toulbar2 User Guide

Release 1.0.0

INRAE

Apr 22, 2022

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 lest than a given upper bound often denoted as k or $\top$. Functions can be typically specified by sparse or full tables but also more concisely as specific functions called "global cost functions" [Schiex2016a].

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

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

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

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. 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.xz**, **.wcsp**, **.wcsp.gz**, **.wcsp.xz**, **.wcnf**, **.wcnf.gz**, **.wcnf.xz**, **.cnf**, **.cnf.gz**, **.cnf.xz**, **.qpbo**, **.qpbo.gz**, **.qpbo.xz**, **.opb**, **.opb.gz**, **.opb.xz**, **.uai**, **.uai.gz**, **.uai.xz**, **.LG**, **.LG.gz**, **.LG.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.
Cost function decomposition time : 1.6e-05 seconds.
Reverse DAC dual bound: 20 (+10.000%)
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)
New solution: 27 (5 backtracks, 15 nodes, depth 5)
Optimality gap: [21, 27] 22.222 % (8 backtracks, 18 nodes)
Optimality gap: [22, 27] 18.519 % (21 backtracks, 55 nodes)
Optimality gap: [23, 27] 14.815 % (49 backtracks, 122 nodes)
Optimality gap: [24, 27] 11.111 % (63 backtracks, 153 nodes)
Optimality gap: [25, 27] 7.407 % (81 backtracks, 217 nodes)
Optimality gap: [27, 27] 0.000 % (89 backtracks, 240 nodes)
Node redundancy during HBFS: 25.417 %
Optimum: 27 in 89 backtracks and 240 nodes ( 460 removals by DEE) and 0.006 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.
Cost function decomposition time : 1.6e-05 seconds.
Reverse DAC dual bound: 20 (+10.000%)
Preprocessing time: 0.001 seconds.
New solution: 27 (0 backtracks, 0 nodes, depth 1)
INCOP solving time: 0.254 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, 27] 25.926%
```

(continues on next page)

(continued from previous page)

```
Optimality gap: [21, 27] 22.222 % (4 backtracks, 8 nodes)
Optimality gap: [22, 27] 18.519 % (42 backtracks, 95 nodes)
Optimality gap: [23, 27] 14.815 % (93 backtracks, 209 nodes)
Optimality gap: [24, 27] 11.111 % (111 backtracks, 253 nodes)
Optimality gap: [25, 27] 7.407 % (121 backtracks, 280 nodes)
Optimality gap: [27, 27] 0.000 % (128 backtracks, 307 nodes)
Node redundancy during HBFS: 16.612 %
Optimum: 27 in 128 backtracks and 307 nodes ( 647 removals by DEE) and 0.263␣
↪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.
Cost function decomposition time : 1.6e-05 seconds.
Reverse DAC dual bound: 20 (+10.000%)
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)
Optimality gap: [21, 27] 22.222 % (6 backtracks, 14 nodes)
Optimality gap: [22, 27] 18.519 % (25 backtracks, 61 nodes)
Optimality gap: [23, 27] 14.815 % (56 backtracks, 133 nodes)
Optimality gap: [24, 27] 11.111 % (60 backtracks, 148 nodes)
Optimality gap: [25, 27] 7.407 % (83 backtracks, 228 nodes)
Optimality gap: [27, 27] 0.000 % (89 backtracks, 265 nodes)
Node redundancy during HBFS: 32.453 %
Optimum: 27 in 89 backtracks and 265 nodes ( 441 removals by DEE) and 0.007 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.
Cost function decomposition time : 0.000133 seconds.
Preprocessing time: 0.154752 seconds.
```

(continues on next page)

(continued from previous page)

