toulbar2 Documentation

Release 1.0.0

INRAE

Dec 13, 2024

CONTENTS

toulbar2 is an open-source C++ solver for cost function networks. It solves various combinatorial optimization problems.

The constraints and objective function are factorized in local functions on discrete variables. Each function returns a cost (a finite positive integer) for any assignment of its variables. Constraints are represented as functions with costs in $\{0, \infty\}$ where ∞ is a large integer representing forbidden assignments. toulbar2 looks for a non-forbidden assignment of all variables that minimizes the sum of all functions.

Its engine uses a hybrid best-first branch-and-bound algorithm exploiting soft arc consistencies. It incorporates a parallel variable neighborhood search method for better performance. See *Publications*.

toulbar2 won several medals in competitions on Max-CSP/COP (CPAI08, XCSP3 2022, 2023, and 2024) and probabilistic graphical models (UAI 2008, 2010, 2014, 2022 MAP task).

toulbar2 is now also able to collaborate with ML code that can learn an additive graphical model (with constraints) from data (see example at cfn-learn).

CHAPTER

ONE

AUTHORS

toulbar2 was originally developed by Toulouse (INRAE MIAT) and Barcelona (UPC, IIIA-CSIC) teams, hence the solver's name.

Additional contributions by:

- Caen University, France (GREYC) and University of Oran, Algeria for (parallel) variable neighborhood search methods
- The Chinese University of Hong Kong and Caen University, France (GREYC) for global cost functions
- Marseille University, France (LSIS) for tree decomposition heuristics
- Ecole des Ponts ParisTech, France (CERMICS/LIGM) for INCOP local search solver
- University College Cork, Ireland (Insight) for a Python interface in Numberjack and a portfolio dedicated to UAI graphical models Proteus
- Artois University, France (CRIL) for an XCSP 2.1 format reader of CSP and WCSP instances
- Université de Toulouse I Capitole (IRIT) and Université du Littoral Côte d'Opale, France (LISIC) for PILS local search solver

CHAPTER

TWO

CITATIONS

- Multi-Language Evaluation of Exact Solvers in Graphical Model Discrete Optimization

  Barry Hurley, Barry O'Sullivan, David Allouche, George Katsirelos, Thomas Schiex, Matthias Zytnicki, Simon de Givry

  Constraints, 21(3):413-434, 2016

- Tractability-preserving Transformations of Global Cost Functions

  David Allouche, Christian Bessiere, Patrice Boizumault, Simon de Givry, Patricia Gutierrez, Jimmy HM. Lee, Ka Lun Leung, Samir Loudni, Jean-Philippe Métivier, Thomas Schiex, Yi Wu

  Artificial Intelligence, 238:166-189, 2016

- Soft arc consistency revisited

  Martin Cooper, Simon de Givry, Marti Sanchez, Thomas Schiex, Matthias Zytnicki, and Thomas Werner

  Artificial Intelligence, 174(7-8):449-478, 2010

CHAPTER

THREE

ACKNOWLEDGMENTS

toulbar2 has been partly funded by the French *Agence Nationale de la Recherche* (projects STAL-DEC-OPT from 2006 to 2008, ANR-10-BLA-0214 Ficolofo from 2011 to 2014, and ANR-16-CE40-0028 DemoGraph from 2017 to 2021, PIA3 ANITI ANR-19-P3IA-0004 from 2019 to 2024) and a PHC PROCORE project number 28680VH (from 2013 to 2015). It is currently supported by ANITI2 (2024-2031) and ANR project GMLaAS (2025-2029).

CHAPTER FOUR

LICENSE

```
MIT License

Copyright (c) 2022 toulbar2 team

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

CHAPTER

FIVE

DOWNLOADS

5.1 Open-source code

- toulbar2 on GitHub

5.2 Packages

- to install **toulbar2** using the package manager in Debian and Debian derived Linux distributions (Ubuntu, Mint,...):

```
apt install toulbar2
```

5.3 Binaries

- **Latest release toulbar2 binaries**

  Linux 64bit | MacOs 64bit | Windows 64bit

5.4 Python package

- **pytoulbar2** module for Linux and MacOS on PyPI

5.5 Docker images

- In Toulbar2 Packages :
  - **toulbar2** : Docker image containing toulbar2 and its pytoulbar2 Python API *(installed from sources with cmake options -DPYTB2=ON and -DXML=ON)*. Install from the command line:

    ```
    docker pull ghcr.io/toulbar2/toulbar2/toulbar2:master
    ```

  - **pytoulbar2** : Docker image containing pytoulbar2 the Python API of toulbar2 *(installed with python3 -m pip install pytoulbar2)*. Install from the command line:

```
docker pull ghcr.io/toulbar2/toulbar2/pytoulbar2:master
```

CHAPTER

SIX

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*
- *Bicriteria 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*
- tuto_rcpsp
- *Golomb ruler problem*
- *Square packing problem*
- *Square soft packing problem*
- *Board coloration problem*
- *Learning to play the Sudoku*
- *Learning car configuration preferences*
- *Visual Sudoku Tutorial*
- *Verbose version of a sudoku code*

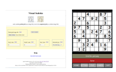
CHAPTER SEVEN

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* 

Misc :

- *A sudoku code*

CHAPTER

EIGHT

BENCHMARK LIBRARIES

- EvalGM : 3026 discrete optimization benchmarks available in various formats (wcsp, wcnf, uai, lp, mzn). [Hurley et al CPAIOR 2016]
- Cost Function Library : an on-going collection of benchmarks from various domains of Artificial Intelligence, Constraint Programming, and Operations Research (formats in wcsp, wcnf, lp, qpbo, uai, opd, and more).

LIST OF ALL EXAMPLES

9.1 Weighted n-queen problem

9.1.1 Brief description

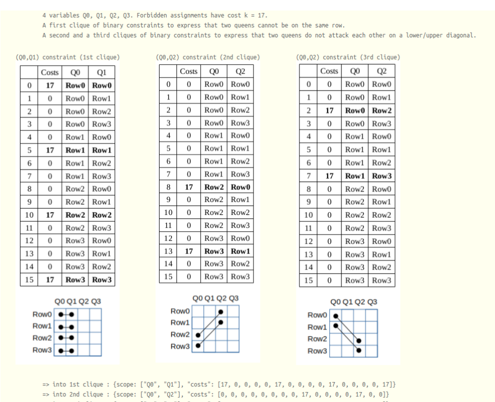
The problem consists in assigning N queens on a NxN chessboard with random costs in (1..N) associated to every cell such that each queen does not attack another queen and the sum of the costs of queen's selected cells is minimized.

9.1.2 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.

9.1.3 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]}

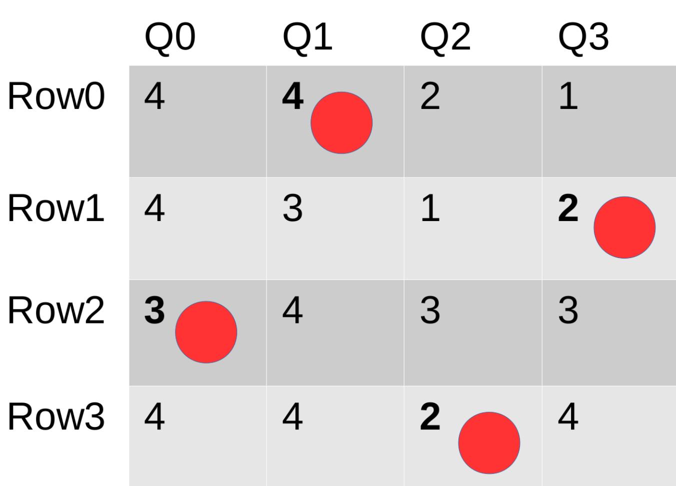
```
{
  problem: { "name": "4-queen", "mustbe": "<17" },
  variables: {"Q0":["Row0", "Row1", "Row2", "Row3"], "Q1":["Row0", "Row1", "Row2", "Row3
↪"],
              "Q2":["Row0", "Row1", "Row2", "Row3"], "Q3":["Row0", "Row1", "Row2", "Row3
```

(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 :



9.1.4 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]))
```

9.2 Weighted latin square problem

9.2.1 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.

9.2.2 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.

9.2.3 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

9.2.4 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]))
```

9.2.5 C++ model

The following code using the C++ toulbar2 library API solves the weighted latin square problem.

latinsquare.cpp

```
#include <iostream>
#include <vector>

#include "core/tb2wcsp.hpp"

using namespace std;

// an alias for storing the variable costs
// first dim is the grid rows and second is the columns
typedef std::vector<std::vector<std::vector<Cost>>> LatinCostArray;

/*!
    \brief generate random costs for each variable (cell)
 */
void initLatinCosts(size_t N, LatinCostArray& costs) {

    // N*N*N values, costs for each cell
    costs.resize(N);
    for(auto& col: costs) {
        col.resize(N);
        for(auto& cell: col) {
            cell.resize(N);
            for(size_t val_ind = 0; val_ind < N; val_ind += 1) {
                cell[val_ind] = (rand()%N)+1;
            }
        }
    }
}

/*!
    \brief print the costs for each unary variabl (cell)
 */
void printCosts(LatinCostArray& costs) {

    for(size_t row_ind = 0; row_ind < costs.size(); row_ind ++) {
        for(size_t col_ind = 0; col_ind < costs[row_ind].size(); col_ind ++) {
            cout << "cell " << row_ind << "_" << col_ind;
            cout << " : ";
            for(auto& cost: costs[row_ind][col_ind]) {
                cout << cost << ", ";
            }
            cout << endl;
        }
    }
}

/*!
    \brief fill in a WCSP object with a latin square problem
```

(continues on next page)

(continued from previous page)

```
 */
void buildWCSP(WeightedCSP& wcsp, LatinCostArray& costs, size_t N, Cost top) {

    // variables
    for(size_t row = 0; row < N; row ++) {
        for(unsigned int col = 0; col < N; col ++) {
            wcsp.makeEnumeratedVariable("Cell_" + to_string(row) + "," + to_string(col),␣
↪0, N-1);
        }
    }

    cout << "number of variables: " << wcsp.numberOfVariables() << endl;

    /* costs for all different constraints (top on diagonal) */
    vector<Cost> alldiff_costs;
    for(unsigned int i = 0; i < N; i ++) {
        for(unsigned int j = 0; j < N; j ++) {
            if(i == j) {
                alldiff_costs.push_back(top);
            } else {
                alldiff_costs.push_back(0);
            }
        }
    }

