NTNU – Trondheim
Norwegian University of
Science and Technology

Sherlock: a Primitive Solver for Numerical Constraint Satisfaction Problems

Antoine Rauzy
Department of Mechanical and Industrial Engineering
S. P. Andersens veg 3, Valgrinda*3.306
Antoine.Rauzy@ntnu.no

Licenses & versions

Sherlock is free software distributed by the AltaRica Association under GNU GPLv3 license.

Version	1.0.0
Date	07/01/2019

Agenda

- Introduction
- Constraint Satisfaction Problems
- Combinatorial Optimization
- References

Appendix

- Grammar
- Known bugs

INTRODUCTION

Rational

Sherlock is a primitive solver for **numerical constraint satisfaction problems**, i.e. problems in which **variables** take their values into **sets or ranges of integers**.

This presentation specifies the input language of Sherlock and presents the algorithms implemented by the tool.

Sherlock is developed in Python, for pedagogical purposes only. Its efficiency is by orders of magnitude worse than those of available commercial tools.

The objective is to familiarize students with constraint satisfaction problems as a **modeling language**.

Installing and Running Sherlock

To install Sherlock you just need to decompress the archive "Sherlock1.0.0.zip" into local directory. Source files are the Python file "Sherlock.py" as well as the directory "src" and its content.

To run Sherlock you have to open the file Python file "Sherlock.py" into your Python environment, set up the names of input file, problem, algorithm and output file and run it.

```
34
35 # 2) Main
36 # -------
37
38 inputFileName = "examples/ProductionPlanning.csp"
39 problemName = "ProductionPlanning"
40 # algorithmName = "GenerateAndTest"
41 # algorithmName = "FindOneSolution"
42 # algorithmName = "FindAllSolutions"
43 # algorithmName = "FindBestSolution"
44 algorithmName = "BranchAndBound"
45 outputFileName = "solution.txt"
46
47 engine = SherlockEngine()
48 engine.LoadAndSolveProblem(inputFileName, problemName, algorithmName, outputFileName)
49
50 sys.stdout.write("Bye\n")
51 sys.stdout.flush()
```

Organization of this Document

The remainder of this document is organized as follows.

- Section Constraint Satisfaction Problems describes Sherlock models for this class of problems as well as algorithms implemented in the tool to solve them. It provides also a list of exercises (solutions can be found in the folder examples).
- Section Combinatorial Optimization describes Sherlock models for this class of problems as well as algorithms implemented in the tool to solve them. It provides also a list of exercises (solutions can be found in the folder examples).
- Appendix Grammar gives the Backus-Naur Form of Sherlock's Grammar.
- Appendix Known bugs reports know problems with the current version of Sherlock

CONSTRAINT SATISFACTION PROBLEMS

Formal Definition

A **constraint satisfaction problem** is a pair (V, C) where:

- V is a finite set of **variables**. Each variable $v \in V$ takes its value in (small) finite set of constants (Boolean, symbols, integers) called the **domain** of v and denoted by dom(v).
- C is a finite set of constraints on variables of V. Each constraint $c \in C$ applies to a subset $\{v_1, v_2, \ldots, v_k\}$ of V. It describes which k-tuples of values are admissible: $c \subseteq dom(v_1) \times dom(v_2) \times \ldots \times dom(v_k)$.

The k-tuples of a constraint can be given **explicitly** (by enumerating them) or **implicitly** (e.g. by means of an equation, as in the case study).

A **variable assignment** in a function from variables of V to their domains. An assignment is a **solution** of (V, C) if it satisfies all of the constraints of C.

Sherlock Models: Domain Declarations

A Sherlock model for a constraint satisfaction problem consists in a number of domain declaration and problem declarations.

A domain declaration associates a name to a domain.

Domains of variables can be either **sets of integers**, e.g. {2, 4, 6}, or **ranges of integers**, e.g. [1, 4]. Ranges of integers are internally expanded into sets, e.g. the range [1, 4] is expanded into the set {1, 2, 3, 4}. E.g.

```
domain Color [1, 3]
domain Color {1, 2, 3}
```

Domains can be given a name and subsequently reused to declare variable. A named domain must be declared before to be used in a variable declaration.

