toulbar2 User Guide

Release 1.0.0

INRAE

Sep 26, 2025

CONTENTS

CHAPTER

ONE

WHAT IS TOULBAR2

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

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

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

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

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

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

CHAPTER

TWO

HOW DO I INSTALL IT ?

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

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

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

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

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

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

CHAPTER

THREE

HOW DO I TEST IT ?

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

```
make test
```

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

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

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

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

CHAPTER

FOUR

USING IT AS A BLACK BOX

Using toulbar2 is just a matter of having a properly formatted input file describing the cost function network, graphical model, PWMaxSAT, PBO or Linkage **.pre** file and executing:

```
toulbar2 [option parameters] <file>
```

and toulbar2 will start solving the optimization problem described in its file argument. By default, the extension of the file (either **.cfn**, **.cfn.gz**, **.cfn.bz2**, **.cfn.xz**, **.wcsp**, **.wcsp.gz**, **.wcsp.bz2**, **.wcsp.xz**, **.wcnf**, **.wcnf.gz**, **.wcnf.bz2**, **.wcnf.xz**, **.cnf**, **.cnf.gz**, **.cnf.bz2**, **.cnf.xz**, **.qpbo**, **.qpbo.gz**, **.qpbo.bz2**, **.qpbo.xz**, **.opb**, **.opb.gz**, **.opb.bz2**, **.opb.xz**, **.wbo**, **.wbo.gz**, **.wbo.bz2**, **.wbo.xz**, **.lp**, **.lp.gz**, **.lp.bz2**, **.lp.xz**, **.uai**, **.uai.gz**, **.uai.bz2**, **.uai.xz**, **.LG**, **.LG.gz**, **.LG.bz2**, **.LG.xz**, **.xml**, **.xml.gz**, **.xml.bz2**, **.xml.xz**, **.pre** or **.bep**) is used to determine the nature of the file (see *Input formats*). There is no specific order for the options or problem file. toulbar2 comes with decently optimized default option parameters. It is however often possible to set it up for different target than pure optimization or tune it for faster action using specific command line options.

CHAPTER

FIVE

QUICK START

- Download a binary weighted constraint satisfaction problem (WCSP) file `example.wcsp.xz`. Solve it with default options:

```
toulbar2 EXAMPLES/example.wcsp.xz
```

```
Read 25 variables, with 5 values at most, and 63 cost functions, with maximum arity␣
↪2.
Reverse original DAC dual bound: 20 (+10.000%)
Cost function decomposition time : 0.000 seconds.
Preprocessing time: 0.001 seconds.
24 unassigned variables, 116 values in all current domains (med. size:5, max␣
↪size:5) and 62 non-unary cost functions (med. arity:2, med. degree:5)
Initial lower and upper bounds: [20, 64] 68.750%
New solution: 28 (0 backtracks, 6 nodes, depth 8, 0.001 seconds)
Optimality gap: [21, 28] 25.000 % (9 backtracks, 20 nodes, 0.002 seconds)
New solution: 27 (9 backtracks, 27 nodes, depth 7, 0.002 seconds)
Optimality gap: [21, 27] 22.222 % (14 backtracks, 32 nodes, 0.002 seconds)
Optimality gap: [22, 27] 18.519 % (17 backtracks, 41 nodes, 0.002 seconds)
Optimality gap: [23, 27] 14.815 % (46 backtracks, 115 nodes, 0.004 seconds)
Optimality gap: [24, 27] 11.111 % (83 backtracks, 210 nodes, 0.005 seconds)
Optimality gap: [25, 27] 7.407 % (102 backtracks, 275 nodes, 0.007 seconds)
Optimality gap: [27, 27] 0.000 % (105 backtracks, 297 nodes, 0.007 seconds)
Node redundancy during HBFS: 28.956
Optimum: 27 in 105 backtracks and 297 nodes ( 544 removals by DEE) and 0.007␣
↪seconds.
end.
```

- Solve a WCSP using INCOP, a local search method [idwalk:cp04] applied just after preprocessing, in order to find a good upper bound before a complete search:

```
toulbar2 EXAMPLES/example.wcsp.xz -i
```

```
Read 25 variables, with 5 values at most, and 63 cost functions, with maximum arity␣
↪2.
Reverse original DAC dual bound: 20 (+10.000%)
Cost function decomposition time : 0.000 seconds.
New solution: 27 (0 backtracks, 0 nodes, depth 1, 0.120 seconds)
INCOP solving time: 0.326 seconds.
Preprocessing time: 0.328 seconds.
24 unassigned variables, 116 values in all current domains (med. size:5, max␣
```

(continues on next page)

(continued from previous page)

```
↪size:5) and 62 non-unary cost functions (med. arity:2, med. degree:5)
Initial lower and upper bounds: [20, 27] 25.926%
Optimality gap: [21, 27] 22.222 % (14 backtracks, 34 nodes, 0.328 seconds)
Optimality gap: [22, 27] 18.519 % (19 backtracks, 48 nodes, 0.329 seconds)
Optimality gap: [23, 27] 14.815 % (79 backtracks, 189 nodes, 0.332 seconds)
Optimality gap: [24, 27] 11.111 % (82 backtracks, 201 nodes, 0.332 seconds)
Optimality gap: [25, 27] 7.407 % (87 backtracks, 222 nodes, 0.333 seconds)
Optimality gap: [27, 27] 0.000 % (99 backtracks, 258 nodes, 0.333 seconds)
Node redundancy during HBFS: 23.256
Optimum: 27 in 99 backtracks and 258 nodes ( 539 removals by DEE) and 0.333 seconds.
end.
```

- Solve a WCSP with an initial upper bound and save its (first) optimal solution in filename "example.sol":

```
toulbar2 EXAMPLES/example.wcsp.xz -ub=28 -w=example.sol
```

```
Read 25 variables, with 5 values at most, and 63 cost functions, with maximum arity␣
↪2.
Reverse original DAC dual bound: 20 (+10.000%)
Cost function decomposition time : 0.000 seconds.
Preprocessing time: 0.001 seconds.
24 unassigned variables, 116 values in all current domains (med. size:5, max␣
↪size:5) and 62 non-unary cost functions (med. arity:2, med. degree:5)
Initial lower and upper bounds: [20, 28] 28.571%
New solution: 27 (0 backtracks, 4 nodes, depth 6, 0.002 seconds)
Optimality gap: [21, 27] 22.222 % (6 backtracks, 14 nodes, 0.002 seconds)
Optimality gap: [22, 27] 18.519 % (23 backtracks, 57 nodes, 0.003 seconds)
Optimality gap: [23, 27] 14.815 % (69 backtracks, 159 nodes, 0.005 seconds)
Optimality gap: [24, 27] 11.111 % (76 backtracks, 180 nodes, 0.006 seconds)
Optimality gap: [25, 27] 7.407 % (92 backtracks, 240 nodes, 0.007 seconds)
Optimality gap: [27, 27] 0.000 % (100 backtracks, 286 nodes, 0.007 seconds)
Node redundancy during HBFS: 29.720
Optimum: 27 in 100 backtracks and 286 nodes ( 569 removals by DEE) and 0.008␣
↪seconds.
end.
```

- … and see this saved "example.sol" file:

```
cat example.sol
# each value corresponds to one variable assignment in problem file order
```

```
1 0 2 2 2 2 0 4 2 0 4 1 0 0 3 0 3 1 2 4 2 1 2 4 1
```

- Download a larger WCSP file `scen06.wcsp.xz`. Solve it using a limited discrepancy search strategy [Ginsberg1995] with a VAC integrality-based variable ordering [Trosser2020a] in order to speed-up the search for finding good upper bounds first (by default, toulbar2 uses another diversification strategy based on hybrid best-first search [Katsirelos2015a]):

```
toulbar2 EXAMPLES/scen06.wcsp.xz -l -vacint
```

```
Read 100 variables, with 44 values at most, and 1222 cost functions, with maximum␣
↪arity 2.
```

(continues on next page)

(continued from previous page)

```
Cost function decomposition time : 6.1e-05 seconds.
Preprocessing time: 0.159001 seconds.
82 unassigned variables, 3273 values in all current domains (med. size:44, max␣
↪size:44) and 327 non-unary cost functions (med. arity:2, med. degree:6)
Initial lower and upper bounds: [0, 248338] 100.000%
--- [1] LDS 0 --- (0 nodes)
c 2097152 Bytes allocated for long long stack.
c 4194304 Bytes allocated for long long stack.
New solution: 7769 (0 backtracks, 98 nodes, depth 3, 0.197 seconds)
--- [1] LDS 1 --- (98 nodes)
c 8388608 Bytes allocated for long long stack.
New solution: 5848 (1 backtracks, 284 nodes, depth 4, 0.280 seconds)
New solution: 5384 (3 backtracks, 399 nodes, depth 4, 0.347 seconds)
New solution: 5039 (4 backtracks, 468 nodes, depth 4, 0.376 seconds)
New solution: 4740 (8 backtracks, 642 nodes, depth 4, 0.510 seconds)
--- [1] LDS 2 --- (739 nodes)
New solution: 4716 (37 backtracks, 976 nodes, depth 5, 0.863 seconds)
New solution: 4693 (44 backtracks, 1020 nodes, depth 5, 0.881 seconds)
New solution: 4690 (49 backtracks, 1125 nodes, depth 5, 0.984 seconds)
New solution: 4683 (79 backtracks, 1481 nodes, depth 5, 1.361 seconds)
New solution: 4679 (97 backtracks, 1607 nodes, depth 4, 1.476 seconds)
New solution: 4527 (105 backtracks, 1773 nodes, depth 5, 1.632 seconds)
New solution: 4494 (109 backtracks, 1867 nodes, depth 5, 1.700 seconds)
New solution: 4487 (128 backtracks, 2068 nodes, depth 5, 1.869 seconds)
New solution: 4457 (139 backtracks, 2184 nodes, depth 5, 1.956 seconds)
New solution: 4442 (150 backtracks, 2272 nodes, depth 5, 2.014 seconds)
New solution: 3986 (169 backtracks, 2444 nodes, depth 5, 2.151 seconds)
New solution: 3858 (171 backtracks, 2488 nodes, depth 5, 2.176 seconds)
New solution: 3709 (180 backtracks, 2570 nodes, depth 5, 2.248 seconds)
--- [1] LDS 4 --- (2654 nodes)
New solution: 3706 (511 backtracks, 3957 nodes, depth 6, 3.490 seconds)
New solution: 3683 (513 backtracks, 4008 nodes, depth 7, 3.511 seconds)
New solution: 3662 (515 backtracks, 4034 nodes, depth 7, 3.522 seconds)
New solution: 3634 (660 backtracks, 4735 nodes, depth 7, 3.971 seconds)
New solution: 3632 (661 backtracks, 4742 nodes, depth 6, 3.973 seconds)
New solution: 3630 (663 backtracks, 4750 nodes, depth 6, 3.974 seconds)
New solution: 3619 (685 backtracks, 4842 nodes, depth 7, 4.029 seconds)
New solution: 3600 (736 backtracks, 5103 nodes, depth 7, 4.178 seconds)
New solution: 3587 (739 backtracks, 5124 nodes, depth 7, 4.184 seconds)
New solution: 3586 (741 backtracks, 5133 nodes, depth 6, 4.185 seconds)
New solution: 3585 (742 backtracks, 5137 nodes, depth 6, 4.185 seconds)
New solution: 3584 (743 backtracks, 5141 nodes, depth 6, 4.185 seconds)
New solution: 3583 (744 backtracks, 5144 nodes, depth 6, 4.185 seconds)
New solution: 3570 (757 backtracks, 5208 nodes, depth 7, 4.214 seconds)
New solution: 3563 (758 backtracks, 5220 nodes, depth 7, 4.217 seconds)
New solution: 3551 (762 backtracks, 5236 nodes, depth 7, 4.222 seconds)
New solution: 3543 (764 backtracks, 5248 nodes, depth 7, 4.225 seconds)
New solution: 3524 (765 backtracks, 5258 nodes, depth 7, 4.227 seconds)
New solution: 3501 (795 backtracks, 5428 nodes, depth 7, 4.382 seconds)
New solution: 3499 (980 backtracks, 6144 nodes, depth 6, 5.110 seconds)
New solution: 3477 (985 backtracks, 6172 nodes, depth 7, 5.119 seconds)
New solution: 3443 (989 backtracks, 6196 nodes, depth 7, 5.125 seconds)
```

(continues on next page)

(continued from previous page)

```
--- [1] Search with no discrepancy limit --- (7446 nodes)
New solution: 3442 (4423 backtracks, 13529 nodes, depth 28, 8.371 seconds)
New solution: 3441 (4424 backtracks, 13534 nodes, depth 28, 8.371 seconds)
New solution: 3424 (4462 backtracks, 13619 nodes, depth 32, 8.388 seconds)
New solution: 3422 (4463 backtracks, 13620 nodes, depth 29, 8.388 seconds)
New solution: 3420 (4464 backtracks, 13624 nodes, depth 28, 8.388 seconds)
New solution: 3412 (4474 backtracks, 13648 nodes, depth 28, 8.395 seconds)
New solution: 3410 (4475 backtracks, 13650 nodes, depth 26, 8.396 seconds)
New solution: 3404 (4582 backtracks, 13870 nodes, depth 28, 8.443 seconds)
New solution: 3402 (4583 backtracks, 13872 nodes, depth 26, 8.443 seconds)
New solution: 3400 (4584 backtracks, 13876 nodes, depth 25, 8.443 seconds)
New solution: 3391 (4586 backtracks, 13887 nodes, depth 29, 8.444 seconds)
New solution: 3389 (4587 backtracks, 13889 nodes, depth 27, 8.444 seconds)
Optimality gap: [100, 3389] 97.049 % (33368 backtracks, 71431 nodes, 31.505 seconds)
Optimality gap: [292, 3389] 91.384 % (50312 backtracks, 105319 nodes, 46.031␣
↪seconds)
Optimality gap: [475, 3389] 85.984 % (52721 backtracks, 110137 nodes, 48.301␣
↪seconds)
Optimality gap: [921, 3389] 72.824 % (62170 backtracks, 129035 nodes, 54.846␣
↪seconds)
Optimality gap: [1864, 3389] 44.999 % (62688 backtracks, 130071 nodes, 55.259␣
↪seconds)
Optimality gap: [3086, 3389] 8.941 % (62722 backtracks, 130139 nodes, 55.307␣
↪seconds)
Optimality gap: [3156, 3389] 6.875 % (62723 backtracks, 130141 nodes, 55.309␣
↪seconds)
Optimum: 3389 in 62724 backtracks and 130143 nodes ( 680102 removals by DEE) and 55.
↪310 seconds.
end.
```

- Download a cluster decomposition file `scen06.dec` (each line corresponds to a cluster of variables, clusters may overlap). Solve the previous WCSP using a variable neighborhood search algorithm (UDGVNS) [Ouali2017] during 10 seconds:

```
toulbar2 EXAMPLES/scen06.wcsp.xz EXAMPLES/scen06.dec -vns -time=10
```

```
Read 100 variables, with 44 values at most, and 1222 cost functions, with maximum␣
↪arity 2.
Cost function decomposition time : 5.5e-05 seconds.
Preprocessing time: 0.154142 seconds.
82 unassigned variables, 3273 values in all current domains (med. size:44, max␣
↪size:44) and 327 non-unary cost functions (med. arity:2, med. degree:6)
Initial lower and upper bounds: [0, 248338] 100.000%
c 2097152 Bytes allocated for long long stack.
c 4194304 Bytes allocated for long long stack.
c 8388608 Bytes allocated for long long stack.
New solution: 7566 (0 backtracks, 108 nodes, depth 109, 0.206 seconds)
Problem decomposition in 55 clusters with size distribution: min: 1 median: 5 mean:␣
↪4.782 max: 12
****** Restart 1 with 1 discrepancies and UB=7566 ****** (108 nodes)
New solution: 7468 (0 backtracks, 108 nodes, depth 1, 0.208 seconds)
New solution: 7360 (0 backtracks, 112 nodes, depth 2, 0.209 seconds)
```

(continues on next page)

(continued from previous page)

```
New solution: 7359 (0 backtracks, 112 nodes, depth 1, 0.209 seconds)
New solution: 7357 (0 backtracks, 112 nodes, depth 1, 0.210 seconds)
New solution: 7337 (0 backtracks, 112 nodes, depth 1, 0.210 seconds)
New solution: 7267 (0 backtracks, 112 nodes, depth 1, 0.211 seconds)
New solution: 5958 (0 backtracks, 124 nodes, depth 2, 0.219 seconds)
New solution: 5900 (0 backtracks, 124 nodes, depth 1, 0.219 seconds)
New solution: 5756 (0 backtracks, 128 nodes, depth 2, 0.222 seconds)
New solution: 5709 (0 backtracks, 128 nodes, depth 1, 0.223 seconds)
New solution: 5653 (0 backtracks, 128 nodes, depth 1, 0.224 seconds)
New solution: 5619 (0 backtracks, 131 nodes, depth 2, 0.227 seconds)
New solution: 5608 (0 backtracks, 133 nodes, depth 2, 0.231 seconds)
New solution: 5171 (5 backtracks, 176 nodes, depth 2, 0.238 seconds)
New solution: 5065 (7 backtracks, 188 nodes, depth 2, 0.240 seconds)
New solution: 5062 (7 backtracks, 189 nodes, depth 2, 0.241 seconds)
New solution: 4972 (7 backtracks, 189 nodes, depth 1, 0.242 seconds)
New solution: 4962 (7 backtracks, 189 nodes, depth 1, 0.243 seconds)
New solution: 4906 (17 backtracks, 238 nodes, depth 2, 0.291 seconds)
New solution: 4905 (22 backtracks, 274 nodes, depth 1, 0.303 seconds)
New solution: 4869 (23 backtracks, 282 nodes, depth 2, 0.309 seconds)
New solution: 4807 (26 backtracks, 321 nodes, depth 2, 0.349 seconds)
New solution: 4797 (26 backtracks, 321 nodes, depth 1, 0.349 seconds)
New solution: 4625 (27 backtracks, 333 nodes, depth 2, 0.354 seconds)
New solution: 4380 (32 backtracks, 354 nodes, depth 2, 0.361 seconds)
New solution: 4372 (32 backtracks, 354 nodes, depth 1, 0.361 seconds)
New solution: 4370 (32 backtracks, 354 nodes, depth 1, 0.362 seconds)
New solution: 4368 (32 backtracks, 354 nodes, depth 1, 0.364 seconds)
New solution: 4349 (32 backtracks, 357 nodes, depth 2, 0.365 seconds)
New solution: 4153 (33 backtracks, 375 nodes, depth 2, 0.372 seconds)
New solution: 4144 (34 backtracks, 379 nodes, depth 2, 0.374 seconds)
New solution: 3714 (43 backtracks, 462 nodes, depth 2, 0.423 seconds)
New solution: 3611 (44 backtracks, 464 nodes, depth 1, 0.426 seconds)
New solution: 3510 (46 backtracks, 482 nodes, depth 2, 0.432 seconds)
****** Restart 2 with 2 discrepancies and UB=3510 ****** (1935 nodes)
New solution: 3477 (1220 backtracks, 6501 nodes, depth 3, 5.336 seconds)
New solution: 3476 (1220 backtracks, 6501 nodes, depth 1, 5.338 seconds)
New solution: 3473 (1220 backtracks, 6501 nodes, depth 1, 5.339 seconds)
New solution: 3461 (1220 backtracks, 6506 nodes, depth 2, 5.343 seconds)
New solution: 3449 (1220 backtracks, 6507 nodes, depth 2, 5.344 seconds)
New solution: 3427 (1220 backtracks, 6507 nodes, depth 1, 5.345 seconds)
New solution: 3426 (1235 backtracks, 6607 nodes, depth 2, 5.426 seconds)
New solution: 3425 (1237 backtracks, 6617 nodes, depth 2, 5.429 seconds)
New solution: 3424 (1237 backtracks, 6619 nodes, depth 2, 5.432 seconds)
New solution: 3407 (1272 backtracks, 6871 nodes, depth 2, 5.662 seconds)
****** Restart 3 with 2 discrepancies and UB=3407 ****** (8284 nodes)
New solution: 3402 (1604 backtracks, 8939 nodes, depth 3, 7.663 seconds)
****** Restart 4 with 2 discrepancies and UB=3402 ****** (10196 nodes)

Time limit expired... Aborting...
Dual bound: 0
Primal bound: 3402 in 2174 backtracks and 11657 nodes ( 96931 removals by DEE) and␣
↪9.975 seconds.
end.
```

