Tutorials

Release 1.0.0

INRAE

Nov 30, 2022

CONTENTS

1 Tutorials 1

2 List of all examples 2

CHAPTER ONE

TUTORIALS

- tutorial materials on cost function networks at ACP/ANITI/GDR-IA/RO Autumn School 2020.
- tutorial on cost function networks at CP2020 (teaser, part1, part2 videos, and script
- tutorial on cost function networks at PFIA 2019 (part1, part2, demo), Toulouse, France, July 4th, 2019.

Here are several examples that can be followed as tutorials. They use toulbar2 in order to resolve different problems. According to cases, they may contain source code, tutorials explaining the examples, possibility to run yourself…

You will find the mentioned examples, among the following exhaustive list of examples.

- *Weighted n-queen problem*
- *Weighted latin square problem*
- *Radio link frequency assignment problem*
- *Frequency assignment problem with polarization*
- *Mendelian error detection problem*
- *Block modeling problem*
- *Airplane landing problem*
- *Warehouse location problem*
- *Square packing problem*
- *Square soft packing problem*
- *Golomb ruler problem*
- *Board coloration problem*
- *Learning to play the Sudoku*
- *Learning car configuration preferences*
- *Visual Sudoku Tutorial*
- *Verbose version of a sudoku code*

CHAPTER

TWO

LIST OF ALL EXAMPLES

2.1 Weighted n-queen problem

2.1.1 Brief description

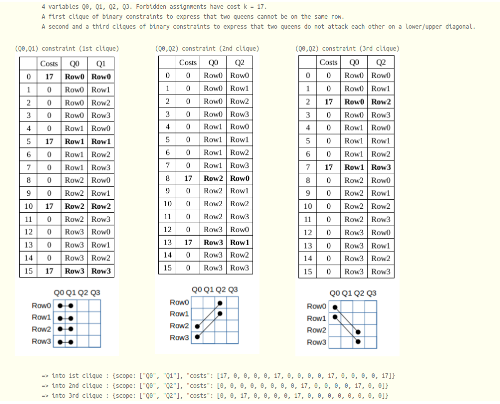
The problem consists in assigning N queens on a NxN chessboard with random costs in (1..N) associated to every cell such that each queen does not attack another queen and the sum of the costs of queen's selected cells is minimized.

2.1.2 CFN model

A solution must have only one queen per column and per row. We create N variables for every column with domain size N to represent the selected row for each queen. A clique of binary constraints is used to express that two queens cannot be on the same row. Forbidden assignments have cost k=N**2+1. Two other cliques of binary constraints are used to express that two queens do not attack each other on a lower/upper diagonal. We add N unary cost functions to create the objective function with random costs on every cell.

2.1.3 Example for N=4 in JSON .cfn format

More details :



```
4 variables Q0, Q1, Q2, Q3. Forbidden assignments have cost k = 17.
A first clique of binary constraints to express that two queens cannot be on the same row.
A second and a third cliques of binary constraints to express that two queens do not attack each other on a lower/upper diagonal.
```

(Q0,Q1) constraint (1st clique)

	Costs	Q0	Q1
0	**17**	**Row0**	**Row0**
1	0	Row0	Row1
2	0	Row0	Row2
3	0	Row0	Row3
4	0	Row1	Row0
5	**17**	**Row1**	**Row1**
6	0	Row1	Row2
7	0	Row1	Row3
8	0	Row2	Row0
9	0	Row2	Row1
10	**17**	**Row2**	**Row2**
11	0	Row2	Row3
12	0	Row3	Row0
13	0	Row3	Row1
14	0	Row3	Row2
15	**17**	**Row3**	**Row3**

(Q0,Q2) constraint (2nd clique)

	Costs	Q0	Q2
0	0	Row0	Row0
1	0	Row0	Row1
2	0	Row0	Row2
3	0	Row0	Row3
4	0	Row1	Row0
5	0	Row1	Row1
6	0	Row1	Row2
7	0	Row1	Row3
8	**17**	**Row2**	**Row0**
9	0	Row2	Row1
10	0	Row2	Row2
11	0	Row2	Row3
12	0	Row3	Row0
13	**17**	**Row3**	**Row1**
14	0	Row3	Row2
15	0	Row3	Row3

(Q0,Q2) constraint (3rd clique)

	Costs	Q0	Q2
0	0	Row0	Row0
1	0	Row0	Row1
2	**17**	**Row0**	**Row2**
3	0	Row0	Row3
4	0	Row1	Row0
5	0	Row1	Row1
6	0	Row1	Row2
7	**17**	**Row1**	**Row3**
8	0	Row2	Row0
9	0	Row2	Row1
10	0	Row2	Row2
11	0	Row2	Row3
12	0	Row3	Row0
13	0	Row3	Row1
14	0	Row3	Row2
15	0	Row3	Row3

```
=> into 1st clique : {scope: ["Q0", "Q1"], "costs": [17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17]}
=> into 2nd clique : {scope: ["Q0", "Q2"], "costs": [0, 0, 0, 0, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0]}
=> into 3rd clique : {scope: ["Q0", "Q2"], "costs": [0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 0, 0, 0, 0]}
```

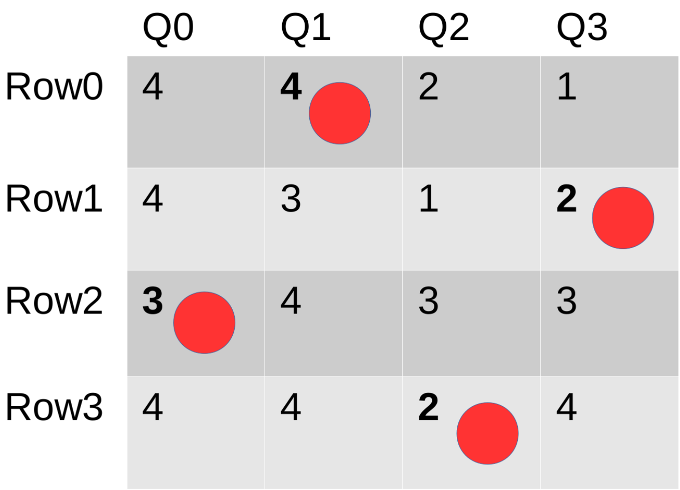
```
{
  problem: { "name": "4-queen", "mustbe": "<17" },
  variables: {"Q0":["Row0", "Row1", "Row2", "Row3"], "Q1":["Row0", "Row1", "Row2", "Row3
↪"],
               "Q2":["Row0", "Row1", "Row2", "Row3"], "Q3":["Row0", "Row1", "Row2", "Row3
```

(continues on next page)

(continued from previous page)

```
↪"]},
  functions: {
    {scope: ["Q0", "Q1"], "costs": [17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17]},
    {scope: ["Q0", "Q2"], "costs": [17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17]},
    {scope: ["Q0", "Q3"], "costs": [17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17]},
    {scope: ["Q1", "Q2"], "costs": [17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17]},
    {scope: ["Q1", "Q3"], "costs": [17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17]},
    {scope: ["Q2", "Q3"], "costs": [17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17]},

    {scope: ["Q0", "Q1"], "costs": [0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0]},
    {scope: ["Q0", "Q2"], "costs": [0, 0, 0, 0, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0]},
    {scope: ["Q0", "Q3"], "costs": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 17, 0, 0, 0]},
    {scope: ["Q1", "Q2"], "costs": [0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0]},
    {scope: ["Q1", "Q3"], "costs": [0, 0, 0, 0, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0]},
    {scope: ["Q2", "Q3"], "costs": [0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0]},

    {scope: ["Q0", "Q1"], "costs": [0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0]},
    {scope: ["Q0", "Q2"], "costs": [0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 0, 0, 0, 0]},
    {scope: ["Q0", "Q3"], "costs": [0, 0, 0, 17, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]},
    {scope: ["Q1", "Q2"], "costs": [0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0]},
    {scope: ["Q1", "Q3"], "costs": [0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 0, 0, 0, 0]},
    {scope: ["Q2", "Q3"], "costs": [0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 0]},

    {scope: ["Q0"], "costs": [4, 4, 3, 4]},
    {scope: ["Q1"], "costs": [4, 3, 4, 4]},
    {scope: ["Q2"], "costs": [2, 1, 3, 2]},
    {scope: ["Q3"], "costs": [1, 2, 3, 4]}}
}
```

Optimal solution with cost 11 for the 4-queen example :



	Q0	Q1	Q2	Q3
Row0	4	**4** ●	2	1
Row1	4	3	1	**2** ●
Row2	**3** ●	4	3	3
Row3	4	4	**2** ●	4

2.1.4 Python model

The following code using the pytoulbar2 library solves the weighted N-queen problem with the first argument being the number of queens N (e.g. “python3 weightedqueens.py 8”).

`weightedqueens.py`

```
import sys
from random import seed, randint
seed(123456789)
import pytoulbar2

N = int(sys.argv[1])

top = N**2 +1

Problem = pytoulbar2.CFN(top)

for i in range(N):
    Problem.AddVariable('Q' + str(i+1), ['row' + str(a+1) for a in range(N)])

for i in range(N):
        for j in range(i+1,N):

                #Two queens cannot be on the same row constraints
                ListConstraintsRow = []
                for a in range(N):
                        for b in range(N):
                                if a != b :
                                        ListConstraintsRow.append(0)
                                else:
                                         ListConstraintsRow.append(top)
                Problem.AddFunction([i, j], ListConstraintsRow)

                #Two queens cannot be on the same upper diagonal constraints
                ListConstraintsUpperD = []
                for a in range(N):
                        for b in range(N):
                                if a + i != b + j :
                                        ListConstraintsUpperD.append(0)
                                else:
                                         ListConstraintsUpperD.append(top)
                Problem.AddFunction([i, j], ListConstraintsUpperD)

                #Two queens cannot be on the same lower diagonal constraints
                ListConstraintsLowerD = []
                for a in range(N):
                        for b in range(N):
                                if a - i != b - j :
                                        ListConstraintsLowerD.append(0)
                                else:
                                         ListConstraintsLowerD.append(top)
                Problem.AddFunction([i, j], ListConstraintsLowerD)
```

(continues on next page)

(continued from previous page)

```
#Random unary costs
for i in range(N):
        ListConstraintsUnaryC = []
        for j in range(N):
                ListConstraintsUnaryC.append(randint(1,N))
        Problem.AddFunction([i], ListConstraintsUnaryC)


#Problem.Dump('WeightQueen.cfn')
Problem.CFN.timer(300)
res = Problem.Solve(showSolutions = 3)
if res:
        for i in range(N):
                row = ['X' if res[0][j]==i else ' ' for j in range(N)]
                print(row)
        # and its cost
        print("Cost:", int(res[1]))
```

2.2 Weighted latin square problem

2.2.1 Brief description

The problem consists in assigning a value from 0 to N-1 to every cell of a NxN chessboard. Each row and each column must be a permutation of N values. For each cell, a random cost in (1…N) is associated to every domain value. The objective is to find a complete assignment where the sum of the costs associated to the selected values for the cells is minimized.

2.2.2 CFN model

We create NxN variables, one for every cell, with domain size N. An AllDifferent hard global constraint is used to model a permutation for every row and every column. Its encoding uses knapsack constraints. Unary cost functions containing random costs associated to domain values are generated for every cell. The worst possible solution is when every cell is associated with a cost of N, so the maximum cost of a solution is N**3, so forbidden assignments have cost k=N**3+1.

