

toulbar2 Documentation

Release 1.0.0

INRAE

Mar 29, 2022

1	Presentation	3
1.1	About toulbar2	3
1.2	Authors	3
1.3	Citations.	3
1.4	Acknowledgements	4
1.5	License.	4
2	Downloads	5
3	Documentation	7
3.1	User and reference manuals	7
3.2	Input formats	7
3.3	Some examples	7
3.4	Command line arguments	8
4	Publications	15
4.1	Related publications	15
4.2	Specific Events.	18
5	Tutorials	19
5.1	Weighted n-queen problem	19
5.2	Weighted latin square problem	23
5.3	Radio link frequency assignment problem.	26
5.4	Frequency assignment problem with polarization	29
5.5	Mendelian error detection problem.	33
5.6	Block modeling problem.	36
5.7	Airplane landing problem	43
5.8	Warehouse location problem	44
5.9	Square packing problem	48
5.10	Square soft packing problem	53
5.11	Learning to play the Sudoku	57
5.12	Renault car configuration system: learning user preferences	57

toulbar2 is an exact solver for cost function networks.

Presentation

1.1 About toulbar2

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 performances. See [Chapter 4](#).

toulbar2 won several competitions on Max-CSP ([CPAI08](#)) and probabilistic graphical models ([UAI 2008](#), [2010](#), [2014](#) 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](#)).

1.2 Authors

toulbar2 was originally developed by Toulouse (INRAE MIAT) and Barcelona (UPC, IIIA-CSIC) teams, hence the name of the solver. Additional global cost functions were provided by the *Chinese University of Hong Kong* and Caen University (GREYC). It also includes codes from Marseille University (LSIS, tree decomposition heuristics) and *Ecole des Ponts ParisTech* (CER-MICS/LIGM, [INCOP](#) local search solver).

A Python interface is now available. Install it with “python3 -m pip install pytoulbar2”. See examples in [Chapter 5](#) using `pytoulbar2` module. An older version is part of [Numberjack](#) (Insight - University College Cork). A portfolio approach dedicated to UAI format instances is available [here](#).

toulbar2 is currently maintained by Simon de Givry (simon.de-givry@inrae.fr) and hosted on [GitHub](#).

1.3 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 H.M. 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

1.4 Acknowledgements

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 [Ficololo](#) from 2011 to 2014, and ANR-16-CE40-0028 [DemoGraph](#) from 2017 to 2021) and a PHC PROCORE project number 28680VH (from 2013 to 2015). It is currently supported by ANITI ANR-19-P3IA-0004 (2019-).

1.5 License

MIT License

Copyright (c) 2019 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.

Downloads

Latest release toulbar2 binaries

[Linux 64bit](#) | [MacOs 64bit](#) | [Windows 64bit](#)

Open-source code

[github](#)

3.1 User and reference manuals

- User manual
- Reference manual

3.2 Input formats

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

- Cost Function Network format (`.cfn` file extensions)
- 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)
- Weigthed Partial Max-SAT (`.cnf`/`.wcnf` file extension)
- Quadratic Unconstrained Pseudo-Boolean Optimization (`.qpbo` file extension)
- Pseudo-Boolean Optimization (`.opb` file extension)

3.3 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 [here](#) and [there](#).