Sherlock Models: Problem Declarations

A problem declaration associates a name to a problem and describes this problem.
It is made of two successive parts:

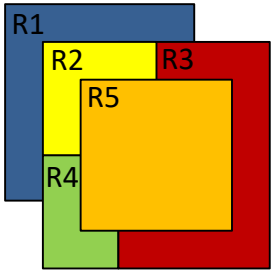
1. The declaration **variables** themselves together with their domains.
2. The description of **constraints** applying on these variables

```
problem Map5
  Color R1, R2, R3, R4, R5;
  constraint
    alldifferent R1, R2;
    alldifferent R1, R3;
    alldifferent R1, R4;
    alldifferent R2, R3;
    alldifferent R2, R4;
    alldifferent R2, R5;
    alldifferent R3, R4;
    alldifferent R3, R5;
    alldifferent R4, R5;
end
```

- The declaration of a problem starts with the keyword "problem" followed by the name of the model.
- Then come variable declarations.
- A variable declaration consists of the domain of the variable followed by the names of the variables separated with commas "," and ending with a semicolon ";".
- Constraints come after the keyword "constraint".
- Here all constraints are of type "alldifferent".
- The declaration finishes with the keyword "end".

Example: the Map5 Problem

Consider the 5 regions map on the right.
Is there a way to color each of the region of this map with k colors so that two adjacent regions are not colored the same?
If k=5, the response is obviously positive. But with k=4 or k=3?




```
domain Color [1, 3]

problem Map5
  Color R1, R2, R3, R4, R5;
  constraint
    alldifferent R1, R2;
    alldifferent R1, R3;
    alldifferent R1, R4;
    alldifferent R2, R3;
    alldifferent R2, R4;
    alldifferent R2, R5;
    alldifferent R3, R4;
    alldifferent R3, R5;
    alldifferent R4, R5;
end
```

Sherlock Models: Available Constraints

The current version of Sherlock provides 3 types of constraints:

- Alldifferent constraints: the given variables must take different values.

```
alldifferent V1, V2, V3, V4;
```

- Clauses: at least one of literals must be verified:

```
or V1!=1, V2!=1, V3==0;
```

  Literals in the form V ® c , where V is a variable, ® is ==, !=, <, >, <= or >=, and c is an integer

- Linear inequalities:

```
2*V1 - V2 - 3*V3 + V4 == 4;
```

Algorithm "Generate and Test"

There exists a quite simple algorithm to solve any constraint satisfaction problem: it consists in generating one by one all possible variable assignments and to test them until a solution is found.

```
GenerateAndTest(V, C):
    GenerateAndTest(∅, V, C)

GenerateAndTest(σ, V, C):
    if σ gives a value to all variables of V:
        if σ satisfies all constraints of C:
            exit with σ
    else:
        select a variable v in V not assigned in σ
        forall constant c in dom(v):
            GenerateAndTest(σ ∪ [v ← c], V, C)
```

Algorithm "FindOneSolution"

FindOneSolution(σ, V, C):

    if σ falsifies one of the constraints of C:
        return None
    else if σ satisfies all constraints of C:
        return σ
    else:
        select a variable v in V not assigned in σ and a value c in dom(v)
        σ' ← Propagate(σ|[dom(v) ← {c}], V, C)
        result ← FindOneSolution(σ', V, C)
        if result=None:
            σ'' ← Propagate(σ|[dom(v) ← dom(v)\{c}], V, C)
            result ← FindOneSolution(σ'', V, C)
        return result

Assigns the value c to v and propagate

Removes the value c from dom(v) and propagate

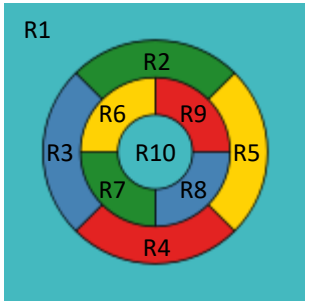
Algorithm "FindAllSolutions"