```
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: 7771 (0 backtracks, 101 nodes, depth 3)
--- [1] LDS 1 --- (101 nodes)
c 8388608 Bytes allocated for long long stack.
New solution: 5848 (1 backtracks, 282 nodes, depth 4)
New solution: 5384 (3 backtracks, 397 nodes, depth 4)
New solution: 5039 (4 backtracks, 466 nodes, depth 4)
New solution: 4740 (8 backtracks, 640 nodes, depth 4)
--- [1] LDS 2 --- (738 nodes)
New solution: 4675 (37 backtracks, 966 nodes, depth 5)
New solution: 4633 (44 backtracks, 1113 nodes, depth 5)
New solution: 4509 (45 backtracks, 1165 nodes, depth 5)
New solution: 4502 (51 backtracks, 1226 nodes, depth 5)
New solution: 4344 (54 backtracks, 1291 nodes, depth 5)
New solution: 4258 (135 backtracks, 1864 nodes, depth 5)
New solution: 4118 (136 backtracks, 1907 nodes, depth 5)
New solution: 4107 (138 backtracks, 1965 nodes, depth 4)
New solution: 4101 (147 backtracks, 2040 nodes, depth 5)
New solution: 4099 (150 backtracks, 2057 nodes, depth 4)
New solution: 4037 (152 backtracks, 2080 nodes, depth 5)
New solution: 3853 (157 backtracks, 2171 nodes, depth 5)
New solution: 3800 (209 backtracks, 2475 nodes, depth 5)
New solution: 3781 (222 backtracks, 2539 nodes, depth 5)
New solution: 3769 (226 backtracks, 2559 nodes, depth 5)
New solution: 3750 (227 backtracks, 2568 nodes, depth 5)
New solution: 3748 (229 backtracks, 2575 nodes, depth 5)
--- [1] LDS 4 --- (2586 nodes)
New solution: 3615 (663 backtracks, 5086 nodes, depth 7)
New solution: 3614 (698 backtracks, 5269 nodes, depth 6)
New solution: 3599 (704 backtracks, 5310 nodes, depth 6)
New solution: 3594 (708 backtracks, 5335 nodes, depth 7)
New solution: 3591 (709 backtracks, 5343 nodes, depth 6)
New solution: 3580 (710 backtracks, 5354 nodes, depth 7)
New solution: 3578 (716 backtracks, 5374 nodes, depth 6)
New solution: 3551 (988 backtracks, 6456 nodes, depth 7)
New solution: 3539 (996 backtracks, 6522 nodes, depth 7)
New solution: 3516 (1000 backtracks, 6554 nodes, depth 7)
New solution: 3507 (1002 backtracks, 6573 nodes, depth 7)
New solution: 3483 (1037 backtracks, 6718 nodes, depth 7)
New solution: 3464 (1038 backtracks, 6739 nodes, depth 7)
New solution: 3438 (1047 backtracks, 6806 nodes, depth 7)
New solution: 3412 (1049 backtracks, 6824 nodes, depth 7)
--- [1] Search with no discrepancy limit --- (9443 nodes)
New solution: 3404 (4415 backtracks, 14613 nodes, depth 27)
New solution: 3402 (4416 backtracks, 14615 nodes, depth 25)
New solution: 3400 (4417 backtracks, 14619 nodes, depth 24)
New solution: 3391 (4419 backtracks, 14630 nodes, depth 28)
```

(continues on next page)

(continued from previous page)

