
Use cases

Release 1.0.0

INRAE

Jun 17, 2022

CONTENTS

1	Use cases	1
2	List of all examples	2

USE CASES

Here are several `toulbar2` use cases, where `toulbar2` has been used in order to resolve different problems. According to cases, they can be used to overview, learn, use `toulbar2`... They may contain source code, explanations, possibility to run yourself...

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

`toulbar2` and Deep Learning :

- *[Visual Sudoku Tutorial](#)*
- *[Visual Sudoku Application](#)*

Some applications based on `toulbar2` :

- **Mendelsoft** : [Mendelsoft](#) detects Mendelian errors in complex pedigree [[Sanchez et al, Constraints 2008](#)].
- **Pompd** : [POsitive Multistate Protein Design](#), [[Vucini et al Bioinformatics 2020](#)]
- *[Visual Sudoku Application](#)*
- *[Sudoku Application](#)*

LIST OF ALL EXAMPLES

2.1 Weighted n-queen problem

2.1.1 Brief description

The problem consists in assigning N queens on a $N \times N$ chessboard with random weights in $(1..N)$ associated to every cell such that each queen does not attack another queen and the sum of the weights 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 $k = N \cdot 2 + 1$. Two other cliques of binary constraints are used to express that two queens do not attack each other on a lower/upper diagonal.

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)

Cost	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)

Cost	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,Q3) constraint (3rd clique)

Cost	Q0	Q3
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

==> 1st 1st clique : {source: ["Q0", "Q1"], "costs": [17, 0, 0, 0, 17, 0, 0, 0, 17, 0, 0, 0, 17, 0, 0, 0, 17]}
 ==> 1st 2nd clique : {source: ["Q0", "Q2"], "costs": [0, 0, 0, 0, 0, 0, 0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 0, 17]}
 ==> 1st 3rd clique : {source: ["Q0", "Q3"], "costs": [0, 0, 17, 0, 0, 0, 0, 17, 0, 0, 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 solver

The following code using python3 interpreter will solve the weighted 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 queen 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')
res = Problem.Solve(showSolutions = 3)
if res:
    for i in range(N):
        row = [' ' for j in range(N)]
        row[res[0][i]] = 'X'
        print(row)

```

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 weight in $(1 \dots N)$ is associated to every domain value.

The objective is to find a complete assignment where the sum of the weights associated to the selected values for the cells is minimized.

2.2.2 CFN model

We create NxN variables, one for all cells, with domain size N. A hard AllDifferent global cost function is used to model a permutation for every row and every column.

Random weights are generated for every cell and domain value.

The worst solution possible is when every cell is associated with a weight of N, so the maximum cost of a solution is N^2 , so forbidden assignments have cost $k=N^2+1$.

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}},

```

(continues on next page)

(continued from previous page)

```

{scope: ["X1_0", "X1_1", "X1_2", "X1_3"], "type:" salldiff, "params": {"metric": "var
↪", "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, weights 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 solver

The following code using python3 interpreter will solve the weighted latin square problem with the first argument being the dimension N of the chessboard and with randomized cost for each cell(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 constraints all different with variable on the same row
    Problem.AddAllDifferent(['Cell(' + str(i) + ',' + str(j) + ')' for j in
↪range(N)])

    #Create a constraints all different with variable on the same column
    Problem.AddAllDifferent(['Cell(' + str(j) + ',' + str(i) + ')' for j in range(N)])

# 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')
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("Optimum:", 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 occurs. Domain

- (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: $a_1*nc_1 + \dots + a_4*nc_4 + b_1*nv_1 + \dots + b_4*nv_4$

where nc_i is the number of violated constraint with Weight Index i and nvi is the number of modified variables with Mobility 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 will be defined as 1 plus the maximum cost if all the soft constraints were violated.

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 solver

The following code using python3 interpreter will solve the associated cost function network and need 4 argument: 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 = {}
        self.ctr = list()
        self.cost = {}
        self.nba = {}
```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

#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 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 :