- Download another difficult instance `scen07.wcsp.xz`. Solve it using a variable neighborhood search algorithm (UDGVNS) with maximum cardinality search cluster decomposition and absorption [Ouali2017] during 5 seconds:

```
toulbar2 EXAMPLES/scen07.wcsp.xz -vns -O=-1 -E -time=5
```

```
Read 200 variables, with 44 values at most, and 2665 cost functions, with maximum␣
↪arity 2.
Cost function decomposition time : 0.000202 seconds.
Preprocessing time: 0.35719 seconds.
162 unassigned variables, 6481 values in all current domains (med. size:44, max␣
↪size:44) and 764 non-unary cost functions (med. arity:2, med. degree:8)
Initial lower and upper bounds: [10000, 436552965] 99.998%
c 2097152 Bytes allocated for long long stack.
c 4194304 Bytes allocated for long long stack.
c 8388608 Bytes allocated for long long stack.
New solution: 2535618 (0 backtracks, 231 nodes, depth 232, 0.459 seconds)
Tree decomposition time: 0.003 seconds.
Problem decomposition in 25 clusters with size distribution: min: 3 median: 10␣
↪mean: 10.360 max: 38
****** Restart 1 with 1 discrepancies and UB=2535618 ****** (231 nodes)
New solution: 2535518 (0 backtracks, 231 nodes, depth 1, 0.467 seconds)
New solution: 2525819 (0 backtracks, 231 nodes, depth 1, 0.472 seconds)
New solution: 2515619 (2 backtracks, 237 nodes, depth 1, 0.475 seconds)
New solution: 2515519 (2 backtracks, 237 nodes, depth 1, 0.477 seconds)
New solution: 2515518 (2 backtracks, 237 nodes, depth 1, 0.477 seconds)
New solution: 2515516 (3 backtracks, 240 nodes, depth 1, 0.481 seconds)
New solution: 2515514 (3 backtracks, 240 nodes, depth 1, 0.481 seconds)
New solution: 2495513 (3 backtracks, 240 nodes, depth 1, 0.482 seconds)
New solution: 2485515 (3 backtracks, 240 nodes, depth 1, 0.483 seconds)
New solution: 2485513 (3 backtracks, 240 nodes, depth 1, 0.484 seconds)
New solution: 2475412 (3 backtracks, 240 nodes, depth 1, 0.486 seconds)
New solution: 2465411 (3 backtracks, 240 nodes, depth 1, 0.487 seconds)
New solution: 2465210 (3 backtracks, 240 nodes, depth 1, 0.488 seconds)
New solution: 2465011 (4 backtracks, 246 nodes, depth 2, 0.496 seconds)
New solution: 2455217 (8 backtracks, 272 nodes, depth 2, 0.502 seconds)
New solution: 2455016 (8 backtracks, 272 nodes, depth 1, 0.504 seconds)
New solution: 2454814 (8 backtracks, 272 nodes, depth 1, 0.505 seconds)
New solution: 2454715 (9 backtracks, 281 nodes, depth 2, 0.510 seconds)
New solution: 2434722 (13 backtracks, 303 nodes, depth 2, 0.524 seconds)
New solution: 2434622 (15 backtracks, 317 nodes, depth 2, 0.528 seconds)
New solution: 2434522 (15 backtracks, 317 nodes, depth 1, 0.530 seconds)
New solution: 2434520 (15 backtracks, 317 nodes, depth 1, 0.532 seconds)
New solution: 2434519 (15 backtracks, 317 nodes, depth 1, 0.534 seconds)
New solution: 2425017 (16 backtracks, 327 nodes, depth 2, 0.538 seconds)
New solution: 2425016 (24 backtracks, 392 nodes, depth 2, 0.558 seconds)
New solution: 1475515 (24 backtracks, 395 nodes, depth 2, 0.561 seconds)
New solution: 1455716 (24 backtracks, 395 nodes, depth 1, 0.565 seconds)
New solution: 1455715 (24 backtracks, 395 nodes, depth 1, 0.565 seconds)
New solution: 1395508 (24 backtracks, 399 nodes, depth 2, 0.569 seconds)
New solution: 1395408 (26 backtracks, 407 nodes, depth 2, 0.573 seconds)
New solution: 1384805 (28 backtracks, 416 nodes, depth 2, 0.577 seconds)
New solution: 1384803 (28 backtracks, 417 nodes, depth 2, 0.581 seconds)
```

(continues on next page)

(continued from previous page)

```
New solution: 1374803 (28 backtracks, 417 nodes, depth 1, 0.584 seconds)
New solution: 1374401 (28 backtracks, 417 nodes, depth 1, 0.585 seconds)
New solution: 1374297 (37 backtracks, 468 nodes, depth 1, 0.594 seconds)
New solution: 1374197 (37 backtracks, 468 nodes, depth 1, 0.597 seconds)
New solution: 1364197 (37 backtracks, 468 nodes, depth 1, 0.598 seconds)
New solution: 1364196 (48 backtracks, 527 nodes, depth 1, 0.632 seconds)
New solution: 1354399 (66 backtracks, 608 nodes, depth 1, 0.666 seconds)
New solution: 1354398 (72 backtracks, 626 nodes, depth 1, 0.674 seconds)
New solution: 1354397 (72 backtracks, 626 nodes, depth 1, 0.675 seconds)
New solution: 1354396 (74 backtracks, 633 nodes, depth 1, 0.682 seconds)
New solution: 1344097 (76 backtracks, 644 nodes, depth 2, 0.694 seconds)
New solution: 1344000 (85 backtracks, 667 nodes, depth 1, 0.705 seconds)
New solution: 1343999 (86 backtracks, 669 nodes, depth 1, 0.708 seconds)
New solution: 1343998 (88 backtracks, 676 nodes, depth 1, 0.713 seconds)
New solution: 1343800 (88 backtracks, 676 nodes, depth 1, 0.714 seconds)
New solution: 1343797 (89 backtracks, 678 nodes, depth 1, 0.719 seconds)
New solution: 1343698 (129 backtracks, 906 nodes, depth 1, 0.858 seconds)
New solution: 1343695 (140 backtracks, 979 nodes, depth 1, 0.889 seconds)
New solution: 373807 (142 backtracks, 987 nodes, depth 2, 0.898 seconds)
New solution: 363805 (155 backtracks, 1052 nodes, depth 2, 0.920 seconds)
New solution: 343800 (155 backtracks, 1058 nodes, depth 2, 0.923 seconds)
New solution: 343799 (158 backtracks, 1070 nodes, depth 1, 0.931 seconds)
New solution: 343798 (160 backtracks, 1074 nodes, depth 1, 0.934 seconds)
New solution: 343697 (161 backtracks, 1078 nodes, depth 1, 0.939 seconds)
New solution: 343593 (162 backtracks, 1084 nodes, depth 2, 0.945 seconds)
****** Restart 2 with 2 discrepancies and UB=343593 ****** (2778 nodes)

Time limit expired... Aborting...
Dual bound: 10000
Primal bound: 343593 in 1127 backtracks and 7051 nodes ( 31649 removals by DEE) and␣
↪4.997 seconds.
end.
```

- Download file `404.wcsp.xz`. Solve it using Depth-First Brand and Bound with Tree Decomposition and HBFS (BTD-HBFS) [Schiex2006a] based on a min-fill variable ordering:

```
toulbar2 EXAMPLES/404.wcsp.xz -O=-3 -B=1
```

```
Read 100 variables, with 4 values at most, and 710 cost functions, with maximum␣
↪arity 3.
Reverse original DAC dual bound: 51 (+19.608%)
Reverse original DAC dual bound: 53 (+3.774%)
Reverse original DAC dual bound: 54 (+1.852%)
Cost function decomposition time : 0.000 seconds.
Preprocessing time: 0.010 seconds.
88 unassigned variables, 228 values in all current domains (med. size:2, max␣
↪size:4) and 591 non-unary cost functions (med. arity:2, med. degree:13)
Initial lower and upper bounds: [54, 164] 67.073%
Get root cluster C44 with max. size: 20
Get root cluster C27 with max. size: 14
Tree decomposition width  : 19
Tree decomposition height : 43
```

(continues on next page)

(continued from previous page)

```
Number of clusters        : 47
Tree decomposition time: 0.003 seconds.
New solution: 126 (33 backtracks, 61 nodes, depth 3, 0.014 seconds)
Optimality gap: [58, 126] 53.968 % (33 backtracks, 61 nodes, 0.014 seconds)
New solution: 125 (46 backtracks, 88 nodes, depth 3, 0.014 seconds)
Optimality gap: [64, 125] 48.800 % (46 backtracks, 88 nodes, 0.014 seconds)
New solution: 117 (136 backtracks, 278 nodes, depth 3, 0.016 seconds)
Optimality gap: [84, 117] 28.205 % (136 backtracks, 278 nodes, 0.017 seconds)
Optimality gap: [86, 117] 26.496 % (185 backtracks, 416 nodes, 0.018 seconds)
Optimality gap: [88, 117] 24.786 % (256 backtracks, 662 nodes, 0.020 seconds)
Optimality gap: [93, 117] 20.513 % (316 backtracks, 812 nodes, 0.021 seconds)
Optimality gap: [96, 117] 17.949 % (327 backtracks, 866 nodes, 0.022 seconds)
Optimality gap: [97, 117] 17.094 % (339 backtracks, 911 nodes, 0.022 seconds)
New solution: 114 (353 backtracks, 969 nodes, depth 3, 0.023 seconds)
Optimality gap: [98, 114] 14.035 % (353 backtracks, 969 nodes, 0.023 seconds)
Optimality gap: [100, 114] 12.281 % (388 backtracks, 1087 nodes, 0.024 seconds)
Optimality gap: [101, 114] 11.404 % (397 backtracks, 1195 nodes, 0.025 seconds)
Optimality gap: [102, 114] 10.526 % (410 backtracks, 1248 nodes, 0.026 seconds)
Optimality gap: [103, 114] 9.649 % (418 backtracks, 1271 nodes, 0.026 seconds)
Optimality gap: [104, 114] 8.772 % (456 backtracks, 1358 nodes, 0.027 seconds)
Optimality gap: [105, 114] 7.895 % (456 backtracks, 1405 nodes, 0.028 seconds)
Optimality gap: [106, 114] 7.018 % (468 backtracks, 1448 nodes, 0.028 seconds)
Optimality gap: [107, 114] 6.140 % (468 backtracks, 1467 nodes, 0.029 seconds)
Optimality gap: [108, 114] 5.263 % (468 backtracks, 1499 nodes, 0.029 seconds)
Optimality gap: [110, 114] 3.509 % (491 backtracks, 1581 nodes, 0.030 seconds)
Optimality gap: [111, 114] 2.632 % (502 backtracks, 1635 nodes, 0.030 seconds)
Optimality gap: [112, 114] 1.754 % (502 backtracks, 1652 nodes, 0.030 seconds)
Optimality gap: [113, 114] 0.877 % (502 backtracks, 1819 nodes, 0.031 seconds)
Optimality gap: [114, 114] 0.000 % (502 backtracks, 1942 nodes, 0.032 seconds)
HBFS open list restarts: 0.000 % and reuse: 16.119 % of 335
Node redundancy during HBFS: 49.691
Optimum: 114 in 502 backtracks and 1942 nodes ( 38 removals by DEE) and 0.032␣
↪seconds.
end.
```

- Solve the same problem using Russian Doll Search exploiting BTD [Sanchez2009a]:

```
toulbar2 EXAMPLES/404.wcsp.xz -O=-3 -B=2
```

```
Read 100 variables, with 4 values at most, and 710 cost functions, with maximum␣
↪arity 3.
Reverse original DAC dual bound: 51 (+19.608%)
Reverse original DAC dual bound: 53 (+3.774%)
Reverse original DAC dual bound: 54 (+1.852%)
Cost function decomposition time : 0.000 seconds.
Preprocessing time: 0.011 seconds.
88 unassigned variables, 228 values in all current domains (med. size:2, max␣
↪size:4) and 591 non-unary cost functions (med. arity:2, med. degree:13)
Initial lower and upper bounds: [54, 164] 67.073%
Get root cluster C44 with max. size: 20
Get root cluster C27 with max. size: 14
Tree decomposition width  : 19
```

(continues on next page)

(continued from previous page)

```
Tree decomposition height : 43
Number of clusters        : 47
Tree decomposition time: 0.003 seconds.
--- Solving cluster subtree 5 ...
New solution: 0 (0 backtracks, 0 nodes, depth 2, 0.014 seconds)
---  done  cost = [0,0] (0 backtracks, 0 nodes, depth 2)

--- Solving cluster subtree 6 ...
New solution: 0 (0 backtracks, 0 nodes, depth 2, 0.014 seconds)
---  done  cost = [0,0] (0 backtracks, 0 nodes, depth 2)

--- Solving cluster subtree 7 ...

...

--- Solving cluster subtree 44 ...
New solution: 53 (493 backtracks, 921 nodes, depth 7, 0.026 seconds)
New solution: 52 (504 backtracks, 938 nodes, depth 7, 0.027 seconds)
New solution: 48 (515 backtracks, 973 nodes, depth 22, 0.027 seconds)
---  done  cost = [48,48] (636 backtracks, 1160 nodes, depth 2)

--- Solving cluster subtree 46 ...
New solution: 114 (636 backtracks, 1160 nodes, depth 2, 0.031 seconds)
---  done  cost = [114,114] (636 backtracks, 1160 nodes, depth 2)

Optimum: 114 in 636 backtracks and 1160 nodes ( 80 removals by DEE) and 0.031␣
↪seconds.
end.
```

- Solve another WCSP using the original Russian Doll Search method [Verfaillie1996] with static variable ordering (following problem file) and soft arc consistency:

```
toulbar2 EXAMPLES/505.wcsp.xz -B=3 -j=1 -svo -k=1
```

```
Read 240 variables, with 4 values at most, and 2242 cost functions, with maximum␣
↪arity 3.
Cost function decomposition time : 0.007695 seconds.
Preprocessing time: 0.031135 seconds.
233 unassigned variables, 666 values in all current domains (med. size:2, max␣
↪size:4) and 1966 non-unary cost functions (med. arity:2, med. degree:16)
Initial lower and upper bounds: [2, 34347] 99.994%
Get root cluster C3 with max. size: 5
Tree decomposition width  : 59
Tree decomposition height : 233
Number of clusters        : 239
Tree decomposition time: 0.026 seconds.
--- Solving cluster subtree 0 ...
New solution: 0 (0 backtracks, 0 nodes, depth 2, 0.058 seconds)
---  done  cost = [0,0] (0 backtracks, 0 nodes, depth 2)

--- Solving cluster subtree 1 ...
New solution: 0 (0 backtracks, 0 nodes, depth 2, 0.059 seconds)
```

(continues on next page)

(continued from previous page)

```
---  done  cost = [0,0] (0 backtracks, 0 nodes, depth 2)

--- Solving cluster subtree 2 ...

...

--- Solving cluster subtree 3 ...
New solution: 21253 (26963 backtracks, 48851 nodes, depth 3, 5.361 seconds)
New solution: 21251 (26991 backtracks, 48883 nodes, depth 4, 5.367 seconds)
---  done  cost = [21251,21251] (26992 backtracks, 48883 nodes, depth 2)

--- Solving cluster subtree 238 ...
New solution: 21253 (26992 backtracks, 48883 nodes, depth 2, 5.367 seconds)
---  done  cost = [21253,21253] (26992 backtracks, 48883 nodes, depth 2)

Optimum: 21253 in 26992 backtracks and 48883 nodes ( 0 removals by DEE) and 5.377␣
↪seconds.
end.
```

- Solve the same WCSP using a parallel variable neighborhood search algorithm (UPDGVNS) with min-fill cluster decomposition [Ouali2017] using 4 cores during 5 seconds:

```
mpirun -n 4 toulbar2 EXAMPLES/505.wcsp.xz -vns -O=-3 -time=5
```

```
Read 240 variables, with 4 values at most, and 2242 cost functions, with maximum␣
↪arity 3.
Reverse original DAC dual bound: 5086 (+59.339%)
Reverse original DAC dual bound: 5089 (+0.059%)
Cost function decomposition time : 0.014 seconds.
Preprocessing time: 0.137 seconds.
233 unassigned variables, 666 values in all current domains (med. size:2, max␣
↪size:4) and 1969 non-unary cost functions (med. arity:2, med. degree:16)
Initial lower and upper bounds: [5089, 34354] 85.187%
Tree decomposition time: 0.018 seconds.
Problem decomposition in 89 clusters with size distribution: min: 4 median: 11␣
↪mean: 11.831 max: 23
New solution: 26266 (0 backtracks, 55 nodes, depth 56, 0.167 seconds)
New solution: 26264 in 0.182 seconds.
New solution: 26263 in 0.187 seconds.
New solution: 26262 in 0.188 seconds.
New solution: 26261 in 0.207 seconds.
New solution: 26260 in 0.221 seconds.
New solution: 25261 in 0.228 seconds.
New solution: 25257 in 0.239 seconds.
New solution: 24262 in 0.243 seconds.
New solution: 23262 in 0.258 seconds.
New solution: 23261 in 0.266 seconds.
New solution: 23259 in 0.270 seconds.
New solution: 22260 in 0.307 seconds.
New solution: 21262 in 0.328 seconds.
New solution: 21261 in 0.334 seconds.
New solution: 21260 in 0.347 seconds.
```