```
New solution: 3389 (4420 backtracks, 14632 nodes, depth 26)
Optimality gap: [100, 3389] 97.049 % (21663 backtracks, 49099 nodes)
Optimality gap: [300, 3389] 91.148 % (24321 backtracks, 54415 nodes)
Optimality gap: [957, 3389] 71.762 % (37965 backtracks, 81703 nodes)
Optimality gap: [1780, 3389] 47.477 % (39060 backtracks, 83893 nodes)
Optimality gap: [1999, 3389] 41.015 % (39252 backtracks, 84277 nodes)
Optimum: 3389 in 39276 backtracks and 84325 nodes ( 444857 removals by DEE) and 36.
↪293 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 : 9.2e-05 seconds.
Preprocessing time: 0.152035 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, 109 nodes, depth 110)
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 ****** (109 nodes)
New solution: 7555 (0 backtracks, 109 nodes, depth 1)
New solution: 7545 (0 backtracks, 111 nodes, depth 2)
New solution: 7397 (0 backtracks, 114 nodes, depth 2)
New solution: 7289 (0 backtracks, 118 nodes, depth 2)
New solution: 7287 (0 backtracks, 118 nodes, depth 1)
New solution: 7277 (0 backtracks, 118 nodes, depth 1)
New solution: 5274 (0 backtracks, 118 nodes, depth 1)
New solution: 5169 (0 backtracks, 118 nodes, depth 1)
New solution: 5159 (0 backtracks, 118 nodes, depth 1)
New solution: 5158 (0 backtracks, 118 nodes, depth 1)
New solution: 5105 (1 backtracks, 120 nodes, depth 1)
New solution: 4767 (2 backtracks, 140 nodes, depth 2)
New solution: 4667 (2 backtracks, 140 nodes, depth 1)
New solution: 4655 (8 backtracks, 164 nodes, depth 2)
New solution: 4588 (8 backtracks, 171 nodes, depth 2)
New solution: 4543 (8 backtracks, 172 nodes, depth 2)
New solution: 4541 (8 backtracks, 172 nodes, depth 1)
New solution: 4424 (8 backtracks, 174 nodes, depth 2)
New solution: 4423 (8 backtracks, 174 nodes, depth 1)
New solution: 4411 (8 backtracks, 174 nodes, depth 1)
New solution: 4401 (8 backtracks, 174 nodes, depth 1)
New solution: 4367 (8 backtracks, 175 nodes, depth 2)
```

(continues on next page)

(continued from previous page)

```
New solution: 4175 (9 backtracks, 177 nodes, depth 1)
New solution: 4174 (9 backtracks, 177 nodes, depth 1)
New solution: 4173 (9 backtracks, 177 nodes, depth 1)
New solution: 4171 (9 backtracks, 177 nodes, depth 1)
New solution: 4152 (9 backtracks, 177 nodes, depth 1)
New solution: 4142 (12 backtracks, 187 nodes, depth 2)
New solution: 4001 (43 backtracks, 562 nodes, depth 2)
New solution: 3900 (43 backtracks, 562 nodes, depth 1)
New solution: 3891 (78 backtracks, 779 nodes, depth 1)
New solution: 3890 (80 backtracks, 788 nodes, depth 1)
New solution: 3816 (130 backtracks, 1192 nodes, depth 2)
New solution: 3768 (137 backtracks, 1217 nodes, depth 1)
New solution: 3740 (205 backtracks, 1660 nodes, depth 2)
New solution: 3738 (205 backtracks, 1660 nodes, depth 1)
New solution: 3730 (229 backtracks, 1780 nodes, depth 1)
New solution: 3723 (230 backtracks, 1786 nodes, depth 2)
New solution: 3721 (230 backtracks, 1786 nodes, depth 1)
New solution: 3711 (236 backtracks, 1819 nodes, depth 1)
New solution: 3633 (239 backtracks, 1850 nodes, depth 2)
New solution: 3628 (245 backtracks, 1941 nodes, depth 2)
New solution: 3621 (245 backtracks, 1943 nodes, depth 2)
New solution: 3609 (245 backtracks, 1943 nodes, depth 1)
New solution: 3608 (411 backtracks, 3079 nodes, depth 2)
New solution: 3600 (518 backtracks, 3775 nodes, depth 2)
New solution: 3598 (525 backtracks, 3806 nodes, depth 2)
New solution: 3597 (525 backtracks, 3806 nodes, depth 1)
New solution: 3587 (525 backtracks, 3806 nodes, depth 1)
New solution: 3565 (534 backtracks, 3846 nodes, depth 2)
New solution: 3554 (536 backtracks, 3856 nodes, depth 1)
New solution: 3534 (538 backtracks, 3860 nodes, depth 1)
New solution: 3522 (538 backtracks, 3861 nodes, depth 2)
New solution: 3507 (560 backtracks, 3987 nodes, depth 2)
New solution: 3505 (584 backtracks, 4130 nodes, depth 2)
New solution: 3500 (598 backtracks, 4255 nodes, depth 2)
New solution: 3498 (600 backtracks, 4281 nodes, depth 2)
New solution: 3493 (657 backtracks, 4648 nodes, depth 2)
****** Restart 2 with 2 discrepancies and UB=3493 ****** (6206 nodes)
New solution: 3492 (1406 backtracks, 9011 nodes, depth 3)
****** Restart 3 with 2 discrepancies and UB=3492 ****** (10128 nodes)
New solution: 3389 (1652 backtracks, 10572 nodes, depth 3)
****** Restart 4 with 2 discrepancies and UB=3389 ****** (11566 nodes)