    /* all different constraints */
    for(unsigned int index = 0; index < N; index ++) {
        for(unsigned int var_ind1 = 0; var_ind1 < N; var_ind1 ++) {
            for(unsigned int var_ind2 = var_ind1+1; var_ind2 < N; var_ind2 ++) {
                /* row constraints */
                wcsp.postBinaryConstraint(N*index+var_ind1, N*index+var_ind2, alldiff_
↪costs);
                /* col constraints */
                wcsp.postBinaryConstraint(index+var_ind1*N, index+var_ind2*N, alldiff_
↪costs);
            }
        }
    }

    /* unary costs */
    size_t var_ind = 0;
    for(size_t row = 0; row < N; row ++) {
        for(size_t col = 0; col < N; col ++) {
            wcsp.postUnaryConstraint(var_ind, costs[row][col]);
            var_ind += 1;
        }
    }

}

int main() {
```

(continues on next page)

(continued from previous page)

```
    srand(123456789);

    size_t N = 5;
    Cost top = N*N*N + 1;

    // N*N*N values, costs for each cell
    LatinCostArray objective_costs;

    // init the costs for each cell
    initLatinCosts(N, objective_costs);

    cout << "Randomly genereated costs : " << endl;
    printCosts(objective_costs);
    cout << endl;

    tb2init();

    ToulBar2::verbose = 0;

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

    // fill in the WeightedCSP object
    WeightedCSP* wcsp = solver->getWCSP();

    buildWCSP(*wcsp, objective_costs, N, top);

    bool result = solver->solve();

    if(result) {

        Cost bestCost = solver->getSolutionValue();
        Cost bestLowerBound = solver->getDDualBound();

        if(!ToulBar2::limited) {
            cout << "Optimal solution found with cost " << bestCost << endl;
        } else {
            cout << "Best solution found with cost " << bestCost << " and best lower␣
↪bound of " << bestLowerBound << endl;
        }

        // retrieve the solution
        std::vector<Value> solution = solver->getSolution();


        cout << endl << "Best solution : " << endl;
        for(size_t var_ind = 0; var_ind < solution.size(); var_ind ++) {
            cout << solution[var_ind] << " ; ";
            if((var_ind+1) % N == 0) {
                cout << endl;
            }
        }
```

(continues on next page)

(continued from previous page)

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

    delete solver;

    return 0;
}
```

9.3 Bicriteria weighted latin square problem

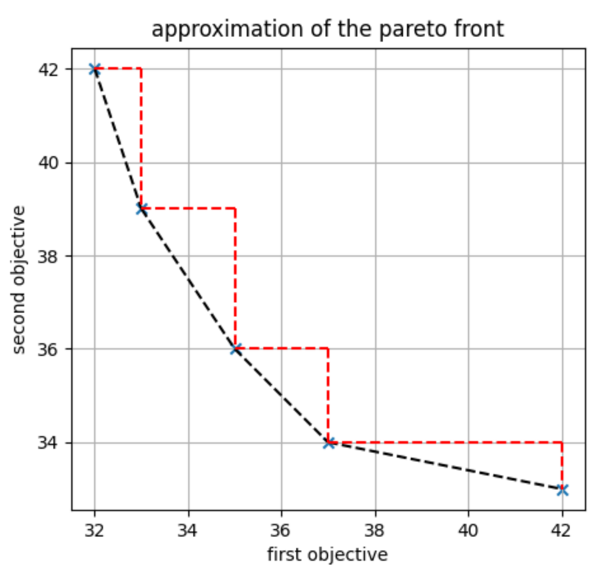
9.3.1 Brief description

In this variant of the *Weighted latin square problem*, two objectives (sum of the costs of the cells) are optimized simultaneously. Each objective is defined by a cost matrix with possible costs for each cell in the chessboard. A subset of the pareto solutions can be obtained by solving linear combinations of the two criteria with various weights on the objectives. This can be achieved in ToulBar2 via a MultiCFN object.

9.3.2 CFN model

Similarly to the *Weighted latin square problem*, NxN variables are created with a domain size N. In this model, the permutation of every row and every column is ensured through infinite costs in binary cost functions. Two different CFN are created to represent the two objectives: a first CFN where unary costs are added only from a first cost matrix, and a second one with unary costs from the second matrix.

Toulbar2 allows to either solve for a chosen weighted sum of the two cost objectives (cost function networks) as input, or approximate the pareto front by enumerating a complete set of non-redundant weights. As it is shown below, the method allows to compute solutions which costs lie in the convex hull of the pareto front. However, potential solutions belonging to the triangles will be missed with this approach.



9.3.3 Python model

The following code using the pytoulbar2 library solves the bicriteria weighted latin square problem with two different pairs of weights for the two objectives.

bicriteria_latinsquare.py

```
import sys
from random import seed, randint
seed(12345678)
import pytoulbar2
from matplotlib import pyplot as plt


N = int(sys.argv[1])

top = N**3 +1

# printing a solution as a grid
def print_solution(sol, N):

  grid = [0 for _ in range(N*N)]
  for k,v in sol.items():
    grid[ int(k[5])*N+int(k[7]) ] = int(v[1:])

  output = ''
  for var_ind in range(len(sol)):
    output += str(grid[var_ind]) + ' '
    if var_ind % N == N-1:
      output += '\n'
  print(output, end='')


# creation of the base problem: variables and hard constraints (alldiff must be␣
↪decomposed into binary constraints)
def create_base_cfn(cfn, N, top):

  # variable creation
  var_indexes = []

  # create N^2 variables, with N values in their domains
  for row in range(N):
    for col in range(N):
      index = cfn.AddVariable('Cell_' + str(row) + '_' + str(col), ['v' + str(val) for␣
↪val in range(N)])
      var_indexes.append(index)

  # all permutation constraints: pairwise all different

  # forbidden values are enforced by infinite costs
  alldiff_costs = [ top if row == col else 0 for row in range(N) for col in range(N) ]

  for index in range(N):
    for var_ind1 in range(N):
```

(continues on next page)

(continued from previous page)

```
      for var_ind2 in range(var_ind1+1, N):

        # permutations in the rows
        cfn.AddFunction([var_indexes[N*index+var_ind1], var_indexes[N*index+var_ind2]],␣
↪alldiff_costs)

        # permutations in the columns
        cfn.AddFunction([var_indexes[index+var_ind1*N], var_indexes[index+var_ind2*N]],␣
↪alldiff_costs)


# generation of two random costs matrice, one for each objective
cell_costs_obj1 = [[randint(1,N) for _ in range(N)] for _ in range(N*N)]
cell_costs_obj2 = [[randint(1,N) for _ in range(N)] for _ in range(N*N)]

# multicfn is the main object for combining multiple cost function networks
multicfn = pytoulbar2.MultiCFN()


# first cfn: costs from the first matrix
cfn = pytoulbar2.CFN(ubinit = top, resolution=6)
cfn.SetName('obj_1')
create_base_cfn(cfn, N, top)
for variable_index in range(N*N):
  cfn.AddFunction([variable_index], cell_costs_obj1[variable_index])
multicfn.PushCFN(cfn)


# first cfn: costs from the second matrix
cfn = pytoulbar2.CFN(ubinit = top, resolution=6)
cfn.SetName('obj_2')
create_base_cfn(cfn, N, top)
for variable_index in range(N*N):
  cfn.AddFunction([variable_index], cell_costs_obj2[variable_index])
multicfn.PushCFN(cfn)


# solve with a first pair of weights
weights = (1., 2.)

multicfn.SetWeight(0, weights[0])
multicfn.SetWeight(1, weights[1])

cfn = pytoulbar2.CFN()
cfn.InitFromMultiCFN(multicfn) # the final cfn is initialized from the combined cfn

# optionaly dumping a weighted sum of the two problems with chosen weights
# cfn.Dump('python_latin_square_bicriteria.cfn')

result = cfn.Solve()

if result:
```

(continues on next page)

(continued from previous page)

```
  print('Solution found with weights', weights, ':')
  sol_costs = multicfn.GetSolutionCosts()
  solution = multicfn.GetSolution()
  print_solution(solution, N)
  print('with costs:', sol_costs, '(sum=', result[1], ')')

print('\n')

# solve a second time with other weights
weights = (2.5, 1.)

multicfn.SetWeight(0, weights[0])
multicfn.SetWeight(1, weights[1])

cfn = pytoulbar2.CFN()
cfn.InitFromMultiCFN(multicfn) # the final cfn is initialized from the combined cfn

# cfn.Dump('python_latin_square_bicriteria.cfn')

result = cfn.Solve()

if result:
  print('Solution found with weights', weights, ':')
  sol_costs = multicfn.GetSolutionCosts()
  solution = multicfn.GetSolution()
  print_solution(solution, N)
  print('with costs:', sol_costs, '(sum=', result[1], ')')


# approximate the pareto front
(solutions, costs) = multicfn.ApproximateParetoFront(0, 'min', 1, 'min')

fig, ax = plt.subplots()
ax.scatter([c[0] for c in costs], [c[1] for c in costs], marker='x')
for index in range(len(costs)-1):
  ax.plot([costs[index][0], costs[index+1][0]], [costs[index][1],costs[index+1][1]], '--
↪', c='k')
  ax.plot([costs[index][0], costs[index+1][0]], [costs[index][1],costs[index][1]], '--',␣
↪c='red')
  ax.plot([costs[index+1][0], costs[index+1][0]], [costs[index][1],costs[index+1][1]], '-
↪-', c='red')

ax.set_xlabel('first objective')
ax.set_ylabel('second objective')
ax.set_title('approximation of the pareto front')
ax.set_aspect('equal')

plt.grid()
plt.show()
```

9.3.4 C++ model

The following code using the C++ toulbar2 library API solves the weighted latin square problem.

latinsquare.cpp

```
#include <iostream>
#include <vector>

#include "core/tb2wcsp.hpp"
#include "mcriteria/multicfn.hpp"
#include "mcriteria/bicriteria.hpp"

using namespace std;

// an alias for storing the variable costs
// first dim is the grid rows and second is the columns
typedef std::vector<std::vector<std::vector<Cost>>> LatinCostArray;


// generate random costs for each variable (cell)
// param N grid size
// param costs the matrix costs
void createCostMatrix(size_t N, LatinCostArray& costs) {

    // N*N*N values, costs for each cell
    costs.resize(N);
    for(auto& col: costs) {
        col.resize(N);
        for(auto& cell: col) {
            cell.resize(N);
            for(size_t val_ind = 0; val_ind < N; val_ind += 1) {
                cell[val_ind] = (rand()%N)+1;
            }
        }
    }
}



// print the costs for each unary variabl (cell)
// param costs the cost matrix
void printCosts(LatinCostArray& costs) {

    for(size_t row_ind = 0; row_ind < costs.size(); row_ind ++) {
        for(size_t col_ind = 0; col_ind < costs[row_ind].size(); col_ind ++) {
            cout << "cell " << row_ind << "_" << col_ind;
            cout << " : ";
            for(auto& cost: costs[row_ind][col_ind]) {
                cout << cost << ", ";
            }
            cout << endl;
        }
    }
```