2.2.3 Example for N=4 in JSON .cfn format

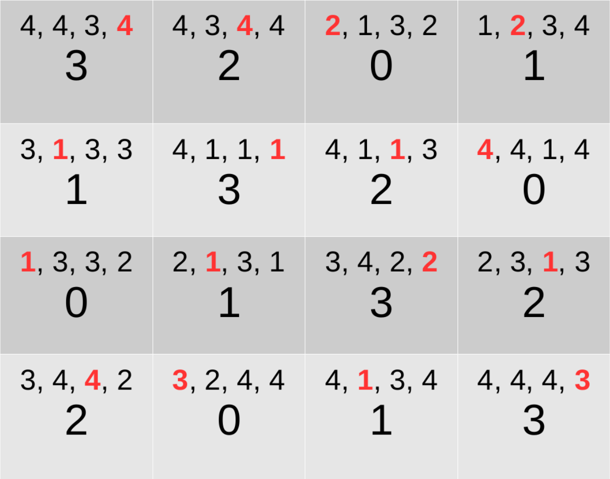
```
{
  problem: { "name": "LatinSquare4", "mustbe": "<65" },
  variables: {"X0_0": 4, "X0_1": 4, "X0_2": 4, "X0_3": 4, "X1_0": 4, "X1_1": 4, "X1_2":␣
↪4, "X1_3": 4, "X2_0": 4, "X2_1": 4, "X2_2": 4, "X2_3": 4, "X3_0": 4, "X3_1": 4, "X3_2
↪": 4, "X3_3": 4},
  functions: {
    {scope: ["X0_0", "X0_1", "X0_2", "X0_3"], "type:" salldiff, "params": {"metric": "var
↪", "cost": 65}},
    {scope: ["X1_0", "X1_1", "X1_2", "X1_3"], "type:" salldiff, "params": {"metric": "var
```

(continues on next page)

(continued from previous page)

```
↪", "cost": 65}},
    {scope: ["X2_0", "X2_1", "X2_2", "X2_3"], "type:" salldiff, "params": {"metric": "var
↪", "cost": 65}},
    {scope: ["X3_0", "X3_1", "X3_2", "X3_3"], "type:" salldiff, "params": {"metric": "var
↪", "cost": 65}},

    {scope: ["X0_0", "X1_0", "X2_0", "X3_0"], "type:" salldiff, "params": {"metric": "var
↪", "cost": 65}},
    {scope: ["X0_1", "X1_1", "X2_1", "X3_1"], "type:" salldiff, "params": {"metric": "var
↪", "cost": 65}},
    {scope: ["X0_2", "X1_2", "X2_2", "X3_2"], "type:" salldiff, "params": {"metric": "var
↪", "cost": 65}},
    {scope: ["X0_3", "X1_3", "X2_3", "X3_3"], "type:" salldiff, "params": {"metric": "var
↪", "cost": 65}},

    {scope: ["X0_0"], "costs": [4, 4, 3, 4]},
    {scope: ["X0_1"], "costs": [4, 3, 4, 4]},
    {scope: ["X0_2"], "costs": [2, 1, 3, 2]},
    {scope: ["X0_3"], "costs": [1, 2, 3, 4]},
    {scope: ["X1_0"], "costs": [3, 1, 3, 3]},
    {scope: ["X1_1"], "costs": [4, 1, 1, 1]},
    {scope: ["X1_2"], "costs": [4, 1, 1, 3]},
    {scope: ["X1_3"], "costs": [4, 4, 1, 4]},
    {scope: ["X2_0"], "costs": [1, 3, 3, 2]},
    {scope: ["X2_1"], "costs": [2, 1, 3, 1]},
    {scope: ["X2_2"], "costs": [3, 4, 2, 2]},
    {scope: ["X2_3"], "costs": [2, 3, 1, 3]},
    {scope: ["X3_0"], "costs": [3, 4, 4, 2]},
    {scope: ["X3_1"], "costs": [3, 2, 4, 4]},
    {scope: ["X3_2"], "costs": [4, 1, 3, 4]},
    {scope: ["X3_3"], "costs": [4, 4, 4, 3]}}
}
```

Optimal solution with cost 35 for the latin 4-square example (in red, costs associated to the selected values) :



4, 4, 3, **4** 3	4, 3, **4**, 4 2	**2**, 1, 3, 2 0	1, **2**, 3, 4 1
3, **1**, 3, 3 1	4, 1, 1, **1** 3	4, 1, **1**, 3 2	**4**, 4, 1, 4 0
1, 3, 3, 2 0	2, **1**, 3, 1 1	3, 4, 2, **2** 3	2, 3, **1**, 3 2
3, 4, **4**, 2 2	**3**, 2, 4, 4 0	4, **1**, 3, 4 1	4, 4, 4, **3** 3

2.2.4 Python model

The following code using the pytoulbar2 library solves the weighted latin square problem with the first argument being the dimension N of the chessboard (e.g. “python3 latinsquare.py 6”).

`latinsquare.py`

```
import sys
from random import seed, randint
seed(123456789)
import pytoulbar2

N = int(sys.argv[1])

top = N**3 +1

Problem = pytoulbar2.CFN(top)

for i in range(N):
    for j in range(N):
        #Create a variable for each square
        Problem.AddVariable('Cell(' + str(i) + ',' + str(j) + ')', range(N))

for i in range(N):
    #Create a constraint all different with variables on the same row
    Problem.AddAllDifferent(['Cell(' + str(i) + ',' + str(j) + ')' for j in range(N)],␣
↪encoding = 'salldiffkp')

    #Create a constraint all different with variables on the same column
    Problem.AddAllDifferent(['Cell(' + str(j) + ',' + str(i) + ')'for j in range(N)],␣
↪encoding = 'salldiffkp')

#Random unary costs
for i in range(N):
    for j in range(N):
        ListConstraintsUnaryC = []
        for l in range(N):
            ListConstraintsUnaryC.append(randint(1,N))
        Problem.AddFunction(['Cell(' + str(i) + ',' + str(j) + ')'],␣
↪ListConstraintsUnaryC)

#Problem.Dump('WeightLatinSquare.cfn')
Problem.CFN.timer(300)
res = Problem.Solve(showSolutions = 3)
if res and len(res[0]) == N*N:
    # pretty print solution
    for i in range(N):
        print([res[0][i * N + j] for j in range(N)])
    # and its cost
    print("Cost:", int(res[1]))
```

2.3 Radio link frequency assignment problem

2.3.1 Brief description

The problem consists in assigning frequencies to radio communication links in such a way that no interferences occur. Domains are set of integers (non necessarily consecutives).

Two types of constraints occur:

- (I) the absolute difference between two frequencies should be greater than a given number d_i (| x - y | > d_i)
- (II) the absolute difference between two frequencies should exactly be equal to a given number d_i (| x - y | = d_i).

Different deviations d_i, i in 0..4, may exist for the same pair of links. d_0 corresponds to hard constraints while higher deviations are soft constraints that can be violated with an associated cost a_i. Moreover, pre-assigned frequencies may be known for some links which are either hard or soft preferences (mobility cost b_i, i in 0..4). The goal is to minimize the weighted sum of violated constraints.

So the goal is to minimize the sum:
a_1*nc1+…+a_4*nc4+b_1*nv1+…+b_4*nv4

where nci is the number of violated constraints with cost a_i and nvi is the number of modified variables with mobility cost b_i.

Cabon, B., de Givry, S., Lobjois, L., Schiex, T., Warners, J.P. Constraints (1999) 4: 79.

2.3.2 CFN model

We create N variables for every radio link with a given integer domain. Hard and soft binary cost functions express interference constraints with possible deviations with cost equal to a_i. Unary cost functions are used to model mobility costs with cost equal to b_i. The initial upper bound is defined as 1 plus the total cost where all the soft constraints are maximally violated (costs a_4/b_4).

2.3.3 Data

Original data files can be download from the cost function library FullRLFAP. Their format is described here. You can try a small example CELAR6-SUB1 (`var.txt`, `dom.txt`, `ctr.txt`, `cst.txt`) with optimum value equal to 2669.

2.3.4 Python model

The following code solves the corresponding cost function network using the pytoulbar2 library and needs 4 arguments: the variable file, the domain file, the constraints file and the cost file (e.g. "python3 rlfap.py var.txt dom.txt ctr.txt cst.txt").

`rlfap.py`

```
import sys
import pytoulbar2

class Data:
        def __init__(self, var, dom, ctr, cst):
                self.var = list()
                self.dom = {}
```

(continues on next page)

(continued from previous page)

```
                self.ctr = list()
                self.cost = {}
                self.nba = {}
                self.nbb = {}
                self.top = 1
                self.Domain = {}

                stream = open(var)
                for line in stream:
                        if len(line.split())>=4:
                                (varnum, vardom, value, mobility) = line.split()[:4]
                                self.Domain[int(varnum)] = int(vardom)
                                self.var.append((int(varnum), int(vardom), int(value),␣
↪int(mobility)))
                                self.nbb["b" + str(mobility)] = self.nbb.get("b" +␣
↪str(mobility), 0) + 1
                        else:
                                (varnum, vardom) = line.split()[:2]
                                self.Domain[int(varnum)] = int(vardom)
                                self.var.append((int(varnum), int(vardom)))
                stream = open(dom)
                for line in stream:
                        domain = line.split()[:]
                        self.dom[int(domain[0])] = [int(f) for f in domain[2:]]

                stream = open(ctr)
                for line in stream:
                        (var1, var2, dummy, operand, deviation, weight) = line.
↪split()[:6]
                        self.ctr.append((int(var1), int(var2), operand, int(deviation),␣
↪int(weight)))
                        self.nba["a" + str(weight)] = self.nba.get("a" + str(weight), 0)␣
↪+ 1

                stream = open(cst)
                for line in stream:
                        if len(line.split()) == 3:
                                (aorbi, eq, cost) = line.split()[:3]
                                if (eq == "="):
                                        self.cost[aorbi] = int(cost)
                                        self.top += int(cost) * self.nba.get(aorbi, self.
↪nbb.get(aorbi, 0))


#collect data
data = Data(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4])

top = data.top
Problem = pytoulbar2.CFN(top)

#create a variable for each link
for e in data.var:
```

(continues on next page)

(continued from previous page)

```
        domain = []
        for f in data.dom[e[1]]:
                domain.append('f' + str(f))
        Problem.AddVariable('link' + str(e[0]), domain)

#binary hard and soft constraints
for (var1, var2, operand, deviation, weight) in data.ctr:
        ListConstraints = []
        for a in data.dom[data.Domain[var1]]:
                for b in data.dom[data.Domain[var2]]:
                        if ((operand==">" and abs(a - b) > deviation) or (operand=="="␣
↪and abs(a - b) == deviation)):
                                ListConstraints.append(0)
                        else:
                                ListConstraints.append(data.cost.get('a' + str(weight),
↪top))
        Problem.AddFunction(['link' + str(var1), 'link' + str(var2)], ListConstraints)

#unary hard and soft constraints
for e in data.var:
        if len(e) >= 3:
                ListConstraints = []
                for a in data.dom[e[1]]:
                        if a == e[2]:
                                ListConstraints.append(0)
                        else:
                                ListConstraints.append(data.cost.get('b' + str(e[3]),
↪top))
                Problem.AddFunction(['link' + str(e[0])], ListConstraints)

#Problem.Dump('Rflap.cfn')
Problem.CFN.timer(300)
res = Problem.Solve(showSolutions=3)
if res:
        print("Best solution found with cost:",int(res[1]),"in", Problem.GetNbNodes(),
↪"search nodes.")
else:
        print('Sorry, no solution found!')
```