3.4 Command line arguments

See 'Available options' below :

```
c /toulbar2/build/bin/Linux/toulbar2 version : 1.1.1-139-gacc22b68-master
(1625740071), copyright (c) 2006-2020, toulbar2 team
*****
* ToulBar2 Help Message *
*****

Command line is:
toulbar2 problem_filename [options]

Available problem formats (specified by the filename extension) are:
  *.cfn : Cost Function Network format (see toulbar2 web site)
  *.wcsp : Weighted CSP format (see toulbar2 web site)
  *.wcnf : Weighted Partial Max-SAT format (see Max-SAT Evaluation)
  *.cnf : (Max-)SAT format
  *.qpbo : quadratic pseudo-Boolean optimization (unconstrained quadratic
programming) format (see also option -qpmult)
  *.opb : pseudo-Boolean optimization format
  *.uai : Bayesian network and Markov Random Field format (see UAI'08
Evaluation) followed by an optional evidence filename (performs MPE task, see
-logz for PR task, and write its solution in file .MPE or .PR using the same
directory as toulbar2)
  *.LG : Bayesian network and Markov Random Field format using logarithms
instead of probabilities
  *.xml : CSP and weighted CSP in XML format XCSP 2.1 (constraints in
extension only)
  *.pre : pedigree format (see doc/MendelSoft.txt for Mendelian error
correction)
  *.pre *.map : pedigree and genetic map formats (see
doc/HaplotypeHalfSib.txt for haplotype reconstruction in half-sib families)
  *.bep : satellite scheduling format (CHOCO benchmark)

  *.order : variable elimination order
  *.cov : tree decomposition given by a list of clusters in topological
order of a rooted forest,
    each line contains a cluster number, then a cluster parent number with
-1 for the root(s) cluster(s), followed by a list of variable indexes
  *.dec : a list of overlapping clusters without the running intersection
property used by VNS-like methods,
    each line contains a list of variable indexes
  *.sol : initial solution for the problem (given as initial upperbound
plus one and as default value heuristic, or only as initial upperbound if
option -x: is added)

Note: cfn, cnf, LG, qpbo, opb, uai, wcnf, wcsp formats can be read in gzip'd
or xz compressed format, e.g., toulbar2 problem.cfn.xz
Warning! File formats are recognized by filename extensions. To change the
default file format extension, use option --old_ext=".new" Examples:
--cfn_ext='.json' --wcspgz_ext='.wgz' --sol_ext='.sol2'

Available options are (use symbol ":" after an option to remove a default
```

```

option):
  -help : shows this help message
  -ub=[decimal] : initial problem upperbound (default value is
512409557603043100)
  -agap=[decimal] : stop search if the absolute optimality gap reduces below
the given value (provides guaranteed approximation) (default value is 0)
  -rgap=[double] : stop search if the relative optimality gap reduces below
the given value (provides guaranteed approximation) (default value is 0)
  -v=[integer] : verbosity level
  -s=[integer] : shows each solution found. 1 prints value numbers, 2 prints
value names, 3 prints also variable names (default 1)
  -w=[filename] : writes last/all solutions in filename (or "sol" if no
parameter is given)
  -w=[integer] : 1 writes value numbers, 2 writes value names, 3 writes also
variable names (default 1)
  -precision=[integer] defines the number of digits that should be
representable on probabilities in uai/pre files (default value is 7)
  -qpmult=[double] defines coefficient multiplier for quadratic terms
(default value is 2)
  -timer=[integer] : CPU time limit in seconds
  -bt=[integer] : limit on the number of backtracks (9223372036854775807 by
default)
  -seed=[integer] : random seed non-negative value or use current time if a
negative value is given (default value is 1)
  --stdin=[format] : read file from pipe ; e.g., cat example.wcsp | toulbar2
--stdin=wcsp
  -var=[integer] : searches by branching only on the first -the given value-
decision variables, assuming the remaining variables are intermediate
variables completely assigned by the decision variables (use a zero if all
variables are decision variables) (default value is 0)
  -b : searches using binary branching always instead of binary branching
for interval domains and n-ary branching for enumerated domains (default
option)
  -svo : searches using a static variable ordering heuristic (same order as
DAC)
  -c : searches using binary branching with last conflict backjumping
variable ordering heuristic (default option)
  -q=[integer] : weighted degree variable ordering heuristic if the number
of cost functions is less than the given value (default value is 1000000)
  -m=[integer] : variable ordering heuristic based on mean (m=1) or median
(m=2) costs (in conjunction with weighted degree heuristic -q) (default value
is 0)
  -d=[integer] : searches using dichotomic branching (d=1 splitting in the
middle of domain range, d=2 splitting in the middle of sorted unary costs)
instead of binary branching when current domain size is strictly greater than
10 (default value is 1)
  -sortd : sorts domains based on increasing unary costs (warning! works
only for binary WCSPs)
  -sortc : sorts constraints based on lexicographic ordering (1), decreasing
DAC ordering (2), decreasing constraint tightness (3), DAC then tightness
(4), tightness then DAC (5), randomly (6) or the opposite order if using a
negative value (default value is 2)
  -solr : solution-based phase saving (default option)
  -e=[integer] : boosting search with variable elimination of small degree
(less than or equal to 3) (default value is 3)
  -p=[integer] : preprocessing only: general variable elimination of degree
less than or equal to the given value (default value is -1)
  -t=[integer] : preprocessing only: simulates restricted path consistency
by adding ternary cost functions on triangles of binary cost functions within
a given maximum space limit (in MB)
  -f=[integer] : preprocessing only: variable elimination of functional
(f=1) (resp. bijective (f=2)) variables (default value is 1)
  -dec : preprocessing only: pairwise decomposition of cost functions with
arity >=3 into smaller arity cost functions (default option)

