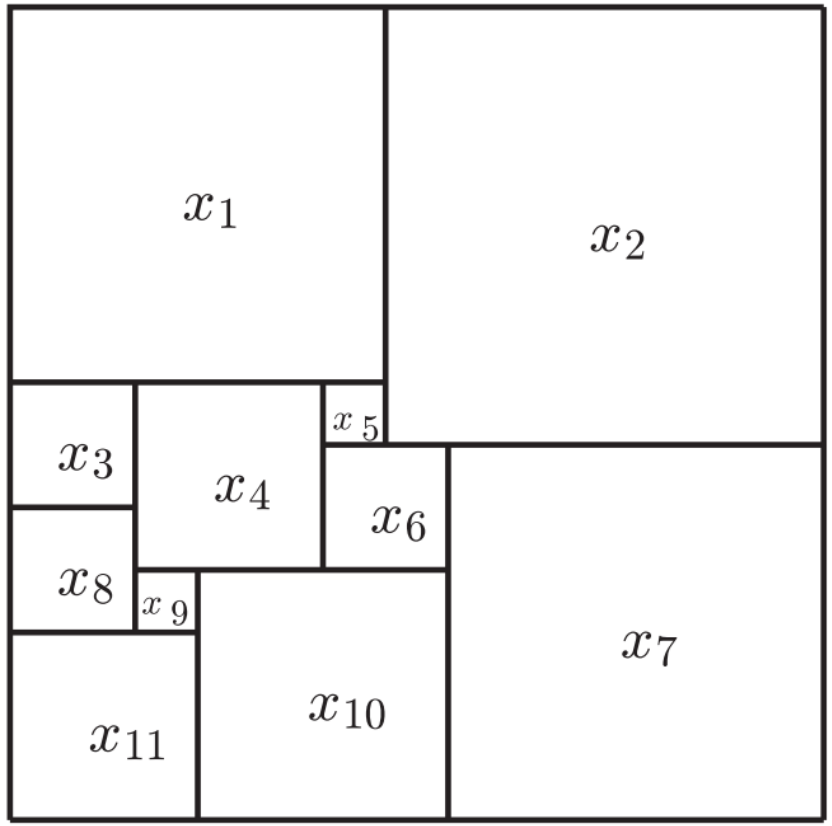
Variables:

$$S, E, N, D, M, O, R, Y \in \{0, \dots, 9\} \textit{ and all different}$$

Constraints:

$$1000S + 100E + 10N + D +$$
$$1000M + 100O + 10R + E =$$
$$10000M + 1000O + 100N + 10E + Y$$

Example



$$x_1 + x_2 = 13$$

$$x_3 + x_4 + x_5 + x_2 = 13$$

$$x_3 + x_4 + x_6 + x_7 = 13$$

$$x_8 + x_4 + x_6 + x_7 = 13$$

$$x_8 + x_9 + x_{10} + x_7 = 13$$

$$x_{11} + x_{10} + x_7 = 13$$

$$x_1 + x_3 + x_8 + x_{11} = 13$$

$$x_1 + x_4 + x_9 + x_{11} = 13$$

$$x_1 + x_4 + x_{10} = 13$$

$$x_1 + x_5 + x_6 + x_{10} = 13$$

$$x_2 + x_6 + x_{10} = 13$$

$$x_2 + x_7 = 13$$

Real life examples

RTS game = factory production allocation:

- We have available resources
- We have facitilies
- We want to plan work to maximize the production

Send more money = pattern recognition:

- We know structure of a problem but doesn’t understand the latent factors

Floor tiling = microchip transistor layout:

- We have transistors and gates of given size
- We want to minimize heat emission and the board size

Others: BTS placement, power plant rooms layout, scheduling systems, items personalization and many more.

Some theory

MIXED INTEGER LINEAR PROGRAMMING


I HAVE NO IDEA
WHAT YOU'RE TALKING ABOUT
quickmeme.com

Mixed Integer Linear Programming

Programming in mathematics means finding a solution to optimization problem.

Linear Programming is a class of a problems with only linear constraints and with linear cost function.

Mixed Integer Linear Programming is a class of a problems with integer constraint for some of variables.