FindAllSolutions(σ, V, C):

    if σ falsifies one of the constraints of C:
        return
    else if σ satisfies all constraints of C:
        if O(σ) is better than the best solution found so far:
            best solution ← σ
    else:
        select a variable v in V not assigned in σ and a value c in dom(v)
        σ' ← Propagate(σ|[dom(v) ← {c}], V, C)
        FindAllSolutions(σ', V, C)
        σ'' ← Propagate(σ|[dom(v) ← dom(v)\{c}], V, C)
        FindAllSolutions(σ'', V, C)

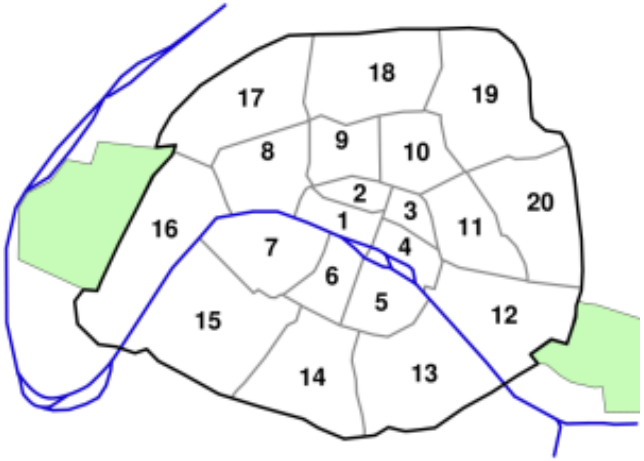
Exercises: 4 Colors Problems

Exercise: [Map10] The following map has colored in 5 colors.



1. Write a model to show that it can be colored in 4 colors.
2. Assess this model with the algorithms `GenerateAndTest`, `FindOneSolution` and `FindAllSolutions`. How running times compare?

Exercise: [Paris] Same question with the districts of Paris.



Exercises: Cryptarithmetic

Exercise: [Money] Write the model to find values for letters (in [0, 9]) so that the following addition works:

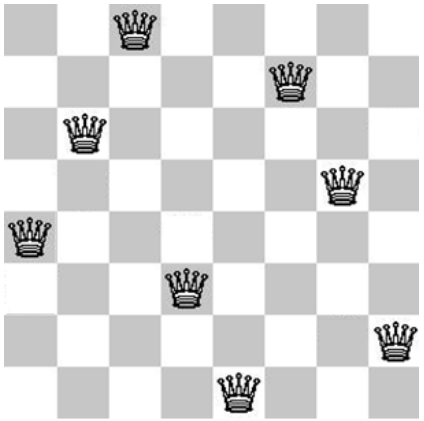
$$SEND + MORE = MONEY$$

Exercise: [Donald] Same question with the following addition.

$$DONALD + GERALD = ROBERT$$

Exercises: N-Queens Problems

Exercise: [8-Queens] Write the model to show that it is possible to place 8 queens on chessboard so that no two queens are "en prise".



Exercise: [k-Queens] Same questions with k queens on a k x k chess board, with k = 4, 5, 6, 7, 9…

Exercises: Puzzle

Exercise: [Zebra] The following version of the puzzle appeared in *Life International* in 1962 (source Wikipedia):

- There are five houses.
- The Englishman lives in the red house.
- The Spaniard owns the dog.
- Coffee is drunk in the green house.
- The Ukrainian drinks tea.
- The green house is immediately to the right of the ivory house.
- The Old Gold smoker owns snails.
- Kools are smoked in the yellow house.
- Milk is drunk in the middle house.
- The Norwegian lives in the first house.
- The man who smokes Chesterfields lives in the house next to the man with the fox.
- Kools are smoked in the house next to the house where the horse is kept.
- The Lucky Strike smoker drinks orange juice.
- The Japanese smokes Parliaments.
- The Norwegian lives next to the blue house.

