Tutorials

Release 1.0.0

INRAE

Feb 28, 2022

CONTENTS

CHAPTER

ONE

WEIGHTED N-QUEEN PROBLEM

1.1 Brief description

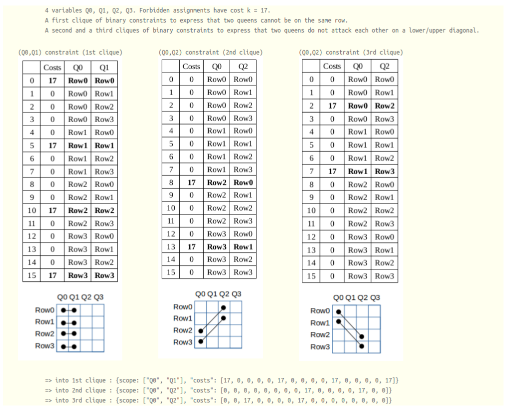
The problem consists in assigning N queens on a NxN chessboard with random weights in (1..N) associated to every cell such that each queen does not attack another queen and the sum of the weights of queen's selected cells is minimized.

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.

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
↪"]},
  functions: {
```

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

1.4 Python model generator

The following code using python3 interpreter will generate the previous example if called without argument. Otherwise the first argument is the number of queens N (e.g. “python3 queens.py 8”).

Note: Notice that the first lines of code (import and functions flatten and cfn) are needed by all the other tutorial examples.

queens.py

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

def flatten(x):
    result = []
    for el in x:
        if hasattr(el, "__iter__") and not isinstance(el, str) and not isinstance(el,␣
↪tuple) and not isinstance(el, dict):
            result.extend(flatten(el))
        else:
            result.append(el)
    return result

def cfn(problem, isMinimization, initPrimalBound, floatPrecision=0):
    globals_key_order = ["rhs", "capacity", "weights", "weightedvalues", "metric", "cost
↪", "bounds", "vars1", "vars2", "nb_states", "starts", "ends", "transitions", "nb_
↪symbols", "nb_values", "start", "terminals", "non_terminals", "min", "max", "values",
↪"defaultcost", "tuples", "comparator", "to"]
    print('{')
    print('\tproblem: { "name": "%s", "mustbe": "%s%.*f" },' % (problem["name"], "<" if␣
↪(isMinimization) else ">", floatPrecision, initPrimalBound))
    print('\tvariables: {', end='')
    for i,e in enumerate(problem["variables"]):
        if i > 0: print(', ', end='')
        print('"%s":' % e[0], end='')
        if isinstance(e[1], int):
            print(' %s' % e[1], end='')
        else:
            print('[', end='')
            for j,a in enumerate(e[1]):
                if j > 0: print(', ', end='')
                print('"%s"' % a, end='')
            print(']', end='')
    print('},')
    print( '\tfunctions: {')
    for i,e in enumerate(flatten(problem["functions"])):
        if i > 0: print(',')
        if e.get("name") is not None: print('\t\t"%s": {scope: [' % e.get("name"), end='
↪')
        else: print('\t\t{scope: [', end='')
        for j,x in enumerate(e.get("scope")):
```

(continues on next page)

(continued from previous page)

```
            if j > 0: print(', ', end='')
            print('"%s"' % x, end='')
        print('], ', end='')
        if e.get("type") is not None:
            print('"type:" %s, ' % e.get("type"), end='')
        if e.get("params") is not None:
            if isinstance(e.get("params"), dict):
                print('"params": {', end='')
                first = True
                for key in globals_key_order:
                    if key in e.get("params"):
                        if not first: print(', ', end='')
                        if isinstance(e.get("params")[key], str): print('"%s": "%s"' %␣
↪(str(key),str(e.get("params")[key]).replace("'", '"')), end='')
                        else: print('"%s": %s' % (str(key),str(e.get("params")[key]).
↪replace("'", '"')), end='')
                        first = False
                print ('}', end='')
            else: print('"params": %s, ' % str(e.get("params")).replace("'",'"'), end='')
        if e.get("defaultcost") is not None:
            print('"defaultcost:" %s, ' % e.get("defaultcost"), end='')
        if e.get("costs") is not None:
            print('"costs": ', end='')
            if isinstance(e.get("costs"), str):
                print('"%s"' % e.get("costs"), end='') # reuse future cost function by␣
↪giving its name here
            else:
                print('[', end='')
                for j,c in enumerate(e.get("costs")):
                    if j > 0: print(', ', end='')
                    if isinstance(c, str) and not c.isdigit():
                        print('"%s"' % c, end='')
                    else:
                        print('%s' % c, end='')
                print(']', end='')
        print('}', end='')
    print('}\n}')

def model(N, k):
    Var = ["Q" + str(i) for i in range(N)]
    Queen = {
        "name": str(N) + "-queen",
        "variables": [(Var[i], ["Row" + str(j) for j in range(N)]) for i in range(N)],
        "functions":
            [
             # permutation constraints expressed by a clique of binary constraints
             [{"scope": [Var[i], Var[j]], "costs": [0 if a != b else k for a in range(N)␣
↪for b in range(N)]} for i in range(N) for j in range(N) if (i < j)],
             # upper diagonal constraints
             [{"scope": [Var[i], Var[j]], "costs": [0 if a + i != b + j else k for a in␣
↪range(N) for b in range(N)]} for i in range(N) for j in range(N) if (i < j)],
             # lower diagonal constraints
```

(continues on next page)

(continued from previous page)

```
             [{"scope": [Var[i], Var[j]], "costs": [0 if a - i != b - j else k for a in␣
↪range(N) for b in range(N)]} for i in range(N) for j in range(N) if (i < j)],
             # random unary costs
             [{"scope": [Var[i]], "costs": [randint(1,N) for a in range(N)]} for i in␣
↪range(N)]
            ]
         }
    return Queen

if __name__ == '__main__':
    # read parameters
    N = int(sys.argv[1]) if len(sys.argv) > 1 else 4
    # infinite cost
    k = N**2+1
    # dump problem into JSON .cfn format for minimization
    cfn(model(N, k), True, k)
    # or for maximization
    #cfn(model(N, -k), False, -k)
```

CHAPTER

TWO

WEIGHTED LATIN SQUARE PROBLEM

2.1 Brief description

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

2.2 CFN model

We create NxN variables for all cells with domain size N. A hard AllDifferent global cost function is used to model a permutation for every row and every column. Random weights are generated for every cell and domain value. Forbidden assignments have cost k=N**3+1.

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
↪", "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}},
```

(continues on next page)

(continued from previous page)

```
    {scope: ["X0_3", "X1_3", "X2_3", "X3_3"], "type:" salldiff, "params": {"metric": "var
↪", "cost": 65}},

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

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

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

2.4 Python model generator

The following code using python3 interpreter will generate the previous example if called without argument. Otherwise the first argument is the dimension N of the chessboard (e.g. "python3 latinsquare.py 6").

latinsquare.py

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

(continues on next page)

(continued from previous page)

```
def flatten(x):
    result = []
    for el in x:
        if hasattr(el, "__iter__") and not isinstance(el, str) and not isinstance(el,␣
↪tuple) and not isinstance(el, dict):
            result.extend(flatten(el))
        else:
            result.append(el)
    return result

def cfn(problem, isMinimization, initPrimalBound, floatPrecision=0):
    globals_key_order = ["rhs", "capacity", "weights", "weightedvalues", "metric", "cost
↪", "bounds", "vars1", "vars2", "nb_states", "starts", "ends", "transitions", "nb_
↪symbols", "nb_values", "start", "terminals", "non_terminals", "min", "max", "values",
↪"defaultcost", "tuples", "comparator", "to"]
    print('{')
    print('\tproblem: { "name": "%s", "mustbe": "%s%.*f" },' % (problem["name"], "<" if␣
↪(isMinimization) else ">", floatPrecision, initPrimalBound))
    print('\tvariables: {', end='')
    for i,e in enumerate(problem["variables"]):
        if i > 0: print(', ', end='')
        print('"%s":' % e[0], end='')
        if isinstance(e[1], int):
            print(' %s' % e[1], end='')
        else:
            print('[', end='')
            for j,a in enumerate(e[1]):
                if j > 0: print(', ', end='')
                print('"%s"' % a, end='')
            print(']', end='')
    print('},')
    print( '\tfunctions: {')
    for i,e in enumerate(flatten(problem["functions"])):
        if i > 0: print(',')
        if e.get("name") is not None: print('\t\t"%s": {scope: [' % e.get("name"), end='
↪')
        else: print('\t\t{scope: [', end='')
        for j,x in enumerate(e.get("scope")):
            if j > 0: print(', ', end='')
            print('"%s"' % x, end='')
        print('], ', end='')
        if e.get("type") is not None:
            print('"type:" %s, ' % e.get("type"), end='')
        if e.get("params") is not None:
            if isinstance(e.get("params"), dict):
                print('"params": {', end='')
                first = True
                for key in globals_key_order:
                    if key in e.get("params"):
                        if not first: print(', ', end='')
                        if isinstance(e.get("params")[key], str): print('"%s": "%s"' %␣
↪(str(key),str(e.get("params")[key]).replace("'", '"')), end='')
```