2.4 Frequency assignment problem with polarization

2.4.1 Brief description

The previously-described *Radio link frequency assignment problem* has been extended to take into account polarization constraints and user-defined relaxation of electromagnetic compatibility constraints. The problem is to assign a pair (frequency,polarization) to every radio communication link (also called a path). Frequencies are integer values taken in finite domains. Polarizations are in {-1,1}. Constraints are :

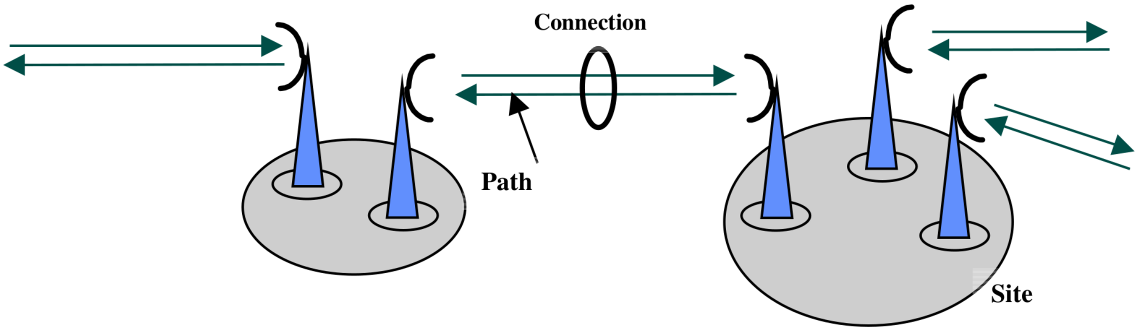
- (I) two paths must use equal or different frequencies (*f_i=f_j* or *f_i<>f_j*),

- (II) the absolute difference between two frequencies should exactly be equal or different to a given number e (|*f_i-f_j*|=*e* or |*f_i-f_j*|<>*e*),
- (III) two paths must use equal or different polarizations (*p_i*=*p_j* or *p_i*<>*p_j*),
- (IV) the absolute difference between two frequencies should be greater at a relaxation level l (0 to 10) than a given number g_l (resp. d_l) if polarization are equal (resp. different) (|*f_i-f_j*|>=*g_l* if *p_i*=*p_j* else |*f_i-f_j*|>=*d_l*), with *g_(l-1)*>*g_l*, *d_(l-1)*>*d_l*, and usually *g_l*>*d_l*.

Constraints (I) to (III) are mandatory constraints, while constraints (IV) can be relaxed. The goal is to find a feasible assignment with the smallest relaxation level l and which minimizes the (weighted) number of violations of (IV) at lower levels. See `ROADEF_Challenge_2001` .

The cost of a given solution will be calculated by the following formula: 10*k*nbsoft**2 + 10*nbsoft*V(k-1) + V(k-2) + V(k-3) + … + V0

where nbsoft is the number of soft constraints in the problem and k the feasible relaxation level and V(i) the number of violated contraints of level i.



2.4.2 CFN model

We create a single variable to represent a pair (frequency,polarization) for every radio link, but be aware, toulbar2 can only read str or int values, so in order to give a tuple to toulbar2 we need to first transform them into string. We use hard binary constraints to modelize (I) to (III) type constraints.

We assume the relaxation level k is given as input. In order to modelize (IV) type constraints we first take in argument the level of relaxation i, and we create 11 constraints, one for each relaxation level from 0 to 10. The first k-2 constraints are soft and with a violation cost of 1. The soft constraint at level k-1 has a violation cost 10*nbsoft (the number of soft constraints) in order to maximize first the number of satisfied constraints at level k-1 and then the other soft constraints. The constraints at levels k to 10 are hard constraints.

The initial upper bound of the problem will be 10*(k+1)*nbsoft**2 +1.

2.4.3 Data

Original data files can be download from ROADEF or fapp. Their format is described here. You can try a small example `exemple1.in` (resp. `exemple2.in`) with optimum 523 at relaxation level 3 with 1 violation at level 2 and 3 below (resp. 13871 at level 7 with 1 violation at level 6 and 11 below). See ROADEF Challenge 2001 results.

2.4.4 Python model

The following code solves the corresponding cost function network using the pytoulbar2 library and needs 4 arguments: the data file and the relaxation level (e.g. “python3 fapp.py exemple1.in 3”). You can also compile `fappeval.c` using “gcc -o fappeval fappeval.c” and download `sol2fapp.awk` in order to evaluate the solutions (e.g., “python3 fapp.py exemple1.in 3 | awk -f ./sol2fapp.awk - exemple1”).

`fapp.py`

```
import sys
import pytoulbar2

class Data:
        def __init__(self, filename, k):
                self.var = {}
                self.dom = {}
                self.ctr = list()
                self.softeq = list()
                self.softne = list()
                self.nbsoft = 0

                stream = open(filename)
                for line in stream:
                        if len(line.split())==3 and line.split()[0]=="DM":
                                (DM, dom, freq) = line.split()[:3]
                                if self.dom.get(int(dom)) is None:
                                        self.dom[int(dom)] = [int(freq)]
                                else:
                                        self.dom[int(dom)].append(int(freq))

                        if len(line.split()) == 4 and line.split()[0]=="TR":
                                (TR, route, dom, polarisation) = line.split()[:4]
                                if int(polarisation) == 0:
                                        self.var[int(route)] = [(f,-1) for f in self.
↪dom[int(dom)]] + [(f,1) for f in self.dom[int(dom)]]
                                if int(polarisation) == -1:
                                        self.var[int(route)] = [(f,-1) for f in self.
↪dom[int(dom)]]
                                if int(polarisation) == 1:
                                        self.var[int(route)] = [(f,1) for f in self.
↪dom[int(dom)]]

                        if len(line.split())==6 and line.split()[0]=="CI":
                                (CI, route1, route2, vartype, operator, deviation) =␣
↪line.split()[:6]
                                self.ctr.append((int(route1), int(route2), vartype,␣
↪operator, int(deviation)))

                        if len(line.split())==14 and line.split()[0]=="CE":
                                (CE, route1, route2, s0, s1, s2, s3, s4, s5, s6, s7, s8,␣
↪s9, s10) = line.split()[:14]
                                self.softeq.append((int(route1), int(route2), [int(s)␣
↪for s in [s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10]]))
                                self.nbsoft += 1
```

(continues on next page)

(continued from previous page)

```
                        if len(line.split())==14 and line.split()[0]=="CD":
                                (CD, route1, route2, s0, s1, s2, s3, s4, s5, s6, s7, s8,␣
↪s9, s10) = line.split()[:14]
                                self.softne.append((int(route1), int(route2), [int(s)␣
↪for s in [s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10]]))

                self.top = 10*(k+1)*self.nbsoft**2 + 1

if len(sys.argv) < 2:
        exit('Command line argument is composed of the problem data filename and the␣
↪relaxation level')
k = int(sys.argv[2])

#collect data
data = Data(sys.argv[1], k)

Problem = pytoulbar2.CFN(data.top)

#create a variable for each link
for e in list(data.var.keys()):
        domain = []
        for i in data.var[e]:
                domain.append(str(i))
        Problem.AddVariable("X" + str(e), domain)

#hard binary constraints
for (route1, route2, vartype, operand, deviation) in data.ctr:
        Constraint = []
        for (f1,p1) in data.var[route1]:
                 for (f2,p2) in data.var[route2]:
                         if vartype == 'F':
                                 if operand == 'E':
                                         if abs(f2 - f1) == deviation:
                                                 Constraint.append(0)
                                         else:
                                                 Constraint.append(data.top)
                                 else:
                                         if abs(f2 - f1) != deviation:
                                                 Constraint.append(0)
                                         else:
                                                 Constraint.append(data.top)
                         else:
                                 if operand == 'E':
                                         if p2 == p1:
                                                 Constraint.append(0)
                                         else:
                                                 Constraint.append(data.top)
                                 else:
                                         if p2 != p1:
                                                 Constraint.append(0)
                                         else:
```

(continues on next page)

(continued from previous page)

```
                                                 Constraint.append(data.top)
        Problem.AddFunction(["X" + str(route1), "X" + str(route2)], Constraint)


#soft binary constraints for equal polarization
for (route1, route2, deviations) in data.softeq:
        for i in range(11):
                ListConstraints = []
                for (f1,p1) in data.var[route1]:
                        for (f2,p2) in data.var[route2]:
                                 if p1!=p2 or abs(f1 - f2) >= deviations[i]:
                                         ListConstraints.append(0)
                                 elif i >= k:
                                         ListConstraints.append(data.top)
                                 elif i == k-1:
                                         ListConstraints.append(10*data.nbsoft)
                                 else:
                                         ListConstraints.append(1)
                Problem.AddFunction(["X" + str(route1), "X" + str(route2)],␣
↪ListConstraints)


#soft binary constraints for not equal polarization
for (route1, route2, deviations) in data.softne:
        for i in range(11):
                ListConstraints = []
                for (f1,p1) in data.var[route1]:
                         for (f2,p2) in data.var[route2]:
                                 if p1==p2 or abs(f1 - f2) >= deviations[i]:
                                         ListConstraints.append(0)
                                 elif i >= k:
                                         ListConstraints.append(data.top)
                                 elif i == k-1:
                                         ListConstraints.append(10*data.nbsoft)
                                 else:
                                         ListConstraints.append(1)
                Problem.AddFunction(["X" + str(route1), "X" + str(route2)],␣
↪ListConstraints)

#zero-arity cost function representing a constant cost corresponding to the relaxation␣
↪at level k
Problem.AddFunction([], 10*k*data.nbsoft**2)

#Problem.Dump('Fapp.cfn')
Problem.CFN.timer(900)
Problem.Solve(showSolutions=3)
```

2.5 Mendelian error detection problem

2.5.1 Brief description

The problem is to detect marker genotyping incompatibilities (Mendelian errors) in complex pedigrees. The input is a pedigree data with partial observed genotyping data at a single locus, we assume the pedigree to be exact, but not the genotyping data. The problem is to assign genotypes (unordered pairs of alleles) to all individuals such that they are compatible with the Mendelian law of heredity (one allele is the same as their father's and one as their mother's). The goal is to maximize the number of matching allelle between the genotyping data and the solution. Each difference from the genotyping data has a cost of 1.

Sanchez, M., de Givry, S. and Schiex, T. Constraints (2008) 13:130.

2.5.2 CFN model

We create N variables, one for each individual genotype with domain being all possible unordered pairs of existing alleles. Hard ternary cost functions express mendelian law of heredity (one allele is the same as their father's and one as their mother's, with mother and father defined in the pedigree data). For each genotyping data, we create one unary soft constraint with violation cost equal to 1 to represent the matching between the genotyping data and the solution.