(continues on next page)

(continued from previous page)

```
New solution: 21258 in 0.452 seconds.
New solution: 21257 in 0.554 seconds.
New solution: 21256 in 0.625 seconds.
New solution: 21255 in 0.640 seconds.
New solution: 21254 in 0.649 seconds.
New solution: 21253 in 1.060 seconds.
```

- Download a cluster decomposition file `example.dec` (each line corresponds to a cluster of variables, clusters may overlap). Solve a WCSP using a variable neighborhood search algorithm (UDGVNS) with a given cluster decomposition:

```
toulbar2 EXAMPLES/example.wcsp.xz EXAMPLES/example.dec -vns
```

```
Read 25 variables, with 5 values at most, and 63 cost functions, with maximum arity␣
↪2.
Reverse original DAC dual bound: 20 (+10.000%)
Cost function decomposition time : 0.000 seconds.
Preprocessing time: 0.001 seconds.
24 unassigned variables, 116 values in all current domains (med. size:5, max␣
↪size:5) and 62 non-unary cost functions (med. arity:2, med. degree:5)
Initial lower and upper bounds: [20, 64] 68.750%
New solution: 29 (0 backtracks, 6 nodes, depth 7, 0.001 seconds)
Problem decomposition in 7 clusters with size distribution: min: 11 median: 15␣
↪mean: 15.143 max: 17
****** Restart 1 with 1 discrepancies and UB=29 ****** (6 nodes)
New solution: 28 (0 backtracks, 6 nodes, depth 1, 0.002 seconds)
New solution: 27 (12 backtracks, 52 nodes, depth 2, 0.004 seconds)
****** Restart 2 with 2 discrepancies and UB=27 ****** (101 nodes)
****** Restart 3 with 4 discrepancies and UB=27 ****** (217 nodes)
****** Restart 4 with 8 discrepancies and UB=27 ****** (376 nodes)
****** Restart 5 with 16 discrepancies and UB=27 ****** (724 nodes)
Optimum: 27 in 446 backtracks and 1002 nodes ( 3035 removals by DEE) and 0.034␣
↪seconds.
end.
```

- Solve a WCSP using a parallel variable neighborhood search algorithm (UPDGVNS) with the same cluster decomposition:

```
mpirun -n 4 toulbar2 EXAMPLES/example.wcsp.xz EXAMPLES/example.dec -vns
```

```
Read 25 variables, with 5 values at most, and 63 cost functions, with maximum arity␣
↪2.
Reverse original DAC dual bound: 20 (+10.000%)
Cost function decomposition time : 0.000 seconds.
Preprocessing time: 0.001 seconds.
24 unassigned variables, 116 values in all current domains (med. size:5, max␣
↪size:5) and 62 non-unary cost functions (med. arity:2, med. degree:5)
Initial lower and upper bounds: [20, 64] 68.750%
Problem decomposition in 7 clusters with size distribution: min: 11 median: 15␣
↪mean: 15.143 max: 17
New solution: 29 (0 backtracks, 7 nodes, depth 8, 0.002 seconds)
New solution: 28 in 0.003 seconds.
```

(continues on next page)

(continued from previous page)

```
New solution: 27 in 0.003 seconds.
Optimum: 27 in 0 backtracks and 7 nodes ( 32 removals by DEE) and 0.063 seconds.
Total CPU time = 0.267 seconds.
Solving real-time = 0.066 seconds (not including reading and preprocessing time).
end.
```

- Download file `example.order`. Solve a WCSP using BTD-HBFS based on a given (min-fill) reverse variable elimination ordering:

```
toulbar2 EXAMPLES/example.wcsp.xz EXAMPLES/example.order -B=1
```

```
Read 25 variables, with 5 values at most, and 63 cost functions, with maximum arity␣
↪2.
Reverse original DAC dual bound: 19 (+5.263%)
Reverse original DAC dual bound: 21 (+9.524%)
Cost function decomposition time : 0.000 seconds.
Preprocessing time: 0.001 seconds.
24 unassigned variables, 118 values in all current domains (med. size:5, max␣
↪size:5) and 62 non-unary cost functions (med. arity:2, med. degree:5)
Initial lower and upper bounds: [21, 64] 67.188%
Get root cluster C14 with max. size: 9
Tree decomposition width  : 8
Tree decomposition height : 16
Number of clusters        : 18
Tree decomposition time: 0.001 seconds.
New solution: 28 (19 backtracks, 30 nodes, depth 3, 0.002 seconds)
Optimality gap: [23, 28] 17.857 % (26 backtracks, 45 nodes, 0.003 seconds)
New solution: 27 (60 backtracks, 127 nodes, depth 3, 0.004 seconds)
Optimality gap: [23, 27] 14.815 % (60 backtracks, 127 nodes, 0.004 seconds)
Optimality gap: [24, 27] 11.111 % (102 backtracks, 214 nodes, 0.006 seconds)
Optimality gap: [25, 27] 7.407 % (148 backtracks, 348 nodes, 0.009 seconds)
Optimality gap: [26, 27] 3.704 % (157 backtracks, 397 nodes, 0.010 seconds)
Optimality gap: [27, 27] 0.000 % (157 backtracks, 410 nodes, 0.010 seconds)
HBFS open list restarts: 0.000 % and reuse: 13.333 % of 45
Node redundancy during HBFS: 25.366
Optimum: 27 in 157 backtracks and 410 nodes ( 238 removals by DEE) and 0.010␣
↪seconds.
end.
```

- Download file `example.cov`. Solve a WCSP using BTD-HBFS based on a given explicit (min-fill path-) tree-decomposition:

```
toulbar2 EXAMPLES/example.wcsp.xz EXAMPLES/example.cov -B=1
```

```
Read 25 variables, with 5 values at most, and 63 cost functions, with maximum arity␣
↪2.
Warning! Cannot apply variable elimination during search with a given tree␣
↪decomposition file.
Warning! Cannot apply functional variable elimination with a given tree␣
↪decomposition file.
Reverse original DAC dual bound: 19 (+10.526%)
Reverse original DAC dual bound: 20 (+5.000%)
```

(continues on next page)

(continued from previous page)

```
Cost function decomposition time : 0.000 seconds.
Preprocessing time: 0.001 seconds.
25 unassigned variables, 120 values in all current domains (med. size:5, max␣
↪size:5) and 63 non-unary cost functions (med. arity:2, med. degree:5)
Initial lower and upper bounds: [20, 64] 68.750%
Tree decomposition width  : 16
Tree decomposition height : 24
Number of clusters        : 9
Tree decomposition time: 0.001 seconds.
New solution: 28 (23 backtracks, 29 nodes, depth 3, 0.003 seconds)
Optimality gap: [21, 28] 25.000 % (61 backtracks, 124 nodes, 0.005 seconds)
Optimality gap: [22, 28] 21.429 % (154 backtracks, 321 nodes, 0.009 seconds)
New solution: 27 (390 backtracks, 814 nodes, depth 3, 0.018 seconds)
Optimality gap: [23, 27] 14.815 % (390 backtracks, 814 nodes, 0.018 seconds)
Optimality gap: [24, 27] 11.111 % (417 backtracks, 884 nodes, 0.020 seconds)
Optimality gap: [25, 27] 7.407 % (435 backtracks, 966 nodes, 0.021 seconds)
Optimality gap: [26, 27] 3.704 % (450 backtracks, 1044 nodes, 0.021 seconds)
Optimality gap: [27, 27] 0.000 % (450 backtracks, 1072 nodes, 0.022 seconds)
HBFS open list restarts: 0.000 % and reuse: 33.333 % of 21
Node redundancy during HBFS: 17.724
Optimum: 27 in 450 backtracks and 1072 nodes ( 248 removals by DEE) and 0.022␣
↪seconds.
end.
```

- Download a Markov Random Field (MRF) file `pedigree9.uai.xz` in UAI format. Solve it using bounded (of degree at most 8) variable elimination enhanced by cost function decomposition in preprocessing [Favier2011a] followed by BTD-HBFS exploiting only small-size (less than four variables) separators:

```
toulbar2 EXAMPLES/pedigree9.uai.xz -O=-3 -p=-8 -B=1 -r=4
```

```
Read 1118 variables, with 7 values at most, and 1118 cost functions, with maximum␣
↪arity 4.
No evidence file specified. Trying EXAMPLES/pedigree9.uai.evid
No evidence file.
Reverse original DAC dual bound: 287536650 energy: 240.632 (+7.642%)
Generic variable elimination of degree 4
Maximum degree of generic variable elimination: 4
Cost function decomposition time : 0.004 seconds.
Preprocessing time: 0.086 seconds.
235 unassigned variables, 523 values in all current domains (med. size:2, max␣
↪size:4) and 433 non-unary cost functions (med. arity:2, med. degree:6)
Initial lower and upper bounds: [560511104, 13246577993] 95.769%
Get root cluster C1034 with max. size: 30
Tree decomposition width  : 230
Tree decomposition height : 233
Number of clusters        : 887
Tree decomposition time: 0.021 seconds.
New solution: 839721541 energy: 295.850 prob: 3.265e-129 (70 backtracks, 133 nodes,␣
↪depth 3, 0.118 seconds)
New solution: 837627377 energy: 295.641 prob: 4.025e-129 (363 backtracks, 808 nodes,
↪ depth 3, 0.148 seconds)
New solution: 815835377 energy: 293.462 prob: 3.558e-128 (590 backtracks, 1287␣
```

(continues on next page)

(continued from previous page)

```
↪nodes, depth 3, 0.183 seconds)
New solution: 771837572 energy: 289.062 prob: 2.898e-126 (1153 backtracks, 2462␣
↪nodes, depth 3, 0.266 seconds)
New solution: 749865327 energy: 286.865 prob: 2.608e-125 (2257 backtracks, 4726␣
↪nodes, depth 3, 0.452 seconds)
New solution: 743190828 energy: 286.197 prob: 5.083e-125 (2962 backtracks, 6589␣
↪nodes, depth 3, 0.529 seconds)
New solution: 718546229 energy: 283.733 prob: 5.976e-124 (5872 backtracks, 12559␣
↪nodes, depth 3, 1.012 seconds)
New solution: 712604395 energy: 283.139 prob: 1.083e-123 (9785 backtracks, 20539␣
↪nodes, depth 3, 1.667 seconds)
New solution: 711184935 energy: 282.997 prob: 1.248e-123 (10606 backtracks, 22820␣
↪nodes, depth 3, 1.778 seconds)
HBFS open list restarts: 0.000 % and reuse: 74.124 % of 3424
Node redundancy during HBFS: 20.718
Optimum: 711184935 energy: 282.997 prob: 1.248e-123 in 165886 backtracks and 418438␣
↪nodes ( 566654 removals by DEE) and 29.071 seconds.
end.
```

- Download another MRF file `GeomSurf-7-gm256.uai.xz`. Solve it using Virtual Arc Consistency (VAC) in preprocessing [Cooper2008] and exploit a VAC-based value [Cooper2010a] and variable [Trosser2020a] ordering heuristics:

```
toulbar2 EXAMPLES/GeomSurf-7-gm256.uai.xz -A -V -vacint
```

```
Read 787 variables, with 7 values at most, and 3527 cost functions, with maximum␣
↪arity 3.
No evidence file specified. Trying EXAMPLES/GeomSurf-7-gm256.uai.evid
No evidence file.
Reverse original DAC dual bound: 5874975174 energy: 1073.679 (+0.003%)
Cost function decomposition time : 0.005 seconds.
Number of VAC iterations: 768
Number of times is VAC: 360 Number of times isvac and itThreshold > 1: 351
Preprocessing time: 2.085 seconds.
729 unassigned variables, 4818 values in all current domains (med. size:7, max␣
↪size:7) and 3132 non-unary cost functions (med. arity:2, med. degree:6)
Initial lower and upper bounds: [5899313907, 111615203929] 94.715%
New solution: 5926230227 energy: 1078.805 prob: 3.027e-469 (0 backtracks, 1 nodes,␣
↪depth 3, 2.088 seconds)
New solution: 5922482579 energy: 1078.430 prob: 4.404e-469 (0 backtracks, 1 nodes,␣
↪depth 2, 2.092 seconds)
Optimality gap: [5922482579, 5922482579] 0.000 % (0 backtracks, 1 nodes, 2.093␣
↪seconds)
Number of VAC iterations: 928
Number of times is VAC: 520 Number of times isvac and itThreshold > 1: 507
Node redundancy during HBFS: 0.000
Optimum: 5922482579 energy: 1078.430 prob: 4.404e-469 in 0 backtracks and 1 nodes (␣
↪481 removals by DEE) and 2.093 seconds.
end.
```

- Download another MRF file `1CM1.uai.xz`. Solve it by applying first an initial upper bound probing, and secondly, use a modified variable ordering heuristic based on VAC-integrality during search [Trosser2020a]:

```
toulbar2 EXAMPLES/1CM1.uai.xz -A=1000 -vacint -rasps -vacthr
```

```
Read 37 variables, with 350 values at most, and 703 cost functions, with maximum␣
↪arity 2.
No evidence file specified. Trying EXAMPLES/1CM1.uai.evid
No evidence file.
Reverse original DAC dual bound: 103967050722 energy: -12488.257 (+0.000%)
Cost function decomposition time : 0.001 seconds.
Number of VAC iterations: 3936
Number of times is VAC: 316 Number of times isvac and itThreshold > 1: 310
Threshold: 2326140792 NbAssignedVar: 0 Ratio: 0.0000000 SumOfDomainSize: 1783
Threshold: 2320180664 NbAssignedVar: 0 Ratio: 0.0000000 SumOfDomainSize: 1775
Threshold: 21288438 NbAssignedVar: 16 Ratio: 0.0000000 SumOfDomainSize: 84
Threshold: 11824293 NbAssignedVar: 17 Ratio: 0.0000000 SumOfDomainSize: 65
Threshold: 8188677 NbAssignedVar: 18 Ratio: 0.0000001 SumOfDomainSize: 52
Threshold: 6858840 NbAssignedVar: 19 Ratio: 0.0000001 SumOfDomainSize: 46
Threshold: 6060563 NbAssignedVar: 20 Ratio: 0.0000001 SumOfDomainSize: 42
Threshold: 5504852 NbAssignedVar: 20 Ratio: 0.0000001 SumOfDomainSize: 42
Threshold: 3972364 NbAssignedVar: 21 Ratio: 0.0000001 SumOfDomainSize: 37
Threshold: 3655432 NbAssignedVar: 21 Ratio: 0.0000002 SumOfDomainSize: 37
Threshold: 3067826 NbAssignedVar: 21 Ratio: 0.0000002 SumOfDomainSize: 37
Threshold: 2174446 NbAssignedVar: 22 Ratio: 0.0000003 SumOfDomainSize: 35
Threshold: 1641827 NbAssignedVar: 22 Ratio: 0.0000004 SumOfDomainSize: 35
Threshold: 1374852 NbAssignedVar: 22 Ratio: 0.0000004 SumOfDomainSize: 35
Threshold: 206113 NbAssignedVar: 23 Ratio: 0.0000030 SumOfDomainSize: 31
Threshold: 103056 NbAssignedVar: 24 Ratio: 0.0000063 SumOfDomainSize: 29
Threshold: 51528 NbAssignedVar: 25 Ratio: 0.0000131 SumOfDomainSize: 27
Threshold: 25764 NbAssignedVar: 25 Ratio: 0.0000262 SumOfDomainSize: 26
Threshold: 12882 NbAssignedVar: 25 Ratio: 0.0000525 SumOfDomainSize: 26
Threshold: 6441 NbAssignedVar: 25 Ratio: 0.0001049 SumOfDomainSize: 26
Threshold: 3220 NbAssignedVar: 25 Ratio: 0.0002098 SumOfDomainSize: 26
Threshold: 1610 NbAssignedVar: 25 Ratio: 0.0004197 SumOfDomainSize: 26
Threshold: 805 NbAssignedVar: 25 Ratio: 0.0008393 SumOfDomainSize: 26
Threshold: 402 NbAssignedVar: 25 Ratio: 0.0016808 SumOfDomainSize: 26
Threshold: 201 NbAssignedVar: 25 Ratio: 0.0033616 SumOfDomainSize: 26
Threshold: 100 NbAssignedVar: 25 Ratio: 0.0067568 SumOfDomainSize: 26
Threshold: 50 NbAssignedVar: 25 Ratio: 0.0135135 SumOfDomainSize: 26
Threshold: 25 NbAssignedVar: 25 Ratio: 0.0270270 SumOfDomainSize: 26
Threshold: 12 NbAssignedVar: 25 Ratio: 0.0563063 SumOfDomainSize: 26
Threshold: 6 NbAssignedVar: 25 Ratio: 0.1126126 SumOfDomainSize: 26
Threshold: 3 NbAssignedVar: 25 Ratio: 0.2252252 SumOfDomainSize: 26
Threshold: 1 NbAssignedVar: 25 Ratio: 0.6756757 SumOfDomainSize: 26
RASPS/VAC threshold: 201
Preprocessing time: 42.878 seconds.
37 unassigned variables, 3387 values in all current domains (med. size:38, max␣
↪size:322) and 627 non-unary cost functions (med. arity:2, med. degree:35)
Initial lower and upper bounds: [103967050722, 239074058140] 56.513%
New solution: 104206588507 energy: -12464.303 prob: inf (0 backtracks, 4 nodes,␣
↪depth 7, 42.886 seconds)
RASPS done in preprocessing at threshold 201 (backtrack: 2 nodes: 6)
New solution: 104174015034 energy: -12467.560 prob: inf (2 backtracks, 9 nodes,␣
↪depth 5, 42.910 seconds)
```

(continues on next page)

(continued from previous page)

```
Optimality gap: [104174015034, 104174015034] 0.000 % (4 backtracks, 11 nodes, 42.
↪910 seconds)
Number of VAC iterations: 4279
Number of times is VAC: 617 Number of times isvac and itThreshold > 1: 606
Node redundancy during HBFS: 0.000
Optimum: 104174015034 energy: -12467.560 prob: inf in 4 backtracks and 11 nodes (␣
↪921 removals by DEE) and 42.911 seconds.
end.
```