(continues on next page)

(continued from previous page)

```
}

// fill in a WCSP object with a latin square problem
// param wcsp the wcsp object to fill
// param LatinCostArray the cost matrix
// param N grid size
// top the top value, problem upper bound (the objective is always lower than top)
void buildLatinSquare(WeightedCSP& wcsp, LatinCostArray& costs, size_t N, Cost top) {

    // variables
    for(size_t row = 0; row < N; row ++) {
        for(unsigned int col = 0; col < N; col ++) {
            wcsp.makeEnumeratedVariable("Cell_" + to_string(row) + "," + to_string(col),␣
↪0, N-1);
        }
    }

    /* costs for all different constraints (top on diagonal) */
    vector<Cost> alldiff_costs;
    for(unsigned int i = 0; i < N; i ++) {
        for(unsigned int j = 0; j < N; j ++) {
            if(i == j) {
                alldiff_costs.push_back(top);
            } else {
                alldiff_costs.push_back(0);
            }
        }
    }

    /* all different constraints */
    for(unsigned int index = 0; index < N; index ++) {
        for(unsigned int var_ind1 = 0; var_ind1 < N; var_ind1 ++) {
            for(unsigned int var_ind2 = var_ind1+1; var_ind2 < N; var_ind2 ++) {
                /* row constraints */
                wcsp.postBinaryConstraint(N*index+var_ind1, N*index+var_ind2, alldiff_
↪costs);
                /* col constraints */
                wcsp.postBinaryConstraint(index+var_ind1*N, index+var_ind2*N, alldiff_
↪costs);
            }
        }
    }

    /* unary costs */
    size_t var_ind = 0;
    for(size_t row = 0; row < N; row ++) {
        for(size_t col = 0; col < N; col ++) {
            wcsp.postUnaryConstraint(var_ind, costs[row][col]);
            var_ind += 1;
        }
    }
```

(continues on next page)

(continued from previous page)

```
}

// print a solution as a grid
// param N the size of the grid
// param solution the multicfn solution (dict)
// param point the objective costs (objective space point)
void printSolution(size_t N, MultiCFN::Solution& solution, Bicriteria::Point& point) {
    for(size_t row = 0; row < N; row ++) {
        for(size_t col = 0; col < N; col ++) {
            string var_name = "Cell_" + to_string(row) + "," + to_string(col);
            cout << solution[var_name].substr(1) << " ";
        }
        cout << endl;
    }
    cout << "obj_1 = " << point.first << " ; obj2 = " << point.second << endl;

}

// main function
int main() {

    srand(123456789);

    size_t N = 4;
    Cost top = N*N*N + 1;

    // two cost matrice
    LatinCostArray costs_obj1, costs_obj2;

    // init the objective with random costs
    createCostMatrix(N, costs_obj1);
    createCostMatrix(N, costs_obj2);

    // cout << "Randomly genereated costs : " << endl;
    // printCosts(costs_obj1);
    // cout << endl << endl;
    // printCosts(costs_obj2);

    tb2init();
    initCosts();

    // create the two wcsp objects
    WeightedCSP* wcsp1 = WeightedCSP::makeWeightedCSP(top);
    WeightedCSP* wcsp2 = WeightedCSP::makeWeightedCSP(top);

    // initialize the objects as a latin square problem objectives with two different␣
↪objectves
    buildLatinSquare(*wcsp1, costs_obj1, N, top);
    buildLatinSquare(*wcsp2, costs_obj2, N, top);

    // creation of the multicfn
    MultiCFN mcfn;
```

(continues on next page)

(continued from previous page)

```
    mcfn.push_back(dynamic_cast<WCSP*>(wcsp1));
    mcfn.push_back(dynamic_cast<WCSP*>(wcsp2));

    // computation iof the supported points of the biobjective problem
    Bicriteria::computeSupportedPoints(&mcfn, std::make_pair(Bicriteria::OptimDir::Optim_
↪Min, Bicriteria::OptimDir::Optim_Min));

    // access to the computed solutions and their objective values
    std::vector<MultiCFN::Solution> solutions = Bicriteria::getSolutions();
    std::vector<Bicriteria::Point> points = Bicriteria::getPoints();

    // print all solutions computed
    cout << "Resulting solutions: " << endl;
    for(size_t sol_index = 0; sol_index < solutions.size(); sol_index ++) {
        printSolution(N, solutions[sol_index], points[sol_index]);
        cout << endl;
    }

    // delete the wcsp objects
    delete wcsp1;
    delete wcsp2;

    return 0;
}
```

The above code can be compiled with the following command:

```
g++ -O3 -std=c++17 -Wall -DBOOST -DLONGLONG_COST -DLONGDOUBLE_PROB -I $(YOUR_TB2_INCLUDE_
↪PATH) main.cpp -c -o main.o
```

Where `$(YOUR_TB2_INCLUDE_PATH)` is the path to the ToulBar2 src directory. And the compiled program is obtained via :

```
g++ -O3 -std=c++17 -Wall -DBOOST -DLONGLONG_COST -DLONGDOUBLE_PROB main.o -o main -L
↪$(YOUR_LIBTB2_PATH) -ltb2 -lgmp -lboost_graph -lboost_iostreams -lz -llzma
```

Where `$(YOUR_LIBTB2_PATH)` is the path to the ToulBar2 compiled library. When running the program, do not forget to set the `$(LD_LIBRARY_PATH)` environment variable in Linux.

9.4 Radio link frequency assignment problem

9.4.1 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 consecutive).

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.

9.4.2 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).

9.4.3 Data

Original data files can be downloaded 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.

9.4.4 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 = {}
                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),␣
```

(continues on next page)

(continued from previous page)

```
↪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)

#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)
```

(continues on next page)

(continued from previous page)

```
                        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!')
```

9.5 Frequency assignment problem with polarization

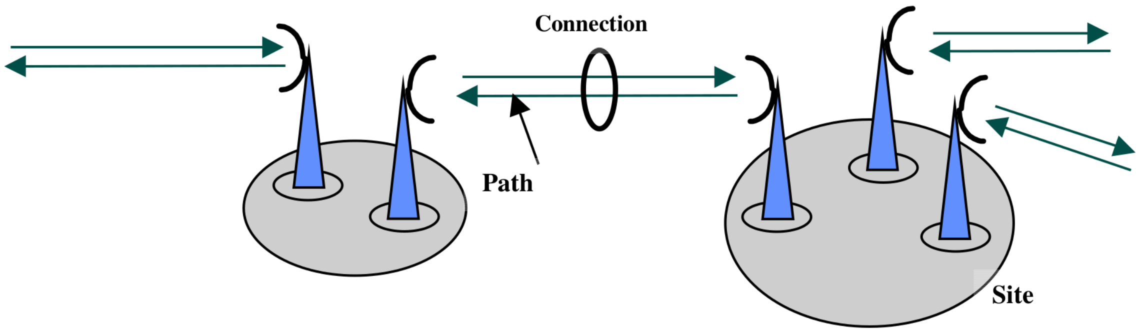
9.5.1 Brief description

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.



9.5.2 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.

9.5.3 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.

9.5.4 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()
```

(continues on next page)

(continued from previous page)

```
                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

                        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
```

(continues on next page)

(continued from previous page)

```
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:
                                                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:
```

(continues on next page)

(continued from previous page)

```
                                        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)
```

9.6 Mendelian error detection problem

9.6.1 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 alleles 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.

9.6.2 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.

9.6.3 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.

9.6.4 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)
                        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)
```

(continues on next page)

(continued from previous page)

```
#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)) :
                                                                                ListConstraintsMendelLaw.
↪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:
                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:
```

(continues on next page)

(continued from previous page)

```
        print('There are',int(res[1]),'difference(s) between the solution and the observation.')
else:
        print('No solution found')
```

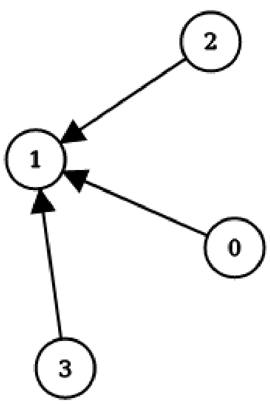
9.7 Block modeling problem

9.7.1 Brief description

This is a clustering problem, occurring 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:

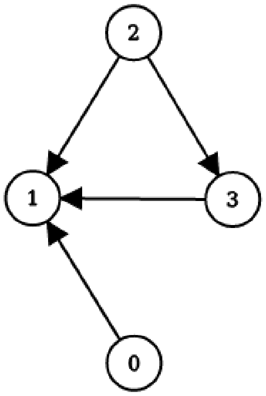
$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

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 minimizes 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.

9.7.2 CFN model

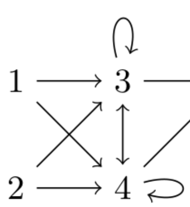
We create N variables, one for every node of the graph, with domain size k representing the clustering. We add k*k Boolean variables for representing M.

For all triplets of two nodes u, v, and one matrix cell M[i,j], we have a ternary cost function that 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

9.7.3 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)

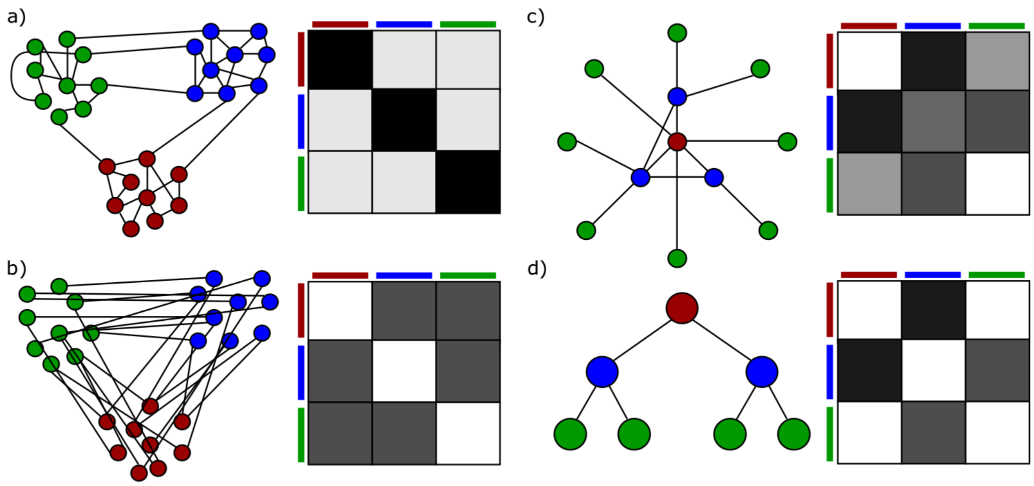


$$G = \left(\begin{array}{cc|cc|c} \mathbf{0} & \mathbf{0} & 1 & 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & 1 & 1 & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & 1 & 1 & 1 \\ \mathbf{0} & \mathbf{0} & 1 & 1 & 1 \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array}\right)$$

$$\{1,2\} \to \{3,4\} \longrightarrow \{5\}$$

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

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`.

9.7.4 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
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):
```

(continues on next page)

(continued from previous page)

```
    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):
        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)
```

(continues on next page)

(continued from previous page)

```
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]))
```

9.8 Airplane landing problem

9.8.1 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 constraint to enforce that the separation time between those planes is larger 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.