Constraints

We can add two variables:

$$a + b$$

We can multiply variable by constant:

$$5 \cdot a$$

We can constrain variable with lower/upper bound:

$$a \leq 7$$

We cannot multiply variables:

$$a \cdot b$$

We cannot use strict constraings (greater than, less than, not equal)

$$a < 10$$

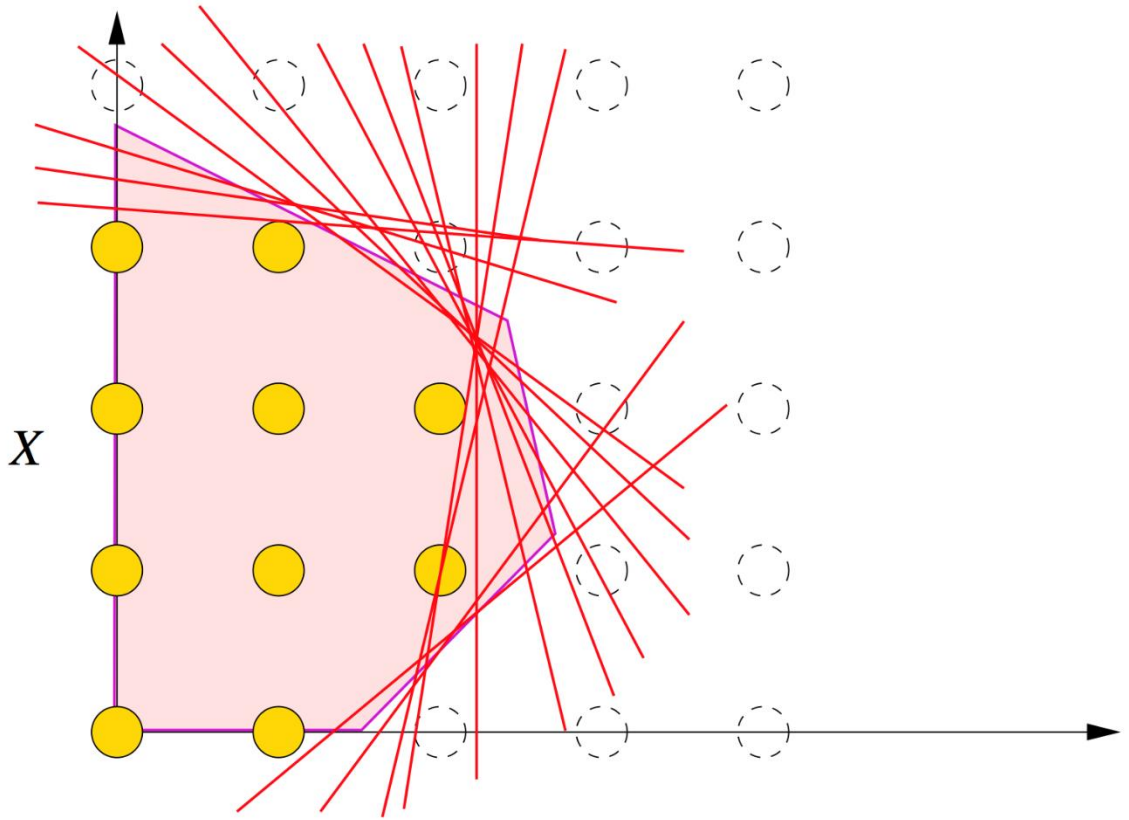
Cost function

Notice that not hiring anyone **was also a solution**!

Problem may have zero, one, multiple or infinitely many solutions. We need to compare them.

We use cost function to specify which one of them is the best.

Cutting plane method



Algorithms

Basic algorithms:

- Cutting plane methods – we solve the problem without integer constraints (continuous version of the problem), and next we cut the plane to get smaller problem
- Branch and bound – we solve the problem without integer constraints, and next we bound some variables and perform next iteration

MILP is NP-complete and we sometimes use heuristics:

- Tabu search
- Simulated annealing
- Ant colony optimization

What can we do?

WE WILL IMPLEMENT

Boole’s logic.

Multiplication.

Comparison operators.

WE WILL NOT IMPLEMENT

Arithmetic: division, remainder, exponentation, roots.

Comparisons: min, max, absolute value.

Number theory: factorial, GCD.

Algorithms: if condition, sorting, loops, lexicographical comparisons, Gray’s code, linear regression.

Set operators: SOS type 1 and 2, approximation.

Graph operators: MST, vertex/edge cover, max flow, connectivity, shortest path, TSP.

Turing machine.

Conjunction — AND operator (&&)

We have two variables: a, b.

We want to create variable x which is $x = a \;\&\&\; b$.

Formula:

$$0 \leq a + b - 2x \leq 1$$

Conjunction — AND operator (&&)

$$0 \leq a + b - 2x \leq 1$$

If both a and b are 1 then x must be 1.