(continues on next page)

(continued from previous page)

```
                            else: print('"%s": %s' % (str(key),str(e.get("params")[key]).
↪replace("'", '"')), end='')
                            first = False
                    print ('}', end='')
                else: print('"params": %s, ' % str(e.get("params")).replace("'",'"'), end='')
            if e.get("defaultcost") is not None:
                print('"defaultcost:" %s, ' % e.get("defaultcost"), end='')
            if e.get("costs") is not None:
                print('"costs": ', end='')
                if isinstance(e.get("costs"), str):
                    print('"%s"' % e.get("costs"), end='') # reuse future cost function by␣
↪giving its name here
                else:
                    print('[', end='')
                    for j,c in enumerate(e.get("costs")):
                        if j > 0: print(', ', end='')
                        if isinstance(c, str) and not c.isdigit():
                            print('"%s"' % c, end='')
                        else:
                            print('%s' % c, end='')
                    print(']', end='')
            print('}', end='')
    print('}\n}')

def model(N, k):
    Var = {(i,j): "X" + str(i) + "_" + str(j) for i in range(N) for j in range(N)}
    LatinSquare = {
        "name": "LatinSquare" + str(N),
        "variables": [(Var[(i,j)], N) for i in range(N) for j in range(N)],
        "functions":
            [# permutation constraints on rows
                [{"scope": [Var[(i,j)] for j in range(N)], "type": "salldiff", "params":
↪{"metric": "var", "cost": k}} for i in range(N)],
             # permutation constraints on columns
             [{"scope": [Var[(i,j)] for i in range(N)], "type": "salldiff", "params": {
↪"metric": "var", "cost": k}} for j in range(N)],
              # random unary costs on every cell
             [{"scope": [Var[(i,j)]], "costs": [randint(1, N) for a in range(N)]} for i␣
↪in range(N) for j in range(N)]
            ]
    }
    return LatinSquare

if __name__ == '__main__':
    # read parameters
    N = int(sys.argv[1]) if len(sys.argv) > 1 else 4
    # infinite cost
    k = N**3+1
    # dump problem into JSON .cfn format for minimization
    cfn(model(N, k), True, k)
```

CHAPTER THREE

RADIO LINK FREQUENCY ASSIGNMENT PROBLEM

3.1 Brief description

The problem consists in assigning frequencies to radio communication links in such a way that no interferences occurs. Domains are set of integers (non necessarily consecutives). Two types of constraints occur: (I) the absolute difference between two frequencies should be greater than a given number d_i (| x - y | > d_i), or (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. Cabon, B., de Givry, S., Lobjois, L., Schiex, T., Warners, J.P. Constraints (1999) 4: 79.

3.2 CFN model

We create N variables for every radio link with a given integer domain. Hard and soft binary cost functions express interference constraints with possible deviations. Unary cost functions are used to model mobility costs.

3.3 Data

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

3.4 Python model generator

The following code using python3 interpreter will generate the corresponding cost function network (e.g. "python3 rlfap.py var.txt dom.txt ctr.txt cst.txt").

rlfap.py

```
import sys

def flatten(x):
    result = []
    for el in x:
        if hasattr(el, "__iter__") and not isinstance(el, str) and not isinstance(el,␣
↪tuple) and not isinstance(el, dict):
```

(continues on next page)

(continued from previous page)

```
            result.extend(flatten(el))
        else:
            result.append(el)
    return result

def cfn(problem, isMinimization, initPrimalBound, floatPrecision=0):
    globals_key_order = ["rhs", "capacity", "weights", "weightedvalues", "metric", "cost
↪", "bounds", "vars1", "vars2", "nb_states", "starts", "ends", "transitions", "nb_
↪symbols", "nb_values", "start", "terminals", "non_terminals", "min", "max", "values",
↪"defaultcost", "tuples", "comparator", "to"]
    print('{')
    print('\tproblem: { "name": "%s", "mustbe": "%s%.*f" },' % (problem["name"], "<" if␣
↪(isMinimization) else ">", floatPrecision, initPrimalBound))
    print('\tvariables: {', end='')
    for i,e in enumerate(problem["variables"]):
        if i > 0: print(', ', end='')
        print('"%s":' % e[0], end='')
        if isinstance(e[1], int):
            print(' %s' % e[1], end='')
        else:
            print('[', end='')
            for j,a in enumerate(e[1]):
                if j > 0: print(', ', end='')
                print('"%s"' % a, end='')
            print(']', end='')
    print('},')
    print( '\tfunctions: {')
    for i,e in enumerate(flatten(problem["functions"])):
        if i > 0: print(',')
        if e.get("name") is not None: print('\t\t"%s": {scope: [' % e.get("name"), end='
↪')
        else: print('\t\t{scope: [', end='')
        for j,x in enumerate(e.get("scope")):
            if j > 0: print(', ', end='')
            print('"%s"' % x, end='')
        print('], ', end='')
        if e.get("type") is not None:
            print('"type:" %s, ' % e.get("type"), end='')
        if e.get("params") is not None:
            if isinstance(e.get("params"), dict):
                print('"params": {', end='')
                first = True
                for key in globals_key_order:
                    if key in e.get("params"):
                        if not first: print(', ', end='')
                        if isinstance(e.get("params")[key], str): print('"%s": "%s"' %␣
↪(str(key),str(e.get("params")[key]).replace("'", '"')), end='')
                        else: print('"%s": %s' % (str(key),str(e.get("params")[key]).
↪replace("'", '"')), end='')
                        first = False
                print ('}', end='')
            else: print('"params": %s, ' % str(e.get("params")).replace("'",'"'), end='')
```

(continues on next page)

(continued from previous page)

```
        if e.get("defaultcost") is not None:
            print('"defaultcost:" %s, ' % e.get("defaultcost"), end='')
        if e.get("costs") is not None:
            print('"costs": ', end='')
            if isinstance(e.get("costs"), str):
                print('"%s"' % e.get("costs"), end='') # reuse future cost function by␣
↪giving its name here
            else:
                print('[', end='')
                for j,c in enumerate(e.get("costs")):
                    if j > 0: print(', ', end='')
                    if isinstance(c, str) and not c.isdigit():
                        print('"%s"' % c, end='')
                    else:
                        print('%s' % c, end='')
                print(']', end='')
        print('}', end='')
    print('}\n}')

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 = 0

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

(continues on next page)

(continued from previous page)

```
            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))

def model(data):
    Var = {e[0]: "X" + str(e[0]) for e in data.var}
    Domain = {e[0]: e[1] for e in data.var}
    RLFAP = {
        "name": "RLFAP",
        "variables": [(Var[e[0]], ["f" + str(f) for f in data.dom[e[1]]]) for e in data.
↪var],
        "functions":
            [# hard and soft interference
             [{"scope": [Var[var1], Var[var2]], "costs": [0 if ((operand==">" and abs(a -
↪ b)>deviation) or (operand=="=" and abs(a - b)==deviation)) else data.cost.get("a
↪"+str(weight),data.top) for a in data.dom[Domain[var1]] for b in data.
↪dom[Domain[var2]]]} for (var1, var2, operand, deviation, weight) in data.ctr],
             # mobility costs
             [{"scope": [Var[e[0]]], "defaultcost": data.cost.get("b"+str(e[3]),data.
↪top), "costs": ["f" + str(e[2]), 0] if e[2] in data.dom[e[1]] else []} for e in data.
↪var if len(e)==4]
            ]
    }
    return RLFAP