```

```

    -n=[integer] : preprocessing only: projects n-ary cost functions on all
binary cost functions if n is lower than the given value (default value is
10)
    -mst : maximum spanning tree DAC ordering
    -nopre : removes all preprocessing options (equivalent to -e: -p: -t: -f:
-dec: -n: -mst: -dee: -trws:)
    -o : ensures optimal worst-case time complexity of DAC and EAC (can be
slower in practice)
    -k=[integer] : soft local consistency level (NC with Strong NIC for global
cost functions=0, (G)AC=1, D(G)AC=2, FD(G)AC=3, (weak) ED(G)AC=4) (default
value is 4)
    -dee=[integer] : restricted dead-end elimination (value pruning by
dominance rule from EAC value (dee>=1 and dee<=3)) and soft neighborhood
substitutability (in preprocessing (dee=2 or dee=4) or during search (dee=3))
(default value is 1)
    -l=[integer] : limited discrepancy search, use a negative value to stop
the search after the given absolute number of discrepancies has been explored
(discrepancy bound = 4 by default)
    -L=[integer] : randomized (quasi-random variable ordering) search with
restart (maximum number of nodes/VNS restarts = 10000 by default)
    -i=["string"] : initial upperbound found by INCOP local search solver.
    string parameter is optional, using "0 1 3 idwa 100000 cv v 0 200 1 0
0" by default with the following meaning:
    stoppinglowerbound randomseed nbiterations method nbmoves
neighborhoodchoice neighborhoodchoice2 minnbneighbors maxnbneighbors
neighborhoodchoice3 autotuning tracemode
    -vns : unified decomposition guided variable neighborhood search (a
problem decomposition can be given as *.dec, *.cov, or *.order input files or
using tree decomposition options such as -O)
    -vnsini=[integer] : initial solution for VNS-like methods found (-1) at
random, (-2) min domain values, (-3) max domain values, (-4) first solution
found by a complete method, (k=0 or more) tree search with k discrepancy max
(-4 by default)
    -ldsmin=[integer] : minimum discrepancy for VNS-like methods (1 by
default)
    -ldsmax=[integer] : maximum discrepancy for VNS-like methods (number of
problem variables multiplied by maximum domain size -1 by default)
    -ldsinc=[integer] : discrepancy increment strategy for VNS-like methods
using (1) Add1, (2) Mult2, (3) Luby operator (2 by default)
    -kmin=[integer] : minimum neighborhood size for VNS-like methods (4 by
default)
    -kmax=[integer] : maximum neighborhood size for VNS-like methods (number
of problem variables by default)
    -kinc=[integer] : neighborhood size increment strategy for VNS-like
methods using (1) Add1, (2) Mult2, (3) Luby operator (4) Add1/Jump (4 by
default)
    -best=[integer] : stop VNS-like methods if a better solution is found
(default value is 0)

    -z=[filename] : saves problem in wcsp (by default) or cfn format (see
below) in filename (or "problem.wcsp/.cfn" if no parameter is given)
    writes also the graphviz dot file and the degree
distribution of the input problem (wcsp format only)
    -z=[integer] : 1 or 3: saves original instance in 1-wcsp or 3-cfn format
(1 by default), 2 or 4: saves after preprocessing in 2-wcsp or 4-cfn format
(this option can be combined with the previous one)
    -Z=[integer] : debug mode (save problem at each node if verbosity option
-v=num >= 1 and -Z=num >=3)
    -x=[(,i[#<>]a)*] : performs an elementary operation ('=':assign,
'#':remove, '<':decrease, '>':increase) with value a on variable of index i
(multiple operations are separated by a comma and no space) (without any
argument, a complete assignment -- used as initial upper bound and as value
heuristic -- read from default file "sol" taken as a certificate or given as
input filename with ".sol" extension)