9.8.2 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.

9.8.3 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.

9.8.4 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()])

# 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):
```

(continues on next page)

(continued from previous page)

```
        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)
```

9.9 Warehouse location problem

9.9.1 Brief description

A company considers opening warehouses at some candidate locations with each of them having a maintenance cost if it is open.

The company controls a set of given stores and each of them needs to take supplies to one of the warehouses, but depending on the warehouse chosen, there will be an additional supply cost.

The objective is to choose which warehouse to open and to divide the stores among the open warehouses in order to minimize the total cost of supply and maintenance costs.

9.9.2 CFN model

We create Boolean variables for the warehouses (i.e., open or not) and integer variables for the stores, 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.

9.9.3 Data

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

9.9.4 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 an 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])  # in [0,9], used to convert cost values from float to␣
↪integer (by 10**precision)

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))

top += sum(CostW)

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

for j in range(M):
```

(continues on next page)

(continued from previous page)

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

# 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):
        Problem.AddFunction([i, N+j], [(top if (a == 0 and b == i) else 0) for a in␣
↪range(2) for b in range(N)])

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

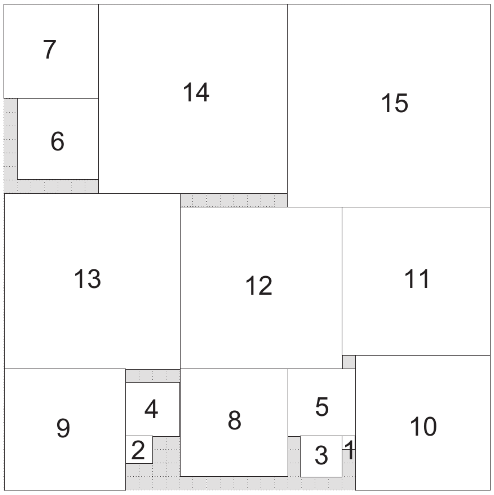
9.10 Square packing problem

9.10.1 Brief description

We have N squares of respective 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.

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



9.10.2 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.

9.10.3 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
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␣
```

(continues on next page)

(continued from previous page)

```
↪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:
                                                        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)
```

(continues on next page)

(continued from previous page)

```
#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!')
```

9.10.4 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 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>
#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
```

(continues on next page)

(continued from previous page)

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

    delete solver;
```

(continues on next page)

(continued from previous page)

```
    return 0;
}
```

9.11 Square soft packing problem

9.11.1 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.

9.11.2 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 stacked together. The cost of this is N**4, so we will take N**4+1 as the initial upper bound.

9.11.3 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()

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):
```

(continues on next page)

(continued from previous page)

```
                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))
                                                        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:
```

(continues on next page)

(continued from previous page)

```
        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!')
```

9.11.4 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>

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
```

(continues on next page)

(continued from previous page)

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

    delete solver;
```

(continues on next page)

(continued from previous page)

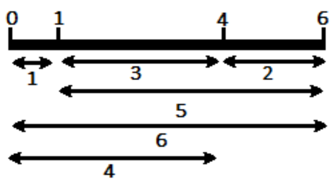
```
    return 0;
}
```

9.12 Golomb ruler problem

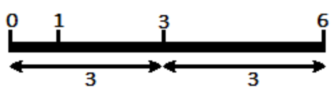
9.12.1 Brief description

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 is 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.

9.12.2 CFN model

We create N variables, one for each integer mark ak. Because we can not create an AllDifferent constraint with differences of variables directly, we also create a variable for each difference and create hard ternary constraints in order to force them be 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 set the first mark to be zero.

9.12.3 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

```python
import sys
import pytoulbar2

N = int(sys.argv[1])
```

(continues on next page)

(continued from previous page)

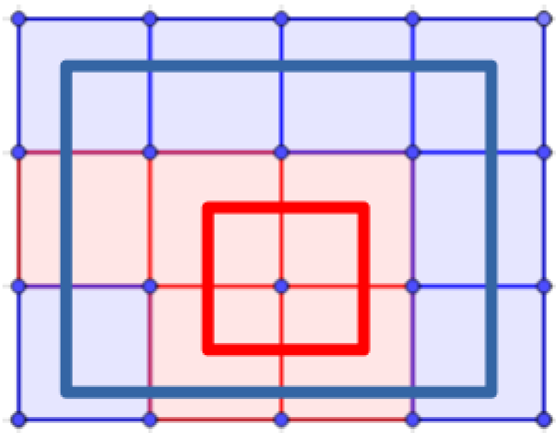
```
top = N**2 + 1

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)
```

9.13 Board coloration problem

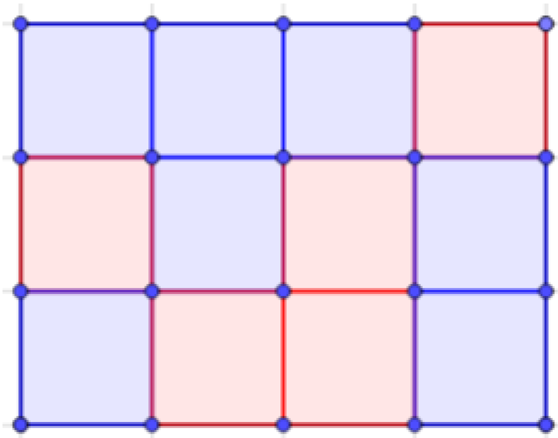
9.13.1 Brief description

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 example, 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:



9.13.2 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 quaternary constraints for every rectangle inside the board with a 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.

9.13.3 Python model

The following code using pytoulbar2 library solves the board coloration problem with the first two arguments being the dimensions 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 integers 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 (encoding with forbidden␣
↪tuples)
ConstraintTuples = []
ConstraintCosts = []
for k in range(n*m):
    #if they are all the same color
    ConstraintTuples.append([k, k, k, k])
    ConstraintCosts.append(top)
#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 valid 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):
                # add a compact function with zero default cost and only forbidden tuples
                Problem.AddCompactFunction(['sq_' + str(i1) + '_' + str(i2), 'sq_' +␣
↪str(i1) + '_' + str(j2), 'sq_' + str(j1) + '_' + str(i2), 'sq_' + str(j1) + '_' +␣
↪str(j2)], 0, ConstraintTuples, ConstraintCosts)
```

(continues on next page)

(continued from previous page)

```
#binary hard constraints to fix the variable max as an upper bound
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)
for i in range(n):
    for j in range(m):
        Problem.AddFunction(['sq_' + str(i) + '_' + str(j), 'max'], Constraint)

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

#symmetry breaking on colors
for i in range(n):
    for j in range(m):
        Constraint = []
        for k in range(n*m):
            if k > i*m+j:
                Constraint.append(top)
            else:
                Constraint.append(0)
        Problem.AddFunction(['sq_' + str(i) + '_' + str(j)], Constraint)