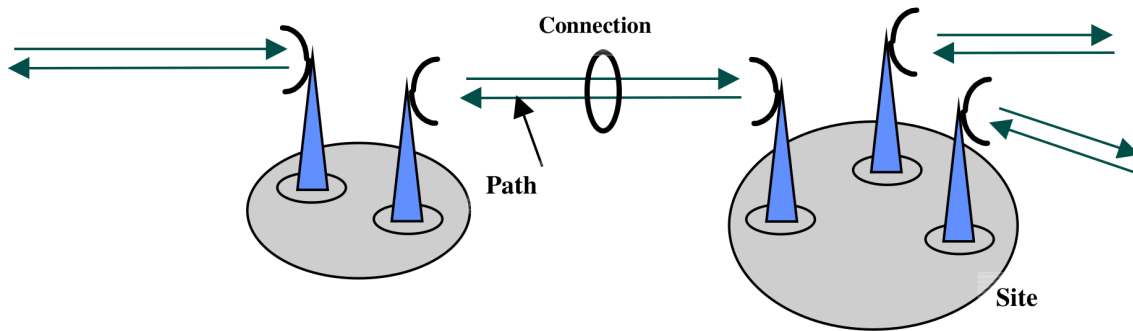
- (I) two paths must use equal or different frequencies ($f_i=f_j$ or $f_i<>f_j$),
- (II) the absolute difference between two frequencies should exactly be equal or different to a given number e ($|f_i-f_j|=e$ or $|f_i-f_j|<>e$),
- (III) two paths must use equal or different polarizations ($p_i=p_j$ 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) ($|f_i - f_j| \geq g_l$ if $p_i = p_j$ else $|f_i - f_j| \geq d_l$), 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 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: $10 * k * \text{nbsoft}^2 + 10 * \text{nbsoft} * V(k) + V(k-1) + V(k-2) + \dots + V_0$

where nbsoft is the number of soft constraints of the problem and k the relaxation level and $V(i)$ the number of violated constraints 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`, so in order to give tuple to `toulbar2` we will need to first transform them into string.

We will use hard binary constraints to modelize (I) to (III) type constraints.

In order to modelize (IV) type constraints we will first take in argument the level of relaxation i , and we will create 11 constraints, one for each relaxation level from 0 to 10. The $i-2$ first constraints will be soft and with cost of 1. The $i-1$ constraint will be of cost $10 * \text{nbsoft}$ (number of soft constraints) in order to first maximize the number of $i-1$ constraints respected and then the other soft constraints. The constraints i to 11 will all be hard binary constraints.

The initial upper bound of the problem will be $10 * (k+1) * \text{nbsoft}^2 + 1$.

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 solver

The following code using python3 interpreter will solve the corresponding cost function network (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 | toulbar2 -stdin=cfn -s=3 | awk -f ./sol2fapp.awk - exemple1”).

Note: Notice that the cost of this model will be $10 \cdot \text{nbsoft} \cdot V(k) + V(k-1) + V(k-2) + \dots + V(0)$, and will not take into account $10 \cdot k \cdot \text{nbsoft} \cdot 2$. So in order to obtain the same results as those said in Data, you will need to add $10 \cdot k \cdot \text{nbsoft} \cdot 2$.

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
        self.top = 1
        self.cst = 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)))
```

(continues on next page)

(continued from previous page)

```

        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

        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.cst = 10*k*self.nbsoft**2
        self.top += self.cst
        self.top += 10*self.nbsoft**2

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 foe 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:
                Constraint.append(0)
    else:
        Constraint.append(data.top)

```

(continues on next page)

(continued from previous page)

```

        if operand == 'E':
            if p2 == p1:
                Constraint.append(0)
            else:
                Constraint.append(data.top)
        else:
            if p2 != p1:
                Constraint.append(0)
            else:
                Constraint.append(data.top)
    Problem.AddFunction(["X" + str(route1), "X" + str(route2)], Constraint)

#soft equivalent binary constraints
    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 greater or equal to binary constraints
    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)

#Problem.Dump('Fapp.cfn')
    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 only) in complex pedigrees.