Now, who drinks water? Who owns the zebra? Design a model to answer these questions.

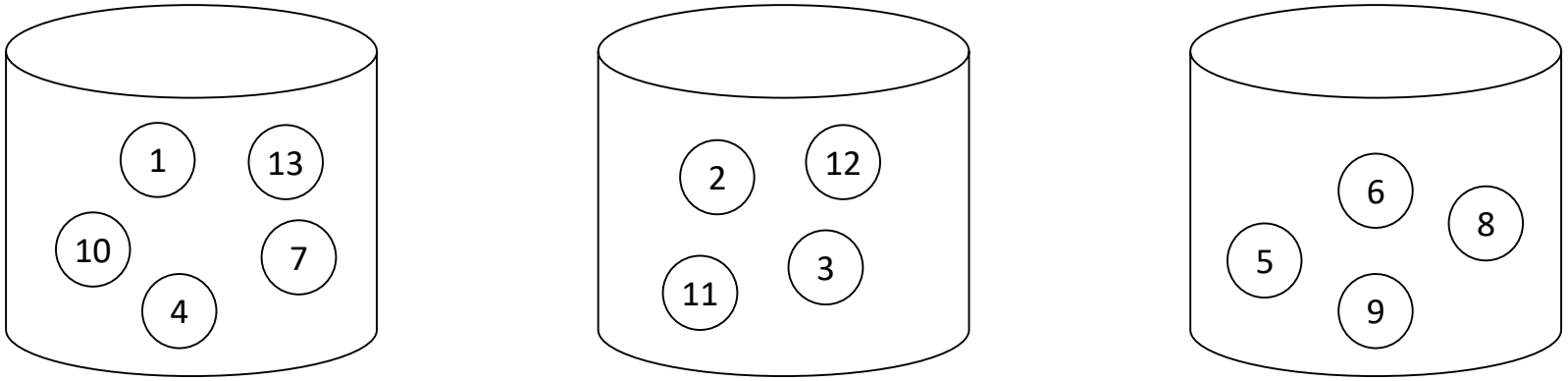
In the interest of clarity, it must be added that each of the five houses is painted a different color, and their inhabitants are of different national extractions, own different pets, drink different beverages and smoke different brands of American cigarettes. One other thing: in statement 6, *right* means *your* right.

Exercises: Mathematical Numbers

Exercise: [Schur's Lemma] The problem consists in determining whether it is possible to put n balls numbered 1, 2,…, n into 3 boxes in such a way that:

- If the ball x is in a box, then the ball 2x is not in the same box.
- If the balls x and y are in a box, then the ball x+y is not in the same box.

Hint: The problem has solutions for $n \leq 13$ and no solution beyond.



Exercises: Mathematical Numbers

Exercise: [Ramsey problem] Color the edges of a complete graph with n nodes using at most k colors, in such a way that there is no monochromatic triangle in the graph, i.e. in any triangle at most two edges have the same color. With 3 colors, the problem has a solution if $n < 17$.

Design a model to show this result (and that a complete graph with 16 nodes can be colored so to fulfil the constraint).

Exercises: Sudoku

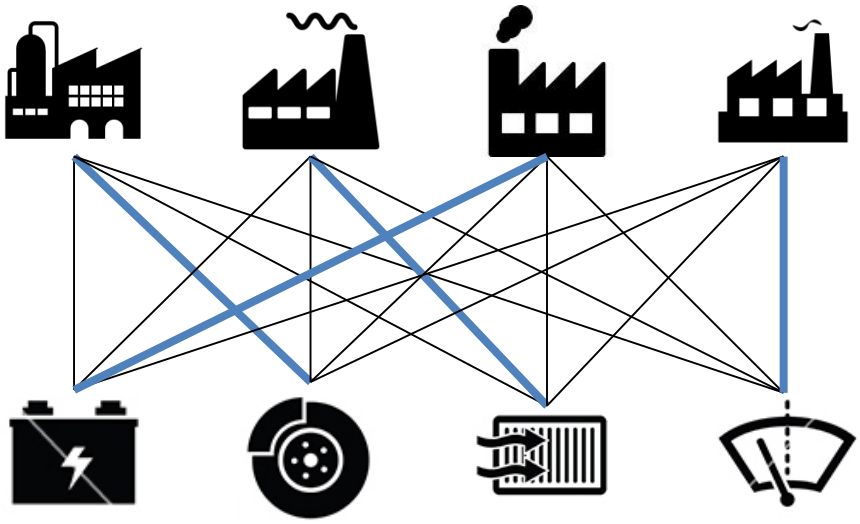
Exercise: [Sudoku] Design a model to solve the following Sudoku grid.