#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 solution found!')
```

9.14 Learning to play the Sudoku

9.14.1 Available

- Presentation
- GitHub code
- Data GitHub code

9.15 Learning car configuration preferences

9.15.1 Brief description

Renault car configuration system: learning user preferences.

9.15.2 Available

- Presentation
- GitHub code
- Data GitHub code

9.16 Visual Sudoku Tutorial

9.16.1 Brief description

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

9.16.2 Available

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

9.17 Visual Sudoku Application

9.17.1 Brief description

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

9.17.2 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.

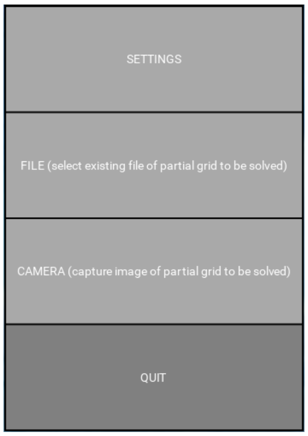
Fig.1 

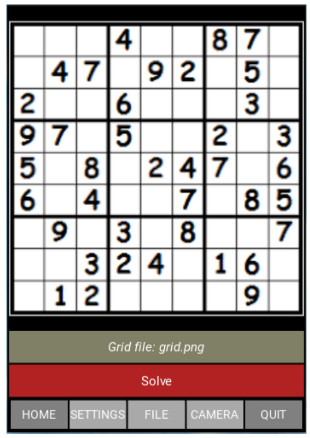
Fig.2 

Fig.3 

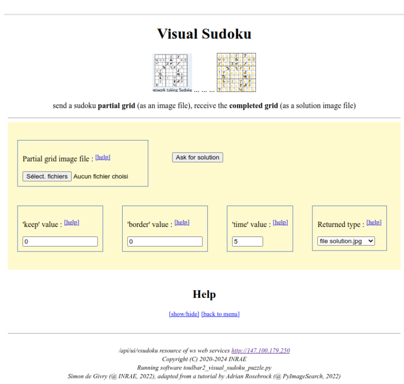
- Fig.1 : Screen of main menu
- Fig.2 : Screen of the grid to be solved
- Fig.3 : Screen of the solution (in yellow) found by the solver

`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** :



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.

9.18 Visual Sudoku App for Android

9.18.1 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*.

9.18.2 Source Code

GitHub code

9.18.3 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.

9.18.4 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.

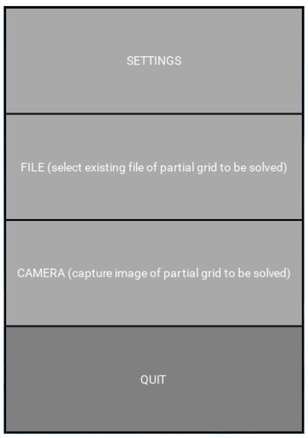
Fig.1 

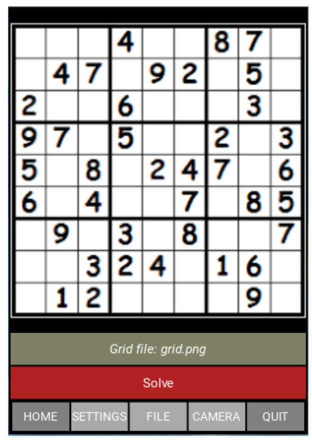
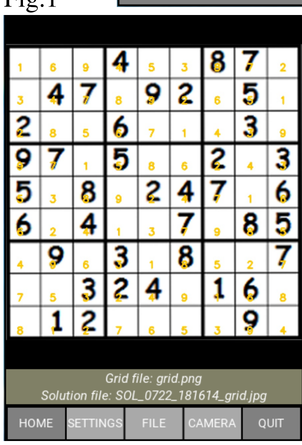
Fig.2 

Fig.3 

- 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`

9.19 A sudoku code

9.19.1 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`.

9.19.2 Available

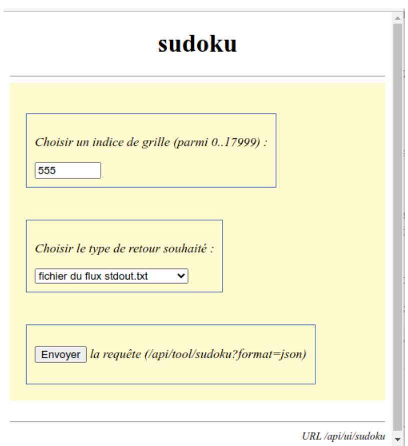
Available as a web service.

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 :

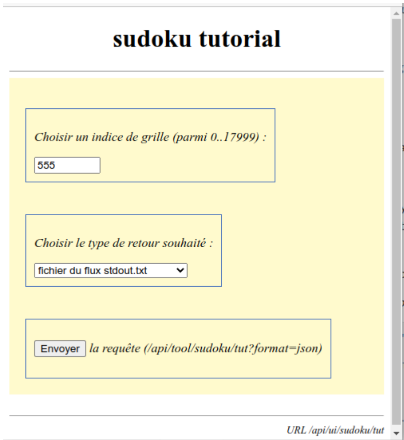
- *returned_type=stdout.txt* : to get the output stream as a .txt file.
- *returned_type=run.zip* : to get the .zip run folder containing the output stream __WS__stdout.txt (+ *the error stream __WS__stderr.txt that may be useful to investigate*).

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



api/ui/sudoku

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



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`

CHAPTER

TEN

USER GUIDE

10.1 What is toulbar2

toulbar2 is an exact black box discrete optimization solver targeted at solving cost function networks (CFN), thus solving the so-called "weighted Constraint Satisfaction Problem" or WCSP. Cost function networks can be simply described by a set of discrete variables each having a specific finite domain and a set of integer cost functions, each involving some of the variables. The WCSP is to find an assignment of all variables such that the sum of all cost functions is minimum and lest than a given upper bound often denoted as k or $\top$. Functions can be typically specified by sparse or full tables but also more concisely as specific functions called "global cost functions" [Schiex2016a].

Using on the fly translation, toulbar2 can also directly solve optimization problems on other graphical models such as Maximum probability Explanation (MPE) on Bayesian networks [koller2009], and Maximum A Posteriori (MAP) on Markov random field [koller2009]. It can also read partial weighted MaxSAT problems, Quadratic Pseudo Boolean problems (MAXCUT) as well as Linkage **.pre** pedigree files for genotyping error detection and correction.

toulbar2 is exact. It will only report an optimal solution when it has both identified the solution and proved its optimality. Because it relies only on integer operations, addition and subtraction, it does not suffer from rounding errors. In the general case, the WCSP, MPE/BN, MAP/MRF, PWMaxSAT, QPBO or MAXCUT being all NP-hard problems and thus toulbar2 may take exponential time to prove optimality. This is however a worst-case behavior and toulbar2 has been shown to be able to solve to optimality problems with half a million non Boolean variables defining a search space as large as $2^{829,440}$. It may also fail to solve in reasonable time problems with a search space smaller than 2^{264}.

toulbar2 provides and uses by default an "anytime" algorithm [Katsirelos2015a] that tries to quickly provide good solutions together with an upper bound on the gap between the cost of each solution and the (unknown) optimal cost. Thus, even if it is unable to prove optimality, it will bound the quality of the solution provided. It can also apply a variable neighborhood search algorithm exploiting a problem decomposition [Ouali2017]. This algorithm is complete (if enough CPU-time is given) and it can be run in parallel using OpenMPI. A parallel version of previous algorithm also exists [Beldjilali2022].

Beyond the service of providing optimal solutions, toulbar2 can also find a greedy sequence of diverse solutions [Ruffini2019a] or exhaustively enumerate solutions below a cost threshold and perform guaranteed approximate weighted counting of solutions. For stochastic graphical models, this means that toulbar2 will compute the partition function (or the normalizing constant Z). These problems being #P-complete, toulbar2 runtimes can quickly increase on such problems.

By exploiting the new toulbar2 python interface, with incremental solving capabilities, it is possible to learn a CFN from data and to combine it with mandatory constraints [Schiex2020b]. See examples at https://forgemia.inra.fr/thomas.schiex/cfn-learn.

10.2 How do I install it ?

toulbar2 is an open source solver distributed under the MIT license as a set of C++ sources managed with git at http://github.com/toulbar2/toulbar2. If you want to use a released version, then you can download there source archives of a specific release that should be easy to compile on most Linux systems.

If you want to compile the latest sources yourself, you will need a modern C++ compiler, CMake, Gnu MP Bignum library, a recent version of boost libraries and optionally the jemalloc memory management and OpenMPI libraries (for more information, see Installation from sources). You can then clone toulbar2 on your machine and compile it by executing:

```
git clone https://github.com/toulbar2/toulbar2.git
cd toulbar2
mkdir build
cd build
# ccmake ..
cmake ..
make
```

Finally, toulbar2 is available in the debian-science section of the unstable/sid Debian version. It should therefore be directly installable using:

```
sudo apt-get install toulbar2
```

If you want to try toulbar2 on crafted, random, or real problems, please look for benchmarks in the Cost Function benchmark Section. Other benchmarks coming from various discrete optimization languages are available at Genotoul EvalGM [Hurley2016b].

10.3 How do I test it ?

Some problem examples are available in the directory **toulbar2/validation**. After compilation with cmake, it is possible to run a series of tests using:

```
make test
```

For debugging toulbar2 (compile with flag `CMAKE_BUILD_TYPE="Debug"`), more test examples are available at Cost Function Library. The following commands run toulbar2 (executable must be found on your system path) on every problems with a 1-hour time limit and compare their optimum with known optima (in .ub files).

```
cd toulbar2
git clone https://forgemia.inra.fr/thomas.schiex/cost-function-library.git
./misc/script/runall.sh ./cost-function-library/trunk/validation
```

Other tests on randomly generated problems can be done where optimal solutions are verified by using an older solver toolbar (executable must be found on your system path).