If x was 0:

$$0 \leq 1 + 1 - 2 \cdot 0 \leq 1$$

$$0 \leq 2 - 0 \leq 1$$

$$0 \leq 2 \leq 1$$

Conjunction — AND operator (&&)

$$0 \leq a + b - 2x \leq 1$$

If both a and b are 1 then x must be 1.

If x is 1:

$$0 \leq 1 + 1 - 2 \cdot 1 \leq 1$$

$$0 \leq 2 - 2 \leq 1$$

$$0 \leq 0 \leq 1$$

Conjunction — AND operator (&&)

$$0 \leq a + b - 2x \leq 1$$

If either a or b is 1 then x must be 0.

If x was 1:

$$0 \leq 1 + 0 - 2 \cdot 1 \leq 1$$

$$0 \leq 1 - 2 \leq 1$$

$$0 \leq -1 \leq 1$$

Conjunction — AND operator (&&)

$$0 \leq a + b - 2x \leq 1$$

If either a or b is 1 then x must be 0.

If x is 0:

$$0 \leq 1 + 0 - 2 \cdot 0 \leq 1$$

$$0 \leq 1 - 0 \leq 1$$

$$0 \leq 1 \leq 1$$

Conjunction — AND operator (&&)

$$0 \leq a + b - 2x \leq 1$$

If both a or b are 0 then x must be 0.

If x was 1:

$$0 \leq 0 + 0 - 2 \cdot 1 \leq 1$$

$$0 \leq 0 - 2 \leq 1$$

$$0 \leq -2 \leq 1$$

Conjunction — AND operator (&&)

$$0 \leq a + b - 2x \leq 1$$

If both a or b are 0 then x must be 0.

If x is 0:

$$0 \leq 0 + 0 - 2 \cdot 0 \leq 1$$

$$0 \leq 0 - 0 \leq 1$$

$$0 \leq 0 \leq 1$$

Conjunction — AND operator (&&)

$$0 \leq a + b - 2x \leq 1$$

If both a and b are 1 then x must be 1.

If either a or b is 1 then x must be 0.

If both a or b are 0 then x must be 0.

Conjunction & Disjunction

Two variables conjunction:

$$0 \leq a + b - 2x \leq 1$$

n variables conjunction:

$$0 \leq a_1 + a_2 + a_3 + \ldots + a_n - nx \leq n - 1$$

Two variables disjunction:

$$-1 \leq a + b - 2x \leq 0$$

n variables disjunction goes the same way

Negation, exclusive or, implication

Negation:

$$x = 1 - a$$

Implication:

$$a \Rightarrow b \equiv \sim a \vee b$$

Exclusive or:

$$(\sim a \wedge b) \vee (a \wedge \sim b)$$

Multiplication of two binary variables is their **conjunction**.

$$a \cdot b \equiv a \wedge b$$

Multiplication of integer variables

Multiplication is possible when we know the maximum possible value of a variable.

We set the upper bound and perform the long multiplication.

It is rather slow approach, it requires $O(n^2)$ temporary variables.

Value decomposition

We decompose the variable to extract digits.

We assume the maximum possible value, because we need to know the number of digits.

Decomposition is straightforward:

$$b = b_0 + 2 \cdot b_1 + 4 \cdot b_2 + 8 \cdot b_3 + \ \ldots + 2^{n-1} \cdot b_{n-1}$$

$$
\begin{array}{cccccccc}
 & & & & 0 & 0 & 1 & 1 \\
\cdot & & & & 0 & 1 & 0 & 0 \\
\hline
 & & & 0\cdot 0 & 0\cdot 0 & 1\cdot 0 & 1\cdot 0 & \\
 & & 0\cdot 0 & 0\cdot 0 & 1\cdot 0 & 1\cdot 0 & & \\
 & 0\cdot 1 & 0\cdot 1 & 1\cdot 1 & 1\cdot 1 & & & \\
+ & 0\cdot 0 & 0\cdot 0 & 1\cdot 0 & 1\cdot 0 & & & \\
\hline
 & 0 & 0 & 0 & 1 & 1 & 0 & 0
\end{array}
$$

$$
\begin{array}{cccccccc}
 & & & & a_3 & a_2 & a_1 & a_0 \\
\cdot & & & & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & & & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
 & & & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & \\
 & & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & \\
+ & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & \\
\hline
\end{array}
$$