				2				1
						5	8	
			8	5		7	4	
		2	5		6			4
3			4		2			7
6			9		7	1		
	2	9		7	1			
	7	6						
8				4				

Exercise: Same exercise with your own Sudoku grid.

Exercises: a Matching Problem*

Exercise: [Matching] design a model so to assign four suppliers S1, S2, S3, S4 to four parts such that each supplier provides on one part and each part is produced by one supplier.
Effectiveness of production is given by the following table (e.g., supplier S1 produces part P1 with effectiveness 7). The total effectiveness must be 19 at least.

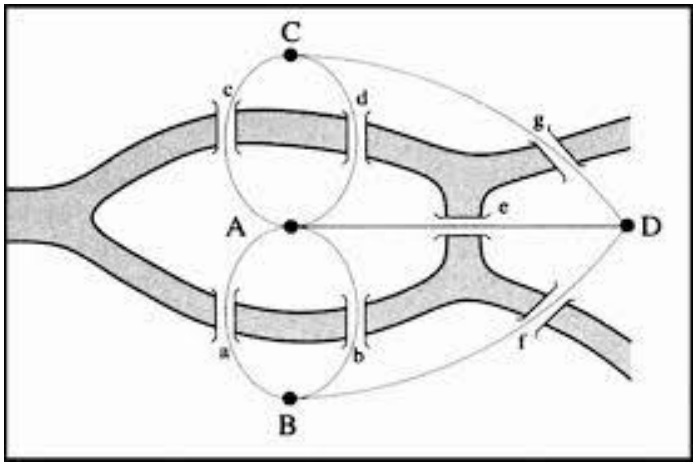


	P1	P2	P3	P4
S1	7	1	3	4
S2	8	2	5	1
S3	4	3	7	2
S4	3	1	6	3

(*) This exercise is borrowed from Hana Rudová "Constraint Programming and Scheduling -- Materials from the course taught at HTWG Constanz, Germany"

Exercises: Planning

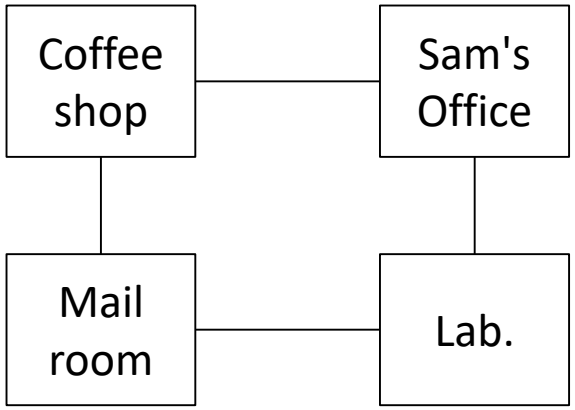
Exercise: [Koenigsberg] The great Swiss mathematician **Leonhard Euler** (1707-1783) has been asked to find a promenade (actually a parad) through the different districts of the city of Königsberg that would go through all of the bridges of the city, but that would not go twice by any bridge.



1. Design a model to show that there is no such promenade.
2. Design a model to find a parade that goes through each district, possibly not taking all the bridges.

Exercises: Planning

Exercise: [LilleRob] Rob is a small robot which hangs around in Sam's institute (see floor map below). Usually, Rob is gently sleeping in the laboratory. But it can also deliver the coffee and the mail to Sam. Not at the same time however: Rob is too small to carry both the coffee and the mail.