- Download a weighted Max-SAT file `brock200_4.clq.wcnf.xz` in wcnf format. Solve it using a modified variable ordering heuristic [Schiex2014a]:

```
toulbar2 EXAMPLES/brock200_4.clq.wcnf.xz -m=1
```

```
c Read 200 variables, with 2 values at most, and 7011 clauses, with maximum arity 2.
Reverse original DAC dual bound: 91 (+86.813%)
Reverse original DAC dual bound: 92 (+1.087%)
Cost function decomposition time : 0.000 seconds.
Preprocessing time: 0.061 seconds.
200 unassigned variables, 400 values in all current domains (med. size:2, max␣
↪size:2) and 6811 non-unary cost functions (med. arity:2, med. degree:68)
Initial lower and upper bounds: [92, 200] 54.000%
New solution: 189 (0 backtracks, 9 nodes, depth 11, 0.066 seconds)
New solution: 188 (45 backtracks, 143 nodes, depth 37, 0.103 seconds)
New solution: 187 (155 backtracks, 473 nodes, depth 47, 0.144 seconds)
New solution: 186 (273 backtracks, 802 nodes, depth 37, 0.170 seconds)
New solution: 185 (1870 backtracks, 4112 nodes, depth 63, 0.279 seconds)
New solution: 184 (19563 backtracks, 41433 nodes, depth 19, 1.430 seconds)
New solution: 183 (207409 backtracks, 459653 nodes, depth 12, 12.669 seconds)
Node redundancy during HBFS: 25.824
Optimum: 183 in 268922 backtracks and 725087 nodes ( 8484 removals by DEE) and 20.
↪814 seconds.
end.
```

- Download another WCSP file `latin4.wcsp.xz`. Count the number of feasible solutions:

```
toulbar2 EXAMPLES/latin4.wcsp.xz -a
```

```
Read 16 variables, with 4 values at most, and 24 cost functions, with maximum arity␣
↪4.
Reverse original DAC dual bound: 48 (+2.083%)
Cost function decomposition time : 0.000 seconds.
Preprocessing time: 0.008 seconds.
16 unassigned variables, 64 values in all current domains (med. size:4, max size:4)␣
↪and 8 non-unary cost functions (med. arity:4, med. degree:6)
Initial lower and upper bounds: [48, 1000] 95.200%
Optimality gap: [49, 1000] 95.100 % (17 backtracks, 41 nodes, 0.024 seconds)
Optimality gap: [58, 1000] 94.200 % (355 backtracks, 812 nodes, 0.175 seconds)
Optimality gap: [72, 1000] 92.800 % (575 backtracks, 1309 nodes, 0.265 seconds)
Optimality gap: [1000, 1000] 0.000 % (575 backtracks, 1318 nodes, 0.266 seconds)
Number of solutions    : =  576
Time                   :    0.266 seconds
```

(continues on next page)

(continued from previous page)

```
... in 575 backtracks and 1318 nodes
end.
```

- Find a greedy sequence of at most 20 diverse solutions with Hamming distance greater than 12 between any pair of solutions:

```
toulbar2 EXAMPLES/latin4.wcsp.xz -a=20 -div=12
```

```
Read 16 variables, with 4 values at most, and 24 cost functions, with maximum arity␣
↪4.
Reverse original DAC dual bound: 48 (+2.083%)
Cost function decomposition time : 0.000 seconds.
Preprocessing time: 0.018 seconds.
320 unassigned variables, 7968 values in all current domains (med. size:26, max␣
↪size:26) and 8 non-unary cost functions (med. arity:4, med. degree:0)
Initial lower and upper bounds: [48, 1000] 95.200%
+++++++++ Search for solution 1 +++++++++
New solution: 49 (0 backtracks, 7 nodes, depth 10, 0.023 seconds)
New solution: 48 (2 backtracks, 11 nodes, depth 3, 0.024 seconds)
Node redundancy during HBFS: 18.182
Optimum: 48 in 2 backtracks and 11 nodes ( 0 removals by DEE) and 0.024 seconds.
+++++++++ Search for solution 2 +++++++++
New solution: 67 (2 backtracks, 15 nodes, depth 7, 0.029 seconds)
New solution: 65 (3 backtracks, 20 nodes, depth 9, 0.032 seconds)
Optimality gap: [49, 65] 24.615 % (9 backtracks, 26 nodes, 0.035 seconds)
New solution: 58 (9 backtracks, 31 nodes, depth 4, 0.038 seconds)
Optimality gap: [50, 58] 13.793 % (10 backtracks, 32 nodes, 0.038 seconds)
New solution: 52 (16 backtracks, 53 nodes, depth 4, 0.046 seconds)
New solution: 51 (17 backtracks, 54 nodes, depth 3, 0.046 seconds)
Optimality gap: [51, 51] 0.000 % (17 backtracks, 54 nodes, 0.046 seconds)
Node redundancy during HBFS: 25.926
Optimum: 51 in 17 backtracks and 54 nodes ( 0 removals by DEE) and 0.046 seconds.
+++++++++ Search for solution 3 +++++++++
+++++++++ predictive bounding: 57.000
Optimality gap: [51, 57] 10.526 % (36 backtracks, 92 nodes, 0.074 seconds)
Optimality gap: [57, 57] 0.000 % (37 backtracks, 98 nodes, 0.078 seconds)
Node redundancy during HBFS: 18.367
No solution in 37 backtracks and 98 nodes ( 1 removals by DEE) and 0.078 seconds.
+++++++++ Search for solution 3 +++++++++
New solution: 64 (37 backtracks, 102 nodes, depth 7, 0.083 seconds)
New solution: 57 (38 backtracks, 105 nodes, depth 7, 0.085 seconds)
Optimality gap: [57, 57] 0.000 % (60 backtracks, 145 nodes, 0.111 seconds)
Node redundancy during HBFS: 12.414
Optimum: 57 in 60 backtracks and 145 nodes ( 2 removals by DEE) and 0.111 seconds.
+++++++++ Search for solution 4 +++++++++
+++++++++ predictive bounding: 69.000
New solution: 65 (66 backtracks, 159 nodes, depth 5, 0.121 seconds)
New solution: 61 (80 backtracks, 188 nodes, depth 6, 0.137 seconds)
New solution: 58 (96 backtracks, 219 nodes, depth 5, 0.156 seconds)
Optimality gap: [58, 58] 0.000 % (110 backtracks, 245 nodes, 0.172 seconds)
Node redundancy during HBFS: 7.347
Optimum: 58 in 110 backtracks and 245 nodes ( 9 removals by DEE) and 0.173 seconds.
```

(continues on next page)

(continued from previous page)

```
+++++++++ Search for solution 5 +++++++++
+++++++++ predictive bounding: 70.000
New solution: 62 (116 backtracks, 260 nodes, depth 6, 0.182 seconds)
New solution: 58 (146 backtracks, 320 nodes, depth 6, 0.221 seconds)
Node redundancy during HBFS: 5.625
Optimum: 58 in 146 backtracks and 320 nodes ( 11 removals by DEE) and 0.221 seconds.
+++++++++ Search for solution 6 +++++++++
+++++++++ predictive bounding: 70.000
New solution: 65 (176 backtracks, 382 nodes, depth 5, 0.253 seconds)
New solution: 62 (233 backtracks, 495 nodes, depth 4, 0.323 seconds)
Optimality gap: [62, 62] 0.000 % (238 backtracks, 504 nodes, 0.331 seconds)
Node redundancy during HBFS: 3.571
Optimum: 62 in 238 backtracks and 504 nodes ( 13 removals by DEE) and 0.331 seconds.
+++++++++ Search for solution 7 +++++++++
+++++++++ predictive bounding: 74.000
c 2097152 Bytes allocated for long long stack.
New solution: 68 (246 backtracks, 523 nodes, depth 6, 0.345 seconds)
New solution: 65 (254 backtracks, 539 nodes, depth 6, 0.354 seconds)
Optimality gap: [49, 65] 24.615 % (304 backtracks, 636 nodes, 0.416 seconds)
Optimality gap: [58, 65] 10.769 % (324 backtracks, 682 nodes, 0.442 seconds)
Optimality gap: [65, 65] 0.000 % (325 backtracks, 692 nodes, 0.444 seconds)
Node redundancy during HBFS: 4.624
Optimum: 65 in 325 backtracks and 692 nodes ( 22 removals by DEE) and 0.444 seconds.
+++++++++ Search for solution 8 +++++++++
+++++++++ predictive bounding: 77.000
Optimality gap: [49, 77] 36.364 % (380 backtracks, 835 nodes, 0.524 seconds)
Optimality gap: [50, 77] 35.065 % (389 backtracks, 874 nodes, 0.538 seconds)
Optimality gap: [51, 77] 33.766 % (392 backtracks, 887 nodes, 0.542 seconds)
Optimality gap: [52, 77] 32.468 % (397 backtracks, 905 nodes, 0.547 seconds)
New solution: 72 (403 backtracks, 923 nodes, depth 4, 0.555 seconds)
Optimality gap: [53, 72] 26.389 % (407 backtracks, 930 nodes, 0.559 seconds)
Optimality gap: [54, 72] 25.000 % (413 backtracks, 960 nodes, 0.567 seconds)
Optimality gap: [57, 72] 20.833 % (419 backtracks, 977 nodes, 0.572 seconds)
New solution: 71 (425 backtracks, 1004 nodes, depth 6, 0.582 seconds)
New solution: 70 (427 backtracks, 1009 nodes, depth 7, 0.584 seconds)
New solution: 65 (435 backtracks, 1022 nodes, depth 4, 0.591 seconds)
Node redundancy during HBFS: 13.796
Optimum: 65 in 435 backtracks and 1022 nodes ( 34 removals by DEE) and 0.591␣
↪seconds.
+++++++++ Search for solution 9 +++++++++
+++++++++ predictive bounding: 77.000
New solution: 73 (452 backtracks, 1059 nodes, depth 6, 0.614 seconds)
New solution: 68 (472 backtracks, 1101 nodes, depth 8, 0.637 seconds)
New solution: 66 (488 backtracks, 1130 nodes, depth 5, 0.652 seconds)
Optimality gap: [66, 66] 0.000 % (537 backtracks, 1226 nodes, 0.713 seconds)
Node redundancy during HBFS: 11.501
Optimum: 66 in 537 backtracks and 1226 nodes ( 51 removals by DEE) and 0.713␣
↪seconds.
+++++++++ Search for solution 10 +++++++++
+++++++++ predictive bounding: 74.000
New solution: 70 (546 backtracks, 1248 nodes, depth 7, 0.729 seconds)
New solution: 68 (560 backtracks, 1275 nodes, depth 6, 0.743 seconds)
```

(continues on next page)

(continued from previous page)

```
Optimality gap: [68, 68] 0.000 % (637 backtracks, 1426 nodes, 0.835 seconds)
Node redundancy during HBFS: 9.888
Optimum: 68 in 637 backtracks and 1426 nodes ( 63 removals by DEE) and 0.835␣
↪seconds.
+++++++++ Search for solution 11 +++++++++
+++++++++ predictive bounding: 76.000
New solution: 72 (638 backtracks, 1431 nodes, depth 6, 0.842 seconds)
New solution: 71 (648 backtracks, 1453 nodes, depth 8, 0.855 seconds)
New solution: 68 (703 backtracks, 1560 nodes, depth 5, 0.918 seconds)
Node redundancy during HBFS: 9.038
Optimum: 68 in 703 backtracks and 1560 nodes ( 71 removals by DEE) and 0.918␣
↪seconds.
+++++++++ Search for solution 12 +++++++++
+++++++++ predictive bounding: 76.000
New solution: 74 (740 backtracks, 1636 nodes, depth 5, 0.966 seconds)
New solution: 72 (770 backtracks, 1696 nodes, depth 5, 1.002 seconds)
New solution: 71 (781 backtracks, 1718 nodes, depth 5, 1.016 seconds)
Optimality gap: [71, 71] 0.000 % (835 backtracks, 1824 nodes, 1.077 seconds)
Node redundancy during HBFS: 7.730
Optimum: 71 in 835 backtracks and 1824 nodes ( 81 removals by DEE) and 1.077␣
↪seconds.
+++++++++ Search for solution 13 +++++++++
+++++++++ predictive bounding: 77.000
c 4194304 Bytes allocated for long long stack.
New solution: 76 (881 backtracks, 1919 nodes, depth 6, 1.144 seconds)
New solution: 72 (950 backtracks, 2056 nodes, depth 5, 1.220 seconds)
Optimality gap: [72, 72] 0.000 % (978 backtracks, 2110 nodes, 1.258 seconds)
Node redundancy during HBFS: 6.682
Optimum: 72 in 978 backtracks and 2110 nodes ( 88 removals by DEE) and 1.258␣
↪seconds.
+++++++++ Search for solution 14 +++++++++
+++++++++ predictive bounding: 78.000
New solution: 77 (1030 backtracks, 2217 nodes, depth 6, 1.321 seconds)
New solution: 76 (1050 backtracks, 2256 nodes, depth 5, 1.345 seconds)
New solution: 74 (1077 backtracks, 2312 nodes, depth 7, 1.384 seconds)
Optimality gap: [74, 74] 0.000 % (1109 backtracks, 2372 nodes, 1.418 seconds)
Node redundancy during HBFS: 5.944
Optimum: 74 in 1109 backtracks and 2372 nodes ( 103 removals by DEE) and 1.418␣
↪seconds.
+++++++++ Search for solution 15 +++++++++
+++++++++ predictive bounding: 80.000
New solution: 79 (1109 backtracks, 2376 nodes, depth 7, 1.427 seconds)
New solution: 78 (1163 backtracks, 2483 nodes, depth 6, 1.483 seconds)
New solution: 76 (1204 backtracks, 2564 nodes, depth 5, 1.533 seconds)
Optimality gap: [76, 76] 0.000 % (1248 backtracks, 2650 nodes, 1.588 seconds)
Node redundancy during HBFS: 5.321
Optimum: 76 in 1248 backtracks and 2650 nodes ( 116 removals by DEE) and 1.588␣
↪seconds.
+++++++++ Search for solution 16 +++++++++
+++++++++ predictive bounding: 82.000
New solution: 80 (1266 backtracks, 2688 nodes, depth 5, 1.615 seconds)
New solution: 79 (1283 backtracks, 2722 nodes, depth 5, 1.636 seconds)
```

(continues on next page)

(continued from previous page)

```
New solution: 78 (1328 backtracks, 2813 nodes, depth 6, 1.691 seconds)
Optimality gap: [78, 78] 0.000 % (1371 backtracks, 2896 nodes, 1.744 seconds)
Node redundancy during HBFS: 4.869
Optimum: 78 in 1371 backtracks and 2896 nodes ( 128 removals by DEE) and 1.744␣
↪seconds.
+++++++++ Search for solution 17 +++++++++
+++++++++ predictive bounding: 84.000
New solution: 80 (1371 backtracks, 2900 nodes, depth 7, 1.753 seconds)
New solution: 79 (1413 backtracks, 2983 nodes, depth 6, 1.798 seconds)
Optimality gap: [79, 79] 0.000 % (1482 backtracks, 3118 nodes, 1.883 seconds)
Node redundancy during HBFS: 4.522
Optimum: 79 in 1482 backtracks and 3118 nodes ( 135 removals by DEE) and 1.883␣
↪seconds.
+++++++++ Search for solution 18 +++++++++
+++++++++ predictive bounding: 85.000
New solution: 80 (1488 backtracks, 3133 nodes, depth 6, 1.898 seconds)
Optimality gap: [59, 80] 26.250 % (1610 backtracks, 3374 nodes, 2.039 seconds)
Optimality gap: [80, 80] 0.000 % (1621 backtracks, 3404 nodes, 2.057 seconds)
Node redundancy during HBFS: 4.377
Optimum: 80 in 1621 backtracks and 3404 nodes ( 136 removals by DEE) and 2.057␣
↪seconds.
+++++++++ Search for solution 19 +++++++++
+++++++++ predictive bounding: 84.000
New solution: 80 (1665 backtracks, 3496 nodes, depth 5, 2.122 seconds)
Node redundancy during HBFS: 4.319
Optimum: 80 in 1665 backtracks and 3496 nodes ( 138 removals by DEE) and 2.122␣
↪seconds.
+++++++++ Search for solution 20 +++++++++
+++++++++ predictive bounding: 84.000
Optimality gap: [49, 84] 41.667 % (1705 backtracks, 3586 nodes, 2.190 seconds)
Optimality gap: [51, 84] 39.286 % (1708 backtracks, 3597 nodes, 2.198 seconds)
Optimality gap: [52, 84] 38.095 % (1716 backtracks, 3640 nodes, 2.219 seconds)
Optimality gap: [53, 84] 36.905 % (1717 backtracks, 3650 nodes, 2.222 seconds)
Optimality gap: [54, 84] 35.714 % (1724 backtracks, 3687 nodes, 2.237 seconds)
Optimality gap: [55, 84] 34.524 % (1725 backtracks, 3694 nodes, 2.242 seconds)
Optimality gap: [56, 84] 33.333 % (1728 backtracks, 3715 nodes, 2.248 seconds)
Optimality gap: [57, 84] 32.143 % (1733 backtracks, 3751 nodes, 2.260 seconds)
Optimality gap: [58, 84] 30.952 % (1734 backtracks, 3760 nodes, 2.263 seconds)
Optimality gap: [60, 84] 28.571 % (1745 backtracks, 3816 nodes, 2.288 seconds)
Optimality gap: [61, 84] 27.381 % (1746 backtracks, 3824 nodes, 2.291 seconds)
Optimality gap: [62, 84] 26.190 % (1747 backtracks, 3834 nodes, 2.293 seconds)
Optimality gap: [63, 84] 25.000 % (1752 backtracks, 3866 nodes, 2.305 seconds)
Optimality gap: [64, 84] 23.810 % (1753 backtracks, 3880 nodes, 2.308 seconds)
Optimality gap: [65, 84] 22.619 % (1758 backtracks, 3908 nodes, 2.320 seconds)
Optimality gap: [66, 84] 21.429 % (1761 backtracks, 3930 nodes, 2.324 seconds)
Optimality gap: [67, 84] 20.238 % (1765 backtracks, 3961 nodes, 2.333 seconds)
Optimality gap: [68, 84] 19.048 % (1767 backtracks, 3987 nodes, 2.336 seconds)
Optimality gap: [71, 84] 15.476 % (1769 backtracks, 3999 nodes, 2.339 seconds)
Optimality gap: [84, 84] 0.000 % (1770 backtracks, 4006 nodes, 2.341 seconds)
Node redundancy during HBFS: 11.258
No solution in 1770 backtracks and 4006 nodes ( 141 removals by DEE) and 2.341␣
↪seconds.
```

(continues on next page)

(continued from previous page)

```
+++++++++ Search for solution 20 +++++++++
Optimality gap: [1000, 1000] 0.000 % (1863 backtracks, 4192 nodes, 2.479 seconds)
Node redundancy during HBFS: 10.759
No solution in 1863 backtracks and 4192 nodes ( 146 removals by DEE) and 2.479␣
↪seconds.
end.
```

- Download a crisp CSP file `GEOM40_6.wcsp.xz` (initial upper bound equal to 1). Count the number of solutions using #BTD [Favier2009a] using a min-fill variable ordering (warning, cannot use BTD to find all solutions in optimization):

```
toulbar2 EXAMPLES/GEOM40_6.wcsp.xz -O=-3 -a -B=1 -ub=1 -hbfs:
```

```
Read 40 variables, with 6 values at most, and 78 cost functions, with maximum arity␣
↪2.
Cost function decomposition time : 1.3e-05 seconds.
Preprocessing time: 0.001466 seconds.
40 unassigned variables, 240 values in all current domains (med. size:6, max␣
↪size:6) and 78 non-unary cost functions (med. arity:2, med. degree:4)
Initial lower and upper bounds: [0, 1] 100.000%
Get root cluster C1 with max. size: 5
Tree decomposition width  : 5
Tree decomposition height : 20
Number of clusters        : 29
Tree decomposition time: 0.001 seconds.
Number of solutions    : =  411110802705928379432960
Number of #goods       :    3993
Number of used #goods  :    17190
Size of sep            :    4
Time                   :    0.052 seconds
... in 13689 backtracks and 27378 nodes
end.
```

- Get a quick approximation of the number of solutions of a CSP with Approx#BTD [Favier2009a]:

```
toulbar2 EXAMPLES/GEOM40_6.wcsp.xz -O=-3 -a -B=1 -D -ub=1 -hbfs:
```

```
Read 40 variables, with 6 values at most, and 78 cost functions, with maximum arity␣
↪2.
Cost function decomposition time : 9e-06 seconds.
Preprocessing time: 0.001419 seconds.
40 unassigned variables, 240 values in all current domains (med. size:6, max␣
↪size:6) and 78 non-unary cost functions (med. arity:2, med. degree:4)
Initial lower and upper bounds: [0, 1] 100.000%

part 1 : 40 variables and 71 constraints (really added)
part 2 : 10 variables and 7 constraints (really added)
--> number of parts : 2
--> time : 0.000 seconds.

Get root cluster C22 with max. size: 5
Get root cluster C25 with max. size: 2
```

(continues on next page)

(continued from previous page)

```
Get root cluster C30 with max. size: 2
Tree decomposition width  : 5
Tree decomposition height : 17
Number of clusters        : 33
Tree decomposition time: 0.001 seconds.

Cartesian product                    :    133674945388437340315549622599 68
Upper bound of number of solutions : <= 1719926784000000000000000
Number of solutions    : ~= 480000000000000000000000
Number of #goods       :    468
Number of used #goods  :    4788
Size of sep            :    3
Time                   :    0.011 seconds
... in 3738 backtracks and 7476 nodes
end.
```

CHAPTER

SIX

POTENTIAL ISSUES

Toulbar2 implements tree search methods using a recursive procedure. It may overcome the current stack memory on very-large problems. On Linux, you can control stacksize limit (in bash, *'ulimit -s unlimited'*).