Multiplication

$$a \cdot b =$$
$$\left((a_0 \wedge b_0) + 2 \cdot (a_0 \wedge b_1) + 4 \cdot (a_0 \wedge b_2) + \ \ldots + 2^{n-1} \cdot (a_0 \wedge b_{n-1})\right)$$
$$+2 \cdot \left((a_1 \wedge b_0) + 2 \cdot (a_1 \wedge b_1) + 4 \cdot (a_1 \wedge b_2) + \cdots 2^{n-1} \cdot (a_1 \wedge b_{n-1})\right)$$
$$+4 \cdot \left((a_2 \wedge b_0) + 2 \cdot (a_2 \wedge b_1) + 4 \cdot (a_2 \wedge b_2) + \ \ldots + 2^{n-1} \cdot (a_2 \wedge b_{n-1})\right)$$
$$+ \cdots$$
$$+ 2^{n-1}((a_{n-1} \wedge b_0) + 2 \cdot (a_{n-1} \wedge b_1) + 4 \cdot (a_{n-1} \wedge b_2) + \ \ldots + 2^{n-1} \cdot (a_{n-1} \wedge b_{n-1}))$$

It can be represented as:

$$a \cdot b = \sum_{i=0}^{n-1} 2^i \left(\sum_{j=0}^{n-1} 2^j a_i \wedge b_j \right)$$

Comparison operators

To calculate whether $a > b$ we can use:

$$0 \leq b - a + 2^{n-1}x \leq 2^{n-1} - 1$$

To check whether numbers differ:

$$x = (a > b) \vee (b > a)$$

There are libraries doing that!

```
IVariable sumOfXAndY = x.Operation(OperationType.Addition, y);
IVariable anotherSumOfXAndY = solver.Operation(OperationType.Addition, x, y);
IVariable negationOfX = x.Operation(OperationType.Negation);
IVariable multiplicationOfXAndY = y.Operation(OperationType.Multiplication, x);
IVariable isYGreaterThanX = y.Operation(OperationType.IsGreaterThan, x);
```

https://github.com/afish/MilpManager

Scheduling System

REAL LIFE EXAMPLE

Idea

At the beginning of every term students must be assigned to classes.

It is hard to make them happy because some of them work, some of them prefer to sleep long, some of them prefer to have lots of days off.

We can try to represent the requirements as a MILP program and find the optimal solution.

Usage

We ask students to assign preferences points to every possible class.

The more points assigned the more student wants to be assigned to that class.

We need to take care of rooms occupancy limits, collisions etc.

Schedule

We are given a schedule of classes in courses

Every class has associated day (Monday – Friday), hour (e.g., 9:30 AM) and duration (e.g., 1:30 hrs).

Every class has associated room with occupancy limit.

Constraints

Variables.

Exactly one class in one course.

Collisions.

Rooms occupancy limits.

Cost function.

Variables

For every person, every course, every class we declare binary variable.

Value 1 means that the student is assigned to this class.

We define:

$$x_{course,class,student}$$

Exactly one class for a student

Every student must attend exactly one class in course.

For every course we need to assign student to exactly one class.

$$\bigwedge_{course,student} \sum_{class} x_{course,class,student} = 1$$

Collisions

Student cannot be in two places at the same time.

$$\bigwedge (course_1, class_1), (course_2, class_2)\ colliding$$

$$\bigwedge_{student} x_{course_1, class_1, student} + x_{course_2, class_2, student} \leq 1$$

Room occupancy limit

Every room has an occupancy limit.

$$\bigwedge_{course,class} \sum_{student} x_{course,class,student} \leq s_{course,class}$$

where $s_{course,class}$ means the size of the room

Preferences cost function

Every student assigned points to classes. We want to maximize the sum of those points.

$$\sum_{course,class,student} p_{course,class,student} \cdot x_{course,class,student}$$

where $p_{course,class,student}$ means number of points

What now?

Students: 150

Courses: 15

Classes: 9 per course

Problem size: ~40 000 variables.

Solving time: 2 seconds.

This solution is optimal, we won't get any better than that!