Time limit expired... Aborting...
```

- 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.
```

(continues on next page)

(continued from previous page)

```
Cost function decomposition time : 0.000303 seconds.
Reverse DAC dual bound: 10001 (+0.010%)
Preprocessing time: 0.351 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: [10001, 436543501] 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: 1455221 (0 backtracks, 232 nodes, depth 233)
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=1455221 ****** (232 nodes)
New solution: 1445522 (0 backtracks, 232 nodes, depth 1)
New solution: 1445520 (0 backtracks, 232 nodes, depth 1)
New solution: 1445320 (0 backtracks, 232 nodes, depth 1)
New solution: 1445319 (0 backtracks, 232 nodes, depth 1)
New solution: 1435218 (0 backtracks, 232 nodes, depth 1)
New solution: 1425218 (0 backtracks, 232 nodes, depth 1)
New solution: 1425217 (0 backtracks, 232 nodes, depth 1)
New solution: 1415216 (0 backtracks, 232 nodes, depth 1)
New solution: 1405218 (0 backtracks, 232 nodes, depth 1)
New solution: 1405216 (9 backtracks, 286 nodes, depth 2)
New solution: 1395016 (9 backtracks, 286 nodes, depth 1)
New solution: 1394815 (9 backtracks, 289 nodes, depth 2)
New solution: 1394716 (9 backtracks, 289 nodes, depth 1)
New solution: 394818 (13 backtracks, 300 nodes, depth 1)
New solution: 394816 (13 backtracks, 300 nodes, depth 1)
New solution: 394716 (15 backtracks, 307 nodes, depth 1)
New solution: 394715 (26 backtracks, 361 nodes, depth 1)
New solution: 394713 (26 backtracks, 361 nodes, depth 1)
New solution: 384515 (30 backtracks, 379 nodes, depth 2)
New solution: 384513 (30 backtracks, 379 nodes, depth 1)
New solution: 384313 (30 backtracks, 379 nodes, depth 1)
New solution: 384213 (33 backtracks, 390 nodes, depth 1)
New solution: 384211 (33 backtracks, 390 nodes, depth 1)
New solution: 384208 (42 backtracks, 426 nodes, depth 1)
New solution: 384207 (42 backtracks, 427 nodes, depth 2)
New solution: 364206 (42 backtracks, 427 nodes, depth 1)
New solution: 353705 (42 backtracks, 438 nodes, depth 2)
New solution: 353703 (42 backtracks, 443 nodes, depth 2)
New solution: 353702 (44 backtracks, 450 nodes, depth 1)
New solution: 353701 (52 backtracks, 482 nodes, depth 1)
New solution: 343898 (88 backtracks, 705 nodes, depth 1)
New solution: 343698 (91 backtracks, 717 nodes, depth 1)
New solution: 343593 (94 backtracks, 726 nodes, depth 1)
****** Restart 2 with 2 discrepancies and UB=343593 ****** (1906 nodes)
New solution: 343592 (319 backtracks, 2203 nodes, depth 3)
****** Restart 3 with 2 discrepancies and UB=343592 ****** (3467 nodes)

Time limit expired... Aborting...
```