If your problem is huge, you may also consider recompiling toulbar2 with some specific cmake options depending on your needs: BINARYWCSP (restricted to binary cost functions only, without on-the-fly variable elimination nor VAC), TERNARYWCSP (restricted to binary and ternary cost functions only), INT_COSTS (32-bit integers for representing costs instead of 64-bit by default), SHORT_COSTS (16-bit integers for representing costs), SHORT_VALUES (16-bit integers for representing domain values instead of 32-bit by default).

CHAPTER

SEVEN

COMMAND LINE OPTIONS

If you just execute:

```
toulbar2
```

toulbar2 will give you its (long) list of optional parameters, that you can see in part *'Available options'* of : `ToulBar2 Help Message`.

To deactivate a default command line option, just use the command-line option followed by `:`. For example:

```
toulbar2 -dee: <file>
```

will disable the default Dead End Elimination [Givry2013a] (aka Soft Neighborhood Substitutability) preprocessing.

We now describe in more detail toulbar2 optional parameters.

7.1 General control

-agap=[decimal]
: stops search if the absolute optimality gap reduces below the given value (provides guaranteed approximation) (default value is 0)

-rgap=[double]
: stops search if the relative optimality gap reduces below the given value (provides guaranteed approximation) (default value is 0)

-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)

-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)

-sigma=[float]
: add a (truncated) random zero-centered gaussian noise for graphical models only (problem file extension .uai)

-timer=[integer]
: gives a CPU time limit in seconds. toulbar2 will stop after the specified CPU time has been consumed. The time limit is a CPU user time limit, not wall clock time limit.

-bt=[integer]
: gives a 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)

7.2 Preprocessing

-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, it assumes a complete assignment – used as initial upper bound and as value heuristic – is read from default file "sol", or, without the option -x, given as input filename with ".sol" extension)

-nopre deactivates all preprocessing options (equivalent to -e: -p: -t: -f: -dec: -n: -mst: -dee: -trws: -pwc: -hve:)

-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, or f=3 to do it before and after PWC) (resp. bijective (f=2)) variables (default value is 1)

-dec preprocessing only: pairwise decomposition [Favier2011a] of cost functions with arity >= 3 into smaller arity cost functions (default option)

-card preprocessing only: when reading opb or lp format, decomposes cardinality equality constraints into a network of binary and ternary constraints (option deactivated by default)

-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). See [Favier2011a].

-amo automatically detects at-most-one constraints and adds them to existing knapsack constraints (positive value) and/or directly in the cost function network up to a given absolute number (non-zero value except -1)

-mst find a maximum spanning tree ordering for DAC

-S=[integer]
preprocessing only: performs singleton consistency restricted to the first variables following the DAC ordering (or all the variables if no parameter is given).

-M=[integer]
preprocessing only: apply the Min Sum Diffusion algorithm (default is inactivated, with a number of iterations of 0). See [Cooper2010a].

-trws=[float]
preprocessing only: enforces TRW-S until a given precision is reached (default value is 0.001). See Kolmogorov 2006.

--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 a basic upper bound every N steps during TRW-S (default value is 100)

-hve=[integer]
hidden variable encoding with a given limit to the maximum domain size of hidden variables (default value is 0) A negative size limit means restoring the original encoding after preprocessing while keeping the improved dual bound. See also option -n to limit the maximum arity of dualized n-ary cost functions.

-pwc=[integer]
pairwise consistency by hidden variable encoding plus intersection constraints, each one bounded by a given maximum space limit (in MB) (default value is 0) A negative size limit means restoring the original encoding after preprocessing while keeping the improved dual bound. See also options -minqual, -hve to limit the domain size of hidden variables, and -n to limit the maximum arity of dualized n-ary cost functions.

-minqual	finds a minimal intersection constraint graph to achieve pairwise consistency (combine with option -pwc) (default option)

7.3 Initial upper bounding

-l=[integer]
limited discrepancy search [Ginsberg1995], 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 upper bound found by INCOP local search solver [idwalk:cp04]. The 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*.

-pils=["string"]
initial upper bound found by PILS local search solver. The string parameter is optional, using "3 0 0.333 100 500 10000 0.1 0.5 0.1 0.1" by default with the following meaning: *nbruns perturb_mode perturb_strength flatMaxIter nbEvalHC nbEvalMax strengthMin strengthMax incrFactor decrFactor*.

-lrbcd=["string"]
initial upperbound found by LR-BCD local search solver. The string parameter is optional, using "5 -2 3" by default with the following meaning: *maxiter rank nbroundings*. (a negative rank means dividing the theoretical rank by the given absolute value)

-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 a value heuristic – read from default file "sol" taken as a certificate or given directly as an additional input filename with ".sol" extension and without **-x**)

-ub=[decimal]
gives an initial upper bound

-rasps=[integer]
VAC-based upper bound probing heuristic (0: disable, >0: max. nb. of backtracks, 1000 if no integer given) (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 degrees)

-raspsini reset weighted degree variable ordering heuristic after doing upper bound probing

7.4 Tree search algorithms and tree decomposition selection

-hbfs=[integer]
hybrid best-first search [Katsirelos2015a], restarting from the root after a given number of backtracks (default value is 16384) (usage for parallel version: "mpirun -n [NbOfProcess] toulbar2 -hbfs problem.wcsp")

-hbfsmin=[integer]
hybrid best-first search compromise between BFS and DFS minimum node redundancy threshold (alpha percentage, default value is 5%)

-hbfsmax=[integer]
hybrid best-first search compromise between BFS and DFS maximum node redundancy threshold (beta percentage default value is 10%)

-open=[integer]
hybrid best-first search limit on the number of stored open nodes (default value is -1, i.e., no limit)

-sopen=[integer]
number of visited open nodes before sorting the remaining open nodes based on weighted degree heuristics (double this limit for the next sorting) (see also option -q) (default value is 0, i.e., no sorting)

-burst in parallel HBFS, workers send their solutions and open nodes as soon as possible (by default) For using a parallel version of HBFS, after compiling with MPI option (cmake -DMPI=ON .) use "mpirun -n [NbOfProcess] toulbar2 problem.wcsp"

-eps=[integer|filename]
Embarrassingly parallel search mode. It outputs a given number of open nodes in -x format and exit (default value is 0). See ./misc/script/eps.sh to run them. Use this option twice to specify the output filename.

-B=[integer]
(0) HBFS, (1) BTD-HBFS [Schiex2006a] [Katsirelos2015a], (2) RDS-BTD [Sanchez2009a], (3) RDS-BTD with path decomposition instead of tree decomposition [Sanchez2009a] (default value is 0)

-O=[filename]
reads either a reverse variable elimination order (given by a list of variable indexes) from a file in order to build a tree decomposition (if BTD-like and/or variable elimination methods are used) or reads a valid tree decomposition directly (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 first/root(s) cluster(s), followed by a list of variable indexes). It is also used as a 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
- (-8) lexicographic ordering of variable names
- (-9) topological ordering (for Bayesian networks only)

If not specified, then use the variable order in which variables appear in the problem file.

-root=[integer]
root cluster heuristic (0:largest, 1:max. size/(height-size), 2:min. size/(height-size), 3:min. height) (default value is 0)

-minheight minimizes cluster tree height when searching for the root cluster (can be slow to perform)

-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 the 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 the 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)

-F=[integer]
merges clusters automatically to give more freedom to variable ordering heuristic in BTD-HBFS (-1: no merging, positive value: maximum iteration value for trying to solve the same subtree given its separator assignment before considering it as unmerged) (default value is -1)

-R=[integer]
choice for a specific root cluster number

-I=[integer]
choice for solving only a particular rooted cluster subtree (with RDS-BTD only)

7.5 Variable neighborhood search algorithms

-vns unified decomposition guided variable neighborhood search [Ouali2017] (UDGVNS). A problem decomposition into clusters can be given as *.dec, *.cov, or *.order input files or using tree decomposition options such as -O. For a parallel version (UPDGVNS), use "mpirun -n [NbOfProcess] toulbar2 -vns problem.wcsp". For doing large neighborhood search (LNS) instead of VNS, and bounded backtrack search instead of LDS, use e.g., "toulbar2 -vns -l: -bt=1000 -kmin=50 -kmax=50 -L=100 problem.wcsp", for exploring 100 random neighborhoods of size 50 variables with at most 1000 backtracks per neighborhood search.

-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 DFBB and VNS-like methods if a better solution is found (default value is 0)

7.6 Node processing & bounding options

-e=[integer]
performs "on the fly" variable elimination of variable with small degree (less than or equal to a specified value, default is 3 creating a maximum of ternary cost functions). See [Larrosa2000].

-k=[integer]
soft local consistency level (NC [Larrosa2002] with Strong NIC for global cost functions=0 [LL2009], (G)AC=1 [Schiex2000b] [Larrosa2002], D(G)AC=2 [CooperFCSP], FD(G)AC=3 [Larrosa2003], (weak) ED(G)AC=4 [Heras2005] [LL2010]) (default value is 4). See also [Cooper2010a] [LL2012asa].

-A=[integer]
enforces VAC [Cooper2008] at each search node with a search depth less than a given value (default value is 0)

-V	VAC-based value ordering heuristic (default option)

-T=[decimal]
threshold cost value for VAC (any decimal cost below this threshold is considered as null by VAC thus speeding-up its convergence, default value is 1, except for the cfn format where it is equal to the decimal cost precision, e.g. 0.001 if 3 digits of precision)

-P=[decimal]
threshold cost value for VAC during the preprocessing phase only (default value is 1, except for the cfn format where it is equal to the decimal cost precision, e.g. 0.001 if 3 digits of precision)

-C=[float]
multiplies all costs internally by this number when loading the problem (cannot be done with cfn format and probabilistic graphical models in uai/LG formats) (default value is 1)

-vaclin	VAC applied on linear constraints (must be combined with option -A)
-vacthr	automatic threshold cost value selection for VAC during search (must be combined with option -A)

-dee=[integer]
restricted dead-end elimination [Givry2013a] (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)

-o ensures an optimal worst-case time complexity of DAC and EAC (can be slower in practice)

-kpdp=[integer]
solves knapsack constraints using dynamic programming (-2: never, -1: only in preprocessing, 0: at every search node, >0: after a given number of nodes) (default value is -2)

7.7 Branching, variable and value ordering

-svo searches using a static variable ordering heuristic. The variable order value used will be the same order as the DAC order.

-b searches using binary branching (by default) instead of n-ary branching. Uses binary branching for interval domains and small domains and dichotomic branching for large enumerated domains (see option -d).

-c searches using binary branching with last conflict backjumping variable ordering heuristic [Lecoutre2009].

-q=[integer]
use weighted degree variable ordering heuristic [boussemart2004] if the number of cost functions is less than the given value (default value is 1000000). A negative number will disconnect weighted degrees in embedded WeightedCSP constraints.

-var=[integer]
searches by branching only on the first [given value] decision variables, assuming the remaining variables are intermediate variables that will be completely assigned by the decision variables (use a zero if all variables are decision variables, default value is 0)

-m=[integer]
use a variable ordering heuristic that selects first variables such that the sum of the mean (m=1) or median (m=2) cost of all incident cost functions is maximum [Schiex2014a] (in conjunction with weighted degree heuristic -q) (default value is 0: unused).

-d=[integer]
searches using dichotomic branching. The default d=1 splits domains in the middle of domain range while d=2 splits domains in the middle of the sorted domain based on unary costs.

-sortd sorts domains in preprocessing based on increasing unary costs (works only for binary WCSPs).

-sortc sorts constraints in preprocessing based on lexicographic ordering (1), decreasing DAC ordering (2 - default option), decreasing constraint tightness (3), DAC then tightness (4), tightness then DAC (5), randomly (6), DAC with special knapsack order (7), increasing arity (8), increasing arity then DAC (9), or the opposite order if using a negative value.

-solr solution-based phase saving (reuse last found solution as preferred value assignment in the value ordering heuristic) (default option).

-vacint VAC-integrality/Full-EAC variable ordering heuristic (can be combined with option -A)

-bisupport=[float]
in bi-objective optimization with the second objective encapsulated by a bounding constraint (see WeightedCSPConstraint), the value heuristic chooses between both EAC supports of first (main) and second objectives by minimum weighted regret (if parameter is non-negative, it is used as the weight for the second objective) or always chooses the EAC support of the first objective (if parameter is zero) or always chooses the second objec-

tive (if parameter is negative, -1: for choosing EAC from the lower bound constraint, -2: from the upper bound constraint, -3: to favor the smallest gap, -4: to favor the largest gap) (default value is 0)

7.8 Diverse solutions

toulbar2 can search for a greedy sequence of diverse solutions with guaranteed local optimality and minimum pairwise Hamming distance [Ruffini2019a].

-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 3)

-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)

7.9 Console output

-help shows the default help message that toulbar2 prints when it gets no argument.

-v=[integer]
sets the verbosity level (default 0).

-Z=[integer]
debug mode (save problem at each node if verbosity option -v=num $>= 1$ and -Z=num $>= 3$)

-s=[integer]
shows each solution found during search. The solution is printed on one line, giving by default (-s=1) the value (integer) of each variable successively in increasing file order. For -s=2, the value name is used instead, and for -s=3, variable name=value name is printed instead.

7.10 File output

-w=[filename]
writes last/all solutions found in the specified filename (or "sol" if no parameter is given). The current directory is used as a relative path.

-w=[integer]
1: writes value numbers, 2: writes value names, 3: writes also variable names (default value is 1, this option can be used in combination with -w=filename).

-z=[filename]
saves problem in wcsp or cfn format in filename (or "problem.wcsp"/"problem.cfn" if no parameter is given) writes also the graphviz dot file and the degree distribution of the input problem (except if using a negative verbosity -v=-1)

-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, -2 or -4: saves after preprocessing but keeps initial domains (this option can be used in combination with -z=filename). If the problem is saved after preprocessing (except for -2 or -4), some variables may be lost (due to variable elimination, see -e or -p or -f).

7.11 Probability representation and numerical control

-precision=[integer]
probability/real precision is a conversion factor (a power of ten) for representing fixed point numbers (default value is 7). It is used by CFN/UAI/LP/QPBO/OPB/Pedigree formats. Note that in CFN format the number of significant digits is given in the problem description by default. This option allows to overwrite this default value.

-epsilon=[float]
approximation factor for computing the partition function (if greater than 1, default value is infinity) or floating-point precision (if smaller than 1, default value is 1e-9)

Note that in CFN format, costs are given as decimal numbers (the same for giving an initial upper bound, an absolute optimality gap or VAC threshold values) whereas in WCSP format costs are non-negative integers only.

7.12 Random problem generation

-random=[bench profile]
bench profile must be specified as follows.

- 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 number of binary cost functions to include
  - the seed parameter is optional

  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 number 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)

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

  nary-{n}-{d}-{t1}-{p2}-{p3}...-{pn}-{seed}
  - pn is the number of n-ary cost functions

  wcolor-{n}-{d}-0-{p2}-{seed} random weighted graph coloring problem
  - p2 is the number of edges in the graph

  vertexcover-{n}-{d}-{t1}-{p2}-{maxcost}-{seed} random vertex cover problem
  - t1 is the tightness (should be equal to 25)
  - p2 is the number of edges in the graph
  - maxcost each vertex has a weight randomly chosen between 0 and maxcost

  bivertexcover-{n}-{d}-{t1}-{p2}-{maxcost}-{ub2}-{seed} random bi-objective vertex cover problem
  - t1 is the tightness (should be equal to 25)
  - p2 is the number of edges in the graph
  - maxcost each vertex has two weights, both randomly chosen between 0 and maxcost

- ub2 upper bound for the bounding constraint on the second objective (see epsilon-constraint method)

salldiff-{n}-{d}-{t1}-{p2}-{p3}...-{pn}-{seed}

- pn is the number of salldiff global cost functions (p2 and p3 still being used for the number of random binary and ternary cost functions). *salldiff* can be replaced by *gcc* or *regular* keywords with three possible forms (*e.g., sgcc, sgccdp, wgcc*) and by *knapsack*.

CHAPTER

EIGHT

INPUT FORMATS

8.1 Introduction

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

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

Some examples :

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

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

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

8.2 Formats details

8.2.1 CFN format (.cfn suffix)

With this JSON-compatible format, it is possible:

- to give a name to variables and functions.
- to associate a local label to every value that is accessible inside toulbar2 (among others for heuristics design purposes).
- to use decimal and possibly negative costs.
- to solve both minimization and maximization problems.
- to debug your **.cfn** files: the parser gives a cause and line number when it fails.
- to use gzip'd or xz compressed files directly as input (.cfn.gz and .cfn.xz).
- to use dense descriptions for dense cost tables.

In a **cfn** file, a Cost Function Network is described as a JSON object with extra freedom and extra constraints.