if __name__ == '__main__':
    # read parameters
    if len(sys.argv) < 5: exit('Command line arguments are filenames: var.txt dom.txt␣
↪ctr.txt cst.txt')
    data = Data(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4])
    # dump problem into JSON .cfn format for minimization
    cfn(model(data), True, data.top)
```

CHAPTER

FOUR

FREQUENCY ASSIGNMENT PROBLEM WITH POLARIZATION

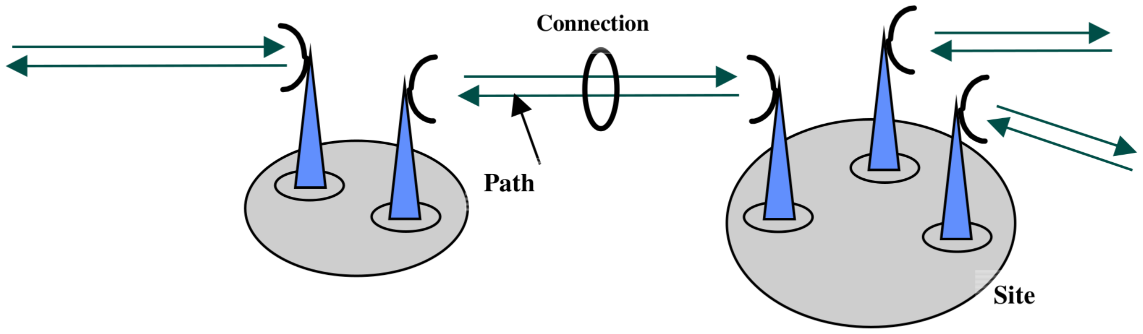
4.1 Brief description

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

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

Constraints (I) to (III) are mandatory constraints, while constraints (IV) can be relaxed. The goal is to find a feasible assignment with the smallest relaxation level l and which minimizes the number of violations of (IV) at lower levels. See ROADEF Challenge 2001.

Physical description and mathematical formulation



4.2 CFN model

In order to benefit from soft local consistencies on binary cost functions, we create a single variable to represent a pair (frequency,polarization) for every radio link.

4.3 Data

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

4.4 Python model generator

The following code using python3 interpreter will generate the corresponding cost function network (e.g. "python3 fapp.py exemple1.in 3"). You can also compile `fappeval.c` using "gcc -o fappeval fappeval.c" and download `sol2fapp.awk` in order to evaluate the solutions (e.g., "python3 fapp.py exemple1.in 3 | toulbar2 –stdin=cfn -s=3 | awk -f ./sol2fapp.awk - exemple1").

`fapp.py`

```
import sys

def flatten(x):
    result = []
    for el in x:
        if hasattr(el, "__iter__") and not isinstance(el, str) and not isinstance(el,␣
↪tuple) and not isinstance(el, dict):
            result.extend(flatten(el))
        else:
            result.append(el)
    return result

def cfn(problem, isMinimization, initPrimalBound, floatPrecision=0):
    globals_key_order = ["metric", "cost", "bounds", "vars1", "vars2", "nb_states",
↪"starts", "ends", "transitions", "nb_symbols", "nb_values", "start", "terminals", "non_
↪terminals", "min", "max", "values", "defaultcost", "tuples", "comparator", "to"]
    print('{')
    print('\tproblem: { "name": "%s", "mustbe": "%s%.*f" },' % (problem["name"], "<" if␣
↪(isMinimization) else ">", floatPrecision, initPrimalBound))
    print('\tvariables: {', end='')
    for i,e in enumerate(problem["variables"]):
        if i > 0: print(', ', end='')
        print('"%s":' % e[0], end='')
        if isinstance(e[1], int):
            print(' %s' % e[1], end='')
        else:
            print('[', end='')
            for j,a in enumerate(e[1]):
                if j > 0: print(', ', end='')
                print('"%s"' % a, end='')
```

(continues on next page)

(continued from previous page)

```
            print(']', end='')
    print('},')
    print( '\tfunctions: {')
    for i,e in enumerate(flatten(problem["functions"])):
        if i > 0: print(',')
        if e.get("name") is not None: print('\t\t"%s": {scope: [' % e.get("name"), end='
↪')
        else: print('\t\t{scope: [', end='')
        for j,x in enumerate(e.get("scope")):
            if j > 0: print(', ', end='')
            print('"%s"' % x, end='')
        print('], ', end='')
        if e.get("type") is not None:
            print('"type:" %s, ' % e.get("type"), end='')
        if e.get("params") is not None:
            if isinstance(e.get("params"), dict):
                print('"params": {', end='')
                first = True
                for key in globals_key_order:
                    if key in e.get("params"):
                        if not first: print(', ', end='')
                        if isinstance(e.get("params")[key], str): print('"%s": "%s"' %␣
↪(str(key),str(e.get("params")[key]).replace("'", '"')), end='')
                        else: print('"%s": %s' % (str(key),str(e.get("params")[key]).
↪replace("'", '"')), end='')
                        first = False
                print ('}', end='')
            else: print('"params": %s, ' % str(e.get("params")).replace("'",'"'), end='')
        if e.get("defaultcost") is not None:
            print('"defaultcost:" %s, ' % e.get("defaultcost"), end='')
        if e.get("costs") is not None:
            print('"costs": ', end='')
            if isinstance(e.get("costs"), str):
                print('"%s"' % e.get("costs"), end='') # reuse future cost function by␣
↪giving its name here
            else:
                print('[', end='')
                for j,c in enumerate(e.get("costs")):
                    if j > 0: print(', ', end='')
                    if isinstance(c, str) and not c.isdigit():
                        print('"%s"' % c, end='')
                    else:
                        print('%s' % c, end='')
                print(']', end='')
        print('}', end='')
    print('}\n}')

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

(continues on next page)

(continued from previous page)

```
        self.softeq = list()
        self.softne = list()
        self.nbsoft = 0
        self.top = 1
        self.cst = 0

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

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

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

def model(data, k):
    Var = {e[0]: "X" + str(e[0]) for e in data.var}
    Domain = {e[0]: e[1] for e in data.var}
    FAPP = {
```

(continues on next page)

(continued from previous page)

```
        "name": "FAPP",
        "variables": [(Var[e[0]], ["f" + str(f) + "p" + str(p) for (f,p) in e[1]]) for e␣
↪in data.var],
        "functions":
            [# hard constraints
             [{"scope": [Var[route1], Var[route2]], "costs": [0 if ((operand=="I" and␣
↪abs((f1 if vartype=="F" else p1) - (f2 if vartype=="F" else p2)) != deviation)
                                                                   or (operand=="E" and␣
↪abs((f1 if vartype=="F" else p1) - (f2 if vartype=="F" else p2)) == deviation)) else␣
↪data.top
                                                             for (f1,p1) in␣
↪Domain[route1] for (f2,p2) in Domain[route2]]}
              for (route1, route2, vartype, operand, deviation) in data.ctr],
             # soft equality constraints
             [{"scope": [Var[route1], Var[route2]], "costs": [0 if p1!=p2 or abs(f1 -␣
↪f2) >= deviations[i] else (data.top if i>=k else (1 if i<k-1 else 10*data.nbsoft))
                                                             for (f1, p1) in␣
↪Domain[route1] for (f2, p2) in Domain[route2]]}
                for i in range(11) for (route1, route2, deviations) in data.softeq],
             # soft inequality constraints
             [{"scope": [Var[route1], Var[route2]], "costs": [0 if p1==p2 or abs(f1 -␣
↪f2) >= deviations[i] else (data.top if i>=k else (1 if i<k-1 else 10*data.nbsoft))
                                                             for (f1, p1) in␣
↪Domain[route1] for (f2, p2) in Domain[route2]]}
                for i in range(11) for (route1, route2, deviations) in data.softne],
             # constant cost to be added corresponding to the relaxation level
             {"scope": [], "defaultcost": data.cst, "costs": []}
             ]
    }
    return FAPP

if __name__ == '__main__':
    # read parameters
    if len(sys.argv) < 2: exit('Command line argument is problem data filename and␣
↪relaxation level')
    k = int(sys.argv[2])
    data = Data(sys.argv[1], k)
    # dump problem into JSON .cfn format for minimization
    cfn(model(data, k), True, data.top)
```

CHAPTER

FIVE

MENDELIAN ERROR DETECTION PROBLEM

5.1 Brief description

The problem is to detect marker genotyping incompatibilities (Mendelian errors only) in complex pedigrees. The input is a pedigree data with partial observed genotyping data at a single locus. The problem is to assign genotypes (unordered pairs of alleles) to all individuals such that they are compatible with the Mendelian law of heredity and with the maximum number of genotyping data. Sanchez, M., de Givry, S. and Schiex, T. Constraints (2008) 13:130.

5.2 CFN model

We create N variables for every individual genotype with domain being all possible unordered pairs of existing alleles. Hard ternary cost functions express mendelian law of heredity. Unary cost functions are used to model potential genotyping errors.