2.5.3 Data

Original data files can be download from the cost function library pedigree. Their format is described here. You can try a small example simple.pre (`simple.pre`) with optimum value equal to 1.

2.5.4 Python model

The following code solves the corresponding cost function network using the pytoulbar2 library (e.g. "python3 mendel.py simple.pre").

`mendel.py`

```
import sys
import pytoulbar2

class Data:
	def __init__(self, ped):
		self.id = list()
		self.father = {}
		self.mother = {}
		self.allelesId = {}
		self.ListAlle = list()
		self.obs = 0

		stream = open(ped)
		for line in stream:
			(locus, id, father, mother, sex, allele1, allele2) = line.
↪split()[:]
			self.id.append(int(id))
			self.father[int(id)] = int(father)
			self.mother[int(id)] = int(mother)
```

(continues on next page)

(continued from previous page)

```
                        self.allelesId[int(id)] = (int(allele1), int(allele2)) if␣
↪int(allele1) < int(allele2) else (int(allele2), int(allele1))
                        if not(int(allele1) in self.ListAlle) and int(allele1) != 0:
                                self.ListAlle.append(int(allele1))
                        if int(allele2) != 0 and not(int(allele2) in self.ListAlle):
                                self.ListAlle.append(int(allele2))
                        if int(allele1) != 0 or int(allele2) != 0:
                                self.obs += 1

#collect data
data = Data(sys.argv[1])
top = int(data.obs+1)

Problem = pytoulbar2.CFN(top)

#create a variable for each individual
for i in data.id:
        domains = []
        for a1 in data.ListAlle:
                for a2 in data.ListAlle:
                        if a1 <= a2:
                                domains.append('a'+str(a1)+'a'+str(a2))
        Problem.AddVariable('g' + str(i) , domains)

#create the constraints that represent the mendel's laws
ListConstraintsMendelLaw = []
for p1 in data.ListAlle:
        for p2 in data.ListAlle:
                if p1 <= p2:        # father alleles
                        for m1 in data.ListAlle:
                                for m2 in data.ListAlle:
                                        if m1 <= m2:        # mother alleles
                                                for a1 in data.ListAlle:
                                                        for a2 in data.ListAlle:
                                                                if a1 <= a2:        #␣
↪child alleles
                                                                        if (a1 in (p1,
↪p2) and a2 in (m1,m2)) or (a2 in (p1,p2) and a1 in (m1,m2)) :
                                                                                ListConstraintsMendelLa
↪append(0)
                                                                        else :
                                                                        ␣
↪         ListConstraintsMendelLaw.append(top)

for i in data.id:
        #ternary constraints representing mendel's laws
        if data.father.get(i, 0) != 0 and data.mother.get(i, 0) != 0:
                Problem.AddFunction(['g' + str(data.father[i]),'g' + str( data.
↪mother[i]), 'g' + str(i)], ListConstraintsMendelLaw)

        #unary constraints linked to the observations
        if data.allelesId[i][0] != 0 and data.allelesId[i][1] != 0:
```

(continues on next page)

(continued from previous page)

```
                ListConstraintsObservation = []
                for a1 in data.ListAlle:
                        for a2 in data.ListAlle:
                                if a1 <= a2:
                                        if (a1,a2) == data.allelesId[i]:
                                                ListConstraintsObservation.append(0)
                                        else :
                                                 ListConstraintsObservation.append(1)
                Problem.AddFunction(['g' + str(i)], ListConstraintsObservation)

#Problem.Dump('Mendel.cfn')
Problem.CFN.timer(300)
res = Problem.Solve(showSolutions=3)
if res:
        print('There are',int(res[1]),'difference(s) between the solution and the␣
↪observation.')
else:
        print('No solution found')
```

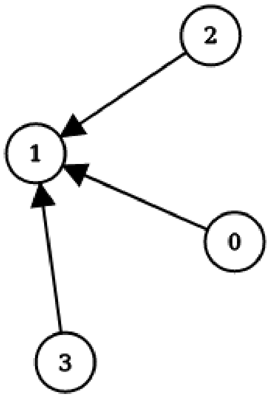
2.6 Block modeling problem

2.6.1 Brief description

This is a clustering problem, occuring in social network analysis.

The problem is to divide a given directed graph G into k clusters such that the interactions between clusters can be summarized by a k*k 0/1 matrix M: if M[i,j]=1 then all the nodes in cluster i should be connected to all the nodes in cluster j in G, else if M[i,j]=0 then there should be no edge in G between the nodes from the two clusters.

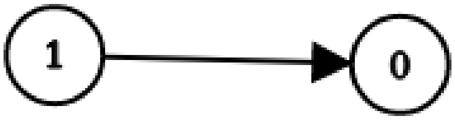
For example, the following graph G is composed of 4 nodes:



and corresponds to the following matrix:

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

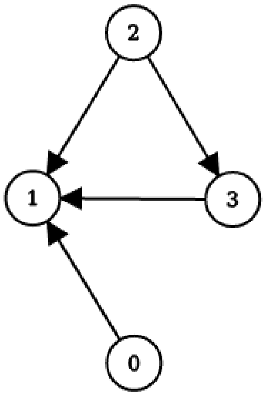
It can be perfectly clusterized into the following graph by clustering together the nodes 0, 2 and 3 in cluster 1 and the node 1 in cluster 0:



and this graph corresponds to the following M matrix:

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

On the contrary, if we decide to cluster the next graph G' in the same way as above, the edge (2, 3) will be 'lost' in the process and the cost of the solution will be 1.



The goal is to find a k-clustering of a given graph and the associated matrix M that minimize the number of erroneous edges.

A Mattenet, I Davidson, S Nijssen, P Schaus. Generic Constraint-Based Block Modeling Using Constraint Programming. CP 2019, pp656-673, Stamford, CT, USA.

2.6.2 CFN model

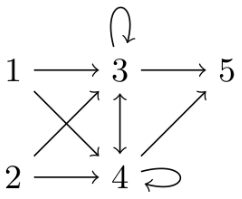
We create N variables, one for every node of the graph, with domain size k representing the clustering. We add k*k Boolean variables for representing M.

For all triplets of two nodes u, v, and one matrix cell M[i,j], we have a ternary cost function which returns a cost of 1 if node u is assigned to cluster i, v to j, and M[i,j]=1 but (u,v) is not in G, or M[i,j]=0 and (u,v) is in G. In order to break symmetries, we constrain the first k-1 node variables to be assigned to a cluster number less than or equal to their index

2.6.3 Data

You can try a small example `simple.mat` with optimum value equal to 0 for 3 clusters.

Perfect solution for the small example with k=3 (Mattenet et al, CP 2019)

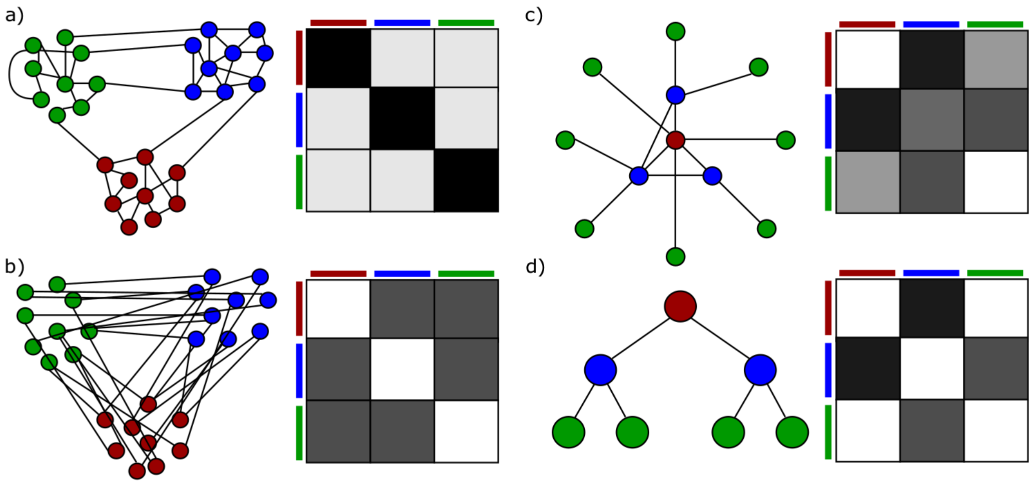


$$G = \left(\begin{array}{cc|cc|c} \mathbf{0} & \mathbf{0} & 1 & 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & 1 & 1 & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & 1 & 1 & 1 \\ \mathbf{0} & \mathbf{0} & 1 & 1 & 1 \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array}\right)$$

$$\begin{tikzcd} \{1,2\} \arrow[r] & \{3,4\} \arrow[r] \arrow[loop above] & \{5\} \end{tikzcd}$$

$$M = \begin{pmatrix} \mathbf{0} & 1 & \mathbf{0} \\ \mathbf{0} & 1 & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}$$

More examples with 3 clusters (Stochastic Block Models [Funke and Becker, Plos One 2019])



See other examples, such as PoliticalActor and more, here : `100.mat` | `150.mat` | `200.mat` | `30.mat` | `50.mat` | `hartford_drug.mat` | `kansas.mat` | `politicalactor.mat` | `sharpstone.mat` | `transatlantic.mat`.

2.6.4 Python model

The following code using pytoulbar2 library solves the corresponding cost function network (e.g. “python3 blockmodel.py simple.mat 3”).

`blockmodel.py`

```
import sys
import pytoulbar2

#read adjency matrix of graph G
```

(continues on next page)

(continued from previous page)

```
Lines = open(sys.argv[1], 'r').readlines()
GMatrix = [[int(e) for e in l.split(' ')] for l in Lines]

N = len(Lines)
Top = N*N + 1

K = int(sys.argv[2])

#give names to node variables
Var = [(chr(65 + i) if N < 28 else "x" + str(i)) for i in range(N)] # Political actor or␣
↪any instance
#    Var = ["ron","tom","frank","boyd","tim","john","jeff","jay","sandy","jerry","darrin
↪","ben","arnie"] # Transatlantic
#    Var = ["justin","harry","whit","brian","paul","ian","mike","jim","dan","ray","cliff
↪","mason","roy"] # Sharpstone
#    Var = ["Sherrif","CivilDef","Coroner","Attorney","HighwayP","ParksRes","GameFish",
↪"KansasDOT","ArmyCorps","ArmyReserve","CrableAmb","FrankCoAmb","LeeRescue","Shawney",
↪"BurlPolice","LyndPolice","RedCross","TopekaFD","CarbFD","TopekaRBW"] # Kansas

Problem = pytoulbar2.CFN(Top)

#create a Boolean variable for each coefficient of the M GMatrix
for u in range(K):
    for v in range(K):
        Problem.AddVariable("M_" + str(u) + "_" + str(v), range(2))

#create a domain variable for each node in graph G
for i in range(N):
    Problem.AddVariable(Var[i], range(K))

#general case for each edge in G
for u in range(K):
    for v in range(K):
        for i in range(N):
            for j in range(N):
                if i != j:
                    ListCost = []
                    for m in range(2):
                        for k in range(K):
                            for l in range(K):
                                if (u == k and v == l and GMatrix[i][j] != m):
                                    ListCost.append(1)
                                else:
                                    ListCost.append(0)
                    Problem.AddFunction(["M_" + str(u) + "_" + str(v), Var[i], Var[j]],
↪ListCost)



# self-loops must be treated separately as they involves only two variables
for u in range(K):
    for i in range(N):
```

(continues on next page)

(continued from previous page)

```
        ListCost = []
        for m in range(2):
            for k in range(K):
                if (u == k and GMatrix[i][i] != m):
                    ListCost.append(1)
                else:
                    ListCost.append(0)
        Problem.AddFunction(["M_" + str(u) + "_" + str(u), Var[i]], ListCost)

# breaking partial symmetries by fixing first (K-1) domain variables to be assigned to a␣
↪cluster number less than or equal to their index
for l in range(K-1):
    Constraint = []
    for k in range(K):
        if k > l:
            Constraint.append(Top)
        else:
            Constraint.append(0)
    Problem.AddFunction([Var[l]], Constraint)

Problem.Dump(sys.argv[1].replace('.mat','.cfn'))
Problem.CFN.timer(300)
res = Problem.Solve(showSolutions = 3)
if res:
    print("M matrix:")
    for u in range(K):
        Line = []
        for v in range(K):
            Line.append(res[0][u*K+v])
        print(Line)
    for k in range(K):
        for i in range(N):
            if res[0][K**2+i] == k:
                print("Node",Var[i],"with index",str(i),"is in cluster",
↪str(res[0][K**2+i]))
```

2.7 Airplane landing problem

2.7.1 Brief description

We consider a single plane's landing runway. Given a set of planes with given target landing time, the objective is to minimize the total weighted deviation from the target landing time for each plane.