```
cd toulbar2
git clone https://forgemia.inra.fr/thomas.schiex/toolbar.git
cd toolbar/toolbar
make toolbar
cd ../..
./misc/script/rungenerate.sh
```

10.4 Input formats

10.4.1 Introduction

The available **file formats** (possibly compressed by gzip or bzip2 or xz, e.g., .cfn.gz, .wcsp.xz, .opb.bz2) are :

- Cost Function Network format (.cfn file extension)
- Weighted Constraint Satisfaction Problem (.wcsp file extension)
- Probabilistic Graphical Model (.uai / .LG file extension ; the file format .LG is identical to .UAI except that we expect log-potentials)
- Weighted Partial Max-SAT (.cnf/.wcnf file extension)
- Quadratic Unconstrained Pseudo-Boolean Optimization (.qpbo file extension)
- Pseudo-Boolean Optimization (.opb file extension)
- Integer Linear Programming (.lp file extension)
- Constraint Satisfaction and Optimization Problem (.xml file extension)

Some examples :

- A simple 2 variables maximization problem maximization.cfn in JSON-compatible CFN format, with decimal positive and negative costs.
- Random binary cost function network `example.wcsp`, with a specific variable ordering `example.order`, a tree decomposition `example.cov`, and a cluster decomposition `example.dec`
- Latin square 4x4 with random costs on each variable `latin4.wcsp`
- Radio link frequency assignment CELAR instances `scen06.wcsp`, `scen06.cov`, `scen06.dec`, `scen07.wcsp`
- Earth observation satellite management SPOT5 instances `404.wcsp` and `505.wcsp` with associated tree/cluster decompositions `404.cov`, `505.cov`, `404.dec`, `505.dec`
- Linkage analysis instance `pedigree9.uai`
- Computer vision superpixel-based image segmentation instance `GeomSurf-7-gm256.uai`
- Protein folding instance `1CM1.uai`
- Max-clique DIMACS instance `brock200_4.clq.wcnf`
- Graph 6-coloring instance `GEOM40_6.wcsp`
- Many more instances available evalgm and Cost Function Library.

Notice that by default toulbar2 distinguishes file formats based on their extension. It is possible to read a file from a unix pipe using option `-stdin=[format]`; *e.g.*, `cat example.wcsp | toulbar2 --stdin=wcsp`

It is also possible to read and combine multiple problem files (warning, they must be all in the same format, either wcsp, cfn, or xml). Variables with the same name are merged (domains must be identical), otherwise the merge is based on variable indexes (wcsp format). Warning, it uses the minimum of all initial upper bounds read from the problem files as the initial upper bound of the merged problem.

10.4.2 Formats details

10.5 How do I use it ?

10.5.1 Using it as a C++ library

See *toulbar2 Reference Manual* which describes the libtb2.so C++ library API.

10.5.2 Using it from Python

A Python interface is now available. Compile toulbar2 with cmake option PYTB2 (and without MPI options) to generate a Python module **pytoulbar2** (in lib directory). See examples in `src/pytoulbar2.cpp` and web/TUTORIALS directory.

An older version of toulbar2 was integrated inside Numberjack. See https://github.com/eomahony/Numberjack.

CHAPTER

ELEVEN

REFERENCE MANUAL

CHAPTER

TWELVE

INTRODUCTION

Cost Function Network Solver	toulbar2
Copyright	toulbar2 team
Source	https://github.com/toulbar2/toulbar2

toulbar2 can be used as a stand-alone solver reading various problem file formats (wcsp, uai, wcnf, qpbo) or as a C++ library.

This document describes the WCSP native file format and the toulbar2 C++ library API.

Note

Use cmake flags `LIBTB2=ON` and `TOULBAR2_ONLY=OFF` to get the toulbar2 C++ library libtb2.so and toulbar2test executable example.

See also : `src/toulbar2test.cpp`.

Exact optimization for cost function networks and additive graphical models

(_README_1)= ## What is toulbar2?

toulbar2 is an open-source black-box C++ optimizer for cost function networks and discrete additive graphical models. This also covers Max-SAT, Max-Cut, QUBO (and constrained variants), among others. It can read a variety of formats. The optimized criteria and feasibility should be provided factorized in local cost functions on discrete variables. Constraints are represented as functions that produce costs that exceed a user-provided primal bound. toulbar2 looks for a non-forbidden assignment of all variables that optimizes the sum of all functions (a decision NP-complete problem).

toulbar2 won several competitions on deterministic and probabilistic graphical models:

- Max-CSP 2008 Competition [CPAI08][cpai08] (winner on 2-ARY-EXT and N-ARY-EXT)
- Probabilistic Inference Evaluation [UAI 2008][uai2008] (winner on several MPE tasks, inra entries)
- 2010 UAI APPROXIMATE INFERENCE CHALLENGE [UAI 2010][uai2010] (winner on 1200-second MPE task)
- The Probabilistic Inference Challenge [PIC 2011][pic2011] (second place by ficolofo on 1-hour MAP task)
- UAI 2014 Inference Competition [UAI 2014][uai2014] (winner on all MAP task categories, see Proteus, Robin, and IncTb entries)
- [XCSP3][xcsp] Competitions (second place on Mini COP and Parallel COP tracks in 2022, first place on Mini COP in 2023, third place in 2024)
- UAI 2022 Inference Competition [UAI 2022][uai2022] (winner on all MPE and MMAP task categories)

[cpai08]: http://www.cril.univ-artois.fr/CPAI08/ [uai2008]: http://graphmod.ics.uci.edu/uai08/Evaluation/Report [uai2010]: http://www.cs.huji.ac.il/project/UAI10/summary.php [pic2011]: http://www.cs.huji.ac.il/project/

PASCAL/board.php [uai2014]: https://personal.utdallas.edu/~vibhav.gogate/uai14-competition/leaders.html [xcsp]: https://xcsp.org/competitions [uai2022]: https://uaicompetition.github.io/uci-2022/results/final-leader-board

toulbar2 is now also able to collaborate with ML code that can learn an additive graphical model (with constraints) from data (see the associated [paper](https://miat.inrae.fr/schiex/Export/Pushing_Data_in_your_CP_model.pdf), [slides](https://miat.inrae.fr/schiex/Export/Pushing_Data_in_your_CP_model-Slides.pdf) and [video](https://www.youtube.com/watch?v=IpUr6KIEjMs) where it is shown how it can learn user preferences or how to play the Sudoku without knowing the rules). The current CFN learning code is available on [GitHub](https://github.com/toulbar2/CFN-learn).

(_README_2)= ## Installation from binaries

You can install toulbar2 directly using the package manager in Debian and Debian derived Linux distributions (Ubuntu, Mint,. . .):

> sudo apt-get update sudo apt-get install toulbar2 toulbar2-doc

For the most recent binary or the Python API, compile from source.

(_README_3)= ## Python interface

An alpha-release Python interface can be tested through pip on Linux and MacOS:

> python3 -m pip install –upgrade pip python3 -m pip install pytoulbar2

The first line is only useful for Linux distributions that ship "old" versions of pip.

Commands for compiling the Python API on Linux/MacOS with cmake (Python module in lib/*/pytb2.cpython*.so):

> pip3 install pybind11 mkdir build cd build cmake -DPYTB2=ON .. make

Move the cpython library and the experimental [pytoulbar2.py](https://github.com/toulbar2/toulbar2/raw/master/pytoulbar2/pytoulbar2.py) python class wrapper in the folder of the python script that does "import pytoulbar2".

(_README_4)= ## Download

Download the latest release from GitHub (https://github.com/toulbar2/toulbar2) or similarly use tag versions, e.g.:

> git clone –branch 1.2.0 https://github.com/toulbar2/toulbar2.git

(_README_5)= ## Installation from sources

Compilation requires git, cmake and a C++-20 capable compiler (in C++20 mode).

Required library: * libgmp-dev * bc (used during cmake)

Recommended libraries (default use): * libboost-graph-dev * libboost-iostreams-dev * libboost-serialization-dev * zlib1g-dev * liblzma-dev * libbz2-dev * libeigen3-dev

Optional libraries: * libjemalloc-dev * pybind11-dev * libopenmpi-dev * libboost-mpi-dev * libicuuc * libicui18n * libicudata * libxml2-dev * libxcsp3parser

On MacOS, run ./misc/script/MacOS-requirements-install.sh to install the recommended libraries. For Mac with ARM64, add option -DBoost=OFF to cmake.

Commands for compiling toulbar2 on Linux/MacOS with cmake (binary in build/bin/*/toulbar2):

> mkdir build cd build cmake .. make

Commands for statically compiling toulbar2 on Linux in directory toulbar2/src without cmake:

> bash cd src echo '#define Toulbar_VERSION "1.2.0"' > ToulbarVersion.hpp g++ -o toulbar2 -std=c++20 -O3 -DNDEBUG -march=native -flto -static -static-libgcc -static-libstdc++ -DBOOST -DLONGDOUBLE_PROB -DLONGLONG_COST -DWCSPFORMATONLY

-I. -I./pils/src tb2*.cpp applis/*.cpp convex/*.cpp core/*.cpp globals/*.cpp incop/*.cpp mcriteria/*.cpp pils/src/exe/*.cpp search/*.cpp utils/*.cpp vns/*.cpp ToulbarVersion.cpp -lboost_graph -lboost_iostreams -lboost_serialization -lgmp -lz -lbz2 -llzma

Use OPENMPI flag and MPI compiler for a parallel version of toulbar2:

bash cd src echo '#define Toulbar_VERSION "1.2.0"' > ToulbarVersion.hpp mpicxx -o toulbar2 -std=c++20 -O3 -DNDEBUG -march=native -flto -DBOOST -DLONGDOUBLE_PROB -DLONGLONG_COST -DWCSPFORMATONLY -DOPENMPI

-I. -I./pils/src tb2*.cpp applis/*.cpp convex/*.cpp core/*.cpp globals/*.cpp incop/*.cpp mcriteria/*.cpp pils/src/exe/*.cpp search/*.cpp utils/*.cpp vns/*.cpp ToulbarVersion.cpp -lboost_graph -lboost_iostreams -lboost_serialization -lboost_mpi -lgmp -lz -lbz2 -llzma

Replace LONGLONG_COST by INT_COST to reduce memory usage by two and reduced cost range (costs must be smaller than 10^8).

Replace WCSPFORMATONLY by XMLFLAG3 and add libxcsp3parser.a from xcsp.org in your current directory for reading XCSP3 files:

bash cd src echo '#define Toulbar_VERSION "1.2.0"' > ToulbarVersion.hpp mpicxx -o toulbar2 -std=c++20 -O3 -DNDEBUG -march=native -flto -DBOOST -DLONGDOUBLE_PROB -DLONGLONG_COST -DXMLFLAG3 -DOPENMPI

-I/usr/include/libxml2 -I. -I./pils/src -I./xmlcsp3 tb2*.cpp applis/*.cpp convex/*.cpp core/*.cpp globals/*.cpp incop/*.cpp mcriteria/*.cpp pils/src/exe/*.cpp search/*.cpp utils/*.cpp vns/*.cpp ToulbarVersion.cpp -lboost_graph -lboost_iostreams -lboost_serialization -lboost_mpi -lxml2 -licuuc -licui18n -licudata libxcsp3parser.a -lgmp -lz -lbz2 -llzma -lm -lpthread -ldl

CHAPTER

THIRTEEN

DOCUMENTATION IN PDF

- Main documentation :

  `toulbar2`

- API Reference :

  `Class Diagram|C++ Library of toulbar2|Python Library of toulbar2`

- Some extracts :

  `User manual|Reference manual`