Rob can thus perform 6 actions:

- Move clockwise from one room to the next one.
- Move counter clockwise from one room to the next one.
- Pick-up the coffee at the coffee shop.
- Deliver the coffee at Sam's office.
- Pick-up the mail in the mail room.
- Deliver the mail at Sam's office.

Design a model to tell Rob the sequence of actions it must perform to deliver the coffee and the mail to Sam, starting from the lab. and going back to the lab. when its job is achieved.

Exercises: Project Management

Exercise: [Project12] A project is made of 12 tasks labelled from A to L, with the following duration (in days) and precedence constraints (a task X precedes another task Y, if X must be completed before Y starts).

Task	Duration	Precedence
A	3	
B	1	A
C	5	A
D	6	B
E	4	B
F	2	C, D, I

Task	Duration	Precedence
G	9	E, F
H	5	
I	8	H
J	2	H
K	3	I
L	7	J, K

Design a model to determine in how many days are required to complete the project.

Exercises: Rostering Problems

Exercise: [Rostering55] You need to organize the shifts of the 5 employees (Adna, Bekka, Carsten, Dina and Egil) of your service over the 5 working days of a week. There are 3 shifts every day: morning, afternoon and night. Constraints are as follows.

- 2 employees are required for morning and afternoon shifts, 1 for night shifts.
- Each employee must take at least one morning shift, one afternoon shift and one night shift.
- Employees cannot work less than 4 shifts and more than 6 shifts a week.
- An employee cannot work two consecutive shifts
- When he takes an afternoon shift, Egil is trained by either Bekka or Carsten.
- Adna and Dina should not work more than Bekka.

Design a model to find an acceptable rostering.

COMBINATORIAL OPTIMIZATION

Formal Definition

A **combinatorial optimization problem** is a triple (V, C, O) where:

- (V,C) is a constraint satisfaction problem, i.e.
  - V is a finite set of **variables**. Each variable $v \in V$ takes its value in (small) finite set of constants (Boolean, symbols, integers) called the **domain** of v and denoted by dom(v).
  - C is a finite set of constraints on variables of V. Each constraint $c \in C$ applies to a subset $\{v_1, v_2, \ldots, v_k\}$ of V. It describes which k-tuples of values are admissible: $c \subseteq dom(v_1) \times dom(v_2) \times \ldots \times dom(v_k)$.

- O is an **objective function**, i.e. either:
  - maximize f(V) for a certain function f of the variables
  - minimize f(V) for a certain function f of the variables

Objective Functions

An optimization problem is just a constraint satisfaction problem with an objective function. Objective functions are introduced by one of the keyword "minimize" or "maximize" (with their obvious meaning). They consists in linear polynomials of the variables of the problem.

```
domain Color [1, 4]

problem Map5
  Color R1, R2, R3, R4, R5;
  constraint
    alldifferent R1, R2;
    alldifferent R1, R3;
    alldifferent R1, R4;
    alldifferent R2, R3;
    alldifferent R2, R4;
    alldifferent R2, R5;
    alldifferent R3, R4;
    alldifferent R3, R5;
    alldifferent R4, R5;
    minimize 2*R2 + R4 - 3*R5;
end
```

Algorithm "FindBestSolution"

FindBestSolution(σ, V, C, O):

    if σ falsifies one of the constraints of C:

        return

    else if σ satisfies all constraints of C:

        if O(σ) is better than the best solution found so far:

            best solution $\leftarrow \sigma$

    else:

        select a variable v in V not assigned in σ and a value c in dom(v)

        $\sigma' \leftarrow$ Propagate(σ|[dom(v) $\leftarrow$ {c}], V, C)

        FindBestSolution(σ', V, C)

        $\sigma'' \leftarrow$ Propagate(σ|[dom(v) $\leftarrow$ dom(v)\{c}], V, C)

        FindBestSolution(σ'', V, C)

Algorithm "BranchAndBound"