There are costs associated with landing either earlier or later than the target landing time for each plane.

Each plane has to land within its predetermined time window. For each pair of planes, there is an additionnal contraints to enforce that separation between those planes in bigger than a given number.

J.E. Beasley, M. Krishnamoorthy, Y.M. Sharaiha and D. Abramson. Scheduling aircraft landings - the static case. Transportation Science, vol.34, 2000.

2.7.2 CFN model

We create N variables, one for each plane, with domain value equal to all their possible landing time.

Binary hard cost functions express separation times between pairs of planes. Unary soft cost functions represent the weighted deviation for each plane.

2.7.3 Data

Original data files can be download from the cost function library airland. Their format is described here. You can try a small example `airland1.txt` with optimum value equal to 700.

2.7.4 Python model solver

The following code uses the pytoulbar2 module to generate the cost function network and solve it (e.g. "python3 airland.py airland1.txt").

`airland.py`

```
import sys
import pytoulbar2

f = open(sys.argv[1], 'r').readlines()

tokens = []
for l in f:
    tokens += l.split()

pos = 0

def token():
    global pos, tokens
    if (pos == len(tokens)):
        return None
    s = tokens[pos]
    pos += 1
    return int(float(s))

N = token()
token() # skip freeze time

LT = []
PC = []
ST = []

for i in range(N):
    token()  # skip appearance time
# Times per plane: {earliest landing time, target landing time, latest landing time}
    LT.append([token(), token(), token()])

# Penalty cost per unit of time per plane:
# [for landing before target, after target]
    PC.append([token(), token()])
```

(continues on next page)

(continued from previous page)

```
# Separation time required after i lands before j can land
    ST.append([token() for j in range(N)])

top = 99999

Problem = pytoulbar2.CFN(top)
for i in range(N):
    Problem.AddVariable('x' + str(i), range(LT[i][0],LT[i][2]+1))

for i in range(N):
    ListCost = []
    for a in range(LT[i][0], LT[i][2]+1):
        if a < LT[i][1]:
            ListCost.append(PC[i][0]*(LT[i][1] - a))
        else:
            ListCost.append(PC[i][1]*(a - LT[i][1]))
    Problem.AddFunction([i], ListCost)

for i in range(N):
    for j in range(i+1,N):
        Constraint = []
        for a in range(LT[i][0], LT[i][2]+1):
            for b in range(LT[j][0], LT[j][2]+1):
                if a+ST[i][j]>b and b+ST[j][i]>a:
                    Constraint.append(top)
                else:
                    Constraint.append(0)
        Problem.AddFunction([i, j],Constraint)

#Problem.Dump('airplane.cfn')
Problem.NoPreprocessing()
Problem.Solve(showSolutions = 3)
```

2.8 Warehouse location problem

2.8.1 Brief description

A compagny consider opening warehouses at some candidate locations with each of them having a maintenance cost if they are open.

The compagny control a set of given stores and each of them need to take suplies to one of the warehouse, but depending the warehouse chosen, there will be an additionnal cost.

The objective is to choose wich warehouse to open and to divide the store among the open warehouse ion order to minimize the total cost of the store’s cost and the maintenance cost.

2.8.2 CFN model

We create Boolean variables for the warehouses (i.e., open or not) and integer variables for the store, with domain size the number of warehouses to represent to wich warehouse the store will take suplies.

Hard binary constraints represent that a store cannot take suplies from a closed warehouse. Soft unary constraints represent the maintenance cost of the warehouses. Soft unary constraints represent the store's cost regarding wich warehouse to take supplies from.

2.8.3 Data

Original data files can be download from the cost function library warehouses. Their format is described here.

2.8.4 Python model solver

The following code uses the pytoulbar2 module to generate the cost function network and solve it (e.g. "python3 warehouse.py cap44.txt 1" found optimum value equal to 10349757). Other instances are available here in cfn format.

`warehouse.py`

```
import sys
import pytoulbar2

f = open(sys.argv[1], 'r').readlines()

precision = int(sys.argv[2])  # used to convert cost values from float to integer

tokens = []
for l in f:
    tokens += l.split()

pos = 0

def token():
    global pos, tokens
    if pos == len(tokens):
        return None
    s = tokens[pos]
    pos += 1
    return s


N = int(token())  # number of warehouses
M = int(token())  # number of stores

top = 1  # sum of all costs plus one

CostW = []  # maintenance cost of warehouses
Capacity = []  # capacity limit of warehouses (not used)

for i in range(N):
    Capacity.append(token())
    CostW.append(int(float(token()) * 10.**precision))
```

(continues on next page)

(continued from previous page)

```
top += sum(CostW)

Demand = []  # demand for each store
CostS = [[] for i in range(M)]  # supply cost matrix

for j in range(M):
    Demand.append(int(token()))
    for i in range(N):
        CostS[j].append(int(float(token()) * 10.**precision))
    top += sum(CostS[-1])

# create a new empty cost function network
Problem = pytoulbar2.CFN(top)
# add warehouse variables
for i in range(N):
    Problem.AddVariable('w' + str(i), range(2))
# add store variables
for j in range(M):
    Problem.AddVariable('s' + str(j), range(N))
# add maintenance costs
for i in range(N):
    Problem.AddFunction([i], [0, CostW[i]])
# add supply costs for each store
for j in range(M):
    Problem.AddFunction([N+j], CostS[j])
# add channeling constraints between warehouses and stores
for i in range(N):
    for j in range(M):
        Constraint = []
        for a in range(2):
            for b in range(N):
                if a == 0 and b == i:
                    Constraint.append(top)
                else:
                    Constraint.append(0)
        Problem.AddFunction([i, N+j], Constraint)

#Problem.Dump('warehouse.cfn')
Problem.Solve(showSolutions=3)
```

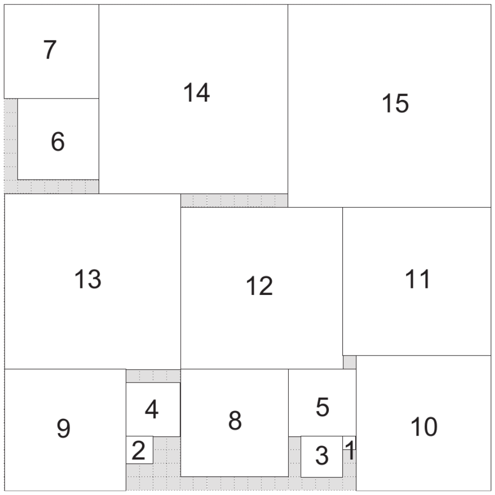
2.9 Square packing problem

2.9.1 Brief description

We have N squares of respectives size 1×1, 2×2,…, NxN. We have to fit them without overlaps into a square of size SxS.

Results up to N=56 are given here.

Optimal solution for 15 squares packed into a 36x36 square (Fig. taken from Takehide Soh)



2.9.2 CFN model

We create an integer variable of domain size (S-i)x(S-i) for each square. The variable represents the position of the top left corner of the square.

The value of a given variable modulo (S-i) gives the x-coordinate, whereas its value divided by (S-i) gives the y-coordinate.

We have hard binary constraints to forbid any overlapping pair of squares.

We make the problem a pure satisfaction problem by fixing the initial upper bound to 1.

2.9.3 Python model

The following code uses the pytoulbar2 library to generate the cost function network and solve it (e.g. "python3 square.py 3 5"). `square.py`

```
import sys
from random import randint, seed
seed(123456789)

import pytoulbar2
try:
        N = int(sys.argv[1])
        S = int(sys.argv[2])
        assert N <= S
```

(continues on next page)

(continued from previous page)

```
except:
        print('Two integers need to be given as arguments: N and S')
        exit()

#pure constraint satisfaction problem
Problem = pytoulbar2.CFN(1)

#create a variable for each square
for i in range(N):
        Problem.AddVariable('sq' + str(i+1), ['(' + str(l) + ',' + str(j) + ')' for l in␣
↪range(S-i) for j in range(S-i)])

#binary hard constraints for overlapping squares
for i in range(N):
        for j in range(i+1,N):
                ListConstraintsOverlaps = []
                for a in [S*k+l for k in range(S-i) for l in range(S-i)]:
                        for b in [S*m+n for m in range(S-j) for n in range(S-j)]:
                                #calculating the coordinates of the squares
                                X_i = a%S
                                X_j = b%S
                                Y_i = a//S
                                Y_j = b//S
                                #calculating if squares are overlapping
                                if X_i >= X_j :
                                        if X_i - X_j < j+1:
                                                if Y_i >= Y_j:
                                                        if Y_i - Y_j < j+1:
                                                                ListConstraintsOverlaps.
↪append(1)
                                                        else:
                                                                ListConstraintsOverlaps.
↪append(0)
                                                else:
                                                        if Y_j - Y_i < i+1:
                                                                ListConstraintsOverlaps.
↪append(1)
                                                        else:
                                                                ListConstraintsOverlaps.
↪append(0)
                                        else:
                                                ListConstraintsOverlaps.append(0)
                                else :
                                        if X_j - X_i < i+1:
                                                if Y_i >= Y_j:
                                                        if Y_i - Y_j < j+1:
                                                                ListConstraintsOverlaps.
↪append(1)
                                                        else:
                                                                ListConstraintsOverlaps.
↪append(0)
                                                else:
```

(continues on next page)

(continued from previous page)

```
                                                        if Y_j - Y_i < i+1:
                                                                ListConstraintsOverlaps.
↪append(1)
                                                        else:
                                                                ListConstraintsOverlaps.
↪append(0)
                                        else:
                                                ListConstraintsOverlaps.append(0)
                Problem.AddFunction(['sq' + str(i+1), 'sq' + str(j+1)],␣
↪ListConstraintsOverlaps)

#Problem.Dump('Square.cfn')
Problem.CFN.timer(300)
res = Problem.Solve(showSolutions=3)
if res:
        for i in range(S):
                row = ''
                for j in range(S):
                        row += ' '
                        for k in range(N-1, -1, -1):
                                if (res[0][k]%(S-k) <= j and j - res[0][k]%(S-k) <= k)␣
↪and (res[0][k]//(S-k) <= i and i - res[0][k]//(S-k) <= k):
                                        row = row[:-1] + chr(65 + k)
                print(row)
else:
        print('No solution found!')
```