```

```

-M=[integer] : preprocessing only: Min Sum Diffusion algorithm (default
number of iterations is 0)
-A=[integer] : enforces VAC at each search node with a search depth less
than the absolute value of a given value, if negative value then VAC is not
performed inside depth-first search of hybrid best-first search (default
value is 0)
-T=[decimal] : threshold cost value for VAC (default value is 1)
-P=[decimal] : threshold cost value for VAC during the preprocessing phase
(default value is 1)
-C=[float] : multiplies all costs internally by this number when loading
the problem (default value is 1)
-S : preprocessing only: performs singleton consistency (only in
conjunction with option "-A")
-V : VAC-based value ordering heuristic (default option)
-vacint : VAC-integrality/Full-EAC variable ordering heuristic
-vacthr : automatic threshold cost value selection for VAC during search
-rasps=[integer] : VAC-based upper bound probing heuristic (0: disable,
>0: max. nb. of backtracks) (default value is 0)
-raspslds=[integer] : VAC-based upper bound probing heuristic using LDS
instead of DFS (0: DFS, >0: max. discrepancy) (default value is 0)
-raspsdeg=[integer] : automatic threshold cost value selection for probing
heuristic (default value is 10°)
-raspsini : reset weighted degree variable ordering heuristic after doing
upper bound probing
-trws=[float] : enforces TRW-S in preprocessing until a given precision is
reached (default value is -1)
--trws-order : replaces DAC order by Kolmogorov's TRW-S order
--trws-n-iters=[integer] : enforce at most N iterations of TRW-S (default
value is 1000)
--trws-n-iters-no-change=[integer] : stop TRW-S when N iterations did not
change the lower bound up the given precision (default value is 5, -1=never)
--trws-n-iters-compute-ub=[integer] : compute UB every N steps in TRW-S
(default value is 100)

-B=[integer] : (0) DFBB, (1) BTD, (2) RDS-BTD, (3) RDS-BTD with path
decomposition instead of tree decomposition (default value is 0)
-O=[filename] : reads a variable elimination order or directly a valid
tree decomposition (given by a list of clusters in topological order of a
rooted forest, each line contains a cluster number,
    followed by a cluster parent number with -1 for the root(s) cluster(s),
    followed by a list of variable indexes) from a file used for BTD-like and
variable elimination methods, and also DAC ordering
-O=[negative integer] : build a tree decomposition (if BTD-like and/or
variable elimination methods are used) and also a compatible DAC ordering
using
    (-1) maximum cardinality search ordering, (-2)
minimum degree ordering, (-3) minimum fill-in ordering,
    (-4) maximum spanning tree ordering (see -mst),
(-5) reverse Cuthill-McKee ordering, (-6) approximate minimum degree
ordering,
    (-7) default file ordering (the same if this
option is missing, i.e. use the variable order in which variables appear in
the problem file)
    (-8) lexicographic ordering of variable names.
-j=[integer] : splits large clusters into a chain of smaller embedded
clusters with a number of proper variables less than this number
    (use options "-B=3 -j=1 -svo -k=1" for pure RDS, use value 0
for no splitting) (default value is 0)
-r=[integer] : limit on maximum cluster separator size (merge cluster with
its father otherwise, use a negative value for no limit) (default value is
-1)
-X=[integer] : limit on minimum number of proper variables in a cluster
(merge cluster with its father otherwise, use a zero for no limit) (default

```

```

value is 0)
  -E=[float] : merges leaf clusters with their fathers if small local
treewidth (in conjunction with option "-e" and positive threshold value) or
ratio of number of separator variables by number of cluster variables above a
given threshold (in conjunction with option "-vns") (default value is 0)
  -R=[integer] : choice for a specific root cluster number
  -I=[integer] : choice for solving only a particular rooted cluster subtree
(with RDS-BTD only)

  -a=[integer] : finds at most a given number of solutions with a cost
strictly lower than the initial upper bound and stops, or if no integer is
given, finds all solutions (or counts the number of zero-cost satisfiable
solutions in conjunction with BTD)
  -div=[integer] : minimum Hamming distance between diverse solutions (use
in conjunction with -a=integer with a limit of 1000 solutions) (default value
is 0)
  -divm=[integer] : diversity encoding method: 0: Dual 1: Hidden 2: Ternary
3: Knapsack (default value is 0)
  -mdd=[integer] : maximum relaxed MDD width for diverse solution global
constraint (default value is 0)
  -mddh=[integer] : MDD relaxation heuristic: 0: random, 1: high div, 2:
small div, 3: high unary costs (default value is 0)
  -D : approximate satisfiable solution count with BTD
  -logz : computes log of probability of evidence (i.e. log partition
function or log(Z) or PR task) for graphical models only (problem file
extension .uai)
  -epsilon=[float] : approximation factor for computing the partition
function (greater than 1, default value is inf)

  -hbfs=[integer] : hybrid best-first search, restarting from the root after
a given number of backtracks (default value is 10000)
  -open=[integer] : hybrid best-first search limit on the number of open
nodes (default value is -1)
-----
Alternatively one can call the random problem generator with the following
options:

  -random=[bench profile] : bench profile must be specified as follow :
                           n and d are respectively the number of variable and
the maximum domain size of the random problem.

      bin-{n}-{d}-{t1}-{p2}-{seed}      :t1 is the tightness in percentage
%of random binary cost functions
                                         :p2 is the num of binary cost
functions to include
                                         :the seed parameter is optional
(and will overwrite -seed)
      or:

      binsub-{n}-{d}-{t1}-{p2}-{p3}-{seed} binary random & submodular cost
functions
                                         t1 is the tightness in percentage %
of random cost functions
                                         p2 is the num of binary cost
functions to include
                                         p3 is the percentage % of
submodular cost functions among p2 cost functions
                                         (plus 10 permutations of two
randomly-chosen values for each domain)
      or:

      tern-{n}-{d}-{t1}-{p2}-{p3}-{seed} p3 is the num of ternary cost
functions
      or:

```



```
nary-{n}-{d}-{t1}-{p2}-{p3}...-{pn}-{seed}  pn is the num of n-ary cost
functions
or:
```

```
salldiff-{n}-{d}-{t1}-{p2}-{p3}...-{pn}-{seed}  pn is the num of
salldiff global cost functions (p2 and p3 still being used for the number of
random binary and ternary cost functions)
```

4.1 Related publications

4.1.1 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 csp, 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 (UDGVN-S/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.

4.1.2 toulbar2 for Combinatorial Optimization in Life Sciences

- **Computational Protein Design**
 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 Bioin-

formatics, 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 Frédéric Garcia, *RAIRO - Operations Research*, 47:151-172, 2013.

4.1.3 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 Roque-maurel, 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**

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 Artificial 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 Hlaváč, In *Proc. of International Conference on Document Analysis and Recognition*, Washington, DC, USA, 1215-1219, 2013.

4.2 Specific Events

- 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.
- 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.

5.1 Weighted n-queen problem

5.1.1 Brief description

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

5.1.2 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.

5.1.3 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
1	0	Row0
2	0	Row0
3	0	Row0
4	0	Row0
5	17	Row1
6	0	Row1
7	0	Row1
8	0	Row1
9	0	Row1
10	17	Row2
11	0	Row2
12	0	Row2
13	0	Row2
14	0	Row2
15	17	Row3
16	0	Row3
17	0	Row3
18	0	Row3

Q0 Q1 Q2 Q3

(Q0,Q2) constraint (2nd clique)

Costs	Q0	Q2
0	0	Row0
1	0	Row0
2	0	Row0
3	0	Row0
4	0	Row0
5	0	Row1
6	0	Row1
7	0	Row1
8	17	Row2
9	0	Row2
10	0	Row2
11	0	Row2
12	0	Row2
13	17	Row3
14	0	Row3
15	0	Row3
16	0	Row3

Q0 Q1 Q2 Q3

(Q0,Q3) constraint (3rd clique)

Costs	Q0	Q3
0	0	Row0
1	0	Row0
2	0	Row0
3	0	Row0
4	0	Row0
5	0	Row1
6	0	Row1
7	17	Row2
8	0	Row2
9	0	Row2
10	0	Row2
11	0	Row2
12	0	Row2
13	0	Row3
14	0	Row3
15	0	Row3
16	0	Row3

Q0 Q1 Q2 Q3

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

```

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

5.1.4 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")):
            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
                [{"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)

```

5.2 Weighted latin square problem

5.2.1 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.

5.2.2 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 $k=N^2+1$.