  `WCSP format|CFN format`

  `Tutorials|Use cases`

CHAPTER

FOURTEEN

PUBLICATIONS

14.1 Conference talks

- talk on toulbar2 at ROADEF 2023, Rennes, France, February 21, 2023.
- ANITI webinar on toulbar2 for industrial applications : slides in English | talk in French
- talk on toulbar2 latest algorithmic features at ISMP 2018, Bordeaux, France, July 6, 2018.
- toulbar2 projects meeting at CP 2016, Toulouse, France, September 5, 2016.

14.2 Related publications

14.2.1 What are the algorithms inside toulbar2 ?

- **Soft arc consistencies (NC, AC, DAC, FDAC)**

  In the quest of the best form of local consistency for Weighted CSP, J. Larrosa & T. Schiex, In Proc. of IJCAI-03. Acapulco, Mexico, 2003.

- **Soft existential arc consistency (EDAC)**

  Existential arc consistency: Getting closer to full arc consistency in weighted csps, S. de Givry, M. Zytnicki, F. Heras, and J. Larrosa, In Proc. of IJCAI-05, Edinburgh, Scotland, 2005.

- **Depth-first Branch and Bound exploiting a tree decomposition (BTD)**

  Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP, S. de Givry, T. Schiex, and G. Verfaillie, In Proc. of AAAI-06, Boston, MA, 2006 .

- **Virtual arc consistency (VAC)**

  Virtual arc consistency for weighted csp, M. Cooper, S. de Givry, M. Sanchez, T. Schiex, and M. Zytnicki In Proc. of AAAI-08, Chicago, IL, 2008.

- **Soft generalized arc consistencies (GAC, FDGAC)**

  Towards Efficient Consistency Enforcement for Global Constraints in Weighted Constraint Satisfaction, J. H. M. Lee and K. L. Leung, In Proc. of IJCAI-09, Pasadena (CA), USA, 2009.

- **Russian doll search exploiting a tree decomposition (RDS-BTD)**

  Russian doll search with tree decomposition, M Sanchez, D Allouche, S de Givry, and T Schiex, In Proc. of IJCAI-09, Pasadena (CA), USA, 2009.

- **Soft bounds arc consistency (BAC)**

  Bounds Arc Consistency for Weighted CSPs, M. Zytnicki, C. Gaspin, S. de Givry, and T. Schiex, Journal of Artificial Intelligence Research, 35:593-621, 2009.

- **Counting solutions in satisfaction (#BTD, Approx_#BTD)**

  Exploiting problem structure for solution counting, A. Favier, S. de Givry, and P. Jégou, In Proc. of CP-09, Lisbon, Portugal, 2009.

- **Soft existential generalized arc consistency (EDGAC)**

  A Stronger Consistency for Soft Global Constraints in Weighted Constraint Satisfaction, J. H. M. Lee and K. L. Leung, In Proc. of AAAI-10, Boston, MA, 2010 .

- **Preprocessing techniques (combines variable elimination and cost function decomposition)**

  Pairwise decomposition for combinatorial optimization in graphical models, A Favier, S de Givry, A Legarra, and T Schiex, In Proc. of IJCAI-11, Barcelona, Spain, 2011.

- **Decomposable global cost functions (wregular, wamong, wsum)**

  Decomposing global cost functions, D Allouche, C Bessiere, P Boizumault, S de Givry, P Gutierrez, S Loudni, JP Métivier, and T Schiex, In Proc. of AAAI-12, Toronto, Canada, 2012.

- **Pruning by dominance (DEE)**

  Dead-End Elimination for Weighted CSP, S de Givry, S Prestwich, and B O'Sullivan, In Proc. of CP-13, pages 263-272, Uppsala, Sweden, 2013.

- **Hybrid best-first search exploiting a tree decomposition (HBFS)**

  Anytime Hybrid Best-First Search with Tree Decomposition for Weighted CSP, D Allouche, S de Givry, G Katsirelos, T Schiex, and M Zytnicki, In Proc. of CP-15, Cork, Ireland, 2015.

- **Unified parallel decomposition guided variable neighborhood search (UDGVNS/UPDGVNS)**

  Iterative Decomposition Guided Variable Neighborhood Search for Graphical Model Energy Minimization, A Ouali, D Allouche, S de Givry, S Loudni, Y Lebbah, F Eckhardt, and L Loukil, In Proc. of UAI-17, pages 550-559, Sydney, Australia, 2017.

  Variable Neighborhood Search for Graphical Model Energy Minimization, A Ouali, D Allouche, S de Givry, S Loudni, Y Lebbah, L Loukil, and P Boizumault, Artificial Intelligence, 2020.

- **Clique cut global cost function (clique)**

  Clique Cuts in Weighted Constraint Satisfaction, S de Givry and G Katsirelos, In Proc. of CP-17, pages 97-113, Melbourne, Australia, 2017.

- **Greedy sequence of diverse solutions (div)**

  Guaranteed diversity & quality for the Weighted CSP, M Ruffini, J Vucinic, S de Givry, G Katsirelos, S Barbe, and T Schiex, In Proc. of ICTAI-19, pages 18-25, Portland, OR, USA, 2019.

- **VAC-integrality based variable heuristics and initial upper-bound probing (vacint and rasps)**

  Relaxation-Aware Heuristics for Exact Optimization in Graphical Models, F Trösser, S de Givry and G Katsirelos, In Proc. of CPAIOR-20, Vienna, Austria, 2020.

- **Partition crossover iterative local search (pils)**

  Iterated local search with partition crossover for computational protein design, François Beuvin, Simon de Givry, Thomas Schiex, Sébastien Verel, and David Simoncini, Proteins: Structure, Function, and Bioinformatics, 2021.

- **Knapsack/generalized linear global constraint (knapsack/knapsackp)**

Multiple-choice knapsack constraint in graphical models, P Montalbano, S de Givry, and G Katsirelos, In Proc. of CP-AI-OR'2022, Los Angeles, CA, 2022.

- **Parallel hybrid best-first search (parallel HBFS)**

  Parallel Hybrid Best-First Search, A Beldjilali, P Montalbano, D Allouche, G Katsirelos, and S de Givry, In Proc. of CP-22, volume 235, pages 7:1-7:10, Haifa, Israel, 2022.

- **Low rank block coordinate descent (LR-BCD)**

  Efficient Low Rank Convex Bounds for Pairwise Discrete Graphical Models, V Durante, G Katsirelos, and T Schiex, In Proc. of the 39th International Conference on Machine Learning (ICML), PMLR 162:5726-5741, 2022.

- **Virtual Pairwise Consistency (pwc, hve)**

  Virtual Pairwise Consistency in Cost Function Networks, P Montalbano, D Allouche, S de Givry, G Katsirelos, and T Werner In Proc. of CP-AI-OR'2023, Nice, France, 2023.

- **Bi-Objective Combinatorial Optimization (global bounding constraint)**

  Bi-Objective Discrete Graphical Model Optimization, S Buchet, D Allouche, S de Givry, and T Schiex In Proc. of CP-AI-OR'2024, Uppsala, Sweden, 2024.

14.2.2 toulbar2 for Combinatorial Optimization in Life Sciences

- **Computational Protein Design**

  Colom, Mireia Solà, et al. Deep Evolutionary Forecasting identifies highly-mutated SARS-CoV-2 variants via functional sequence-landscape enumeration. Research Square, 2022.

  XENet: Using a new graph convolution to accelerate the timeline for protein design on quantum computers Jack B. Maguire, Daniele Grattarola, Vikram Khipple Mulligan, Eugene Klyshko, Hans Melo. Plos Comp. Biology, 2021.

  Designing Peptides on a Quantum Computer, Vikram Khipple Mulligan, Hans Melo, Haley Irene Merritt, Stewart Slocum, Brian D. Weitzner, Andrew M. Watkins, P. Douglas Renfrew, Craig Pelissier, Paramjit S. Arora, and Richard Bonneau, bioRxiv, 2019.

  Computational design of symmetrical eight-bladed β-propeller proteins, Noguchi, H., Addy, C., Simoncini, D., Wouters, S., Mylemans, B., Van Meervelt, L., Schiex, T., Zhang, K., Tameb, J., and Voet, A., IUCrJ, 6(1), 2019.

  Positive Multi-State Protein Design, Jelena Vučinić, David Simoncini, Manon Ruffini, Sophie Barbe, Thomas Schiex, Bioinformatics, 2019.

  Cost function network-based design of protein-protein interactions: predicting changes in binding affinity, Clément Viricel, Simon de Givry, Thomas Schiex, and Sophie Barbe, Bioinformatics, 2018.

  Algorithms for protein design, Pablo Gainza, Hunter M Nisonoff, Bruce R Donald, Current Opinion in Structural Biology, 39:6-26, 2016.

  Fast search algorithms for computational protein design, Seydou Traoré, Kyle E Roberts, David Allouche, Bruce R Donald, Isabelle André, Thomas Schiex, and Sophie Barbe, Journal of computational chemistry, 2016.

  Comparing three stochastic search algorithms for computational protein design: Monte Carlo, replica exchange Monte Carlo, and a multistart, steepest-descent heuristic, David Mignon, Thomas Simonson, Journal of computational chemistry, 2016.

  Protein sidechain conformation predictions with an mmgbsa energy function, Thomas Gaillard, Nicolas Panel, and Thomas Simonson, Proteins: Structure, Function, and Bioinformatics, 2016.

  Improved energy bound accuracy enhances the efficiency of continuous protein design, Kyle E Roberts and Bruce R Donald, Proteins: Structure, Function, and Bioinformatics, 83(6):1151-1164, 2015.

Guaranteed discrete energy optimization on large protein design problems, D. Simoncini, D. Allouche, S. de Givry, C. Delmas, S. Barbe, and T. Schiex, Journal of Chemical Theory and Computation, 2015.

Computational protein design as an optimization problem, David Allouche, Isabelle André, Sophie Barbe, Jessica Davies, Simon de Givry, George Katsirelos, Barry O'Sullivan, Steve Prestwich, Thomas Schiex, and Seydou Traoré, Journal of Artificial Intelligence, 212:59-79, 2014.

A new framework for computational protein design through cost function network optimization, Seydou Traoré, David Allouche, Isabelle André, Simon de Givry, George Katsirelos, Thomas Schiex, and Sophie Barbe, Bioinformatics, 29(17):2129-2136, 2013.

- **Genetics**

  Optimal haplotype reconstruction in half-sib families, Aurélie Favier, Jean-Michel Elsen, Simon de Givry, and Andrès Legarra, ICLP-10 workshop on Constraint Based Methods for Bioinformatics, Edinburgh, UK, 2010.

  Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques, Marti Sanchez, Simon de Givry, and Thomas Schiex, Constraints, 13(1-2):130-154, 2008. See also Mendelsoft integrated in the QTLmap Quantitative Genetics platform from INRA GA dept.

- **RNA motif search**

  Darn! a weighted constraint solver for RNA motif localization, Matthias Zytnicki, Christine Gaspin, and Thomas Schiex, Constraints, 13(1-2):91-109, 2008.

- **Agronomy**

  Solving the crop allocation problem using hard and soft constraints, Mahuna Akplogan, Simon de Givry, Jean-Philippe Métivier, Gauthier Quesnel, Alexandre Joannon, and Frédérick Garcia, RAIRO - Operations Research, 47:151-172, 2013.

14.2.3 Other publications mentioning toulbar2

- **Constraint Satisfaction, Distributed Constraint Optimization**

  Graph Based Optimization For Multiagent Cooperation, Arambam James Singh, Akshat Kumar, In Proc. of AAMAS, 2019.

  Probabilistic Inference Based Message-Passing for Resource Constrained DCOPs, Supriyo Ghosh, Akshat Kumar, Pradeep Varakantham, In Proc. of IJCAI, 2015.