Freedom:

- the double quotes around strings are not compulsory: both `"problem"` and `problem` are strings.
- double quotes can also be added around numbers: both `1.20` and `"1.20"` will be interpreted as decimal numbers.
- the commas that separate the fields inside an array or object are not compulsory. Any separator will do (comma, white space). So `[1, 2]` or `[1,2]` or `[1 2]` are all describing the same array.
- the delimiters for objects and arrays (`{}` and `[]`) can be used arbitrarily for both types of items.
- the colon (`:`) that separates the name of a field in an object from the contents of the field is not compulsory.
- It is possible to comment a line with a `#` the first position of a line.

Constraints:

- strings should not start with a character in `0123456789-.+` and cannot contain `/#[]{}:,` or a space character (tabs...).
- numbers can only be integers or decimals. No scientific notation.
- the order of fields inside an object is compulsory and cannot be changed.

A CFN is an object with 3 data: a definition of the main problem properties (tag `problem`), of variables and their domains (tag `variables`) and of cost functions (tag `functions`), in this order:

```
{ "problem": <problem properties>,
  "variables": <variables and domains>,
  "functions": <functions descriptions> }
```

Problem properties:

An object with two fields:

1. `"name"` : the name of the problem.

2. "mustbe" : specifies the direction of optimization and a global (upper/lower) bound on the objective. This is the concatenation of a comparator (> or <) immediately followed by a decimal number, described as a string. The comparator specifies the direction of optimization:

- "<": we are minimizing and the decimal indicates a global upper bound (all costs equal to or larger than this are considered as unfeasible).
- ">": we are maximizing and and the decimal indicates a global lower bound (all costs equal to or less than this are considered as unfeasible).

The number of significant digits in the decimal number gives the precision that will be used for all cost computations inside toulbar2.

As an example, "mustbe": "<10.00" means that the CFN describes a function where all costs larger than or equal to 10.00 are considered as infinite. All costs will also be handled with 2 digits of precision after the decimal point.

The two fields must appear in this order:

```
{ "name": "test_problem", "mustbe": "<-12.100" }
```

or

```
{test.problem <12.100}
```

in a more concise non-JSON-compatible form.

Variables and domains:

An object with as many fields as variables. All fields must have different names. The contents of a variable field can be an array or an integer. An array gives the sequence of values (defined by their name) of the variable domain. An integer gives the domain cardinality, without naming values (values are represented by their position in the domain, starting at 0). If a negative domain size is given, the variable is an interval variable instead of a finite domain variable and it has domain [0,-domainsize-1].

```
{ "fdv1": ["a", "b", "c"], "fdv2" : 2, "iv1" : -100}
```

defines 3 variables, two finite domain variables and 1 interval variable. The first domain variable has 3 values, "a" "b" and "c". the second has two anonymous values and the interval variable has domain [0,99].

As an extra freedom, it is possible to give no name to variables. This can be achieved using an array instead of an object. The example above can therefore be written:

```
[[a b c] 2 -100]
```

or even just

```
[3 2 -100]
```

in a dense non JSON-compatible format.

Functions:

An object with as many fields as functions. Every function is an object with different possible fields. All functions have a scope which is an array of variables (names or indices). The rest of the fields depends on the type of the cost function: table cost function or global (including arithmetic functions).

Table cost functions:

Sparse functions format:* useful for functions that are dominantly constant. A numerical `defaultcost` must be given after the scope. The `costs` table must be an array of tuple costs: a sequence of value names or indices followed by a numeric cost or `inf` to represent a forbidden tuple. The `defaultcost` is used to define the cost of any missing tuple.

```
{"scope": ["fdv1", "fdv2"],
 "defaultcost": 0.234,
 "costs": ["a", 0, 5,
           "a", 1, 6.2,
           "c", 0, -7.21] }
```

is a possible sparse function definition. Here only 3 tuples are defined with their costs. All 3 remaining tuples will have cost `0.234`.

Dense function format: if the `defaultcost` tag is absent, a complete lexicographically ordered list of costs is expected instead.

```
{"scope": [ "fdv1", "fdv2" ],
 "costs": [4.2, 3.67, -12.1, 7.1, -3.1, 100.2] }
```

describes the 6 costs of the 6 tuples insides the cartesian product of the two variables `"fdv1"` and `"fdv2"`. To assign costs to tuples, all possible tuples of the cartesian product are lexicographically ordered using the declared value order in the domain of each variable. In the example above, the order over the six pairs will be `("a",0) ("a",1) ("b",0) ("b",1) ("c",0) ("c",1)` that will be associated to the costs `4.2, 3.67, -12.1, 7.1, -3.1` and `100.2` in this order. This lexicographic ordering is used for all arities.

Shared function format: If instead of an array, a string is given for the cost table, then this string must be the name of a yet undefined function. The actual function will have the same cost table as the future indicated function (on the specified scope). The domain sizes of the two functions must match.

```
{"scope": [ "v1", "v3" ],
 "costs": "f12" }
```

defines a function on variables `v1` and `v3` that will have the same cost table as the function i:code:*f12* that must be defined later in the file.

Global and arithmetic cost functions

These functions are defined by a `scope`, a `type` and `parameters`. The `type` is a string that defines the specific function to use, the `parameters` is an array of objects. The composition of the `parameters` depends on the `type` of the function.

At this point, in maximization mode, most of the global cost functions have restricted usage (with the exception of wregular).

Arithmetic functions:

These functions have all arity 2 and it is assumed here that these variables are called x and y . The values are considered as representing their index in the domain and are therefore integer. The `type` can be either:

- ">=" : with `parameters` array $[cst, \delta]$ where cst and δ are two costs, to express cost function $max(0, y + cst - x \leq \delta ? y + cst - x : upperbound)$. This is a soft inequality with hard threshold δ.
- ">": similar with a strict inequality and semantics $max(0, y + 1 + cst - x \leq \delta ? y + 1 + cst - x : upperbound)$
- "<=": similar with an inverted inequality and semantics: $max(0, x - cst - y \leq \delta ? x - cst - y : upperbound)$
- "<": similar with a strict inequality and semantics $max(0, x - cst + 1 - y \leq \delta ? x - cst + 1 - y : upperbound)$
- "=": similar with an equality and semantics: similar with a strict inequality and semantics $|y + cst - x| \leq \delta ? |y + cst - x| : upperbound)$

- "disj": takes a parameters array $[cstx, csty, w]$ to express soft binary disjunctive cost function with semantics $((x \geq y + csty) \vee (y \geq x + cstx))?0 : w)$
- "sdisj": takes a parameters array $[cstx, csty, xmax, ymaxwxwy]$ to express a special disjunctive cost function with three implicit constraints $x \leq xmax$, $y \leq ymax$ and $(x < xmax \wedge y < ymax) \Rightarrow (x \geq y + csty \vee y \geq x + cstx)$ and an additional cost function $((x = xmax)?wx : 0) + ((y = ymax?wy : 0)$.

Example : arithmetic function with >= operator :

```
"arith0": {"scope": ["v5", "v6"],
           "type": ">=",
           "params": [1, 3]}
```

Global cost functions:

We use an informal syntactical description of each global cost function below. the "|" is used for alternative keywords and parentheses together with ?, * and + to denote optional or repeated groups of items (+ requires that at least one repetition exists). For more details on semantics and implementation, see:

> 1. Lee, J. H. M., & Leung, K. L. (2012). Consistency techniques for flow-based projection-safe global cost functions in weighted constraint satisfaction. *Journal of Artificial Intelligence Research*, 43, 257-292. *Artificial Intelligence*, 238, 166-189. 2. Allouche, D., Bessiere, C., Boizumault, P., De Givry, S., Gutierrez, P., Lee, J. H., … & Wu, Y. (2016). Tractability-preserving transformations of global cost functions. *Artificial Intelligence*, 238, 166-189.

Using a flow-based propagator:

- salldiff" with parameters array [metric: "var"|"dec"|"decbi" cost: cost] expresses a soft alldifferent with either variable-based (var keyword) or decomposition-based (dec and decbi keywords) cost semantic with a given cost per violation (decbi decomposes into a complete binary cost function network).
  - example :

    ```
    "f1": {"scope": ["v1" "v2" "v3" "v4"],
           "type": "salldiff",
           "params": {"metric": "var" "cost": 0.7}}
    ```

    generates a cost of 0.7 per variable assignment that needs to be changed for all variables to take a different value.
- "sgcc" with parameters array [metric:"var"|"dec"|"wdec" cost: cost bounds: [[value lower_bound upper_bound (shortage_weight excess_weight)?]*] expresses a soft global cardinality constraint with either variable-based (var keyword) or decomposition-based (dec keyword) cost semantic with a given cost per violation and for each value its lower and upper bound (value shortage and excess weights penalties must be given iff wdec is used).
  - example :

    ```
    name: {scope: [v1 v2 v3 v4]
           type: sgcc
           params: {
              metric: wdec
              cost: 0.5
              bounds: [[0 1 2 0.2 0.2]
                       [1 3 4 0.2 0.1]]
              }
          }
    ```

- "ssame" with parameters array [cost: cost vars1: [(variable)*] vars2: [(variable)*]] to express a permutation constraint on two lists of variables of equal size with implicit variable-based cost semantic
  - example :

```
name: {scope: [v1 v2 v3 v4]
       type : ssame
       params : {
          cost : 6.2
          vars1 : [v1 v2]
          vars2 : [v3 v4]
          }
      }
```

- "sregular" with parameters array [metric: "var"|"edit" cost: cost starts: [(state)*] ends: [(state)*] transitions: [(start-state symbol_value end_state)*] to express a soft regular constraint with either variable-based (var keyword) or edit distance-based (edit keyword) cost semantics with a given cost per violation followed by the definition of a deterministic finite automaton with arrays of initial and final states, and an array of state transitions where symbols are domain values indices.
  - example :

```
name: {scope: [v1 v2 v3 v4]
       type : sregular
       params : {
          metric: var
          cost: 1.0
          nb_states: 2
          starts: [0]
          ends: [0 1]
          transitions: [[0 0 0][0 1 1][1 1 1]]
          }
      }
```

Global cost functions using a dynamic programming DAG-based propagator:

- "sregulardp" with parameters array [metric: "var" cost: cost nb_states: nb_states starts: [(state)*] ends: [(state)*] transitions: [(start_state value_index end_state)*] to express a soft regular constraint with a variable-based (var keyword) cost semantic with a given cost per violation followed by the definition of a deterministic finite automaton with arrays of initial and final states, and an array of state transitions where symbols are domain value indices.
  - example: see sregular above.
- "sgrammar"|"sgrammardp" with parameters array [metric: "var"|"weight" cost: cost nb_symbols: nb_symbols nb_values: nb_values start: start_symbol terminals: [(terminal_symbol value (cost)?)*] non_terminals: [(nonterminal_in nonterminal_out_left nonterminal_out_right (cost)?)*] to express a soft/weighted grammar in Chomsky normal form. The costs inside the rules and terminals should be used only with the weight metric.
  - example:

```
name: {scope: [v1 v2 v3 v4]
       type : sgrammardp
       params: {
          metric : var
          cost : 1.012
```

(continues on next page)

(continued from previous page)

```
          nb_symbols : 4
          nb_values : 2
          start : 0
          terminals : [[1 0][3 1]]
          non_terminals : [[0 0 0][0 1 2][0 1 3][2 0 3]]
          }
      }
```

- "samong"|"samongdp" with parameters array [metric: "var" cost: cost min: lower_bound max: upper_bound values: [(value)*]] to express a soft among constraint to restrict the number of variables taking their value into a given set of value indices
  - example:

```
name: {scope: [v1 v2 v3 v4]
       type : samong
       params: {
          metric : var
          cost : 1.0
          min: 2
          max: 2
          values: [0]
          }
      }
```

- "salldiffdp" with parameters array [metric: "var" cost: cost] to express a soft alldifferent constraint with variable-based ("var" keyword) cost semantic with a given cost per violation (decomposes into samongdp cost functions)
  - example:

```
name: {scope: [v1 v2 v3 v4]
       type: salldiffdp
       params: {
          metric: var
          cost: 0.7
          }
      }
```

- "sgccdp" with parameters array [metric: "var" cost: "cost" bounds: [(value lower_bound upper_bound)*]] to express a soft global cardinality constraint with variable-based ("var" keyword) cost semantic with a given cost per violation and for each value its lower and upper bound (decomposes into samongdp cost functions)
  - example:

```
name: {scope: [v1 v2 v3 v4]
       type: sgccdp
       params: {
          metric: var
          cost: 1.1
          bounds: [[0 0 1] [1 2 3]]
          }
      }
```

- "max|smaxdp" with parameters array [defaultcost: defcost tuples: [(variable value cost)*]] to express a weighted max cost function to find the maximum cost over a set of unary cost functions associated to a set of variables (by default, defCost if unspecified)
  - example:

```
name: {scope: [v1 v2 v3 v4]
       type : smaxdp
       params: {
          defaultcost: 3
          tuples: [[0 0 4] [1 1 3][2 2 2][3 3 1]]
          }
       }
```

- "MST"|"smstdp" with empty parameters expresses a hard spanning tree constraint where each variable is assigned to its parent variable index in order to build a spanning tree (the root being assigned to itself)
  - example:

```
name: { scope: [v1 v2 v3 v4]
        type: MST params: []}
```

Global cost functions using a cost function network-based propagator (decompose to bounded arity table cost functions):

- "wregular" with parameters nb_states: nbstates starts: [[state cost]*] ends: [[state cost]*] transitions: [[state value_index state cost]*] to express a weighted regular constraint with weights on initial states, final states, and transitions, followed by the definition of a deterministic finite automaton with number of states, list of initial and final states with their costs, and list of weighted state transitions where symbols are domain value indices
  - example :

```
name: {scope: [v1 v2 v4 v3]
       type : wregular
       params: {
          nb_states: 4
          starts : [[0 0.0][1 0.5]]
          ends : [[2 -1.0] [3 0.0]]
          transitions : [[0 0 1 0.5][0 1 2 0.0]
                         [2 0 2 1.0][1 1 3 -1.0]]
          }
       }
```

- "walldiff" with parameters array [hard|lin|quad] cost to express a soft alldifferent constraint as a set of wamong hard constraint (hard keyword) or decomposition-based (lin and quad keywords) cost semantic with a given cost per violation.
  - example:

```
name: {scope: [v1 v2 v3 v4]
       type : walldiff
       params: {
          metric: lin
          cost: 0.8
          }
      }
```

- "wgcc" with parameters metric: hard|lin|quad cost: cost bounds: [[value lower_bound upper_bound]*] to express a soft global cardinality constraint as either a hard constraint (hard keyword) or with decomposition-based (lin and quad keyword) cost semantic with a given cost per violation and for each value its lower and upper bound
  - example:

```
name: {scope: [v1 v2 v3 v4]
       type : wgcc
       params: {
          metric: lin
          cost: 3.3
          bounds: [[0 0 1][1 2 2][2 0 1]]
          }
      }
```

- "wsame" with parameters a metric: hard|lin|quad cost: cost to express a permutation constraint on two lists of variables of equal size (implicitly concatenated in the scope) using implicit decomposition-based cost semantic
  - example:

```
name: { scope: [v1 v2 v3 v4]
        type : wsame
        params: {
           metric: lin
           cost: 3.3
           }
      }
```

- "wsamegcc" with parameters array metric: hard|lin|quad cost: cost bounds: [[value lower_bound upper_bound]*] to express the combination of a soft global cardinality constraint and a permutation constraint.
  - example:

```
name: {scope: [v1 v2 v3 v4]
       type : wsamegcc
       params: {
          metric: lin
          cost: 3.3
          bounds: [[0 0 1][1 0 1][2 0 1][3 0 0]]
          }
      }
```

- "wamong" with parameters metric: hard|lin|quad cost: cost values: [(value)*] min: lower_bound max: upper_bound to express a soft among constraint to restrict the number of variables taking their value into a given set of values.
  - example:

```
name: {scope: [v1 v2 v3 v4]
       type: wamong
       params: {
          metric: lin
          cost: 1
```

(continues on next page)

(continued from previous page)

```
          values: [0]
          min: 1
          max: 1
          }
      }
```

- "wvaramong" with parameters array `metric: hard cost: cost values: [(value)*]` to express a hard among constraint to restrict the number of variables taking their value into a given set of values to be equal to the last variable in the scope.
  - example:

```
name: {scope: [v1 v2 v3 v4 v5]
       type: wvaramong
       params: {
          metric: hard
          cost: 12.0
          values: [1]
          }
      }
```

- "woverlap" with parameters `metric: hard|lin|quad cost: cost comparator: comparator to: righthandside]` overlaps between two sequences of variables X, Y (i.e. set the fact that Xi and Yi take the same value (not equal to zero))
  - example:

```
name: {scope: [v1 v2 v3 v4]
       type: woverlap
       params: {
          metric: hard
          cost: 2.01
          comparator: ">"
          to: 1
          }
      }
```

- "wdiverse" with parameters `distance: integer values: [(value)*]` to express a hard diversity constraint using a dual encoding such that there is a given minimum Hamming distance to a given variable assignment (values).
  - example:

```
name: { scope: [v1 v2 v3 v4]
        type : wdiverse
        params: {
           distance: 2
           values: [0 1 0 1]
           }
      }
```

- "whdiverse" with parameters `distance: integer values: [(value)*]` to express a hard diversity constraint using a hidden encoding such that there is a given minimum Hamming distance to a given variable assignment (values).

  - example:

```
name: { scope: [v1 v2 v3 v4]
        type : whdiverse
        params: {
           distance: 2
           values: [0 1 0 1]
           }
      }
```

- "wtdiverse" with parameters distance: integer values: [(value)*] to express a hard diversity constraint using a ternary encoding such that there is a given minimum Hamming distance to a given variable assignment (values).

  - example:

```
name: { scope: [v1 v2 v3 v4]
        type : wtdiverse
        params: {
           distance: 2
           values: [0 1 0 1]
           }
      }
```

- "wsum" parameters metric: hard|lin|quad cost: cost comparator: comparator to: righthandside to express a soft sum constraint with unit coefficients to test if the sum of a set of variables matches with a given comparator and right-hand-side value.

  - example:

```
name: {scope: [v1 v2 v3 v4]
       type: wsum
       params: {
          metric: quad
          cost: 1.0
          comparator: "<="
          to: 4
          }
      }
```

- "wvarsum" with parameters metric: hard cost: cost comparator: comparator to express a hard sum constraint to restrict the sum to be comparator to the value of the last variable in the scope.