- 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.
Cost function decomposition time : 6.6e-05 seconds.
Reverse DAC dual bound: 64 (+35.938%)
Reverse DAC dual bound: 66 (+3.030%)
Reverse DAC dual bound: 67 (+1.493%)
Preprocessing time: 0.008 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: [67, 155] 56.774%
Tree decomposition width  : 19
Tree decomposition height : 43
Number of clusters        : 47
Tree decomposition time: 0.002 seconds.
New solution: 124 (20 backtracks, 35 nodes, depth 3)
Optimality gap: [70, 124] 43.548 % (20 backtracks, 35 nodes)
New solution: 123 (34 backtracks, 64 nodes, depth 3)
Optimality gap: [77, 123] 37.398 % (34 backtracks, 64 nodes)
New solution: 119 (173 backtracks, 348 nodes, depth 3)
Optimality gap: [88, 119] 26.050 % (173 backtracks, 348 nodes)
Optimality gap: [91, 119] 23.529 % (202 backtracks, 442 nodes)
New solution: 117 (261 backtracks, 609 nodes, depth 3)
Optimality gap: [97, 117] 17.094 % (261 backtracks, 609 nodes)
New solution: 114 (342 backtracks, 858 nodes, depth 3)
Optimality gap: [98, 114] 14.035 % (342 backtracks, 858 nodes)
Optimality gap: [100, 114] 12.281 % (373 backtracks, 984 nodes)
Optimality gap: [101, 114] 11.404 % (437 backtracks, 1123 nodes)
Optimality gap: [102, 114] 10.526 % (446 backtracks, 1152 nodes)
Optimality gap: [103, 114] 9.649 % (484 backtracks, 1232 nodes)
Optimality gap: [104, 114] 8.772 % (521 backtracks, 1334 nodes)
Optimality gap: [105, 114] 7.895 % (521 backtracks, 1353 nodes)
Optimality gap: [106, 114] 7.018 % (525 backtracks, 1364 nodes)
Optimality gap: [107, 114] 6.140 % (525 backtracks, 1379 nodes)
Optimality gap: [109, 114] 4.386 % (534 backtracks, 1539 nodes)
Optimality gap: [111, 114] 2.632 % (536 backtracks, 1559 nodes)
Optimality gap: [113, 114] 0.877 % (536 backtracks, 1564 nodes)
Optimality gap: [114, 114] 0.000 % (536 backtracks, 1598 nodes)
HBFS open list restarts: 0.000 % and reuse: 11.080 % of 352
Node redundancy during HBFS: 34.355 %
Optimum: 114 in 536 backtracks and 1598 nodes ( 21 removals by DEE) and 0.031␣
↪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.
```

(continues on next page)

(continued from previous page)

```
Cost function decomposition time : 6.6e-05 seconds.
Reverse DAC dual bound: 64 (+35.938%)
Reverse DAC dual bound: 66 (+3.030%)
Reverse DAC dual bound: 67 (+1.493%)
Preprocessing time: 0.008 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: [67, 155] 56.774%
Tree decomposition width  : 19
Tree decomposition height : 43
Number of clusters        : 47
Tree decomposition time: 0.002 seconds.
--- Solving cluster subtree 5 ...
New solution: 0 (0 backtracks, 0 nodes, depth 2)
---  done  cost = [0,0] (0 backtracks, 0 nodes, depth 2)

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

--- Solving cluster subtree 7 ...

...