Implementation consideration

GUROBI OPTIMIZATION

IBM CPLEX

Google Optimization Tools

Microsoft® Solver Foundation

neos SERVER

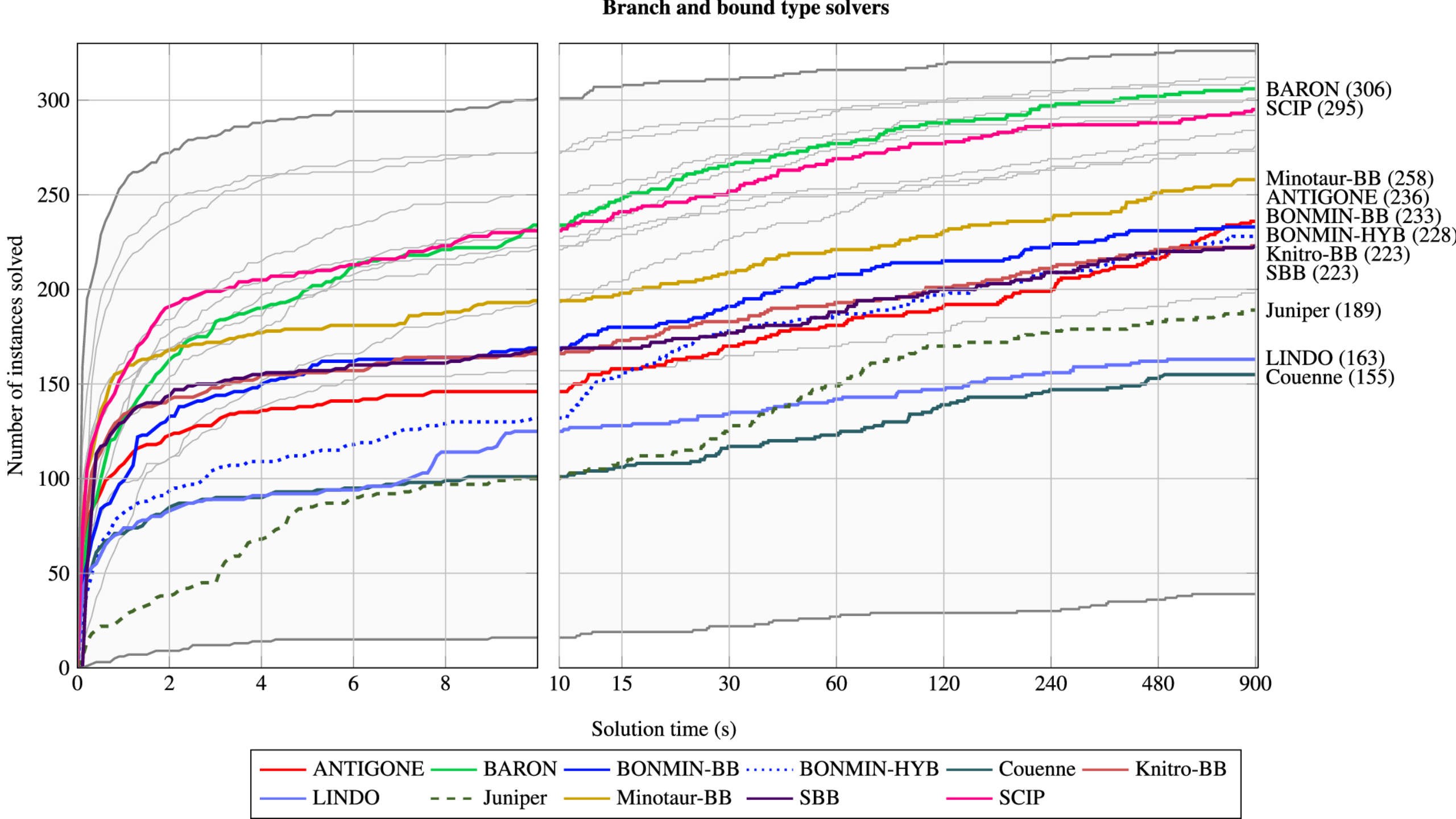
Licenses

Various approaches:

- License per workstation
- License per person
- Licensing server with tickets

Licenses for universities and research work.

Licenses for students.


Branch and bound type solvers
Number of instances solved
0
50
100
150
200
250
300
0
2
4
6
8
10
15
30
60
120
240
480
900
Solution time (s)
BARON (306)
SCIP (295)
Minotaur-BB (258)
ANTIGONE (236)
BONMIN-BB (233)
BONMIN-HYB (228)
Knitro-BB (223)
SBB (223)
Juniper (189)
LINDO (163)
Couenne (155)
ANTIGONE
BARON
BONMIN-BB
BONMIN-HYB
Couenne
Knitro-BB
LINDO
Juniper
Minotaur-BB
SBB
SCIP