  - example:

```
mywsum: {scope: [v1 v2 v3 v4]
         type : wvarsum
         params: {
            metric: hard
            cost: 3
            comparator: "=="
            }
        }
```

  Comparators: let us note <> the comparator, K the right-hand-side (to:) value associated to the comparator, and Sum the result of the sum over the variables. For each comparator, the gap is defined according to the distance

as follows:

- if <> is == : gap = abs(K - Sum)
- if <> is <= : gap = max(0,Sum - K)
- if <> is < : gap = max(0,Sum - K - 1)
- if <> is != : gap = 1 if Sum != K and gap = 0 otherwise
- if <> is > : gap = max(0,K - Sum + 1);
- if <> is >= : gap = max(0,K - Sum);

Warning: the decomposition of `wsum` and `wvarsum` may use an exponential size (sum of domain sizes). list_size1 and list_size2 must be equal in `ssame`.

Global cost functions using a dedicated propagator:

- "`knapsack`" with parameters `capacity: capacity weights: [(coefficient)*]` to express a hard global reverse knapsack constraint (i.e., a linear constraint on 0/1 variables with >= operator) where capacity and coefficients (one for each variable in the scope) are positive or negative integers. Use negative numbers to express a linear constraint with <= operator. See below a simple example encoding `v1+v2+v3+v4 >= 1`.
  - example:

```
myknapsack: {scope: [v1 v2 v3 v4]
         type : knapsack
         params: {
            capacity: 1
            weights: [1 1 1 1]
            }
        }
```

- "`knapsackv`" with parameters `capacity: capacity weightedvalues: [([variable value coefficient])*]` to express a hard global reverse knapsack constraint (i.e., a generalized linear constraint on domain variables with >= operator) where capacity and coefficients are positive or negative integers. Use negative numbers to express a generalized linear constraint with <= operator. Variables can be names or indices in the whole problem. They must also belong to the scope. See below a simple example encoding `(v1=1)+(v2=1)+(v3=1)+(v4=1) >= 1`.
  - example:

```
myknapsackv: {scope: [v1 v2 v3 v4]
         type : knapsackv
         params: {
            capacity: 1
            weightedvalues: [[v1 1 1] [v2 1 1] [v3 1 1] [v4 1 1]]
            }
        }
```

- "`salldiffkp`" with parameters array `[metric: "hard" cost: inf]` to express a hard alldifferent constraint (decomposes into `knapsackv` cost functions)
  - example:

```
name: {scope: [v1 v2 v3 v4]
       type: salldiffkp
       params: {
          metric: hard
```

(continues on next page)

(continued from previous page)

```
          cost: inf
          }
     }
```

- "clique" with parameters rhs: 1 values: [([(value)*])*] to express a hard global clique constraint to restrict the number of variables taking their value into a given set of values (one set per variable) to at most 1 occurrence for all the variables. A clique of binary constraints must also be added to forbid any two variables from using both the restricted values.
  - example:

```
f01: { scope: [v0 v1] defaultcost: 0 costs: [1 1 inf]}
f02: { scope: [v0 v2] defaultcost: 0 costs: [1 1 inf]}
f03: { scope: [v0 v3] defaultcost: 0 costs: [1 1 inf]}
f12: { scope: [v1 v2] defaultcost: 0 costs: [1 1 inf]}
f13: { scope: [v1 v3] defaultcost: 0 costs: [1 1 inf]}
f23: { scope: [v2 v3] defaultcost: 0 costs: [1 1 inf]}
myclique: {scope: [v0 v1 v2 v3]
         type : clique
         params: {
            rhs: 1
            values: [[1], [1], [1], [1]]
            }
        }
```

- "cfnconstraint" with parameters cfn: cost-function-network lb: cost ub: cost duplicatehard: value strongduality: value to express a hard global constraint on the cost of an input weighted constraint satisfaction problem in cfn format such that its valid solutions must have a cost value in [lb,ub[.
  - "duplicatehard" (0|1): if true then it assumes any forbidden tuple in the original input problem is also forbidden by another constraint in the main model (you must duplicate any hard constraints in your input model into the main model).
  - "strongduality" (0|1): if true then it assumes the propagation is complete when all channeling variables in the scope are assigned and the semantic of the constraint enforces that the optimum and ONLY the optimum on the remaining variables is between lb and ub.
  - example :

```
name: {scope: [v1 v2 v4]
       type : cfnconstraint
       params: {
          cfn:
            {
            problem: {name: "subcfn", mustbe: "<1000.0"}
            variables: {v1:2, v2:2, v4:2}
            functions: {
               {scope: [v1], costs: [0.0, -3.0]},
               {scope: [v2], costs: [-1.0, 0.0]},
               {scope: [v4], costs: [0.0, 2.0]}}
            }
          lb : -1.0
          ub : 0.0
```

(continues on next page)

(continued from previous page)

```
        duplicatehard: 0
        strongduality: 0
        }
    }
```

Warning: the same floatting-point precision and optimization sense (minimization or maximization) should be used by the encapsulated cost function network and the main model. Warning: the list of variables of the encapsulated cost function network should be exactly the same as the scope (and with the same order).

8.2.2 Weighted Constraint Satisfaction Problem file format (wcsp)

group **Weighted Constraint Satisfaction Problem file format (wcsp)**

It is a text format composed of a list of numerical and string terms separated by spaces. Instead of using names for making reference to variables, variable indexes are employed. The same for domain values. All indexes start at zero.

Cost functions can be defined in intention (see below) or in extension, by their list of tuples. A default cost value is defined per function in order to reduce the size of the list. Only tuples with a different cost value should be given (not mandatory). All the cost values must be positive. The arity of a cost function in extension may be equal to zero. In this case, there is no tuples and the default cost value is added to the cost of any solution. This can be used to represent a global lower bound constant of the problem.

The wcsp file format is composed of three parts: a problem header, the list of variable domain sizes, and the list of cost functions.

- Header definition for a given problem:

```
<Problem name>
<Number of variables (N)>
<Maximum domain size>
<Number of cost functions>
<Initial global upper bound of the problem (UB)>
```

  The goal is to find an assignment of all the variables with minimum total cost, strictly lower than UB. Tuples with a cost greater than or equal to UB are forbidden (hard constraint).

- Definition of domain sizes

```
<Domain size of variable with index 0>
...
<Domain size of variable with index N - 1>
```

Note : domain values range from zero to *size-1*

Note : a negative domain size is interpreted as a variable with an interval domain in $[0, -size - 1]$

Warning : variables with interval domains are restricted to arithmetic and disjunctive cost functions in intention (see below)

- General definition of cost functions
  - Definition of a cost function in extension

```
<Arity of the cost function>
<Index of the first variable in the scope of the cost function>
...
<Index of the last variable in the scope of the cost function>
<Default cost value>
<Number of tuples with a cost different than the default cost>
```

followed by for every tuple with a cost different than the default cost:

```
<Index of the value assigned to the first variable in the scope>
...
<Index of the value assigned to the last variable in the scope>
<Cost of the tuple>
```

Note : Shared cost function: A cost function in extension can be shared by several cost functions with the same arity (and same domain sizes) but different scopes. In order to do that, the cost function to be shared must start by a negative scope size. Each shared cost function implicitly receives an occurrence number starting from 1 and incremented at each new shared definition. New cost functions in extension can reuse some previously defined shared cost functions in extension by using a negative number of tuples representing the occurrence number of the desired shared cost function. Note that default costs should be the same in the shared and new cost functions. Here is an example of 4 variables with domain size 4 and one AllDifferent hard constraint decomposed into 6 binary constraints.

- Shared CF used inside a small example in wcsp format:

```
AllDifferentDecomposedIntoBinaryConstraints 4 4 6 1
4 4 4 4
-2 0 1 0 4
0 0 1
1 1 1
2 2 1
3 3 1
2 0 2 0 -1
2 0 3 0 -1
2 1 2 0 -1
2 1 3 0 -1
2 2 3 0 -1
```

- Definition of a cost function in intension by replacing the default cost value by -1 and by giving its keyword name and its K parameters

```
<Arity of the cost function>
<Index of the first variable in the scope of the cost function>
...
<Index of the last variable in the scope of the cost function>
-1
<keyword>
<parameter1>
...
<parameterK>
```

Possible keywords of cost functions defined in intension followed by their specific parameters:

- >= *cst delta* to express soft binary constraint $x \geq y+cst$ with associated cost function $max((y+cst-x \leq delta)?(y+cst-x) : UB, 0)$
- > *cst delta* to express soft binary constraint $x > y+cst$ with associated cost function $max((y+cst+1-x \leq delta)?(y+cst+1-x) : UB, 0)$
- <= *cst delta* to express soft binary constraint $x \leq y+cst$ with associated cost function $max((x-cst-y \leq delta)?(x-cst-y) : UB, 0)$
- < *cst delta* to express soft binary constraint $x < y+cst$ with associated cost function $max((x-cst+1-y \leq delta)?(x-cst+1-y) : UB, 0)$
- = *cst delta* to express soft binary constraint $x = y + cst$ with associated cost function $(|y+cst-x| \leq delta)?|y+cst-x| : UB$
- disj *cstx csty penalty* to express soft binary disjunctive constraint $x \geq y + csty \vee y \geq x + cstx$ with associated cost function $(x \geq y + csty \vee y \geq x + cstx)?0 : penalty$
- sdisj *cstx csty xinfty yinfty costx costy* to express a special disjunctive constraint with three implicit hard constraints $x \leq xinfty$ and $y \leq yinfty$ and $x < xinfty \wedge y < yinfty \Rightarrow (x \geq y+csty \vee y \geq x+cstx)$ and an additional cost function $((x = xinfty)?costx : 0) + ((y = yinfty)?costy : 0)$
- Global cost functions using a dedicated propagator:
  - clique *1* (*nb_values* (*value*)*)* to express a hard clique cut to restrict the number of variables taking their value into a given set of values (per variable) to at most *1* occurrence for all the variables (warning! it assumes also a clique of binary constraints already exists to forbid any two variables using both the restricted values)
  - knapsack *capacity* (*weight*)* to express a reverse knapsack constraint (i.e., a linear constraint on 0/1 variables with >= operator) with capacity and weights are positive or negative integer coefficients (use negative numbers to express a linear constraint with <= operator)
  - knapsackc *capacity* (*weight*)* *nb_AMO* (*nb_variables* (*variable value*)*)* to express a reverse knapsack constraint (i.e., a linear constraint on 0/1 variables with >= operator) combined with a list of non-overlapping at-most-one constraints
  - knapsackp *capacity* (*nb_values* (*value weight*)*)* to express a reverse knapsack constraint with for each variable the list of values to select the item in the knapsack with their corresponding weight
  - knapsackv *capacity nb_triplets* (*variable value weight*)* to express a reverse knapsack constraint with a list of triplets variable, value, and its corresponding weight
  - salldiffkp hard *UB* to express a hard alldifferent constraint (decomposes into knapsack cost functions)
  - wcsp *lb ub duplicatehard strongduality wcsp* to express a hard global constraint on the cost of an input weighted constraint satisfaction problem in wcsp format such that its valid solutions must have a cost value in [lb,ub[.
- Global cost functions using a flow-based propagator:
  - salldiff var|dec|decbi *cost* to express a soft alldifferent constraint with either variable-based (*var* keyword) or decomposition-based (*dec* and *decbi* keywords) cost semantic with a given *cost* per violation (*decbi* decomposes into a binary cost function complete network)
  - sgcc var|dec|wdec *cost nb_values* (*value lower_bound upper_bound* (*shortage_weight excess_weight*)?)* to express a soft global cardinality constraint with either variable-based (*var* keyword) or decomposition-based (*dec* keyword) cost semantic with a given *cost* per violation and for each value its lower and upper bound (if *wdec* then violation cost depends on each value shortage or excess weights)
  - ssame *cost list_size1 list_size2* (*variable_index*)* (*variable_index*)* to express a permutation constraint on two lists of variables of equal size (implicit variable-based cost semantic)

  - sregular var|edit *cost nb_states nb_initial_states* (*state*)* *nb_final_states* (*state*)* *nb_transitions* (*start_state symbol_value end_state*)* to express a soft regular constraint with either variable-based (*var* keyword) or edit distance-based (*edit* keyword) cost semantic with a given *cost* per violation followed by the definition of a deterministic finite automaton with number of states, list of initial and final states, and list of state transitions where symbols are domain values

- Global cost functions using a dynamic programming DAG-based propagator:
  - sregulardp var *cost nb_states nb_initial_states* (*state*)* *nb_final_states* (*state*)* *nb_transitions* (*start_state symbol_value end_state*)* to express a soft regular constraint with a variable-based (*var* keyword) cost semantic with a given *cost* per violation followed by the definition of a deterministic finite automaton with number of states, list of initial and final states, and list of state transitions where symbols are domain values
  - sgrammar|sgrammardp var|weight *cost nb_symbols nb_values start_symbol nb_rules* ((0 *terminal_symbol value*)|(1 *nonterminal_in nonterminal_out_left nonterminal_out_right*)|(2 *terminal_symbol value weight*)|(3 *nonterminal_in nonterminal_out_left nonterminal_out_right weight*))* to express a soft/weighted grammar in Chomsky normal form
  - samong|samongdp var *cost lower_bound upper_bound nb_values* (*value*)* to express a soft among constraint to restrict the number of variables taking their value into a given set of values
  - salldiffdp var *cost* to express a soft alldifferent constraint with variable-based (*var* keyword) cost semantic with a given *cost* per violation (decomposes into samongdp cost functions)
  - sgccdp var *cost nb_values* (*value lower_bound upper_bound*)* to express a soft global cardinality constraint with variable-based (*var* keyword) cost semantic with a given *cost* per violation and for each value its lower and upper bound (decomposes into samongdp cost functions)
  - max|smaxdp *defCost nbtuples* (*variable value cost*)* to express a weighted max cost function to find the maximum cost over a set of unary cost functions associated to a set of variables (by default, *defCost* if unspecified)
  - MST|smstdp to express a spanning tree hard constraint where each variable is assigned to its parent variable index in order to build a spanning tree (the root being assigned to itself)

- Global cost functions using a cost function network-based propagator:
  - wregular *nb_states nb_initial_states* (*state* and cost)* *nb_final_states* (*state* and cost)* *nb_transitions* (*start_state symbol_value end_state cost*)* to express a weighted regular constraint with weights on initial states, final states, and transitions, followed by the definition of a deterministic finite automaton with number of states, list of initial and final states with their costs, and list of weighted state transitions where symbols are domain values
  - walldiff hard|lin|quad *cost* to express a soft alldifferent constraint as a set of wamong hard constraint (*hard* keyword) or decomposition-based (*lin* and *quad* keywords) cost semantic with a given *cost* per violation
  - wgcc hard|lin|quad *cost nb_values* (*value lower_bound upper_bound*)* to express a soft global cardinality constraint as either a hard constraint (*hard* keyword) or with decomposition-based (*lin* and *quad* keyword) cost semantic with a given *cost* per violation and for each value its lower and upper bound
  - wsame hard|lin|quad *cost* to express a permutation constraint on two lists of variables of equal size (implicitly concatenated in the scope) using implicit decomposition-based cost semantic
  - wsamegcc hard|lin|quad *cost nb_values* (*value lower_bound upper_bound*)* to express the combination of a soft global cardinality constraint and a permutation constraint
  - wamong hard|lin|quad *cost nb_values* (*value*)* *lower_bound upper_bound* to express a soft among constraint to restrict the number of variables taking their value into a given set of values

- wvaramong hard *cost nb_values* (*value*)* to express a hard among constraint to restrict the number of variables taking their value into a given set of values to be equal to the last variable in the scope
- woverlap hard|lin|quad *cost comparator righthandside* overlaps between two sequences of variables X, Y (i.e. set the fact that Xi and Yi take the same value (not equal to zero))
- wsum hard|lin|quad *cost comparator righthandside* to express a soft sum constraint with unit coefficients to test if the sum of a set of variables matches with a given comparator and right-hand-side value
- wvarsum hard *cost comparator* to express a hard sum constraint to restrict the sum to be *comparator* to the value of the last variable in the scope
- wdiverse *distance* (*value*)* to express a hard diversity constraint using a dual encoding such that there is a given minimum Hamming distance to a given variable assignment
- whdiverse *distance* (*value*)* to express a hard diversity constraint using a hidden encoding such that there is a given minimum Hamming distance to a given variable assignment
- wtdiverse *distance* (*value*)* to express a hard diversity constraint using a ternary encoding such that there is a given minimum Hamming distance to a given variable assignment

  Let us note <> the comparator, K the right-hand-side value associated to the comparator, and Sum the result of the sum over the variables. For each comparator, the gap is defined according to the distance as follows:

  * if <> is == : gap = abs(K - Sum)
  * if <> is <= : gap = max(0,Sum - K)
  * if <> is < : gap = max(0,Sum - K - 1)
  * if <> is != : gap = 1 if Sum != K and gap = 0 otherwise
  * if <> is > : gap = max(0,K - Sum + 1);
  * if <> is >= : gap = max(0,K - Sum);

Warning : The decomposition of wsum and wvarsum may use an exponential size (sum of domain sizes).

Warning : *list_size1* and *list_size2* must be equal in *ssame*.

Warning : Cost functions defined in intention cannot be shared.