5.3 Data

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

5.4 Python model generator

The following code using python3 interpreter will generate the corresponding cost function network (e.g. "python3 mendel.py simple.pre").

mendel.py

```
import sys

def flatten(x):
    result = []
    for el in x:
        if hasattr(el, "__iter__") and not isinstance(el, str) and not isinstance(el,␣
↪tuple) and not isinstance(el, dict):
            result.extend(flatten(el))
        else:
            result.append(el)
```

(continues on next page)

(continued from previous page)

```
    return result

def cfn(problem, isMinimization, initPrimalBound, floatPrecision=0):
    globals_key_order = ["rhs", "capacity", "weights", "weightedvalues", "metric", "cost
↪", "bounds", "vars1", "vars2", "nb_states", "starts", "ends", "transitions", "nb_
↪symbols", "nb_values", "start", "terminals", "non_terminals", "min", "max", "values",
↪"defaultcost", "tuples", "comparator", "to"]
    print('{')
    print('\tproblem: { "name": "%s", "mustbe": "%s%.*f" },' % (problem["name"], "<" if␣
↪(isMinimization) else ">", floatPrecision, initPrimalBound))
    print('\tvariables: {', end='')
    for i,e in enumerate(problem["variables"]):
        if i > 0: print(', ', end='')
        print('"%s":' % e[0], end='')
        if isinstance(e[1], int):
            print(' %s' % e[1], end='')
        else:
            print('[', end='')
            for j,a in enumerate(e[1]):
                if j > 0: print(', ', end='')
                print('"%s"' % a, end='')
            print(']', end='')
    print('},')
    print( '\tfunctions: {')
    for i,e in enumerate(flatten(problem["functions"])):
        if i > 0: print(',')
        if e.get("name") is not None: print('\t\t"%s": {scope: [' % e.get("name"), end='
↪')
        else: print('\t\t{scope: [', end='')
        for j,x in enumerate(e.get("scope")):
            if j > 0: print(', ', end='')
            print('"%s"' % x, end='')
        print('], ', end='')
        if e.get("type") is not None:
            print('"type:" %s, ' % e.get("type"), end='')
        if e.get("params") is not None:
            if isinstance(e.get("params"), dict):
                print('"params": {', end='')
                first = True
                for key in globals_key_order:
                    if key in e.get("params"):
                        if not first: print(', ', end='')
                        if isinstance(e.get("params")[key], str): print('"%s": "%s"' %␣
↪(str(key),str(e.get("params")[key]).replace("'", '"')), end='')
                        else: print('"%s": %s' % (str(key),str(e.get("params")[key]).
↪replace("'", '"')), end='')
                        first = False
                print ('}', end='')
            else: print('"params": %s, ' % str(e.get("params")).replace("'",'"'), end='')
        if e.get("defaultcost") is not None:
            print('"defaultcost:" %s, ' % e.get("defaultcost"), end='')
        if e.get("costs") is not None:
```

(continues on next page)

(continued from previous page)

```
            print('"costs": ', end='')
            if isinstance(e.get("costs"), str):
                print('"%s"' % e.get("costs"), end='') # reuse future cost function by␣
↪giving its name here
            else:
                print('[', end='')
                for j,c in enumerate(e.get("costs")):
                    if j > 0: print(', ', end='')
                    if isinstance(c, str) and not c.isdigit():
                        print('"%s"' % c, end='')
                    else:
                        print('%s' % c, end='')
                print(']', end='')
        print('}', end='')
    print('}\n}')

class Data:
    def __init__(self, ped):
        self.id = list()
        self.father = {}
        self.mother = {}
        self.alleles = {}
        self.freq = {}
        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.alleles[int(id)] = (int(allele1), int(allele2)) if int(allele1) <␣
↪int(allele2) else (int(allele2), int(allele1))
            if int(allele1) != 0: self.freq[int(allele1)] = self.freq.get(int(allele1),␣
↪0) + 1
            if int(allele2) != 0: self.freq[int(allele2)] = self.freq.get(int(allele2),␣
↪0) + 1
            if int(allele1) != 0 or int(allele2) != 0: self.obs += 1

def model(data, k):
    Var = {g: "g" + str(g) for g in data.id}
    Domain = ["a" + str(a1) + "a" + str(a2)  for a1 in data.freq for a2 in data.freq if␣
↪a1 <= a2]
    Mendel = {
        "name": "Mendel",
        "variables": [(Var[g], Domain) for g in data.id],
        "functions":
            [# mendelian law of heredity
             [{"scope": [Var[data.father[g]], Var[data.mother[g]], Var[g]],
               "costs": [0 if (a1 in (p1,p2) and a2 in (m1,m2)) or (a2 in (p1,p2) and a1␣
↪in (m1,m2)) else k
                         for p1 in data.freq for p2 in data.freq
```

(continues on next page)

(continued from previous page)

```
                            for m1 in data.freq for m2 in data.freq
                            for a1 in data.freq for a2 in data.freq if p1 <= p2 and m1 <=␣
↪m2 and a1 <= a2]}
                 for g in data.id if data.father.get(g, 0) != 0 and data.mother.get(g, 0) !
↪= 0],
                # observation costs
                [{"scope": [Var[g]],
                  "costs": [0 if (a1,a2) == data.alleles[g] else 1 for a1 in data.freq for␣
↪a2 in data.freq if a1 <= a2]}
                 for g in data.id if data.alleles[g][0] != 0 and data.alleles[g][1] != 0]
                ]
    }
    return Mendel

if __name__ == '__main__':
    # read parameters
    if len(sys.argv) < 2: exit('Command line arguments are PEDFILE filename: simple.pre')
    data = Data(sys.argv[1])
    # dump problem into JSON .cfn format for minimization
    cfn(model(data, data.obs + 1), True, data.obs + 1)
```

CHAPTER

SIX

BLOCK MODELING PROBLEM

6.1 Brief description

This is a clustering problem, occuring in social network analysis. The problem is to divide a given graph G into k clusters such that the interactions between clusters can be summarized by a k*k 0/1 matrix M: if M[i,j]=1 then all the nodes in cluster i should be connected to all the nodes in cluster j in G, else if M[i,j]=0 then there should be no edge between the nodes in G. The goal is to find a k-clustering and the associated matrix M minimizing 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.

6.2 CFN model

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

6.3 Data

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

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

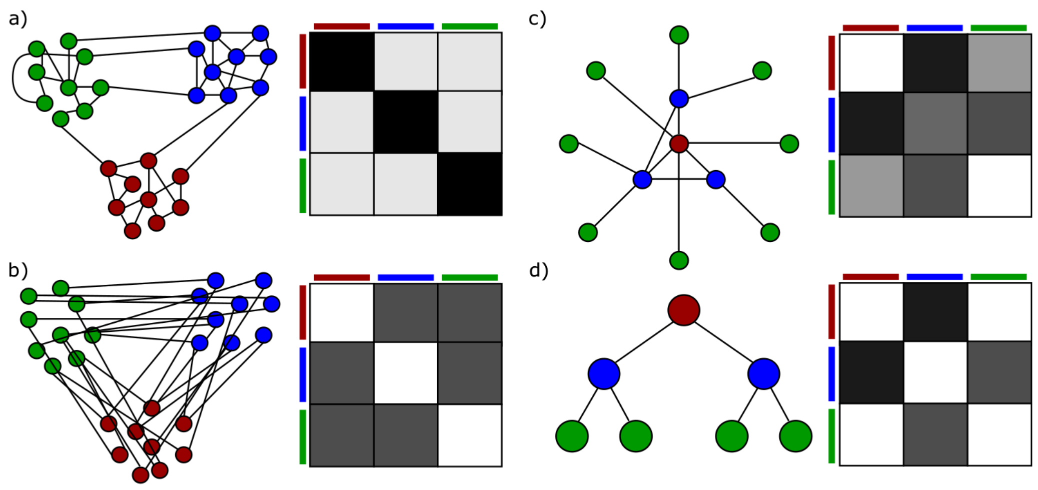
$$\begin{tikzcd}
1 \arrow[r] \arrow[dr] & 3 \arrow[r] \arrow[d, leftrightarrow] \arrow[loop above] & 5 \\
2 \arrow[r] \arrow[ur] & 4 \arrow[ur] \arrow[loop right] &
\end{tikzcd}$$

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

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

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

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



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

6.4 Python model generator

The following code using python3 interpreter will generate the corresponding cost function network (e.g. "python3 blockmodel.py simple.mat 3"). Download the AWK script `sol2block.awk` to pretty print the results (e.g., "python3 blockmodel.py simple.mat 3 | toulbar2 –stdin=cfn -s=3 | awk -f ./sol2block.awk").