The input is a pedigree data with partial observed genotyping data at a single locus, we will 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 allele between the genotyping data and the solution. Each distinction from the genotyping data will be of cost 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 will create one unary soft constraint with 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 solver

The following code using python3 interpreter will solve 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)
```

(continues on next page)

(continued from previous page)

```

        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)
            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:
            for m1 in data.ListAlle:
                for m2 in data.ListAlle:
                    if m1 <= m2:
                        for a1 in data.ListAlle:
                            for a2 in data.ListAlle:
                                if a1 <= a2:
                                    if (a1 in (p1,
    ↪p2) and a2 in (m1,m2)) or (a2 in (p1,p2) and a1 in (m1,m2)) :
                                        ListConstraintsMendellaw
    ↪append(0)
                                else :
                                    ↵

    ↪ ListConstraintsMendellaw.append(top)

for i in data.id:
    #ternary constraints representing mendel's laws

```

(continues on next page)

(continued from previous page)

```

    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)], ListConstraintsMendellLaw)

    #unary constraints linked to the observations
    if data.allelesId[i][0] != 0 and data.allelesId[i][1] != 0:
        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')
res = Problem.Solve(showSolutions =3)

if res:
    print(f'There is {int(res[1])} difference(s) between the solution and the_
↪observation.')
else:
    print('No solution found')

```

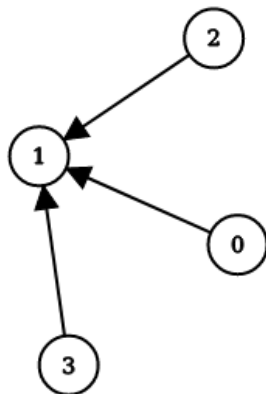
2.6 Block modeling problem

2.6.1 Brief description

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

The problem is to divide a given graph G into k clusters such that the interactions between clusters can be summarized by a $k \times k$ 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 between the nodes in G .

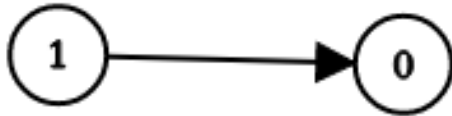
For example, the folowing graph is composed of 4 nodes:



and is corresponding to the following matrix:

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

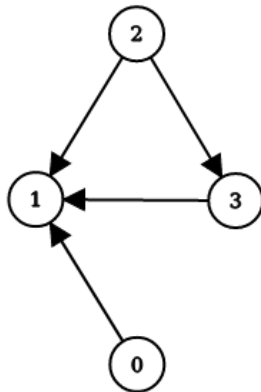
But he can be perfectly clusterized into the following graph by clustering together the node 0, 2 and 3:



and this graph is corresponding to the following matrix:

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

But if we decide to cluster the following graph into the same graph as above, the edges (2, 3) will be 'lost' in the process and the cost of the solution above for the following graph 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

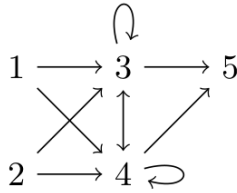
We create N variables, one for every node of the graph, with domain size k . We add $k*k$ Boolean variables for representing M .

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) in G . In order to break symmetries, we constrain the first $k-1$ node variables to be assigned to cluster index less than or equal to their index

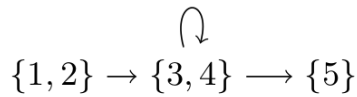
2.6.3 Data

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)

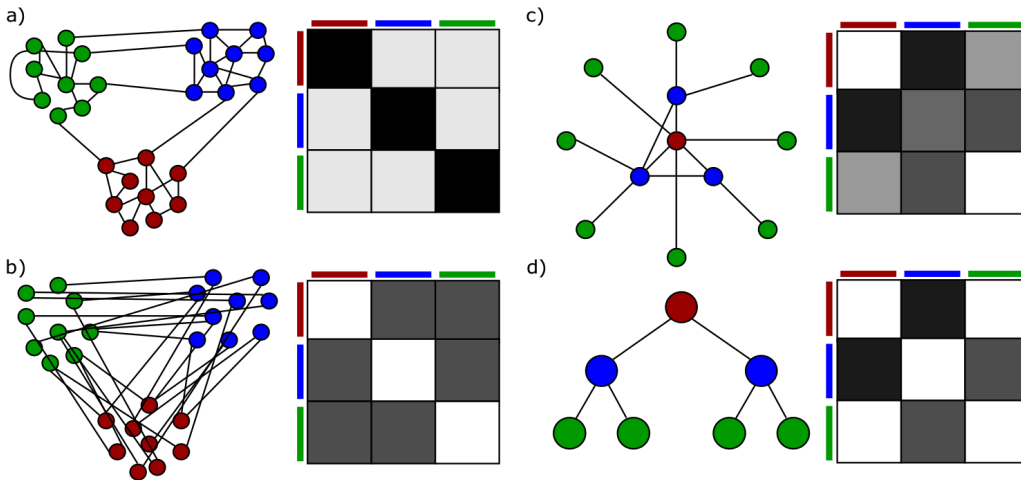


$$G = \left(\begin{array}{cc|cc|c} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 \end{array} \right)$$



$$M = \left(\begin{array}{ccc} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{array} \right)$$

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 solver

The following code using python3 interpreter will solve the corresponding cost function network (e.g. “python3 BlockModel.py simple.mat 3”).

BlockModel.py