  SAT-based MaxSAT algorithms, Carlos Ansótegui and Maria Luisa Bonet and Jordi Levy, Artificial Intelligence, 196:77-105, 2013.

  Local Consistency and SAT-Solvers, P. Jeavons and J. Petke, Journal of Artificial Intelligence Research, 43:329-351, 2012.

- **Data Mining and Machine Learning**

  Pushing Data in CP Models Using Graphical Model Learning and Solving, Céline Brouard, Simon de Givry, and Thomas Schiex, In Proc. of CP-20, Louvain-la-neuve, Belgium, 2020.

  A constraint programming approach for mining sequential patterns in a sequence database, Jean-Philippe Métivier, Samir Loudni, and Thierry Charnois, In Proc. of the ECML/PKDD Workshop on Languages for Data Mining and Machine Learning, Praha, Czech republic, 2013.

- **Timetabling, planning and POMDP**

  Solving a Judge Assignment Problem Using Conjunctions of Global Cost Functions, S de Givry, J.H.M. Lee, K.L. Leung, and Y.W. Shum, In Proc. of CP-14, pages 797-812, Lyon, France, 2014.

  Optimally solving Dec-POMDPs as continuous-state MDPs, Jilles Steeve Dibangoye, Christopher Amato, Olivier Buffet, and François Charpillet, In Proc. of IJCAI, pages 90-96, 2013.

A weighted csp approach to cost-optimal planning, Martin C Cooper, Marie de Roquemaurel, and Pierre Régnier, Ai Communications, 24(1):1-29, 2011.

Point-based backup for decentralized POMDPs: Complexity and new algorithms, Akshat Kumar and Shlomo Zilberstein, In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, 1:1315-1322, 2010.

- **Inference, Sampling, and Diagnostic**

  Dubray, A., Derval, G., Nijssen, S., Schaus, P. Optimal Decoding of Hidden Markov Models with Consistency Constraints. In Proc. of Discovery Science (DS), LNCS 13601, 2022.

  Mohamed-Hamza Ibrahim, Christopher Pal and Gilles Pesant, Leveraging cluster backbones for improving MAP inference in statistical relational models, In Ann. Math. Artif. Intell. 88, No. 8, 907-949, 2020.

  C. Viricel, D. Simoncini, D. Allouche, S. de Givry, S. Barbe, and T. Schiex, Approximate counting with deterministic guarantees for affinity computations, In Proc. of Modeling, Computation and Optimization in Information Systems and Management Sciences - MCO'15, Metz, France, 2015.

  Discrete sampling with universal hashing, Stefano Ermon, Carla P Gomes, Ashish Sabharwal, and Bart Selman, In Proc. of NIPS, pages 2085-2093, 2013.

  Compiling ai engineering models for probabilistic inference, Paul Maier, Dominik Jain, and Martin Sachenbacher, In KI 2011: Advances in Artifcial Intelligence, pages 191-203, 2011.

  Diagnostic hypothesis enumeration vs. probabilistic inference for hierarchical automata models, Paul Maier, Dominik Jain, and Martin Sachenbacher, In Proc. of the International Workshop on Principles of Diagnosis, Murnau, Germany, 2011.

- **Computer Vision and Energy Minimization**

  Exact MAP-inference by Confining Combinatorial Search with LP Relaxation, Stefan Haller, Paul Swoboda, Bogdan Savchynskyy, In Proc. of AAAI, 2018.

- **Computer Music**

  Exploiting structural relationships in audio music signals using markov logic networks, Hélène Papadopoulos and George Tzanetakis, In Proc. of 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pages 4493-4497, Canada, 2013.

  Modeling chord and key structure with markov logic, Hélène Papadopoulos and George Tzanetakis, In Proc. of the Society for Music Information Retrieval (ISMIR), pages 121-126, 2012.

- **Inductive Logic Programming**

  Extension of the top-down data-driven strategy to ILP, Erick Alphonse and Céline Rouveirol, In Proc. of Inductive Logic Programming, pages 49-63, 2007.

- **Other domains**

  An automated model abstraction operator implemented in the multiple modeling environment MOM, Peter Struss, Alessandro Fraracci, and D Nyga, In Proc. of the 25th International Workshop on Qualitative Reasoning, Barcelona, Spain, 2011.

  Modeling Flowchart Structure Recognition as a Max-Sum Problem, Martin Bresler, Daniel Prusa, Václav Hlavác, In Proc. of International Conference on Document Analysis and Recognition, Washington, DC, USA, 1215-1219, 2013.

14.3 References

See 'BIBLIOGRAPHY' at the end of the document.

BIBLIOGRAPHY

[Beldjilali22] A Beldjilali, P Montalbano, D Allouche, G Katsirelos and S de Givry. Parallel Hybrid Best-First Search. In *Proc. of CP-22*, Haifa, Israel, 2022.

[Schiex2020b] Céline Brouard and Simon de Givry and Thomas Schiex. Pushing Data in CP Models Using Graphical Model Learning and Solving. In *Proc. of CP-20*, Louvain-la-neuve, Belgium, 2020.

[Trosser2020a] Fulya Trösser, Simon de Givry and George Katsirelos. Relaxation-Aware Heuristics for Exact Optimization in Graphical Models. In *Proc.of CP-AI-OR'2020*, Vienna, Austria, 2020.

[Ruffini2019a] M. Ruffini, J. Vucinic, S. de Givry, G. Katsirelos, S. Barbe and T. Schiex. Guaranteed Diversity & Quality for the Weighted CSP. In *Proc. of ICTAI-19*, pages 18-25, Portland, OR, USA, 2019.

[Ouali2017] Abdelkader Ouali, David Allouche, Simon de Givry, Samir Loudni, Yahia Lebbah, Francisco Eckhardt, Lakhdar Loukil. Iterative Decomposition Guided Variable Neighborhood Search for Graphical Model Energy Minimization. In *Proc. of UAI-17*, pages 550-559, Sydney, Australia, 2017.

[Schiex2016a] David Allouche, Christian Bessière, Patrice Boizumault, Simon de Givry, Patricia Gutierrez, Jimmy H.M. Lee, Ka Lun Leung, Samir Loudni, Jean-Philippe Métivier, Thomas Schiex and Yi Wu. Tractability-preserving transformations of global cost functions. *Artificial Intelligence*, 238:166-189, 2016.

[Hurley2016b] B Hurley, B O'Sullivan, D Allouche, G Katsirelos, T Schiex, M Zytnicki and S de Givry. Multi-Language Evaluation of Exact Solvers in Graphical Model Discrete Optimization. *Constraints*, 21(3):413-434, 2016. Presentation at CPAIOR'16, Banff, Canada, http://www.inra.fr/mia/T/degivry/cpaior16sdg.pdf.

[Katsirelos2015a] D Allouche, S de Givry, G Katsirelos, T Schiex and M Zytnicki. Anytime Hybrid Best-First Search with Tree Decomposition for Weighted CSP. In *Proc. of CP-15*, pages 12-28, Cork, Ireland, 2015.

[Schiex2014a] David Allouche, Jessica Davies, Simon de Givry, George Katsirelos, Thomas Schiex, Seydou Traoré, Isabelle André, Sophie Barbe, Steve Prestwich and Barry O'Sullivan. Computational Protein Design as an Optimization Problem. *Artificial Intelligence*, 212:59-79, 2014.

[Givry2013a] S de Givry, S Prestwich and B O'Sullivan. Dead-End Elimination for Weighted CSP. In *Proc. of CP-13*, pages 263-272, Uppsala, Sweden, 2013.

[Ficolofo2012] D Allouche, C Bessiere, P Boizumault, S de Givry, P Gutierrez, S Loudni, JP Métivier and T Schiex. Decomposing Global Cost Functions. In *Proc. of AAAI-12*, Toronto, Canada, 2012. http://www.inra.fr/mia/T/degivry/Ficolofo2012poster.pdf (poster).

[Favier2011a] A Favier, S de Givry, A Legarra and T Schiex. Pairwise decomposition for combinatorial optimization in graphical models. In *Proc. of IJCAI-11*, Barcelona, Spain, 2011. Video demonstration at http://www.inra.fr/mia/T/degivry/Favier11.mov.

[Cooper2010a] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki and T. Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174(7-8):449-478, 2010.

[Favier2009a] A. Favier, S. de Givry and P. Jégou. Exploiting Problem Structure for Solution Counting. In *Proc. of CP-09*, pages 335-343, Lisbon, Portugal, 2009.

[Sanchez2009a] M Sanchez, D Allouche, S de Givry and T Schiex. Russian Doll Search with Tree Decomposition. In *Proc. of IJCAI'09*, Pasadena (CA), USA, 2009. http://www.inra.fr/mia/T/degivry/rdsbtd_ijcai09_sdg.ppt.

[Cooper2008] M. Cooper, S. de Givry, M. Sanchez, T. Schiex and M. Zytnicki. Virtual Arc Consistency for Weighted CSP. In *Proc. of AAAI-08*, Chicago, IL, 2008.

[Schiex2006a] S. de Givry, T. Schiex and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *Proc. of AAAI-06*, Boston, MA, 2006. http://www.inra.fr/mia/T/degivry/VerfaillieAAAI06pres.pdf (slides).

[Heras2005] S. de Givry, M. Zytnicki, F. Heras and J. Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. In *Proc. of IJCAI-05*, pages 84-89, Edinburgh, Scotland, 2005.

[Larrosa2000] J. Larrosa. Boosting search with variable elimination. In *Principles and Practice of Constraint Programming - CP 2000*, volume 1894 of LNCS, pages 291-305, Singapore, September 2000.

[koller2009] D Koller and N Friedman. Probabilistic graphical models: principles and techniques. The MIT Press, 2009.

[Ginsberg1995] W. D. Harvey and M. L. Ginsberg. Limited Discrepency Search. In *Proc. of IJCAI-95*, Montréal, Canada, 1995.

[Lecoutre2009] C. Lecoutre, L. Saïs, S. Tabary and V. Vidal. Reasoning from last conflict(s) in constraint programming. *Artificial Intelligence*, 173:1592,1614, 2009.

[boussemart2004] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre and Lakhdar Sais. Boosting systematic search by weighting constraints. In *ECAI*, volume 16, page 146, 2004.

[idwalk:cp04] Bertrand Neveu, Gilles Trombettoni and Fred Glover. ID Walk: A Candidate List Strategy with a Simple Diversification Device. In *Proc. of CP*, pages 423-437, Toronto, Canada, 2004.

[Verfaillie1996] G. Verfaillie, M. Lemaître and T. Schiex. Russian Doll Search. In *Proc. of AAAI-96*, pages 181-187, Portland, OR, 1996.

[LL2009] J. H. M. Lee and K. L. Leung. Towards Efficient Consistency Enforcement for Global Constraints in Weighted Constraint Satisfaction. In *Proceedings of IJCAI'09*, pages 559-565, 2009.

[LL2010] J. H. M. Lee and K. L. Leung. A Stronger Consistency for Soft Global Constraints in Weighted Constraint Satisfaction. In *Proceedings of AAAI'10*, pages 121-127, 2010.

[LL2012asa] J. H. M. Lee and K. L. Leung. Consistency Techniques for Global Cost Functions in Weighted Constraint Satisfaction. *Journal of Artificial Intelligence Research*, 43:257-292, 2012.

[Larrosa2002] J. Larrosa. On Arc and Node Consistency in weighted {CSP}. In *Proc. AAAI'02*, pages 48-53, Edmondton, (CA), 2002.

[Larrosa2003] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for Weighted CSP. In *Proc. of the 18th IJCAI*, pages 239-244, Acapulco, Mexico, August 2003.

[Schiex2000b] T. Schiex. Arc consistency for soft constraints. In *Principles and Practice of Constraint Programming - CP 2000*, volume 1894 of *LNCS*, pages 411-424, Singapore, September 2000.

[CooperFCSP] M.C. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3):311-342, 2003.