2.9.4 C++ program using libtb2.so

The following code uses the C++ toulbar2 library. Compile toulbar2 with "cmake -DLIBTB2=ON -DPYTB2=ON . ; make" and copy the library in your current directory "cp lib/Linux/libtb2.so ." before compiling "g++ -o square square.cpp -Isrc -Llib/Linux -std=c++11 -O3 -DNDEBUG -DBOOST -DLONGDOUBLE_PROB -DLONGLONG_COST -DWCSPFORMATONLY libtb2.so" and running the example (e.g. "./square 15 36").

square.cpp

```

/**
 * Square Packing Problem
 */

// Compile with cmake option -DLIBTB2=ON -DPYTB2=ON to get C++ toulbar2 library lib/
↪Linux/libtb2.so
// Then,
// g++ -o square square.cpp -Isrc -Llib/Linux -std=c++11 -O3 -DNDEBUG -DBOOST -
↪DLONGDOUBLE_PROB -DLONGLONG_COST -DWCSPFORMATONLY libtb2.so

#include "toulbar2lib.hpp"

#include <string.h>
#include <stdio.h>
```

(continues on next page)

(continued from previous page)

```
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char* argv[])
{
    int N = atoi(argv[1]);
    int S = atoi(argv[2]);

    tb2init(); // must be call before setting specific ToulBar2 options and creating a␣
↪model

    ToulBar2::verbose = 0; // change to 0 or higher values to see more trace information

    initCosts(); // last check for compatibility issues between ToulBar2 options and␣
↪Cost data-type

    Cost top = UNIT_COST;
    WeightedCSPSolver* solver = WeightedCSPSolver::makeWeightedCSPSolver(top);

    for (int i=0; i<N; i++) {
        solver->getWCSP()->makeEnumeratedVariable("sq" + to_string(i+1), 0, (S-i)*(S-i) -
↪ 1);
    }

    for (int i=0; i<N; i++) {
        for (int j=i+1; j<N; j++) {
            vector<Cost> costs((S-i)*(S-i)*(S-j)*(S-j), MIN_COST);
                for (int a=0; a<(S-i)*(S-i); a++) {
                    for (int b=0; b<(S-j)*(S-j); b++) {
                    costs[a*(S-j)*(S-j)+b] = ((((a%(S-i)) + i + 1 <= (b%(S-j))) || ((b
↪%(S-j)) + j + 1 <= (a%(S-i))) || ((a/(S-i)) + i + 1 <= (b/(S-j))) || ((b/(S-j)) + j +␣
↪1 <= (a/(S-i))))?MIN_COST:top);
                }
            }
            solver->getWCSP()->postBinaryConstraint(i, j, costs);
        }
    }

    solver->getWCSP()->sortConstraints(); // must be done at the end of the modeling

    tb2checkOptions();
    if (solver->solve()) {
            vector<Value> sol;
            solver->getSolution(sol);
                for (int y=0; y<S; y++) {
                for (int x=0; x<S; x++) {
                    char c = ' ';
                    for (int i=0; i<N; i++) {
                        if (x >= (sol[i]%(S-i)) && x < (sol[i]%(S-i) ) + i + 1 && y >=␣
↪(sol[i]/(S-i)) && y < (sol[i]/(S-i)) + i + 1) {
                            c = 65+i;
                            break;
```

(continues on next page)

(continued from previous page)

```
                        }
                     }
                     cout << c;
                }
                cout << endl;
            }
    } else {
            cout << "No solution found!" << endl;
    }

    delete solver;
    return 0;
}
```

2.10 Square soft packing problem

2.10.1 Brief description

The problem is almost identical to the square packing problem with the difference that we now allow overlaps but we want to minimize them.

2.10.2 CFN model

We reuse the *Square packing problem* model except that binary constraints are replaced by cost functions returning the overlapping size or zero if no overlaps.

To calculate an initial upper bound we simply compute the worst case scenario where N squares of size N*N are all stacks together. The cost of this is N**4, so we will take N**4+1 as the initial upperbound.

2.10.3 Python model

The following code using pytoulbar2 library solves the corresponding cost function network (e.g. "python3 squaresoft.py 10 20").

`squaresoft.py`

```
import sys
from random import randint, seed
seed(123456789)

import pytoulbar2
try:
        N = int(sys.argv[1])
        S = int(sys.argv[2])
        assert N <= S
except:
        print('Two integers need to be given as arguments: N and S')
        exit()
```

(continues on next page)

(continued from previous page)

```
Problem = pytoulbar2.CFN(N**4 + 1)

#create a variable for each square
for i in range(N):
        Problem.AddVariable('sq' + str(i+1), ['(' + str(l) + ',' + str(j) + ')' for l in␣
↪range(S-i) for j in range(S-i)])


#binary soft constraints for overlapping squares
for i in range(N):
        for j in range(i+1,N):
                ListConstraintsOverlaps = []
                for a in [S*k+l for k in range(S-i) for l in range(S-i)]:
                        for b in [S*m+n for m in range(S-j) for n in range(S-j)]:
                                #calculating the coordinates of the squares
                                X_i = a%S
                                X_j = b%S
                                Y_i = a//S
                                Y_j = b//S
                                #calculating if squares are overlapping
                                if X_i >= X_j :
                                        if X_i - X_j < j+1:
                                                if Y_i >= Y_j:
                                                        if Y_i - Y_j < j+1:
                                                                ListConstraintsOverlaps.
↪append(min(j+1-(X_i - X_j),i+1)*min(j+1-(Y_i - Y_j),i+1))
                                                        else:
                                                                ListConstraintsOverlaps.
↪append(0)
                                                else:
                                                        if Y_j - Y_i < i+1:
                                                                ListConstraintsOverlaps.
↪append(min(j+1-(X_i - X_j),i+1)*min(i+1-(Y_j - Y_i),j+1))
                                                        else:
                                                                ListConstraintsOverlaps.
↪append(0)
                                        else:
                                                ListConstraintsOverlaps.append(0)
                                else :
                                        if X_j - X_i < i+1:
                                                if Y_i >= Y_j:
                                                        if Y_i - Y_j < j+1:
                                                                ListConstraintsOverlaps.
↪append(min(i+1-(X_j - X_i),j+1)*min(j+1-(Y_i - Y_j),i+1))
                                                        else:
                                                                ListConstraintsOverlaps.
↪append(0)
                                                else:
                                                        if Y_j - Y_i < i+1:
                                                                ListConstraintsOverlaps.
↪append(min(i+1-(X_j - X_i),j+1)*min(i+1-(Y_j - Y_i),j+1))
```

(continues on next page)

(continued from previous page)

```
                                                        else:
                                                                ListConstraintsOverlaps.
↪append(0)
                                        else:
                                                ListConstraintsOverlaps.append(0)
                Problem.AddFunction(['sq' + str(i+1), 'sq' + str(j+1)],␣
↪ListConstraintsOverlaps)

#Problem.Dump('SquareSoft.cfn')
Problem.CFN.timer(300)
res = Problem.Solve(showSolutions=3)
if res:
        for i in range(S):
                row = ''
                for j in range(S):
                        row += ' '
                        for k in range(N-1, -1, -1):
                                if (res[0][k]%(S-k) <= j and j - res[0][k]%(S-k) <= k)␣
↪and (res[0][k]//(S-k) <= i and i - res[0][k]//(S-k) <= k):
                                        row = row[:-1] + chr(65 + k)
                print(row)
else:
        print('No solution found!')
```

2.10.4 C++ program using libtb2.so

The following code uses the C++ toulbar2 library. Compile toulbar2 with “cmake -DLIBTB2=ON -DPYTB2=ON . ; make” and copy the library in your current directory “cp lib/Linux/libtb2.so .” before compiling “g++ -o squaresoft squaresoft.cpp -I./src -L./lib/Linux -std=c++11 -O3 -DNDEBUG -DBOOST -DLONGDOUBLE_PROB -DLONGLONG_COST -DWCSPFORMATONLY libtb2.so” and running the example (e.g. “./squaresoft 10 20”).

`squaresoft.cpp`

```

/**
 * Square Soft Packing Problem
 */

// Compile with cmake option -DLIBTB2=ON -DPYTB2=ON to get C++ toulbar2 library lib/
↪Linux/libtb2.so
// Then,
// g++ -o squaresoft squaresoft.cpp -Isrc -Llib/Linux -std=c++11 -O3 -DNDEBUG -DBOOST -
↪DLONGDOUBLE_PROB -DLONGLONG_COST -DWCSPFORMATONLY libtb2.so

#include "toulbar2lib.hpp"

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

(continues on next page)

(continued from previous page)

```
int main(int argc, char* argv[])
{
    int N = atoi(argv[1]);
    int S = atoi(argv[2]);

    tb2init(); // must be call before setting specific ToulBar2 options and creating a␣
↪model

    ToulBar2::verbose = 0; // change to 0 or higher values to see more trace information

    initCosts(); // last check for compatibility issues between ToulBar2 options and␣
↪Cost data-type

    Cost top = N*(N*(N-1)*(2*N-1))/6 + 1;
    WeightedCSPSolver* solver = WeightedCSPSolver::makeWeightedCSPSolver(top);

    for (int i=0; i < N; i++) {
        solver->getWCSP()->makeEnumeratedVariable("sq" + to_string(i+1), 0, (S-i)*(S-i) -
↪ 1);
    }

    for (int i=0; i < N; i++) {
        for (int j=i+1; j < N; j++) {
            vector<Cost> costs((S-i)*(S-i)*(S-j)*(S-j), MIN_COST);
                for (int a=0; a < (S-i)*(S-i); a++) {
                    for (int b=0; b < (S-j)*(S-j); b++) {
                    costs[a*(S-j)*(S-j)+b] = ((((a%(S-i)) + i + 1 <= (b%(S-j))) || ((b
↪%(S-j)) + j + 1 <= (a%(S-i))) || ((a/(S-i)) + i + 1 <= (b/(S-j))) || ((b/(S-j)) + j +␣
↪1 <= (a/(S-i))))?MIN_COST:(min((a%(S-i)) + i + 1 - (b%(S-j)), (b%(S-j)) + j + 1 - (a
↪%(S-i))) * min((a/(S-i)) + i + 1 - (b/(S-j)), (b/(S-j)) + j + 1 - (a/(S-i)))));
                }
            }
            solver->getWCSP()->postBinaryConstraint(i, j, costs);
        }
    }