```
import sys
import pytoulbar2

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

(continues on next page)

(continued from previous page)

```

N = len(Lines)
Matrix = [[int(e) for e in l.split(' ')] for l in Lines]
Top = 1 + N*N

K = int(sys.argv[2])

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 variable for each coeficient of the matrix
for u in range(K):
    for v in range(K):
        Problem.AddVariable("M_" + str(u) + "_" + str(v), range(2))

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

#general case for cost function
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 Matrix[i][j] != m):
                                    ListCost.
↳ append(1)
                                else:
                                    ListCost.
↳ append(0)
                    Problem.AddFunction(["M_" + str(u) + "_" +
↳ str(v), Var[i], Var[j]],ListCost)

# self-loops
for u in range(K):

```

(continues on next page)

(continued from previous page)

```

    for i in range(N):
        ListCost = []
        for m in range(2):
            for k in range(K):
                if (u == k and Matrix[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
↳ cluster less than or equal to their index
for l in range(K-1):
    Constraints = []
    for k in range(K):
        if k > l:
            Constraints.append(Top)
        else:
            Constraints.append(0)
    Problem.AddFunction([Var[l]], Constraints)

#Problem.Dump(sys.argv[1].replace('.mat','.cfn'))
res = Problem.Solve(showSolutions = 3)

if res:
    for i in range(K):
        Line = []
        for j in range(K):
            Line.append(res[0][i*K+j])
        print(Line)
    for i in range(N):
        print("The node number " + str(i+1) + " 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 additional constraints to enforce that separation between those planes is 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 = pyoulbar2.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 which warehouse the store will take supplies.

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

2.8.3 Data

Original data files can be downloaded 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 `cfm` 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):
```

(continues on next page)

(continued from previous page)

```

        Capacity.append(token())
        CostW.append(int(float(token()) * 10.**precision))

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)

```

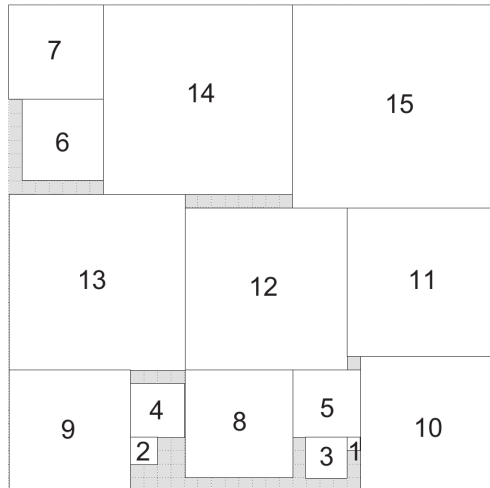
2.9 Square packing problem

2.9.1 Brief description

We have N squares of respective size $1 \times 1, 2 \times 2, \dots, N \times N$. We have to fit them without overlaps into a square of size $S \times S$.

Results up to $N=56$ are given [here](#).

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



2.9.2 CFN model

We create an integer variable of domain size $(S-i) \times (S-i)$ for each square. The variable will represent 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 S . The initial upper bound is 1.

2.9.3 Python model solver

The following code uses the `pytoulbar2` module 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 integer need to be in argument: N then S')
    exit()

top = N**4 + 1

Problem = pyoulbar2.CFN(top)
#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 sqaures
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 coodonate 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)

```

(continues on next page)

(continued from previous page)

```

else:
    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('SquareSoft.cfn')
res = Problem.Solve(showSolutions =3)

#visual print of the soluitions
if res:
    for i in range(S):
        row = []
        for j in range(S):
            row.append(' 0')
            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[-1] = f'{k+1:2}'
            print(row)
else:
    print('No solutions found')

```

2.9.4 C++ program using libtb2.so

The following code uses the C++ toulbar2 library libtb2.so. 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 -DWCSFFORMATONLY 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 -DWCSFFORMATONLY libtb2.so

#include "toulbar2lib.hpp"