--- Solving cluster subtree 44 ...
New solution: 42 (420 backtracks, 723 nodes, depth 7)
New solution: 39 (431 backtracks, 743 nodes, depth 9)
New solution: 35 (447 backtracks, 785 nodes, depth 22)
---  done  cost = [35,35] (557 backtracks, 960 nodes, depth 2)

--- Solving cluster subtree 46 ...
New solution: 114 (557 backtracks, 960 nodes, depth 2)
---  done  cost = [114,114] (557 backtracks, 960 nodes, depth 2)

Optimum: 114 in 557 backtracks and 960 nodes ( 50 removals by DEE) and 0.026␣
↪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 : 0.000911 seconds.
Preprocessing time: 0.013967 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%
Tree decomposition width  : 59
Tree decomposition height : 233
```

(continues on next page)

(continued from previous page)

```
Number of clusters        : 239
Tree decomposition time: 0.017 seconds.
--- Solving cluster subtree 0 ...
New solution: 0 (0 backtracks, 0 nodes, depth 2)
---  done  cost = [0,0] (0 backtracks, 0 nodes, depth 2)

--- Solving cluster subtree 1 ...
New solution: 0 (0 backtracks, 0 nodes, depth 2)
---  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)
New solution: 21251 (26991 backtracks, 48883 nodes, depth 4)
---  done  cost = [21251,21251] (26992 backtracks, 48883 nodes, depth 2)

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

Optimum: 21253 in 26992 backtracks and 48883 nodes ( 0 removals by DEE) and 6.180␣
↪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.
Cost function decomposition time : 0.002201 seconds.
Reverse DAC dual bound: 11120 (+81.403%)
Reverse DAC dual bound: 11128 (+0.072%)
Preprocessing time: 0.079 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: [11128, 34354] 67.608%
Tree decomposition time: 0.017 seconds.
Problem decomposition in 89 clusters with size distribution: min: 4 median: 11␣
↪mean: 11.831 max: 23
New solution: 26266 (0 backtracks, 59 nodes, depth 60)
New solution: 26265 in 0.038 seconds.
New solution: 26264 in 0.046 seconds.
New solution: 25266 in 0.047 seconds.
New solution: 25265 in 0.060 seconds.
New solution: 25260 in 0.071 seconds.
New solution: 24262 in 0.080 seconds.
New solution: 23262 in 0.090 seconds.
```

(continues on next page)

(continued from previous page)

```
New solution: 23260 in 0.098 seconds.
New solution: 23259 in 0.108 seconds.
New solution: 22262 in 0.108 seconds.
New solution: 22261 in 0.110 seconds.
New solution: 22260 in 0.113 seconds.
New solution: 22259 in 0.118 seconds.
New solution: 22258 in 0.128 seconds.
New solution: 22257 in 0.138 seconds.
New solution: 22255 in 0.154 seconds.
New solution: 22254 in 0.170 seconds.
New solution: 22252 in 0.206 seconds.
New solution: 21257 in 0.227 seconds.
New solution: 21256 in 0.256 seconds.
New solution: 21254 in 0.380 seconds.
New solution: 21253 in 0.478 seconds.
--------------------------------------------------------------------------
MPI_ABORT was invoked on rank 1 in communicator MPI_COMM_WORLD
with errorcode 0.

NOTE: invoking MPI_ABORT causes Open MPI to kill all MPI processes.
You may or may not see output from other processes, depending on
exactly when Open MPI kills them.
--------------------------------------------------------------------------