`blockmodel.py`

```
import sys

def flatten(x):
    result = []
    for el in x:
        if hasattr(el, "__iter__") and not isinstance(el, str) and not isinstance(el,␣
↪tuple) and not isinstance(el, dict):
            result.extend(flatten(el))
        else:
            result.append(el)
    return result

def cfn(problem, isMinimization, initPrimalBound, floatPrecision=0):
    globals_key_order = ["rhs", "capacity", "weights", "weightedvalues", "metric", "cost
↪", "bounds", "vars1", "vars2", "nb_states", "starts", "ends", "transitions", "nb_
↪symbols", "nb_values", "start", "terminals", "non_terminals", "min", "max", "values",
↪"defaultcost", "tuples", "comparator", "to"]
    print('{')
    print('\tproblem: { "name": "%s", "mustbe": "%s%.*f" },' % (problem["name"], "<" if␣
↪(isMinimization) else ">", floatPrecision, initPrimalBound))
    print('\tvariables: {', end='')
    for i,e in enumerate(flatten(problem["variables"])):
        if i > 0: print(', ', end='')
```

(continues on next page)

(continued from previous page)

```
        print('"%s":' % e[0], end='')
        if isinstance(e[1], int):
            print(' %s' % e[1], end='')
        else:
            print('[', end='')
            for j,a in enumerate(e[1]):
                if j > 0: print(', ', end='')
                print('"%s"' % a, end='')
            print(']', end='')
    print('},')
    print( '\tfunctions: {')
    for i,e in enumerate(flatten(problem["functions"])):
        if i > 0: print(',')
        if e.get("name") is not None: print('\t\t"%s": {scope: [' % e.get("name"), end='
↪')
        else: print('\t\t{scope: [', end='')
        scope = {}
        for j,x in enumerate(e.get("scope")):
            if (x in scope): sys.exit(str(e) + '\nError: scope of function ' + str(i) +
↪' with the same variable twice is forbidden!')
            if j > 0: print(', ', end='')
            print('"%s"' % x, end='')
            scope[x]=j
        print('], ', end='')
        if e.get("type") is not None:
            print('"type:" %s, ' % e.get("type"), end='')
        if e.get("params") is not None:
            if isinstance(e.get("params"), dict):
                print('"params": {', end='')
                first = True
                for key in globals_key_order:
                    if key in e.get("params"):
                        if not first: print(', ', end='')
                        if isinstance(e.get("params")[key], str): print('"%s": "%s"' %␣
↪(str(key),str(e.get("params")[key]).replace('"', '')), end='')
                        else: print('"%s": %s' % (str(key),str(e.get("params")[key]).
↪replace('"', '')), end='')
                        first = False
                print ('}', end='')
            else: print('"params": %s, ' % str(e.get("params")).replace('"',''), end='')
        if e.get("defaultcost") is not None:
            print('"defaultcost:" %s, ' % e.get("defaultcost"), end='')
        if e.get("costs") is not None:
            print('"costs": ', end='')
            if isinstance(e.get("costs"), str):
                print('"%s"' % e.get("costs"), end='') # reuse future cost function by␣
↪giving its name here
            else:
                print('[', end='')
                for j,c in enumerate(e.get("costs")):
                    if j > 0: print(', ', end='')
                    if isinstance(c, str) and not c.isdigit():
```

(continues on next page)

(continued from previous page)

```
                        print('"%s"' % c, end='')
                    else:
                        print('%s' % c, end='')
                print(']', end='')
        print('}', end='')
    print('}\n}')

class Data:
    def __init__(self, filename, k):
        lines = open(filename).readlines()
        self.n = len(lines)
        self.matrix = [[int(e) for e in l.split(' ')] for l in lines]
        self.top = 1 + self.n*self.n

def model(data, K):
    Var = [(chr(65 + i) if data.n < 28 else "x" + str(i)) for i in range(data.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
    BlockModeling = {
        "name": "BlockModel_N" + str(data.n) + "_K" + str(K),
        "variables": [[("M_" + str(u) + "_" + str(v), 2) for u in range(K) for v in␣
↪range(K)],
                      [(Var[i], K)  for i in range(data.n)]],
        "functions":
            [
                # objective function
                [{"scope": ["M_" + str(u) + "_" + str(v), Var[i], Var[j]],
                  "costs": [1 if (u == k and v == l and data.matrix[i][j] != m)
                            else 0
                            for m in range(2)
                            for k in range(K)
                            for l in range(K)]}
                 for u in range(K) for v in range(K) for i in range(data.n) for j in␣
↪range(data.n) if i != j],

                # self-loops
                [{"scope": ["M_" + str(u) + "_" + str(u), Var[i]],
                  "costs": [1 if (u == k and data.matrix[i][i] != m)
                            else 0
                            for m in range(2)
                            for k in range(K)]}
                 for u in range(K) for i in range(data.n)],

                # breaking partial symmetries by fixing first (K-1) domain variables to␣
↪be assigned to cluster less than or equal to their index
                [{"scope": [Var[l]],
```

(continues on next page)

(continued from previous page)

```
                  "costs": [data.top if k > l else 0 for k in range(K)]}
                  for l in range(K-1)]
             ]
    }
    return BlockModeling

if __name__ == '__main__':
    # read parameters
    if len(sys.argv) < 2: exit('Command line argument is problem data filename and␣
↪number of blocks')
    K = int(sys.argv[2])
    data = Data(sys.argv[1], K)
    # dump problem into JSON .cfn format for minimization by toulbar2 solver
    cfn(model(data, K), True, data.top)
```

We improve the previous model by sorting node variables by decreasing out degree and removing the lower triangular matrix of M if the input graph is undirected (symmetric adjacency matrix).

blockmodel2.py

```

import sys

def flatten(x):
    result = []
    for el in x:
        if hasattr(el, "__iter__") and not isinstance(el, str) and not isinstance(el,␣
↪tuple) and not isinstance(el, dict):
            result.extend(flatten(el))
        else:
            result.append(el)
    return result

def cfn(problem, isMinimization, initPrimalBound, floatPrecision=0):
    globals_key_order = ["rhs", "capacity", "weights", "weightedvalues", "metric", "cost
↪", "bounds", "vars1", "vars2", "nb_states", "starts", "ends", "transitions", "nb_
↪symbols", "nb_values", "start", "terminals", "non_terminals", "min", "max", "values",
↪"defaultcost", "tuples", "comparator", "to"]
    print('{')
    print('\tproblem: { "name": "%s", "mustbe": "%s%.*f" },' % (problem["name"], "<" if␣
↪(isMinimization) else ">", floatPrecision, initPrimalBound))
    print('\tvariables: {', end='')
    for i,e in enumerate(flatten(problem["variables"])):
        if i > 0: print(', ', end='')
        print('"%s":' % e[0], end='')
        if isinstance(e[1], int):
            print(' %s' % e[1], end='')
        else:
            print('[', end='')
            for j,a in enumerate(e[1]):
                if j > 0: print(', ', end='')
                print('"%s"' % a, end='')
            print(']', end='')
```

(continues on next page)

(continued from previous page)

```
    print('},')
    print( '\tfunctions: {')
    for i,e in enumerate(flatten(problem["functions"])):
        if i > 0: print(',')
        if e.get("name") is not None: print('\t\t"%s": {scope: [' % e.get("name"), end='
↪')
        else: print('\t\t{scope: [', end='')
        scope = {}
        for j,x in enumerate(e.get("scope")):
            if (x in scope): sys.exit(str(e) + '\nError: scope of function ' + str(i) +
↪' with the same variable twice is forbidden!')
            if j > 0: print(', ', end='')
            print('"%s"' % x, end='')
            scope[x]=j
        print('], ', end='')
        if e.get("type") is not None:
            print('"type:" %s, ' % e.get("type"), end='')
        if e.get("params") is not None:
            if isinstance(e.get("params"), dict):
                print('"params": {', end='')
                first = True
                for key in globals_key_order:
                    if key in e.get("params"):
                        if not first: print(', ', end='')
                        if isinstance(e.get("params")[key], str): print('"%s": "%s"' %␣
↪(str(key),str(e.get("params")[key]).replace("'", '"')), end='')
                        else: print('"%s": %s' % (str(key),str(e.get("params")[key]).
↪replace("'", '"')), end='')
                        first = False
                print ('}', end='')
            else: print('"params": %s, ' % str(e.get("params")).replace("'",'"'), end='')
        if e.get("defaultcost") is not None:
            print('"defaultcost:" %s, ' % e.get("defaultcost"), end='')
        if e.get("costs") is not None:
            print('"costs": ', end='')
            if isinstance(e.get("costs"), str):
                print('"%s"' % e.get("costs"), end='') # reuse future cost function by␣
↪giving its name here
            else:
                print('[', end='')
                for j,c in enumerate(e.get("costs")):
                    if j > 0: print(', ', end='')
                    if isinstance(c, str) and not c.isdigit():
                        print('"%s"' % c, end='')
                    else:
                        print('%s' % c, end='')
                print(']', end='')
        print('}', end='')
    print('}\n}')

class Data:
    def __init__(self, filename, k):
```