Note More about network-based global cost functions can be found on ./misc/doc/DecomposableGlobalCostFunctions.html

Examples:

- quadratic cost function $x0 * x1$ in extension with variable domains $\{0, 1\}$ (equivalent to a soft clause $\neg x0 \vee \neg x1$):

```
2 0 1 0 1 1 1 1
```

- simple arithmetic hard constraint $x1 < x2$:

```
2 1 2 -1 < 0 0
```

- hard temporal disjunction $x1 \geq x2 + 2 \vee x2 \geq x1 + 1$:

```
2 1 2 -1 disj 1 2 UB
```

- clique cut ({x0,x1,x2,x3}) on Boolean variables such that value 1 is used at most once:

```
4 0 1 2 3 -1 clique 1 1 1 1 1 1 1 1 1
```

- knapsack constraint ($2 * x0 + 3 * x1 + 4 * x2 + 5 * x3 >= 10$) on four Boolean 0/1 variables:

```
4 0 1 2 3 -1 knapsack 10 2 3 4 5
```

- knapsackc constraint ($2 * x0 + 3 * x1 + 4 * x2 + 5 * x3 >= 10$, $x1 + x2 <= 1$) on four Boolean 0/1 variables:

```
4 0 1 2 3 -1 knapsackc 10 2 3 4 5 1 2 1 1 2 1
```

- knapsackp constraint ($2 * (x0 = 0) + 3 * (x1 = 1) + 4 * (x2 = 2) + 5 * (x3 = 0 \vee x3 = 1) >= 10$) on four {0,1,2}-domain variables:

```
4 0 1 2 3 -1 knapsackp 10 1 0 2 1 1 3 1 2 4 2 0 5 1 5
```

- knapsackv constraint ($2 * (x0 = 0) + 3 * (x1 = 1) + 4 * (x2 = 2) + 5 * (x3 = 0 \vee x3 = 1) >= 10$) on four {0,1,2}-domain variables:

```
4 0 1 2 3 -1 knapsackv 10 5 0 0 2 1 1 3 2 2 4 3 0 5 3 1 5
```

- wcsp constraint ($3 <= 2 * x1 * x2 + 3 * x1 * x4 + 4 * x2 * x4 < 5$) on three Boolean 0/1 variables:

```
3 1 2 4 -1 wcsp 3 5 0 0 name 3 2 3 1000 2 2 2 2 0 1 0 1 1 1 2 2 0 2 0 1 1 1 3 2␣
↪1 2 0 1 1 1 4
```

- soft_alldifferent({x0,x1,x2,x3}):

```
4 0 1 2 3 -1 salldiff var 1
```

- soft_gcc({x1,x2,x3,x4}) with each value v from 1 to 4 only appearing at least v-1 and at most v+1 times:

```
4 1 2 3 4 -1 sgcc var 1 4 1 0 2 2 1 3 3 2 4 4 3 5
```

- soft_same({x0,x1,x2,x3},{x4,x5,x6,x7}):

```
8 0 1 2 3 4 5 6 7 -1 ssame 1 4 4 0 1 2 3 4 5 6 7
```

- soft_regular({x1,x2,x3,x4}) with DFA (3*)+(4*):

```
4 1 2 3 4 -1 sregular var 1 2 1 0 2 0 1 3 0 3 0 0 4 1 1 4 1
```

- soft_grammar({x0,x1,x2,x3}) with hard cost (1000) producing well-formed parenthesis expressions:

```
4 0 1 2 3 -1 sgrammardp var 1000 4 2 0 6 1 0 0 0 1 0 1 2 1 0 1 3 1 2 0 3 0 1 0␣
↪0 3 1
```

- soft_among({x1,x2,x3,x4}) with hard cost (1000) if $\sum_{i=1}^{4}(x_i \in \{1,2\}) < 1$ or $\sum_{i=1}^{4}(x_i \in \{1,2\}) > 3$:

```
4 1 2 3 4 -1 samongdp var 1000 1 3 2 1 2
```

- soft max({x0,x1,x2,x3}) with cost equal to $\max_{i=0}^{3}((x_i! = i)?1000 : (4 - i))$:

```
4 0 1 2 3 -1 smaxdp 1000 4 0 0 4 1 1 3 2 2 2 3 3 1
```

- wregular({x0,x1,x2,x3}) with DFA (0(10)*2*):

```
4 0 1 2 3 -1 wregular 3 1 0 0 1 2 0 9 0 0 1 0 0 1 1 1 0 2 1 1 1 1 0 0 1 0 0 1 1␣
↪2 0 1 1 2 2 0 1 0 2 1 1 1 2 1
```

- wamong({x1,x2,x3,x4}) with hard cost (1000) if $\sum_{i=1}^{4}(x_i \in \{1,2\}) < 1$ or $\sum_{i=1}^{4}(x_i \in \{1,2\}) > 3$:

```
4 1 2 3 4 -1 wamong hard 1000 2 1 2 1 3
```

- wvaramong({x1,x2,x3,x4}) with hard cost (1000) if $\sum_{i=1}^{3}(x_i \in \{1,2\}) \neq x_4$:

```
4 1 2 3 4 -1 wvaramong hard 1000 2 1 2
```

- woverlap({x1,x2,x3,x4}) with hard cost (1000) if $\sum_{i=1}^{2}(x_i = x_{i+2}) \geq 1$:

```
4 1 2 3 4 -1 woverlap hard 1000 < 1
```

- wsum({x1,x2,x3,x4}) with hard cost (1000) if $\sum_{i=1}^{4}(x_i) \neq 4$:

```
4 1 2 3 4 -1 wsum hard 1000 == 4
```

- wvarsum({x1,x2,x3,x4}) with hard cost (1000) if $\sum_{i=1}^{3}(x_i) \neq x_4$:

```
4 1 2 3 4 -1 wvarsum hard 1000 ==
```

- wdiverse({x0,x1,x2,x3}) hard constraint on four variables with minimum Hamming distance of 2 to the value assignment (1,1,0,0):

```
4 0 1 2 3 -1 wdiverse 2 1 1 0 0
```

Latin Square 4 x 4 crisp CSP example in wcsp format:

```
latin4 16 4 8 1
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 0 1 2 3 -1 salldiff var 1
4 4 5 6 7 -1 salldiff var 1
4 8 9 10 11 -1 salldiff var 1
4 12 13 14 15 -1 salldiff var 1
4 0 4 8 12 -1 salldiff var 1
4 1 5 9 13 -1 salldiff var 1
4 2 6 10 14 -1 salldiff var 1
4 3 7 11 15 -1 salldiff var 1
```

4-queens binary weighted CSP example with random unary costs in wcsp format:

```
4-WQUEENS 4 4 10 5
4 4 4 4
2 0 1 0 10
0 0 5
0 1 5
1 0 5
1 1 5
1 2 5
2 1 5
2 2 5
2 3 5
```

(continues on next page)

(continued from previous page)

```
3 2 5
3 3 5
2 0 2 0 8
0 0 5
0 2 5
1 1 5
1 3 5
2 0 5
2 2 5
3 1 5
3 3 5
2 0 3 0 6
0 0 5
0 3 5
1 1 5
2 2 5
3 0 5
3 3 5
2 1 2 0 10
0 0 5
0 1 5
1 0 5
1 1 5
1 2 5
2 1 5
2 2 5
2 3 5
3 2 5
3 3 5
2 1 3 0 8
0 0 5
0 2 5
1 1 5
1 3 5
2 0 5
2 2 5
3 1 5
3 3 5
2 2 3 0 10
0 0 5
0 1 5
1 0 5
1 1 5
1 2 5
2 1 5
2 2 5
2 3 5
3 2 5
3 3 5
1 0 0 2
1 1
3 1
```

(continues on next page)

(continued from previous page)

```
1 1 0 2
1 1
2 1
1 2 0 2
1 1
2 1
1 3 0 2
0 1
2 1
```

8.2.3 UAI and LG formats (.uai, .LG)

It is a simple text file format specified below to describe probabilistic graphical model instances. The format is a generalization of the Ergo file format initially developed by Noetic Systems Inc. for their Ergo software.

- **Structure**

  A file in the UAI format consists of the following two parts, in that order:

```
<Preamble>

<Function tables>
```

  The contents of each section (denoted $< \ldots >$ above) are described in the following:

- **Preamble**

  The preamble starts with one line denoting the type of network. This will be either BAYES (if the network is a Bayesian network) or MARKOV (in case of a Markov network). This is followed by a line containing the number of variables. The next line specifies each variable's domain size, one at a time, separated by whitespace (note that this implies an order on the variables which will be used throughout the file).

  The fourth line contains only one integer, denoting the number of functions in the problem (conditional probability tables for Bayesian networks, general factors for Markov networks). Then, one function per line, the scope of each function is given as follows: The first integer in each line specifies the size of the function's scope, followed by the actual indexes of the variables in the scope. The order of this list is not restricted, except when specifying a conditional probability table (CPT) in a Bayesian network, where the child variable has to come last. Also note that variables are indexed starting with 0.

  For instance, a general function over variables 0, 5 and 11 would have this entry:

```
3 0 5 11
```

  A simple Markov network preamble with three variables and two functions might for instance look like this:

```
MARKOV
3
2 2 3
2
2 0 1
3 0 1 2
```

  The first line denotes the Markov network, the second line tells us the problem consists of three variables, let's refer to them as X, Y, and Z. Their domain size is 2, 2, and 3 respectively (from the third line). Line four specifies that there are 2 functions. The scope of the first function is X,Y, while the second function is defined over X,Y,Z.

  An example preamble for a Belief network over three variables (and therefore with three functions) might be:

```
BAYES
3
2 2 3
3
1 0
2 0 1
2 1 2
```

The first line signals a Bayesian network. This example has three variables, let's call them X, Y, and Z, with domain size 2, 2, and 3, respectively (from lines two and three). Line four says that there are 3 functions (CPTs in this case). The scope of the first function is given in line five as just X (the probability P(X)), the second one is defined over X and Y (this is (Y | X)). The third function, from line seven, is the CPT P(Z | Y). We can therefore deduce that the joint probability for this problem factors as P(X,Y,Z) = P(X).P(Y | X).P(Z | Y).

- **Function tables**

  In this section each function is specified by giving its full table (i.e, specifying the function value for each tuple). The order of the functions is identical to the one in which they were introduced in the preamble.

  For each function table, first the number of entries is given (this should be equal to the product of the domain sizes of the variables in the scope). Then, one by one, separated by whitespace, the values for each assignment to the variables in the function's scope are enumerated. Tuples are implicitly assumed in ascending order, with the last variable in the scope as the 'least significant'.

  To illustrate, we continue with our Bayesian network example from above, let's assume the following conditional probability tables:

```
X       P(X)
0       0.436
1       0.564

X       Y          P(Y | X)
0       0          0.128
0       1          0.872
1       0          0.920
1       1          0.080

Y       Z          P(Z | Y)
0       0          0.210
0       1          0.333
0       2          0.457
1       0          0.811
1       1          0.000
1       2          0.189
```

The corresponding function tables in the file would then look like this:

```
2
 0.436 0.564

4
 0.128 0.872
 0.920 0.080

6
```

(continues on next page)

(continued from previous page)

```
 0.210 0.333 0.457
 0.811 0.000 0.189
```

(Note that line breaks and empty lines are effectively just whitespace, exactly like plain spaces " ". They are used here to improve readability.)

In the LG format, probabilities are replaced by their logarithm.

- **Summary**

  To sum up, a problem file consists of 2 sections: the preamble and the full the function tables, the names and the labels.

  For our Markov network example above, the full file could be:

```
MARKOV
3
2 2 3
2
2 0 1
3 0 1 2

4
 4.000 2.400
 1.000 0.000

12
 2.2500 3.2500 3.7500
 0.0000 0.0000 10.0000
 1.8750 4.0000 3.3330
 2.0000 2.0000 3.4000
```

Here is the full Bayesian network example from above:

```
BAYES
3
2 2 3
3
1 0
2 0 1
2 1 2

2
 0.436 0.564

4
 0.128 0.872
 0.920 0.080

6
 0.210 0.333 0.457
 0.811 0.000 0.189
```

- **Expressing evidence**

Evidence is specified in a separate file. This file has the same name as the original problems file but an added .evid extension at the end. For instance, problem.uai will have evidence in problem.uai.evid.

The file simply starts with a line specifying the number of evidence variables. This is followed by the pairs of variable and value indexes for each observed variable, one pair per line. The indexes correspond to the ones implied by the original problem file.

If, for our above example, we want to specify that variable Y has been observed as having its first value and Z with its second value, the file example.uai.evid would contain the following:

```
2
 1 0
 2 1
```

8.2.4 Partial Weighted MaxSAT format

Max-SAT input format (.cnf)}

The input file format for Max-SAT will be in DIMACS format:

```
c
c comments Max-SAT
c
p cnf 3 4
1 -2 0
-1 2 -3 0
-3 2 0
1 3 0
```

- The file can start with comments, that is lines beginning with the character 'c'.
- Right after the comments, there is the line "p cnf nbvar nbclauses" indicating that the instance is in CNF format; nbvar is the number of variables appearing in the file; nbclauses is the exact number of clauses contained in the file.
- Then the clauses follow. Each clause is a sequence of distinct non-null numbers between -nbvar and nbvar ending with 0 on the same line. Positive numbers denote the corresponding variables. Negative numbers denote the negations of the corresponding variables.

Weighted Max-SAT input format (.wcnf)

In Weighted Max-SAT, the parameters line is "p wcnf nbvar nbclauses". The weights of each clause will be identified by the first integer in each clause line. The weight of each clause is an integer greater than or equal to 1.

Example of Weighted Max-SAT formula:

```
c
c comments Weighted Max-SAT
c
p wcnf 3 4
10 1 -2 0
3 -1 2 -3 0
8 -3 2 0
5 1 3 0
```

Partial Max-SAT input format (.wcnf)

In Partial Max-SAT, the parameters line is "p wcnf nbvar nbclauses top". We associate a weight with each clause, which is the first integer in the clause. Weights must be greater than or equal to 1. Hard clauses have weight top and soft clauses have weight 1. We assume that top is a weight always greater than the sum of the weights of violated soft clauses.

Example of Partial Max-SAT formula:

```
c
c comments Partial Max-SAT
c
p wcnf 4 5 15
15 1 -2 4 0
15 -1 -2 3 0
1 -2 -4 0
1 -3 2 0
1 1 3 0
```

Weighted Partial Max-SAT input format (.wcnf)

In Weighted Partial Max-SAT, the parameters line is "p wcnf nbvar nbclauses top". We associate a weight with each clause, which is the first integer in the clause. Weights must be greater than or equal to 1. Hard clauses have weight top and soft clauses have a weight smaller than top. We assume that top is a weight always greater than the sum of the weights of violated soft clauses.

Example of Weighted Partial Max-SAT formula:

```
c
c comments Weighted Partial Max-SAT
c
p wcnf 4 5 16
16 1 -2 4 0
16 -1 -2 3 0
8 -2 -4 0
4 -3 2 0
3 1 3 0
```

8.2.5 QPBO format (.qpbo)

In the quadratic pseudo-Boolean optimization (unconstrained quadratic programming) format, the goal is to minimize or maximize the quadratic function:

$X' * W * X = \sum_{i=1}^{N} \sum_{j=1}^{N} W_{ij} * X_i * X_j$

where W is a symmetric squared $N \times N$ matrix expressed by all its non-zero half ($i \leq j$) squared matrix coefficients, X is a vector of N binary variables with domain values in $\{0, 1\}$ or $\{1, -1\}$, and X' is the transposed vector of X.

Note that for two indices $i \neq j$, coefficient $W_{ij} = W_{ji}$ (symmetric matrix) and it appears twice in the previous sum. It can be controled by the option {tt -qpmult=[double]} which defines a coefficient multiplier for quadratic terms (default value is 2).

Note also that coefficients can be positive or negative and are real float numbers. They are converted to fixed-point real numbers by multiplying them by $10^{precision}$ (see option {em -precision} to modify it, default value is 7). Infinite coefficients are forbidden.

Notice that depending on the sign of the number of variables in the first text line, the domain of all variables is either $\{0, 1\}$ or $\{1, -1\}$.

Warning! The encoding in Weighted CSP of variable domain $\{1, -1\}$ associates for each variable value the following index: value 1 has index 0 and value -1 has index 1 in the solutions found by toulbar2. The encoding of variable domain $\{0, 1\}$ is direct.

Qpbo is a file text format:

- First line contains the number of variables N and the number of non-zero coefficients M.

  If N is negative then domain values are in $\{1, -1\}$, otherwise $\{0, 1\}$. If M is negative then it will maximize the quadratic function, otherwise it will minimize it.

- Followed by $|M|$ lines where each text line contains three values separated by spaces: position index i (integer belonging to $[1, |N|]$), position index j (integer belonging to $[1, |N|]$), coefficient W_{ij} (float number) such that $i \leq j$ and $W_{ij} \neq 0$.

8.2.6 OPB format (.opb)

The OPB file format is used to express pseudo-Boolean satisfaction and optimization models. These models may only contain $0/1$ Boolean variables. The format is defined by an optional objective function followed by a set of linear constraints. Variables may be multiplied together in the objective function, but currently not in the constraints due to some restriction in the reader. The objective function must start with the **min:** or **max:** keyword followed by **coef_1 varname_1_1 varname_1_2 ... coef2 varname_2_1 ...** and end with a **;**. Linear constraints are composed in the same way, ended by a comparison operator (**<=**, **>=**, or **!=**) followed by the right-hand side coefficient and **;**. Each coefficient must be an integer beginning with its sign (**+** or **-** with no extra space). Comment lines start with a *.

An example with a quadratic objective and 7 linear constraints is:

```
max: +1 x1 x2 +2 x3 x4;
+1 x2 +1 x1 >= 1;
+1 x3 +1 x1 >= 1;
+1 x4 +1 x1 >= 1;
+1 x3 +1 x2 >= 1;
+1 x4 +1 x2 >= 1;
+1 x4 +1 x3 >= 1;
+2 x1 +2 x2 +2 x3 +2 x4 <= 7;
```

Internally, all integer costs are multiplied by a power of ten depending on the -precision option. For problems with big integers, try to reduce the precision (*e.g.*, use option -precision 0).

8.2.7 WBO format (.wbo)

The WBO file format is used to express pseudo-Boolean optimization models with hard and soft constraints. It should contain no objective function. Instead a first line with the keyword **soft:** followed by a positive integer cost corresponding to a forbidden assignment and **;**. Each soft constraint starts with **[cost]** where **cost** is a positive integer representing the violation cost of the constraint. The precision is forced to be 0.

An example with 4 soft linear constraints and 2 hard linear constraints is:

```
soft: 8 ;
[2] +1 x1 >= 1 ;
[3] +1 x2 >= 1 ;
[4] +1 x3 >= 1 ;
[5] +1 x4 >= 1 ;
-1 x1 -1 x2 >= -1 ;
-1 x3 -1 x4 >= -1 ;
```

8.2.8 CPLEX format (.lp)

This textual format expresses a linear program with a linear or quadratic objective. Currently, quadratic constraints cannot be read and real variables are discretized by keeping only integer values inside their domains. Real coefficients in the objective or constraints are decimal numbers with the number of significant digits read from the file. This precision can be bounded by a command line option (-precision). When reading bound or linear constraints on a single variable, basic floating-point operations, directly reducing variable domains, are performed based on approximate operations (floor/ceil) with a bounded floating-point precision (see option -epsilon). Given the required numerical precisions, computations should be exact, and the solver should return optimal solutions without any floating-point errors. See, for example, MIPLIB instance app2-2.lp with optimum equal to 212040.500.

A complete description of the lp file format can be found at https://www.ibm.com/docs/en/icos/22.1.2?topic=cplex-lp-file-format-algebraic-representation or https://lpsolve.sourceforge.net/5.5/lp-format.htm.

Warning, unbounded variable domains cannot be represented.

Thanks to Gauthier Quesnel (INRAE) for this reader.

8.2.9 XCSP2.1 format (.xml)

CSP and weighted CSP in XML format XCSP 2.1, with constraints in extension only, can be read. See a description of this deprecated format here http://www.cril.univ-artois.fr/CPAI08/XCSP2_1.pdf.

Warning, toulbar2 must be compiled with a specific option XML in the cmake.

8.2.10 XCSP3 format (.xml)

CSP and COP format in XML format XCSP3 core can be read (still on-going work for including globals). See a description of this format here http://xcsp.org.

Warning, toulbar2 must be compiled with specific options XML and XCSP3 in the cmake.

8.2.11 Linkage format (.pre)

See **mendelsoft** companion software at http://miat.inrae.fr/MendelSoft for pedigree correction. See also https://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/HaplotypeInference for haplotype inference in half-sib families.

CHAPTER

NINE

HOW DO I USE IT ?

9.1 Using it as a C++ library

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

9.2 Using it from Python

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

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

CHAPTER

TEN

REFERENCES

See ‘BIBLIOGRAPHY’ at the end of the document.

BIBLIOGRAPHY

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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