5.2.3 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}},
    {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

5.2.4 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)

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(' \t\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}')

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)

```

5.3 Radio link frequency assignment problem

5.3.1 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.

5.3.2 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.

5.3.3 Data

Original data files can be download from the cost function library [FullIRLFAP](#). 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.

5.3.4 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):
            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='')
        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()[:1]
            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))
```



```

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)

```

5.4 Frequency assignment problem with polarization

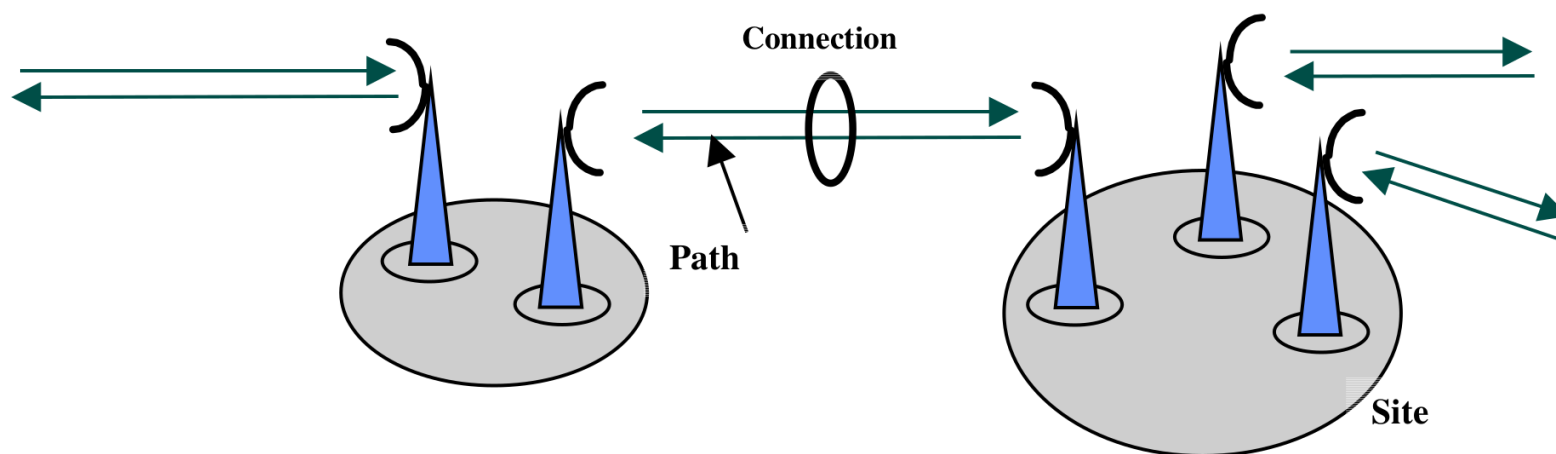
5.4.1 Brief description

The previously-described [Section 5.3](#) 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



5.4.2 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.

5.4.3 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](#).

5.4.4 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='')
        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()
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()[1:3]
        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()[1:4]
        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, routel, route2, vartype, operator, deviation) =
line.split()[1:6]
        self.ctr.append((int(routel), int(route2), vartype, operator,
int(deviation)))

    if len(line.split())==14 and line.split()[0]=="CE":
        (CE, routel, route2, s0, s1, s2, s3, s4, s5, s6, s7, s8, s9,
s10) = line.split()[1:14]
        self.softeq.append((int(routel), 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, routel, route2, s0, s1, s2, s3, s4, s5, s6, s7, s8, s9,
s10) = line.split()[1:14]
        self.softne.append((int(routel), 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 = {
        "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[routel], 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)

```

5.5 Mendelian error detection problem

5.5.1 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.

5.5.2 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.

5.5.3 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.

5.5.4 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)
    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:
        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()[1:]
            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
        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)
```

5.6 Block modeling problem

5.6.1 Brief description

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

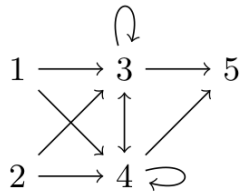
5.6.2 CFN model

We create N variables for every node of the graph with domain size k . We add $k \times k$ 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