BranchAndBound(σ, V, C, O):

    if σ falsifies one of the constraints of C:
        return
    else if σ satisfies all constraints of C:
        if O(σ) is better than the cost of the best solution found so far:
            best solution ← σ
    else if O(σ) can still be better than the cost of the best solution found so far:
        select a variable v in V not assigned in σ and a value c in dom(v)
        σ' ← Propagate(σ|[dom(v) ← {c}], V, C)
        BranchAndBound(σ', V, C)
        σ'' ← Propagate(σ|[dom(v) ← dom(v)\{c}], V, C)
        BranchAndBound(σ'', V, C)

Exercises: Production Planning*

Exercise: [ProductionPlanning] A firm produces products P1, P2 and P3

- To produce 1 unit of product P1, the firm uses 3 kg of raw material.
- To produce 1 unit of product P2, the firm uses 2 kg of raw material and 1 unit of product P1.
- To produce 1 unit of product P3, the firm uses 2 kg of raw material, 2 units of product P1 and 1 unit of product P2.
- There are 1000 kg of raw material available.

Products P1 and P2 that are used as semi-finished products can also be sold themselves.

- Prices of products P1, P2 and P3 are respectively 5, 10 and 30€.

The objective is to maximize total revenues from products sold.

Design a model to reach this objective.

(*) Borrowed to Jan Fàbri

Exercises: Cutting Stock Problem*

Exercise: [CuttingStock] Airm produces garden laths fence. There are only standard laths of 200 cm long at disposal at storehouse.

To produce a fence, the firm needs exactly 12 laths 80 cm long, 31 laths 50 cm long and 21 laths 30 cm long.

You have to design a cutting plan to minimize total amount of laths 200 cm long.

You can assume that the cutting width is null.

(*) Borrowed to Jan Fàbri

Exercises: Knapsack Problem*

Exercise: [Knapsack] There are 5 projects characterized by the investment cost and return. The budget 50 000 € is available to select such projects that assure the highest total return.

	P1	P2	P3	P4	P5
Cost	12 000	10 000	15 000	18 000	16 000
Return	20 000	18 000	22 000	26 000	21 000

Design a model to propose the best investment.

(*) Borrowed to Jan Fàbri

Exercises: Perfect Matching* (1)

Exercise: [Perfect Matching] Ten students go for a school trip. To assign them to double rooms, they are asked to express their preferences (see the table, 0-min, 10-max). For i < j , the value cij is the preference value expressing how much student i wants to be in the room with student j , for i > j , the value cij is the preference value expressing how much student j wants to be in the room with student i.

Design a model so to assign students to rooms in order to maximize global happiness of the group.

(*) Borrowed to Jan Fàbri

Exercises: Perfect Matching (2)

	1	2	3	4	5	6	7	8	9	10
1		7	6	2	4	7	4	1	8	3
2	1		3	1	10	5	2	9	4	2
3	10	1		5	6	1	8	2	7	4
4	1	8	4		10	7	5	4	2	7
5	8	7	3	5		2	1	5	2	9
6	2	2	3	7	8		8	2	1	5
7	1	7	6	1	7	7		8	1	5
8	6	8	1	1	10	8	1		4	7
9	4	1	2	2	8	1	7	5		2
10	1	5	4	3	9	7	1	4	6	

Bottleneck Assignment*

Exercise: [BottleneckAssignment] A project consists of 5 independent parts. In the company, 5 departments can manage the parts individually. Historical data shows average times (in days) departments finished similar tasks (see the table below). N.A. represents the fact a department did not work on such task in the past. The company wants to finish the whole project as soon as possible.

Time	Part1	Part2	Part3	Part4	Part5
Dept1	25	15	N.A.	17	25
Dept2	22	N.A.	22	20	22
Dept3	20	18	25	16	23
Dept4	N.A.	20	30	21	28
Dept5	27	19	27	18	N.A.

Design a model to help the company to com

(*) Borrowed to Jan Fàbri

Quadratic Assignment*

Exercise: [QuadraticAssignment] A company intends to establish 5 warehouses in 5 cities. In the first table below, distances (in km) between cities are given. The second table shows a number of necessary travels between warehouses within 1 month.