(continues on next page)

(continued from previous page)

```
        lines = open(filename).readlines()
        self.n = len(lines)
        self.matrix = [[int(e) for e in l.split(' ')] for l in lines]
        self.top = 1 + self.n*self.n

def model(data, K):
    symmetric = all([data.matrix[i][j] == data.matrix[j][i] for i in range(data.n) for j␣
↪in range(data.n) if j>i])
    Var = [(chr(65 + i) if data.n < 28 else "x" + str(i)) for i in range(data.n)]

    # sort node variables by decreasing out degree
    degree = [(i, sum(data.matrix[i])) for i in range(data.n)]
    degree.sort(key=lambda tup: -tup[1])
    indexes = [e[0] for e in degree]

    BlockModeling = {
        "name": "BlockModel_N" + str(data.n) + "_K" + str(K) + "_Sym" + str(symmetric),
        # order node variables before matrix M variables
        # order matrix M variables starting from the main diagonal and moving away␣
↪progressively
        # if input graph is symmetric then keep only the upper triangular matrix of M
        "variables": [[("M_" + str(u) + "_" + str(u), 2) for u in range(K)],
                      [("M_" + str(u) + "_" + str(v), 2) for d in range(K) for u in␣
↪range(K) for v in range(K)
                       if u != v and (not symmetric or u < v) and abs(u - v) == d],
                      [(Var[indexes[i]], K)  for i in range(data.n)]],
        "functions":
            [
                # objective function
                # if input graph is symmetric then cost tables are also symmetric wrt␣
↪node variables
                [{"scope": ["M_" + str(u) + "_" + str(v), Var[indexes[i]],␣
↪Var[indexes[j]]],
                  "costs": [1 if (((u == k and v == l) or (symmetric and u == l and v ==␣
↪k))
                                  and data.matrix[indexes[i]][indexes[j]] != m)
                            else 0
                            for m in range(2)
                            for k in range(K)
                            for l in range(K)]}
                 for u in range(K) for v in range(K) for i in range(data.n) for j in␣
↪range(data.n)
                 if i != j and (not symmetric or u <= v)],

                # self-loops
                [{"scope": ["M_" + str(u) + "_" + str(u), Var[indexes[i]]],
                  "costs": [1 if (u == k and data.matrix[indexes[i]][indexes[i]] != m)
                            else 0
                            for m in range(2)
                            for k in range(K)]}
                 for u in range(K) for i in range(data.n)],
```

(continues on next page)

(continued from previous page)

```
                # breaking partial symmetries by fixing first (K-1) domain variables to␣
↪be assigned to cluster less than or equal to their index
                [{"scope": [Var[indexes[l]]],
                  "costs": [data.top if k > l else 0 for k in range(K)]}
                 for l in range(K-1)]
            ]
    }
    return BlockModeling

if __name__ == '__main__':
    # read parameters
    if len(sys.argv) < 2: exit('Command line argument is problem data filename and␣
↪number of blocks')
    K = int(sys.argv[2])
    data = Data(sys.argv[1], K)
    # dump problem into JSON .cfn format for minimization by toulbar2 solver
    cfn(model(data, K), True, data.top)
```

CHAPTER

SEVEN

AIRPLANE LANDING PROBLEM

7.1 Brief description (from CHOCO-SOLVER)

Given a set of planes and runways, the objective is to minimize the total weighted deviation from the target landing time for each plane. We consider only a single runway. There are costs associated with landing either earlier or later than a target landing time for each plane. Each plane has to land within its predetermined time window such that separation times between all pairs of planes are satisfied. J.E. Beasley, M. Krishnamoorthy, Y.M. Sharaiha and D. Abramson. Scheduling aircraft landings - the static case. Transportation Science, vol.34, 2000.

7.2 CFN model

We create N variables for every plane landing time. Binary cost functions express separation times between pairs of planes. Unary cost functions represent the weighted deviation for each plane.

7.3 Data

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

7.4 Python model and solve

The following code uses the pytoulbar2 module to generate the cost function network and solve it (e.g. "python3 airland.py airland1.txt"). Compile toulbar2 with "cmake -DPYTB2=ON . ; make" and copy the resulting module in pytoulbar2 folder "cp lib/Linux/pytb2.cpython* pytoulbar2".

`airland.py`

```python
import sys
import pytoulbar2

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

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

(continues on next page)

(continued from previous page)

```
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):
    Problem.AddFunction([i], [PC[i][0]*abs(a-LT[i][1]) for a in range(LT[i][0],␣
↪LT[i][2]+1)])

for i in range(N):
    for j in range(i+1,N):
        Problem.AddFunction([i, j], [top*(a+ST[i][j]>b and b+ST[j][i]>a) for a in␣
↪range(LT[i][0], LT[i][2]+1) for b in range(LT[j][0], LT[j][2]+1)])

Problem.Dump('airplane.cfn')
Problem.NoPreprocessing()
Problem.Solve()
```

CHAPTER

EIGHT

WAREHOUSE LOCATION PROBLEM

8.1 Brief description

See a problem description in CSPLib-034. We are dealing with the uncapacitated case only for the moment.

8.2 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). Channeling constraints link both of them. The objective function is linear and decomposed into one unary cost function per variable (maintenance and supply costs).

8.3 Data

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

8.4 Python model generator

The following code using python3 interpreter will generate the corresponding cost function network with a user given floating-point precision (e.g. "python3 warehouse.py cap44.txt 5").

warehouse.py

```

import sys

def flatten(x):
    result = []
    for el in x:
        if hasattr(el, "__iter__") and not isinstance(el, str) and not isinstance(el,␣
↪tuple) and not isinstance(el, dict):
            result.extend(flatten(el))
        else:
            result.append(el)
    return result

def cfn(problem, isMinimization, initPrimalBound, floatPrecision=0):
```

(continues on next page)

(continued from previous page)

```
    globals_key_order = ["rhs", "capacity", "weights", "weightedvalues", "metric", "cost
↪", "bounds", "vars1", "vars2", "nb_states", "starts", "ends", "transitions", "nb_
↪symbols", "nb_values", "start", "terminals", "non_terminals", "min", "max", "values",
↪"defaultcost", "tuples", "comparator", "to"]
    print('{')
    print('\tproblem: { "name": "%s", "mustbe": "%s%.*f" },' % (problem["name"], "<" if␣
↪(isMinimization) else ">", floatPrecision, initPrimalBound))
    print('\tvariables: {', end='')
    for i,e in enumerate(flatten(problem["variables"])):
        if i > 0: print(', ', end='')
        print('"%s":' % e[0], end='')
        if isinstance(e[1], int):
            print(' %s' % e[1], end='')
        else:
            print('[', end='')
            for j,a in enumerate(e[1]):
                if j > 0: print(', ', end='')
                print('"%s"' % a, end='')
            print(']', end='')
    print('},')
    print( '\tfunctions: {')
    for i,e in enumerate(flatten(problem["functions"])):
        if i > 0: print(',')
        if e.get("name") is not None: print('\t\t"%s": {scope: [' % e.get("name"), end='
↪')
        else: print('\t\t{scope: [', end='')
        scope = {}
        for j,x in enumerate(e.get("scope")):
            if (x in scope): sys.exit(str(e) + '\nError: scope of function ' + str(i) +
↪' with the same variable twice is forbidden!')
            if j > 0: print(', ', end='')
            print('"%s"' % x, end='')
            scope[x]=j
        print('], ', end='')
        if e.get("type") is not None:
            print('"type": %s, ' % e.get("type"), end='')
        if e.get("params") is not None:
            if isinstance(e.get("params"), dict):
                print('"params": {', end='')
                first = True
                for key in globals_key_order:
                    if key in e.get("params"):
                        if not first: print(', ', end='')
                        if isinstance(e.get("params")[key], str): print('"%s": "%s"' %␣
↪(str(key),str(e.get("params")[key]).replace("'", '"')), end='')
                        else: print('"%s": %s' % (str(key),str(e.get("params")[key]).
↪replace("'", '"')), end='')
                        first = False
                print ('}', end='')
            else: print('"params": %s, ' % str(e.get("params")).replace("'",'"'), end='')
        if e.get("defaultcost") is not None:
            print('"defaultcost:" %s, ' % e.get("defaultcost"), end='')
```

(continues on next page)

(continued from previous page)