```

(continues on next page)

(continued from previous page)

```

#include <string.h>
#include <stdio.h>
#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) {

```

(continues on next page)

(continued from previous page)

```

        c = 65+i;
        break;
    }
    }
    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 it.

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 will simply say that the worst case scenario can't be worse if we had $N \times N$ square that are all stacks together. The cost of this is N^4 , so we will take N^4 as the initial upperbound.

2.10.3 Python model solver

The following code using python3 interpreter will solve 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:

```

(continues on next page)

(continued from previous page)

```

    print('Two integer need to be in argument: N then S')
    exit()

top = N**4 + 1

Problem = pyoulbar2.CFN(top)
#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+1 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 coodonate 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:

```

(continues on next page)

(continued from previous page)

```

                                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))
                                else:
                                    ListConstraintsOverlaps.
→append(0)

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

#Problem.Dump('SquareSoft.cfn')
res = Problem.Solve(showSolutions =3)

#visual print of the soluitions
if res:
    for i in range(S):
        row = []
        for j in range(S):
            row.append(' 0')
            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[-1] = f'{k+1:2}'
            print(row)
else:
    print('No solutions found')

```

2.10.4 C++ program using libtb2.so

The following code uses the C++ toulbar2 library libtb2.so. 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 -Isrc -Llib/Linux -std=c++11 -O3 -DNDEBUG -DBOOST -DLONGDOUBLE_PROB -DLONGLONG_COST -DWCSPPFORMATONLY 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 -DWCSPPFORMATONLY libtb2.so

#include "toulbar2lib.hpp"

#include <string.h>

```

(continues on next page)

(continued from previous page)

```

#include <stdio.h>
#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 = 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) {

```

(continues on next page)

(continued from previous page)

```

        if (c != ' ') {
            c = 97+i;
        } else {
            c = 65+i;
        }
    }
    cout << c;
}
cout << endl;
}
} else {
    cout << "No solution found!" << endl;
}

delete solver;
return 0;
}

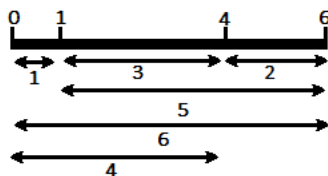
```

2.11 Golomb ruler problem

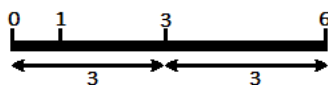
2.11.1 Brief description

A golomb ruler is a set of integer $0=a_1 < a_2 < a_3 < a_4 < \dots < a_n$ such that each differences between two a_k are unique.

For examples, this is a golomb ruler:



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



The size of a golomb ruler is equal to a_n , the greatest number of the ruler. The goal is to find the smallest golomb ruler given the number n of integer.

2.11.2 CFN model

We create N variables, one for each integer a_k . Because we cannot create an AllDifferent function with differences of variable, we also create a variable for each difference and create hard ternary function in order to fix them equal to the difference.

Because we do not put absolute value when creating the hard function, it will force the variable a_k by increasing order.

Then we create one variable AllDifferent with all the differences and one soft function on the an variable in order to minimize the size of the ruler.

2.11.3 Python model solver

The following code using python3 interpreter will solve the golomb ruler problem with the first argument being the number of integer 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 square
for i in range(N):
    Problem.AddVariable('X' + str(i), range(N**2))

#binary soft constraints for overlapping squares
for i in range(N):
    for j in range(i+1, N):
        Problem.AddVariable('X' + str(i) + '-X' + str(j), 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(i) + '-X' +
↪str(j)], Constraint)

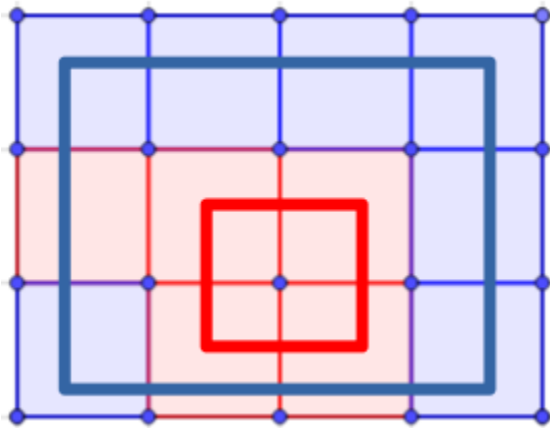
Problem.AddAllDifferent(['X' + str(i) + '-X' + str(j) for i in range(N) for j in_
↪range(i+1,N)])
Problem.AddFunction(['X' + str(N-1)], range(N**2))
#Problem.Dump('SquareSoft.cfn')
res = Problem.Solve(showSolutions =3)
```