Design a model to help the company to allocate the warehouses so to minimize total travelling cost.

Dist.	City1	City2	City3	City4	City5
City1	0	50	60	130	100
City2	50	0	70	150	120
City3	60	70	0	80	40
City4	130	150	80	0	50
City5	100	120	40	50	0

Travels	WH1	WH2	WH3	WH4	WH5
WH1	0	10	15	12	8
WH2	9	0	18	16	10
WH3	20	8	0	10	12
WH4	10	15	11	0	22
WH5	17	12	9	11	0

REFERENCES

References

Reference books on constraints:

There exists a vast scientific and technical literature on constraint. Some historical refrences:

Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press Inc (août 1993). ISBN-10: 0127016104. ISBN-13: 978-0127016108

Rina Dechter. *Constraint Processing*. The Morgan Kaufmann Series in Artificial Intelligence. May 19, 2003. ISBN-10: 1558608907. ISBN-13: 978-1558608900.

Francesca Rossi, Peter Van Beek, Toby Walsh (editors): Handbook of Constraint Programming, Elsevier, 2006

http://www.csplib.org/Problems

References on satisfiability modulo theories:

Daniel Kroening and Ofer Strichman Decision. Procedures : An Algorithmic Point of View. Springer-Verlag Berlin and Heidelberg GmbH & Co. K; 2nd ed. 2016. Texts in Theoretical Computer Science. An EATCS Series ISBN-13: 978-3662504963. 2017

http://smtlib.cs.uiowa.edu/

References on combinatorial optimization:

Bernhard Korte and Jens Vygen. Combinatorial Optimization: Theory and Algorithms. Springer-Verlag Berlin and Heidelberg GmbH & Co. K; Édition : 6th ed. Algorithms and Combinatorics. ISBN-13: 978-3662560389. 2018

Christos H. Papadimitriou and Kenneth Steiglitz. Combinatorial Optimization: Algorithms and Complexity. Dover Publications Inc.; Dover Books on Computer Science. ISBN-13: 978-0486402581. 1998.

APPENDIX

GRAMMAR

Models

```
Model ::= (DomainDeclaration | Problem)*

DomainDeclaration ::= domain Identifier (Set | Range)

Set ::= "{" Integer ("," Integer)* "}"
Range ::= "[" Integer "," Integer "]"
Domain ::= Set | Range | Identifier

Problem ::=
    problem Identifier
        VariableDeclaration*
        ConstraintClause?
        ObjectiveFunction?
    end

VariableDeclaration ::= Domain Identifier ( "," Identifier )* ";"
ConstraintClause ::= constraint Constraint+
```

Constraints

```
Constraint ::= AllDifferent | Inequality | Clause

AllDifferent ::= alldifferent Variable ( "," Variable )* ";"

Inequality ::= Polynomial Comparator Integer ";"
Polynomial ::= Monomial (("+" Monomial) | ("-" UnsignedMonomial))*
Monomial ::= "-"? UnsignedMonomial
UnsignedMonomial ::= UnsignedInteger "*" Variable | Variable

Clause ::= or Literal ( "," Literal )* ";"
Literal ::= Variable Comparator Integer

Comparator ::= "==" | "!=" | "<" | ">" | "<=" | ">="
Variable ::= Identifier

Identifier ::= [a-zA-Z][a-zA-Z0-9_]+
Integer ::= "-"? UnsignedInteger
UnsignedInteger ::= [0-9]+
```

Objective Functions and Comments

```
ObjectiveFunction ::= (minimize | maximize) Polynomial ";"
```

Comments can be added everywhere in the code.

- Single line comments introduced by //, which comment out the rest of the line.
- Multiline comments which comment out the text between /* and */.

Known Bugs

Known bugs

- It is not possible to have more than one occurrence of a variable in a polynomial (both for inequalities and objective functions).
- It is not possible to give a null coefficient to a variable in a polynomial