    solver->getWCSP()->sortConstraints(); // must be done at the end of the modeling

    tb2checkOptions();
    if (solver->solve()) {
            vector<Value> sol;
            solver->getSolution(sol);
                for (int y=0; y < S; y++) {
                for (int x=0; x < S; x++) {
                    char c = ' ';
                    for (int i=N-1; i >= 0; i--) {
                        if (x >= (sol[i]%(S-i)) && x < (sol[i]%(S-i) ) + i + 1 && y >=␣
↪(sol[i]/(S-i)) && y < (sol[i]/(S-i)) + i + 1) {
                            if (c != ' ') {
                                c = 97+i;
                            } else {
                                c = 65+i;
```

(continues on next page)

(continued from previous page)

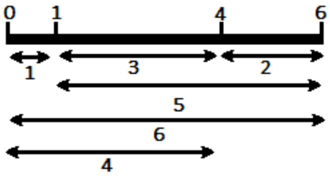
```
                            }
                        }
                     }
                     cout << c;
                }
                cout << endl;
            }
    } else {
            cout << "No solution found!" << endl;
    }

    delete solver;
    return 0;
}
```

2.11 Golomb ruler problem

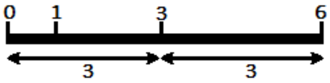
2.11.1 Brief description

A golomb ruler of order N is a set of integer marks 0=a1<a2<a3<a4<. . . .<aN such that each difference between two ak's are unique.

For example, this is a golomb ruler:



We can see that all differences are unique, rather than in this other ruler where 0-3 and 3-6 are both equal to 3.



The size of a golomb ruler is equal to aN, the greatest number of the ruler. The goal is to find the smallest golomb ruler given N.

2.11.2 CFN model

We create N variables, one for each integer mark ak. Because we can not create an AllDifferent constraint with differences of variables, we also create a variable for each difference and create hard ternary constraints in order to fix them equal to the difference. Because we do not use an absolute value when creating the hard constraints, it forces the assignment of ak's variables to follow an increasing order.

Then we create an AllDifferent constraint on all the difference variables and one unary cost function on the last aN variable in order to minimize the size of the ruler. In order to break symmetries, we fix the first mark to be zero.

2.11.3 Python model

The following code using pytoulbar2 library solves the golomb ruler problem with the first argument being the number of marks N (e.g. “python3 golomb.py 8”).

golomb.py

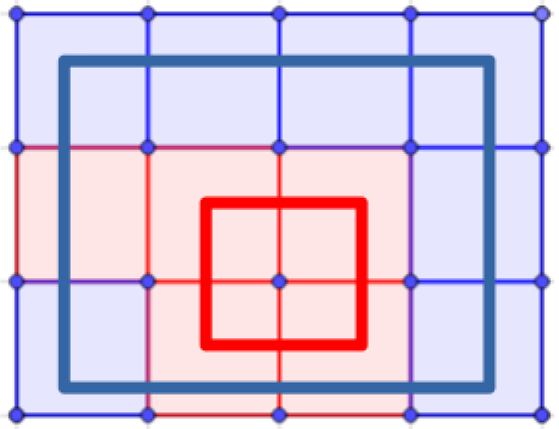
```
import sys
import pytoulbar2

N = int(sys.argv[1])

top = N**2 + 1

Problem = pytoulbar2.CFN(top)

#create a variable for each mark
for i in range(N):
    Problem.AddVariable('X' + str(i), range(N**2))

#ternary constraints to link new variables of difference with the original variables
for i in range(N):
    for j in range(i+1, N):
        Problem.AddVariable('X' + str(j) + '-X' + str(i), range(N**2))
        Constraint = []
        for k in range(N**2):
            for l in range(N**2):
                for m in range(N**2):
                    if l-k == m:
                        Constraint.append(0)
                    else:
                        Constraint.append(top)
        Problem.AddFunction(['X' + str(i), 'X' + str(j), 'X' + str(j) + '-X' + str(i)],␣
↪Constraint)

Problem.AddAllDifferent(['X' + str(j) + '-X' + str(i) for i in range(N) for j in␣
↪range(i+1,N)])

Problem.AddFunction(['X' + str(N-1)], range(N**2))

#fix the first mark to be zero
Problem.AddFunction(['X0'], [0] + [top] * (N**2 - 1))

#Problem.Dump('golomb.cfn')
Problem.CFN.timer(300)
res = Problem.Solve(showSolutions=3)
if res:
    ruler = '0'
    for i in range(1,N):
        ruler += ' '*(res[0][i]-res[0][i-1]-1) + str(res[0][i])
    print('Golomb ruler of size:',int(res[1]))
    print(ruler)
```

2.12 Board coloration problem

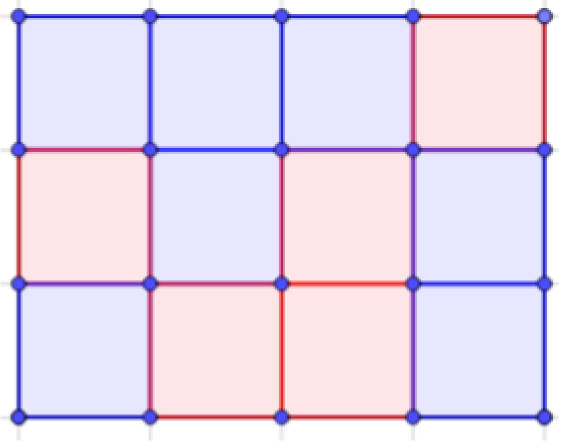
2.12.1 Brief description

Given a rectangular board with dimension n*m, the goal is to color the cells such that any inner rectangle included inside the board doesn't have all its corners colored with the same color. The goal is to minimize the number of colors used.

For examples, this is not a valid solution of the 3*4 problem, because the red and blue rectangles have both their 4 corners having the same color:



On the contrary the following coloration is a valid solution of the 3*4 problem because every inner rectangle inside the board does not have a unique color for its corners:



2.12.2 CFN basic model

We create n*m variables, one for each square of the board, with domain size equal to n*m representing all the possible colors. We also create one variable for the number of colors.

We create hard quaterny constraints for every rectangle inside the board with cost equal to 0 if the 4 variables have different values and a forbidden cost if not.

We then create hard binary constraints between the variable of the number of colors for each cell to fix the variable for the number of colors as an upper bound.

Then we create a soft constraint on the number of colors to minimize it.

2.12.3 Python model

The following code using pytoulbar2 library solves the board coloration problem with the first two arguments being the dimension n and m of the board (e.g. “python3 boardcoloration.py 3 4”).

`boardcoloration.py`

```
import sys
from random import randint, seed
seed(123456789)
import pytoulbar2

try:
    n = int(sys.argv[1])
    m = int(sys.argv[2])
except:
    print('Two integer need to be in arguments: number of rows n, number of columns m')
    exit()

top = n*m + 1

Problem = pytoulbar2.CFN(top)

#create a variable for each cell
for i in range(n):
    for j in range(m):
        Problem.AddVariable('sq(' + str(i) + ',' + str(j) + ')', range(n*m))

#create a variable for the maximum of colors
Problem.AddVariable('max', range(n*m))

#quaterny hard constraints for rectangle with same color angles
#for each cell on the chessboard
for i1 in range(n):
    for i2 in range(m):
        #for every cell on the chessboard that could form a rectangle with the first␣
↪cell as up left corner and this cell as down right corner
        for j1 in range(i1+1, n):
            for j2 in range(i2+1, m):
                Constraint = []
                for k in range(n*m):
                    for l in range(n*m):
                        for o in range(n*m):
                            for p in range(n*m):
                                if k ==l and l == o and o == p:
                                    #if they are all the same color
                                    Constraint.append(top)
                                else:
                                    Constraint.append(0)
                Problem.AddFunction(['sq(' + str(i1) + ',' + str(i2) + ')', 'sq(' +␣
↪str(i1) + ',' + str(j2) + ')', 'sq(' + str(j1) + ',' + str(i2) + ')', 'sq(' + str(j1)␣
```

(continues on next page)

(continued from previous page)

```
↪+ ',' + str(j2) + ')'], Constraint)

#binary hard constraints to fix the variable max as an upper bound
for i in range(n):
    for j in range(m):
        Constraint = []
        for k in range(n*m):
            for l in range(n*m):
                if k>l:
                    #if the color of the square is more than the number of the max
                    Constraint.append(top)
                else:
                    Constraint.append(0)
        Problem.AddFunction(['sq(' + str(i) + ',' + str(j) + ')', 'max'], Constraint)

#minimize the number of colors
Problem.AddFunction(['max'], range(n*m))

#Problem.Dump('boardcoloration.cfn')
Problem.CFN.timer(300)
res = Problem.Solve(showSolutions =3)
if res:
    for i in range(n):
        row = []
        for j in range(m):
            row.append(res[0][m*i+j])
        print(row)
else:
    print('No solutions found')
                                                                70,2-9        Bas
```

2.13 Learning to play the Sudoku

2.13.1 Available

- Presentation
- GitHub code
- Data GitHub code

2.14 Learning car configuration preferences

2.14.1 Brief description

Renault car configuration system: learning user preferences.

2.14.2 Available

- Presentation
- GitHub code
- Data GitHub code

2.15 Visual Sudoku Tutorial

2.15.1 Brief description

A simple case mixing **Deep Learning** and **Graphical models**.

2.15.2 Available

- You can run it directly from your browser as a Jupyter Notebook

2.16 Visual Sudoku Application

2.16.1 Brief description

An automatic Sudoku puzzle **solver** using **OpenCV**, **Deep Learning**, and **Optical Character Recognition** (**OCR**).

2.16.2 Available

Software

Software adapted by Simon de Givry (@ INRAE, 2022) in order to use **toulbar2** solver, from a tutorial by Adrian Rosebrock (@ PyImageSearch, 2022) : GitHub code

As an APK

Based on this software, a ‘Visual Sudoku’ application for Android has been developed to be used from a smartphone.

See the *detailed presentation* (description, source, download…).

The application allows to capture a grid from its own camera (*‘CAMERA’ menu*) or to select a grid among the smartphone existing files (*‘FILE’ menu*), for example files coming from ‘DCIM’, in .jpg or .png formats. The grid image must have been captured in portrait orientation. Once the grid has been chosen, the *‘Solve’* button allows to get the solution.

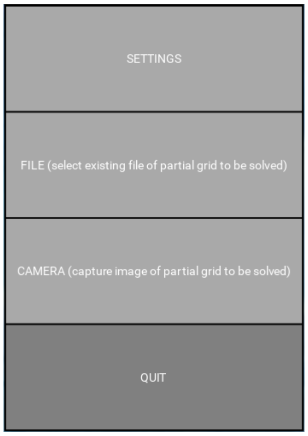
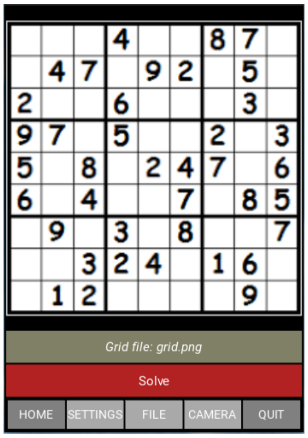
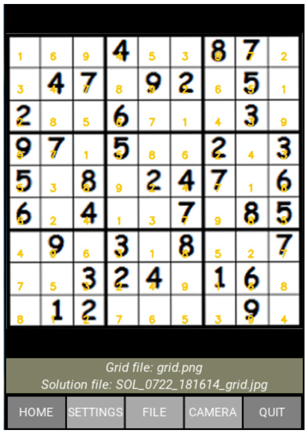
Fig.1

SETTINGS
FILE (select existing file of partial grid to be solved)
CAMERA (capture image of partial grid to be solved)
QUIT

Fig.2


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

Grid file: grid.png

Fig.3


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

Grid file: grid.png
Solution file: SOL_0722_181614_grid.jpg

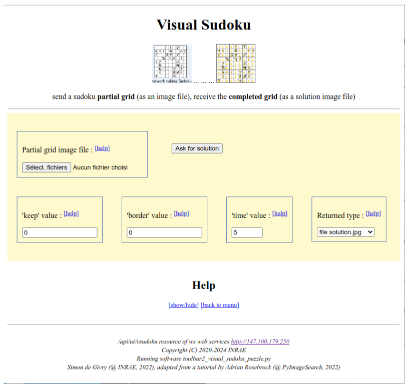
- Fig.1 : Screen of main menu
- Fig.2 : Screen of the grid to be solved
- Fig.3 : Screen of the solution (in yellow) found by the solver

`Examples of some input grids and their solved grids`

As a Web service

The software is available as a web service. The **visual sudoku web service**, hosted by the ws web services (based on HTTP protocol), can be called by many ways : from a **browser** (like above), from any softwares written in a language supporting HTTP protocol (**Python**, **R**, **C++**, **Java**, **Php**…), from command line tools (**cURL**…)…

- Calling the visual sudoku web service from a **browser** :



Visual Sudoku

send a sudoku **partial grid** (as an image file), receive the **completed grid** (as a solution image file)

Partial grid image file : [help]
Sélect. fichiers Aucun fichier choisi

'keep' value : [help] 0

'border' value : [help] 0

'time' value : [help] 5

Returned type : [help] file solution.jpg

Help

[show/hide] [back to menu]

/api/ui/vsudoku resource of ws web services http://147.100.179.250
Copyright (C) 2020-2024 INRAE
Running software toulbar2_visual_sudoku_puzzle.py
Simon de Givry (@ INRAE, 2022), adapted from a tutorial by Adrian Rosebrock (@ PyImageSearch, 2022)

api/ui/vsudoku

- Example of calling the visual sudoku web service from a **terminal** by **cURL** :

  Commands (*replace mygridfilename.jpg by your own image file name*) :