Mixed Integer Linear Programming

- Cbc [AMPL Input][GAMS Input][MPS Input]
- CPLEX [AMPL Input][GAMS Input][LP Input][MPS Input][NL Input]
- feaspump [AMPL Input][CPLEX Input][MPS Input]
- FICO-Xpress [AMPL Input][GAMS Input][MOSEL Input][MPS Input][NL Input]
- Gurobi [AMPL Input][GAMS Input][LP Input][MPS Input][NL Input]
- MINTO [AMPL Input]
- MOSEK [AMPL Input][GAMS Input][LP Input][MPS Input][NL Input]
- proxy [CPLEX Input][MPS Input]
- qsopt_ex [AMPL Input][LP Input][MPS Input]
- scip [AMPL Input][CPLEX Input][GAMS Input][MPS Input][OSIL Input][Python Input][ZIMPL Input]
- SYMPHONY [MPS Input]

https://neos-server.org/neos/solvers/index.html

Beyond MILP

SAT/SMT.

Quadratic programming.

Prolog.

Specialized models for various problems.

Favorable Adjustment of Periods for Reduced Hyperperiods in Real-Time Systems

Our approach is not limited to strictly-periodical systems. Whether each task has exactly one period associated to it, or whether it is triggered by a more complex activation pattern composed of multiple basic periods is not critical for applying the presented approach. However, associating one dedicated period T_i to each task τ_i eases both notations, applicability to common task set generators like UUnifast and comparison to previous work.

Capital letters like, e.g., T_i are used to express *constants* in the upcoming ILP model. ILP *variables* are denoted by lower-case letters in any formulas, e.g., $\tilde{t}_i$.

Subsequently, we set up a constraint for each task τ_i to find a common multiple of all periods $\tilde{t}_i$.

$$\tilde{t}_i \cdot f_i = g,\ 0 \leq i < N \tag{7}$$

$$f_i > 0,\ 0 \leq i < N \tag{8}$$

The ILP variable f_i represents any integer factor which is multiplied with the period $\tilde{t}_i$ to achieve a common multiple g of all periods. The variable g is bound to integer values as well.

Obviously, Eq. (7) is not linear, as two ILP variables are multiplied. Yet, we show in the following how the multiplication of two ILP variables can be described efficiently using only linear terms.

Unsigned Multiplication inside an ILP: The following principles were introduced first by Furmanek [1]. The key idea is to execute the multiplication using only binary values. We will start with the multiplication of two unsigned ILP variables a and b.

$$y = a \cdot b \tag{9}$$

Subsequently, the variable b is decomposed to the base 2 as shown

$$\min : g \tag{14}$$

We choose to describe the optimization problem using ILP instead of quadratic constrained programming (QCP) to avoid convexity-requirement issues of solvers. It turned out that both state-of-the-art optimizers CPLEX and Gurobi are not capable of solving the presented set of constraints (without re-formulating Eq. (7)), as they do not fulfill their QCP solver's requirements. Yet, both are capable of solving the ILP problem using the presented re-formulations.

Summary

Declarative programming allows you to focus on a problem, not on an algorithm!

If something is slow — just replace the solver.

There are many models, choose as powerful as you can (to make modelling easy) and as primitive as possible (to make solving fast).

Q&A

References

Alexander Schrijver — „Theory of Linear and Integer Programming”

Dennis Yurichev — „SAT/SMT by example”

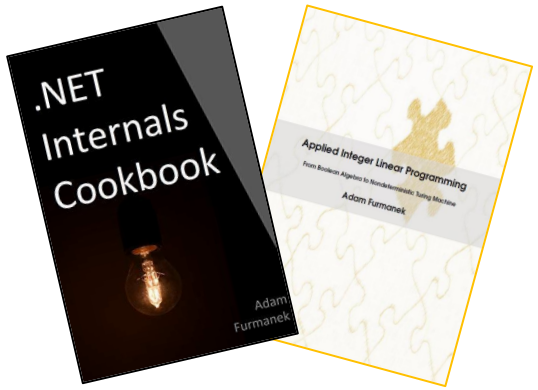
Adam Furmanek – „.NET Internals Cookbook”

http://blog.adamfurmanek.pl/2015/08/22/ilp-part-1/ — a lot about ILP

https://github.com/afish/MilpManager — library for modelling

https://neos-server.org/neos/solvers/index.html — NEOS cloud for solving problems

https://tore.tuhh.de/bitstream/11420/2548/1/201905-scopes-oehlert.pdf — Practical usage of ILP versus QP



Random IT Utensils

IT, operating systems, maths, and more.

Thanks!

CONTACT@ADAMFURMANEK.PL

HTTP://BLOG.ADAMFURMANEK.PL

FURMANEKADAM