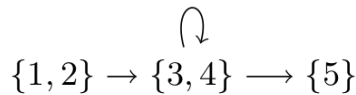
5.6.3 Data

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

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

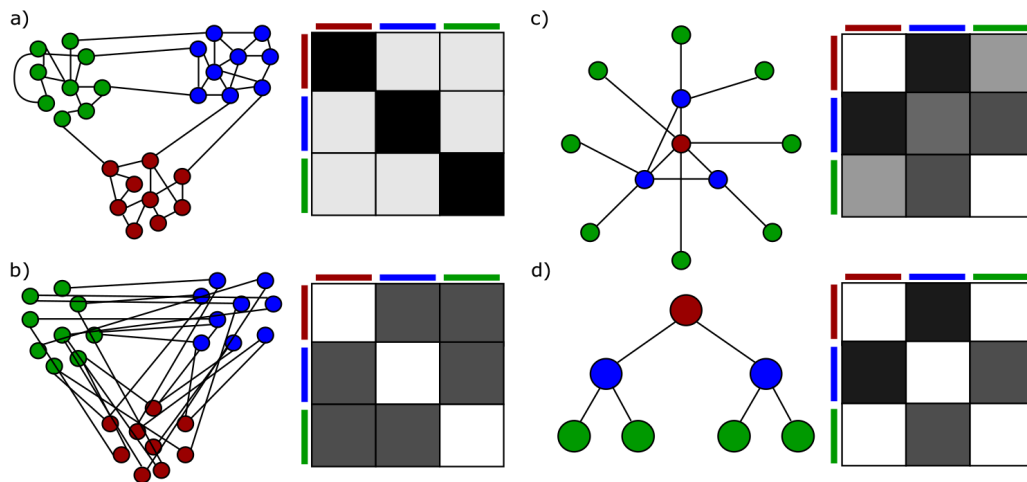


$$G = \left(\begin{array}{cc|cc|c} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 \end{array} \right)$$



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

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



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

5.6.4 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 | toolbar2 -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", "varsl", "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='')
        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):
        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", "Ar
    # 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[1]],
"costs": [data.top if k > 1 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='')
    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):
        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(v), 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)],

            # 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[1]]],
            "costs": [data.top if k > 1 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)

```

5.7 Airplane landing problem

5.7.1 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.

5.7.2 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.

5.7.3 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.

5.7.4 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`

```
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):
    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()
```

5.8 Warehouse location problem

5.8.1 Brief description

See a problem description in [CSPLib-034](#). We are dealing with the uncapacitated case only for the moment.

5.8.2 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).

5.8.3 Data

Original data files can be download from the cost function library [warehouses](#). Their format is described [here](#).

5.8.4 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):
    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='')
    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],
                          [(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]))

```

5.8.5 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()

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

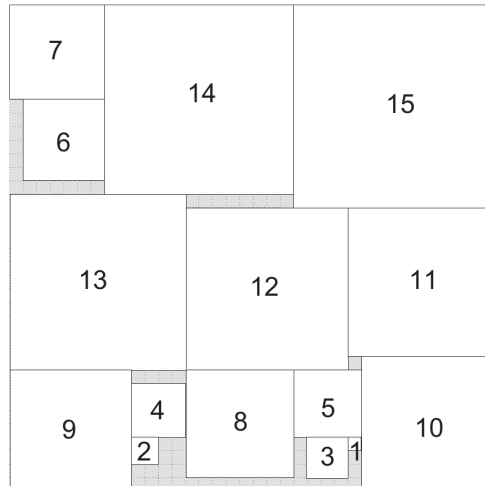
Problem.Dump('warehouse.cfn')
Problem.Option.FullEAC = False
Problem.Option.showSolutions = False
Problem.Solve()
```

5.9 Square packing problem

5.9.1 Brief description

Find a packing of squares of size 1×1 , 2×2 , ..., $N \times N$ into a given container square $S \times S$ 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 36×36 square (Fig. taken from Takehide Soh)



5.9.2 CFN model

We create an integer variable of domain size $(S-i) \times (S-i)$ 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.

5.9.3 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"])):
        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)],
        "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)

```

5.9.4 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)))]

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

```

5.9.5 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);

    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;
}

```

5.10 Square soft packing problem

5.10.1 Brief description

Find a packing of squares of size 1×1 , 2×2 , ..., $N \times N$ into a given container square $S \times S$ minimizing total sum of overlaps.

5.10.2 CFN model

We reuse the [Section 5.9](#) 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.

5.10.3 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(e1, tuple) and not isinstance(e1, dict):
    result.extend(flatten(e1))
else:
    result.append(e1)
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"])):
        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)],
        "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)

```

5.10.4 C++ program using libtb2.so

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

squaresoft.cpp

```

/**
 * Square Soft Packing Problem
 */

// Compile with cmake option -DLIBTB2=ON -DPYTB2=ON to get C++ toulbar2
library lib/Linux/libtb2.so

```

```
// Then,
// g++ -o squaresoft squaresoft.cpp -Isrc -Llib/Linux -std=c++11 -O3 -DNDEBUG
// -DBOOST -DLONGDOUBLE_PROB -DLONGLONG_COST -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);
        }
    }

    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;
}
```

5.11 Learning to play the Sudoku

5.12 Renault car configuration system: learning user preferences