```
        if e.get("costs") is not None:
            print('"costs": ', end='')
            if isinstance(e.get("costs"), str):
                print('"%s"' % e.get("costs"), end='') # reuse future cost function by␣
↪giving its name here
            else:
                print('[', end='')
                for j,c in enumerate(e.get("costs")):
                    if j > 0: print(', ', end='')
                    if isinstance(c, str) and not c.isdigit():
                        print('"%s"' % c, end='')
                    else:
                        print('%s' % c, end='')
                print(']', end='')
        print('}', end='')
    print('}\n}')

class Data:
    def __init__(self, filename):
        lines = open(filename).readlines()
        tokens = flatten([[e for e in l.split()] for l in lines])
        p = 0
        self.n = int(tokens[p])
        p += 1
        self.m = int(tokens[p])
        p += 1
        self.top = 1.  # sum of all costs plus one
        self.CostW = []  # maintenance cost of warehouses
        self.Capacity = []  # capacity limit of warehouses (not used)
        for i in range(self.n):
            self.Capacity.append(int(tokens[p]))
            p += 1
            self.CostW.append(float(tokens[p]))
            p += 1
        self.top += sum(self.CostW)
        self.Demand = []  # demand for each store (not used)
        self.CostS = []  # supply cost matrix
        for j in range(self.m):
            self.Demand.append(int(tokens[p]))
            p += 1
            self.CostS.append([])
            for i in range(self.n):
                self.CostS[j].append(float(tokens[p]))
                p += 1
            self.top += sum(self.CostS[-1])

def model(data):
    Warehouse = ["w" + str(i) for i in range(data.n)]
    Store = ["s" + str(i) for i in range(data.m)]
    Model = {
        "name": "Warehouse_" + str(data.n) + "_" + str(data.m),
        "variables": [(e, 2) for e in Warehouse],
```

(continues on next page)

(continued from previous page)

```
                    [(e, data.n)  for e in Store]],
        "functions":
            [
                # maintenance costs
                [{"scope": [Warehouse[i]],
                  "costs": [0, data.CostW[i]]}
                 for i in range(data.n)],
                # supply costs
                [{"scope": [Store[i]],
                  "costs": data.CostS[i]}
                 for i in range(data.m)],
                # channeling constraints between warehouses and stores
                [{"scope": [Warehouse[i], Store[j]],
                  "costs": [(data.top if (a == 0 and b == i) else 0) for a in range(2)␣
↪for b in range(data.n)]}
                 for i in range(data.n) for j in range(data.m)]
            ]
    }
    return Model

if __name__ == '__main__':
    # read parameters
    if len(sys.argv) < 2: exit('Command line argument is problem data filename and␣
↪number of precision digits after the floating point')
    data = Data(sys.argv[1])
    # dump problem into JSON .cfn format for minimization
    cfn(model(data), True, data.top, int(sys.argv[2]))
```

8.5 Python model and solve using pytoulbar2

The following code uses the pytoulbar2 module to generate the cost function network and solve it (e.g. "python3 warehouse2.py cap44.txt 1" found optimum value equal to 10349757). Other instances are available here in cfn format. Compile toulbar2 with "cmake -DPYTB2=ON . ; make" and copy the resulting module in pytoulbar2 folder "cp lib/Linux/pytb2.cpython* pytoulbar2".

warehouse2.py

```

import sys
import pytoulbar2

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

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

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

(continues on next page)

(continued from previous page)

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

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

(continues on next page)

(continued from previous page)

```
        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.Option.FullEAC = False
Problem.Option.showSolutions = False
Problem.Solve()
```

CHAPTER

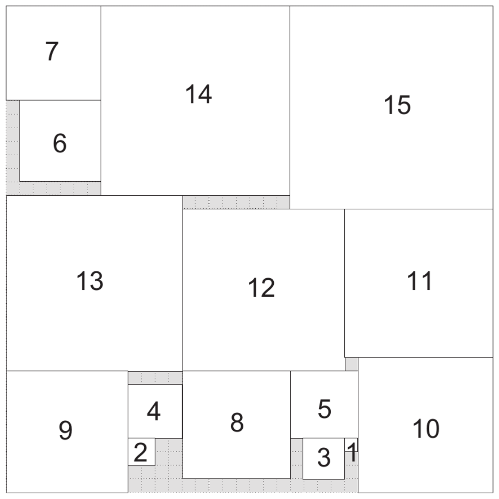
NINE

SQUARE PACKING PROBLEM

9.1 Brief description

Find a packing of squares of size 1×1, 2×2,. . . , NxN into a given container square SxS without overlaps. See a problem description in CSPLib-009. Results up to N=56 are given here.

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



9.2 CFN model

We create an integer variable of domain size (S-i)x(S-i) for each square i in [0,N-1] of size i+1 representing its top-left position in the container. Its value modulo (S-i) gives the x-coordinate, whereas its value divided by (S-i) gives the y-coordinate. We have binary constraints to forbid any overlapping pair of squares. We make the problem a pure satisfaction problem by fixing S. The initial upper bound is 1.

9.3 Python model generator

The following code using python3 interpreter will generate the corresponding cost function network (e.g. “python3 square.py 3 5”).

square.py

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

def flatten(x):
    result = []
    for el in x:
        if hasattr(el, "__iter__") and not isinstance(el, str) and not isinstance(el,␣
↪tuple) and not isinstance(el, dict):
            result.extend(flatten(el))
        else:
            result.append(el)
    return result

def cfn(problem, isMinimization, initPrimalBound, floatPrecision=0):
    globals_key_order = ["rhs", "capacity", "weights", "weightedvalues", "metric", "cost
↪", "bounds", "vars1", "vars2", "nb_states", "starts", "ends", "transitions", "nb_
↪symbols", "nb_values", "start", "terminals", "non_terminals", "min", "max", "values",
↪"defaultcost", "tuples", "comparator", "to"]
    print('{')
    print('\tproblem: { "name": "%s", "mustbe": "%s%.*f" },' % (problem["name"], "<" if␣
↪(isMinimization) else ">", floatPrecision, initPrimalBound))
    print('\tvariables: {', end='')
    for i,e in enumerate(flatten(problem["variables"])):
        if i > 0: print(', ', end='')
        print('"%s":' % e[0], end='')
        if isinstance(e[1], int):
            print(' %s' % e[1], end='')
        else:
            print('[', end='')
            for j,a in enumerate(e[1]):
                if j > 0: print(', ', end='')
                print('"%s"' % a, end='')
            print(']', end='')
    print('},')
    print( '\tfunctions: {')
    for i,e in enumerate(flatten(problem["functions"])):
```

(continues on next page)

(continued from previous page)

```
        if i > 0: print(',')
        if e.get("name") is not None: print('\t\t"%s": {scope: [' % e.get("name"), end='
↪')
        else: print('\t\t{scope: [', end='')
        scope = {}
        for j,x in enumerate(e.get("scope")):
            if (x in scope): sys.exit(str(e) + '\nError: scope of function ' + str(i) +
↪' with the same variable twice is forbidden!')
            if j > 0: print(', ', end='')
            print('"%s"' % x, end='')
            scope[x]=j
        print('], ', end='')
        if e.get("type") is not None:
            print('"type": %s, ' % e.get("type"), end='')
        if e.get("params") is not None:
            if isinstance(e.get("params"), dict):
                print('"params": {', end='')
                first = True
                for key in globals_key_order:
                    if key in e.get("params"):
                        if not first: print(', ', end='')
                        if isinstance(e.get("params")[key], str): print('"%s": "%s"' %␣
↪(str(key),str(e.get("params")[key]).replace("'", '"')), end='')
                        else: print('"%s": %s' % (str(key),str(e.get("params")[key]).
↪replace("'", '"')), end='')
                        first = False
                print ('}', end='')
            else: print('"params": %s, ' % str(e.get("params")).replace("'",'"'), end='')
        if e.get("defaultcost") is not None:
            print('"defaultcost:" %s, ' % e.get("defaultcost"), end='')
        if e.get("costs") is not None:
            print('"costs": ', end='')
            if isinstance(e.get("costs"), str):
                print('"%s"' % e.get("costs"), end='') # reuse future cost function by␣
↪giving its name here
            else:
                print('[', end='')
                for j,c in enumerate(e.get("costs")):
                    if j > 0: print(', ', end='')
                    if isinstance(c, str) and not c.isdigit():
                        print('"%s"' % c, end='')
                    else:
                        print('%s' % c, end='')
                print(']', end='')
        print('}', end='')
    print('}\n}')

def model(N, S, top):
    Var = ["sq" + str(i+1) for i in range(N)]
    Model = {
        "name": "SquarePacking" + str(N) + "_" + str(S),
        "variables": [(Var[i], (S-i)*(S-i)) for i in range(N)],
```

(continues on next page)

(continued from previous page)