2.12 Board Coloration problem

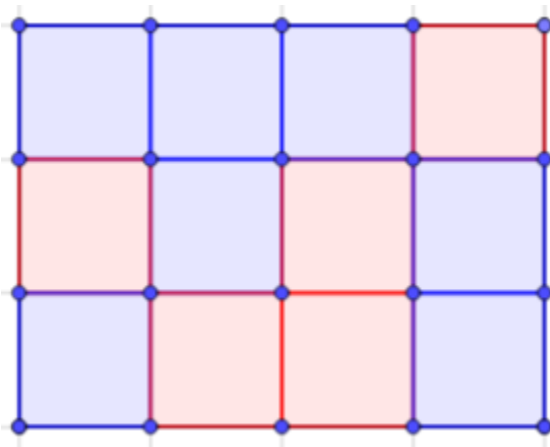
2.12.1 Brief description

Given a squared board with dimension $n \times m$, the goal is to colored the squares so any given rectangle included inside the board doesn't have each corner colored with the same color.

For examples, this is not a valid solution of the 3×4 problem, because the red and blue square have both their 4 corner with the same color:



On the contrary the following coloration is a valid solution of the 3×4 problem because every given rectangle inside the board don't have matching color corner:



2.12.2 CFN model

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

We create hard quaternary constraints for every rectangle inside the board with cost equal to 0 if the 4 variable have different value and top if not.

We then create hard binary constraints with the variable of the number of color for each square to fix the variable for the number of color as an upper bound.

Then we create a soft constraints on the number of color to minimize it.

2.12.3 Python model solver

The following code using python3 interpreter will solve the board coloration problem with the first two arguments being the dimension n and m of the board (e.g. “python3 BoardColoration.py 8”).

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 argument: n then m')
    exit()

top = n*m + 1

Problem = pytoulbar2.CFN(top)
#Create a variable for each square
for i in range(n):
    for j in range(m):
        Problem.AddVariable('sq(' + str(i) + ',' + str(j) + ')', range(n*m))

#create a variable for maximum of color
Problem.AddVariable('max', range(n*m))
#quaterny hard constraints for rectangle with same color angles
#for each square on the chessboard
for i1 in range(n):
    for i2 in range(m):
        #for every square on the chessboard that could form a rectangle with
        ↪ first square as up left corner and this square as down right corner
        for j1 in range(i1+1, n):
            for j2 in range(i2+1, m):
                Constraints = []
                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
                                    Constraints.
                                ↪ append(top)
                                else:
                                    Constraints.
                                ↪ append(0)
                Problem.AddFunction(['sq(' + str(i1) + ',' + str(i2) + ')',
                ↪ ', 'sq(' + str(i1) + ',' + str(j2) + ')', 'sq(' + str(j1) + ',' + str(i2) + ')', 'sq(' +
                ↪ + str(j1) + ',' + str(j2) + ')'], Constraints)
```

(continues on next page)

(continued from previous page)

```

#binary hard constraints to fix the variable max as an upper bound
for i in range(n):
    for j in range(m):
        Constraints = []
        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
                    Constraints.append(top)
                else:
                    Constraints.append(0)
        Problem.AddFunction(['sq(' + str(i) + ',' + str(j) + ')', 'max'],↪
↪Constraints)

#minimize the number of colors
Problem.AddFunction(['max'], range(n*m))
#Problem.Dump('BoardColoration.cfn')
res = Problem.Solve(showSolutions =3)

#visual print of the soltuions
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')

```

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

[GitHub code](#) 

- **As a web service :**

– You can run this software directly from your browser as a [web service](#)



- Other ways to call the 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...).

Example from a terminal by cURL (*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
```

2.17 Sudoku Application

2.17.1 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.

2.17.2 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) : [api/ui/sudoku](#)
- web service to further get a detailed description of what the program does : [api/ui/sudoku/tut](#) (verbose version)



version)

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...).

For example, 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