```
curl --output mysolutionfilename.jpg -F 'file=@mygridfilename.jpg' -F 'keep=40' -F
↪'border=15' http://147.100.179.250/api/tool/vsudoku
```

- The ‘Visual Sudoku’ APK calls the visual sudoku web service.

2.17 Visual Sudoku App for Android

2.17.1 A visual sudoku solver based on cost function networks

This application solves the sudoku problem from a smartphone by reading the grid using its camera. The cost function network solver toulbar2 is used to deal with the uncertainty on the digit recognition produced by the neural network. This uncertainty, combined with the sudoku logical rules, makes it possible to correct perceptual errors. It is particularly useful in the case of hand-written digits or poor image quality. It is also possible to solve a partially filled-in grid with printed and hand-written digits. The solver will always suggest a valid solution that best adapts to the retrieved digit information. It will naturally detect (a small number of) errors in a partially filled-in grid and could be used later as a diagnosis tool (future work). This software demonstration emphasizes the tight relation between constraint programming, computer vision, and deep learning.

We used the open-source C++ solver toulbar2 in order to find the maximum a posteriori solution of a constrained probabilistic graphical model. With its dedicated numerical (soft) local consistency bounds, toulbar2 outperforms traditional CP solvers on this problem. Grid perception and cell extraction are performed by the computer vision library OpenCV. Digit recognition is done by **Keras** and **TensorFlow**. The current android application is written in Python using the Kivy framework. It is inspired from a tutorial by Adrian Rosebrock. It uses the ws RESTful web services in order to run the solver.

See also : *Visual Sudoku Application*.

2.17.2 Source Code

GitHub code

2.17.3 Download and Install

To install the ‘Visual Sudoku’ application on smartphone :

1) **Download** the **visualsudoku-release.apk** APK file from Github repository :

   https://github.com/toulbar2/visualsudoku/releases/latest

2) Click on the downloaded **visualsudoku-release.apk** APK file to ask for **installation** (*you have to accept to ‘install anyway’ from unknown developer*).

3) In your parameter settings for the app, give permissions to the ‘Visual Sudoku’ application (smartphone menu ‘Parameters’ > ‘Applications’ > ‘Visual Sudoku’) : allow camera (required to capture grids), files and multimedia contents (required to save images as files). Re-run the app.

Warnings :

- The application may fail at first start and you may have to launch it twice.
- While setting up successfully, the application should have created itself the required ‘VisualSudoku’ folder (under the smartphone ‘Internal storage’ folder) but if not, you will have to create it by yourself manually.
- Since the application calls a web service, an internet connection is required.

2.17.4 Description

The ‘SETTINGS’ menu allows to save grids or solutions as image files (‘savinginputfile’, ‘savingoutputfile’ parameters) and to access to some ‘expert’ parameters in order to enhance the resolution process (‘keep’, ‘border’, ‘time’ parameters).

The application allows to capture a grid from its own camera (*‘CAMERA’ menu*) or to select a grid among the smartphone existing files (*‘FILE’ menu*), for example files coming from ‘DCIM’, in .jpg or .png formats. The grid image must have been captured in portrait orientation. Once the grid has been chosen, the *‘Solve’* button allows to get the solution.

Fig.1

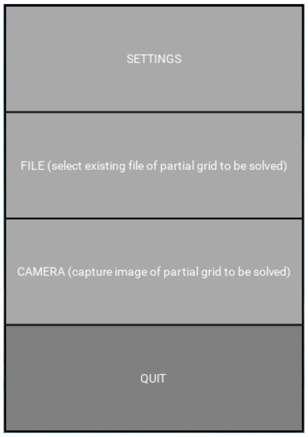
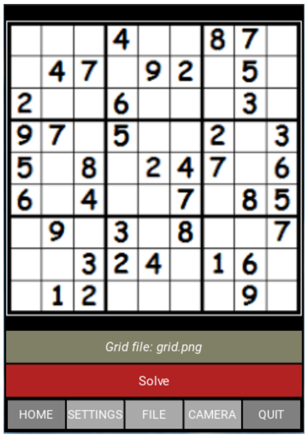

SETTINGS
FILE (select existing file of partial grid to be solved)
CAMERA (capture image of partial grid to be solved)
QUIT

Fig.2

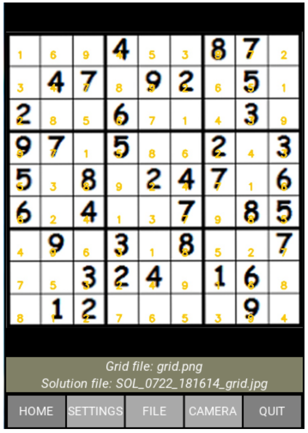


			4			8	7	
	4	7		9	2		5	
2			6				3	
9	7		5			2		3
5		8		2	4	7		6
6		4			7		8	5
	9		3		8			7
		3	2	4		1	6	
	1	2					9	

Grid file: grid.png

Solve
HOME
SETTINGS
FILE
CAMERA
QUIT

Fig.3



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

Grid file: grid.png
Solution file: SOL_0722_181614_grid.jpg

HOME
SETTINGS
FILE
CAMERA
QUIT

- Fig.1 : Screen of main menu
- Fig.2 : Screen of the grid to be solved
- Fig.3 : Screen of the solution (in yellow) found by the solver

`Examples of some input grids and their solved grids`

2.18 A sudoku code

2.18.1 Brief description

A Sudoku code returning a sudoku partial grid (sudoku problem) and the corresponding completed grid (sudoku solution), such as `partial and completed grids`.

The verbose version, that further gives a detailed description of what the program does, could be useful as tutorial example. Example : `partial and completed grids with explanations`.

2.18.2 Available

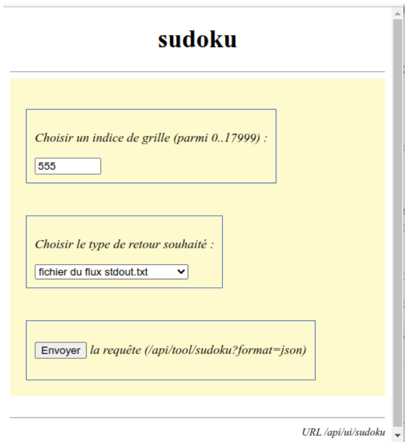
Available as a web service.

You can run the software directly from your **browser** as a web service :

Grids information is returned into the output stream. The **returned_type** parameter of the web service allows to choose how to receive it :

- *returned_type=stdout.txt* : to get the output stream as a .txt file.
- *returned_type=run.zip* : to get the .zip run folder containing the output stream __WS__stdout.txt (+ *the error stream __WS__stderr.txt that may be useful to investigate*).

Web service to get one sudoku grids (both partial and completed) :



sudoku

Choisir un indice de grille (parmi 0..17999) : 555

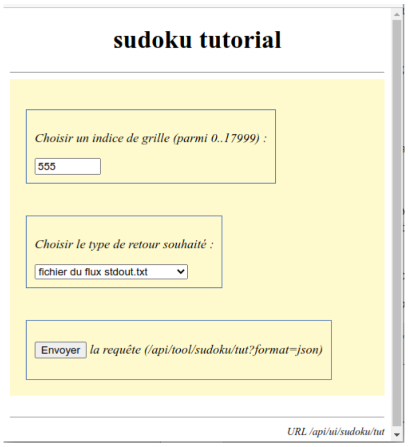
Choisir le type de retour souhaité : fichier du flux stdout.txt

Envoyer *la requête (/api/tool/sudoku?format=json)*

URL /api/ui/sudoku

api/ui/sudoku

Web service to further get a detailed description of what the program does (verbose version) :



sudoku tutorial

Choisir un indice de grille (parmi 0..17999) :

555

Choisir le type de retour souhaité :

fichier du flux stdout.txt

la requête (/api/tool/sudoku/tut?format=json)

URL /api/ui/sudoku/tut

api/ui/sudoku/tut (verbose version)

Note: The **sudoku web services**, hosted by the ws web services (based on HTTP protocol), can be called by many other ways : from a **browser** (like above), from any softwares written in a language supporting HTTP protocol (**Python**, **R**, **C++**, **Java**, **Php**…), from command line tools (**cURL**…)…

Example of calling the sudoku web services from a terminal by cURL :

- Commands (*replace indice value by any value in 1…17999*) :

```
curl --output mygrids.txt -F 'indice=778' -F 'returned_type=stdout.txt' http://147.
↪100.179.250/api/tool/sudoku

curl --output myrun.zip -F 'indice=778' -F 'returned_type=run.zip' http://147.100.
↪179.250/api/tool/sudoku

# verbose version

curl --output mygrids_details.txt -F 'indice=778' -F 'returned_type=stdout.txt'␣
↪http://147.100.179.250/api/tool/sudoku/tut

curl --output myrun_details.zip -F 'indice=778' -F 'returned_type=run.zip' http://
↪147.100.179.250/api/tool/sudoku/tut
```

- Responses corresponding with the requests above :
  - `mygrids.txt`
  - __WS__stdout.txt into myrun.zip has the same content as `mygrids.txt`
  - `mygrids_details.txt` (*__WS__stdout.txt into myrun_details.zip has the same content*)
  - __WS__stdout.txt into myrun_details.zip has the same content as `mygrids_details.txt`