```
        "functions":
            [
             # no overlapping constraint
             [{"scope": [Var[i], Var[j]], "costs": [(0 if ((a%(S-i)) + i + 1 <= (b%(S-
↪j))) or ((b%(S-j)) + j + 1 <= (a%(S-i))) or (int(a/(S-i)) + i + 1 <= int(b/(S-j))) or␣
↪(int(b/(S-j)) + j + 1 <= int(a/(S-i))) else top) for a in range((S-i)*(S-i)) for b in␣
↪range((S-j)*(S-j))]} for i in range(N) for j in range(N) if (i < j)]
            ]
        }
    return Model

if __name__ == '__main__':
    # read parameters
    N = int(sys.argv[1])
    S = int(sys.argv[2])
    # infinite cost
    top = 1
    # dump problem into JSON .cfn format for minimization
    cfn(model(N, S, top), True, top)
```

9.4 Python model and solve using pytoulbar2

The following code uses the pytoulbar2 module to generate the cost function network and solve it (e.g. "python3 square2.py 3 5"). Compile toulbar2 with "cmake -DPYTB2=ON . ; make" and copy the resulting module in pytoulbar2 folder "cp lib/Linux/pytb2.cpython* pytoulbar2".

square2.py

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

import pytoulbar2

N = int(sys.argv[1])
S = int(sys.argv[2])

top = 1

Problem = pytoulbar2.CFN(top)

for i in range(N):
    Problem.AddVariable('sq' + str(i+1), range((S-i)*(S-i)))

for i in range(N):
    for j in range(i+1,N):
        Problem.AddFunction([i, j], [0 if ((a%(S-i)) + i + 1 <= (b%(S-j))) or ((b%(S-j))␣
↪+ j + 1 <= (a%(S-i))) or (int(a/(S-i)) + i + 1 <= int(b/(S-j))) or (int(b/(S-j)) + j +␣
↪1 <= int(a/(S-i))) else top for a in range((S-i)*(S-i)) for b in range((S-j)*(S-j))])
```

(continues on next page)

(continued from previous page)

```
#Problem.Dump('square.cfn')
Problem.Option.FullEAC = False
Problem.Option.showSolutions = True
Problem.Solve()
```

9.5 C++ program using libtb2.so

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

    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);
```

(continues on next page)

(continued from previous page)

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

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

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

    tb2checkOptions();
    if (solver->solve()) {
            vector<Value> sol;
            solver->getSolution(sol);
                for (int y=0; y<S; y++) {
                for (int x=0; x<S; x++) {
                    char c = ' ';
                    for (int i=0; i<N; i++) {
                        if (x >= (sol[i]%(S-i)) && x < (sol[i]%(S-i) ) + i + 1 && y >=␣
↪(sol[i]/(S-i)) && y < (sol[i]/(S-i)) + i + 1) {
                            c = 65+i;
                            break;
                        }
                     }
                     cout << c;
                }
                cout << endl;
            }
    } else {
            cout << "No solution found!" << endl;
    }

    delete solver;
    return 0;
}
```

CHAPTER

TEN

SQUARE SOFT PACKING PROBLEM

10.1 Brief description

Find a packing of squares of size 1×1, 2×2,. . . , NxN into a given container square SxS minimizing total sum of overlaps.

10.2 CFN model

We reuse the *Square packing problem* model except that binary constraints are replaced by cost functions returning the overlapping size or zero if no overlaps. The initial upper bound is a worst-case upper estimation of total sum of overlaps.

10.3 Python model generator

The following code using python3 interpreter will generate the corresponding cost function network (e.g. "python3 squaresoft 10 20").

squaresoft.py

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

def flatten(x):
    result = []
    for el in x:
        if hasattr(el, "__iter__") and not isinstance(el, str) and not isinstance(el,␣
↪tuple) and not isinstance(el, dict):
            result.extend(flatten(el))
        else:
            result.append(el)
    return result

def cfn(problem, isMinimization, initPrimalBound, floatPrecision=0):
    globals_key_order = ["rhs", "capacity", "weights", "weightedvalues", "metric", "cost
↪", "bounds", "vars1", "vars2", "nb_states", "starts", "ends", "transitions", "nb_
↪symbols", "nb_values", "start", "terminals", "non_terminals", "min", "max", "values",
↪"defaultcost", "tuples", "comparator", "to"]
```

(continues on next page)

(continued from previous page)

```
    print('{')
    print('\tproblem: { "name": "%s", "mustbe": "%s%.*f" },' % (problem["name"], "<" if␣
↪(isMinimization) else ">", floatPrecision, initPrimalBound))
    print('\tvariables: {', end='')
    for i,e in enumerate(flatten(problem["variables"])):
        if i > 0: print(', ', end='')
        print('"%s":' % e[0], end='')
        if isinstance(e[1], int):
            print(' %s' % e[1], end='')
        else:
            print('[', end='')
            for j,a in enumerate(e[1]):
                if j > 0: print(', ', end='')
                print('"%s"' % a, end='')
            print(']', end='')
    print('},')
    print( '\tfunctions: {')
    for i,e in enumerate(flatten(problem["functions"])):
        if i > 0: print(',')
        if e.get("name") is not None: print('\t\t"%s": {scope: [' % e.get("name"), end='
↪')
        else: print('\t\t{scope: [', end='')
        scope = {}
        for j,x in enumerate(e.get("scope")):
            if (x in scope): sys.exit(str(e) + '\nError: scope of function ' + str(i) +
↪' with the same variable twice is forbidden!')
            if j > 0: print(', ', end='')
            print('"%s"' % x, end='')
            scope[x]=j
        print('], ', end='')
        if e.get("type") is not None:
            print('"type": %s, ' % e.get("type"), end='')
        if e.get("params") is not None:
            if isinstance(e.get("params"), dict):
                print('"params": {', end='')
                first = True
                for key in globals_key_order:
                    if key in e.get("params"):
                        if not first: print(', ', end='')
                        if isinstance(e.get("params")[key], str): print('"%s": "%s"' %␣
↪(str(key),str(e.get("params")[key]).replace("'", '"')), end='')
                        else: print('"%s": %s' % (str(key),str(e.get("params")[key]).
↪replace("'", '"')), end='')
                        first = False
                print ('}', end='')
            else: print('"params": %s, ' % str(e.get("params")).replace("'",'"'), end='')
        if e.get("defaultcost") is not None:
            print('"defaultcost:" %s, ' % e.get("defaultcost"), end='')
        if e.get("costs") is not None:
            print('"costs": ', end='')
            if isinstance(e.get("costs"), str):
                print('"%s"' % e.get("costs"), end='') # reuse future cost function by␣
↪giving its name here
```

(continues on next page)

(continued from previous page)

```
            else:
                print('[', end='')
                for j,c in enumerate(e.get("costs")):
                    if j > 0: print(', ', end='')
                    if isinstance(c, str) and not c.isdigit():
                        print('"%s"' % c, end='')
                    else:
                        print('%s' % c, end='')
                print(']', end='')
        print('}', end='')
    print('}\n}')

def model(N, S, top):
    Var = ["sq" + str(i+1) for i in range(N)]
    Model = {
        "name": "SquarePacking" + str(N) + "_" + str(S),
        "variables": [(Var[i], (S-i)*(S-i)) for i in range(N)],
        "functions":
            [
             # no overlapping constraint
             [{"scope": [Var[i], Var[j]], "costs": [(0 if ((a%(S-i)) + i + 1 <= (b%(S-
↪j))) or ((b%(S-j)) + j + 1 <= (a%(S-i))) or (int(a/(S-i)) + i + 1 <= int(b/(S-j))) or␣
↪(int(b/(S-j)) + j + 1 <= int(a/(S-i))) else min((a%(S-i)) + i + 1 - (b%(S-j)), (b%(S-
↪j)) + j + 1 - (a%(S-i))) * min(int(a/(S-i)) + i + 1 - int(b/(S-j)), int(b/(S-j)) + j +␣
↪1 - int(a/(S-i)))) for a in range((S-i)*(S-i)) for b in range((S-j)*(S-j))]} for i in␣
↪range(N) for j in range(N) if (i < j)]
            ]
        }
    return Model

if __name__ == '__main__':
    # read parameters
    N = int(sys.argv[1])
    S = int(sys.argv[2])
    # infinite cost
    top = int((N*N*(N-1)*(2*N-1))/6 + 1)
    # dump problem into JSON .cfn format for minimization
    cfn(model(N, S, top), True, top)
```

10.4 C++ program using libtb2.so

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

squaresoft.cpp

(continues on next page)

(continued from previous page)

```
/**
 * 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

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

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

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

(continues on next page)

(continued from previous page)

```
    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;
    return 0;
}
```

CHAPTER

ELEVEN

LEARNING TO PLAY THE SUDOKU

CHAPTER

TWELVE

RENAULT CAR CONFIGURATION SYSTEM: LEARNING USER PREFERENCES