Time limit expired... Aborting...
```

- 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.
Cost function decomposition time : 1.6e-05 seconds.
Reverse DAC dual bound: 20 (+10.000%)
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 7)
Problem decomposition in 7 clusters with size distribution: min: 11 median: 15␣
↪mean: 15.143 max: 17
****** Restart 1 with 1 discrepancies and UB=28 ****** (6 nodes)
New solution: 27 (0 backtracks, 6 nodes, depth 1)
****** Restart 2 with 2 discrepancies and UB=27 ****** (57 nodes)
****** Restart 3 with 4 discrepancies and UB=27 ****** (143 nodes)
****** Restart 4 with 8 discrepancies and UB=27 ****** (418 nodes)
****** Restart 5 with 16 discrepancies and UB=27 ****** (846 nodes)
Optimum: 27 in 521 backtracks and 1156 nodes ( 3066 removals by DEE) and 0.039␣
↪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.
Cost function decomposition time : 2.7e-05 seconds.
Reverse DAC dual bound: 20 (+10.000%)
Preprocessing time: 0.002 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: 28 (0 backtracks, 7 nodes, depth 8)
New solution: 27 in 0.001 seconds.
Optimum: 27 in 0 backtracks and 7 nodes ( 36 removals by DEE) and 0.064 seconds.
Total CPU time = 0.288 seconds
Solving real-time = 0.071 seconds (not including 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.
Cost function decomposition time : 1.5e-05 seconds.
Reverse DAC dual bound: 20 (+10.000%)
Reverse DAC dual bound: 21 (+4.762%)
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: [21, 64] 67.188%
Tree decomposition width  : 8
Tree decomposition height : 16
Number of clusters        : 18
Tree decomposition time: 0.000 seconds.
New solution: 29 (19 backtracks, 30 nodes, depth 3)
New solution: 28 (37 backtracks, 62 nodes, depth 3)
Optimality gap: [22, 28] 21.429 % (37 backtracks, 62 nodes)
Optimality gap: [23, 28] 17.857 % (309 backtracks, 629 nodes)
New solution: 27 (328 backtracks, 672 nodes, depth 3)
Optimality gap: [23, 27] 14.815 % (328 backtracks, 672 nodes)
Optimality gap: [24, 27] 11.111 % (347 backtracks, 724 nodes)
Optimality gap: [25, 27] 7.407 % (372 backtracks, 819 nodes)
Optimality gap: [26, 27] 3.704 % (372 backtracks, 829 nodes)
Optimality gap: [27, 27] 0.000 % (372 backtracks, 873 nodes)
HBFS open list restarts: 0.000 % and reuse: 10.769 % of 65
Node redundancy during HBFS: 16.724 %
Optimum: 27 in 372 backtracks and 873 nodes ( 463 removals by DEE) and 0.020␣
↪seconds.
```

(continues on next page)

(continued from previous page)

```
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.
Cost function decomposition time : 1.6e-05 seconds.
Reverse DAC dual bound: 20 (+15.000%)
Reverse DAC dual bound: 22 (+9.091%)
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: [22, 64] 65.625%
Tree decomposition width  : 16
Tree decomposition height : 24
Number of clusters         : 9
Tree decomposition time: 0.000 seconds.
New solution: 29 (23 backtracks, 29 nodes, depth 3)
New solution: 28 (32 backtracks, 46 nodes, depth 3)
Optimality gap: [23, 28] 17.857 % (37 backtracks, 58 nodes)
New solution: 27 (61 backtracks, 122 nodes, depth 3)
Optimality gap: [23, 27] 14.815 % (61 backtracks, 122 nodes)
Optimality gap: [24, 27] 11.111 % (132 backtracks, 269 nodes)
Optimality gap: [25, 27] 7.407 % (177 backtracks, 395 nodes)
Optimality gap: [26, 27] 3.704 % (189 backtracks, 467 nodes)
Optimality gap: [27, 27] 0.000 % (189 backtracks, 482 nodes)
HBFS open list restarts: 0.000 % and reuse: 25.926 % of 27
Node redundancy during HBFS: 25.519 %
Optimum: 27 in 189 backtracks and 482 nodes ( 95 removals by DEE) and 0.010 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.xz.evid
No evidence file.
Generic variable elimination of degree 4
Maximum degree of generic variable elimination: 4
Cost function decomposition time : 0.003733 seconds.
Preprocessing time: 0.073664 seconds.
```

(continues on next page)

(continued from previous page)

```
232 unassigned variables, 517 values in all current domains (med. size:2, max␣
↪size:4) and 415 non-unary cost functions (med. arity:2, med. degree:6)
Initial lower and upper bounds: [553902779, 13246577453] 95.819%
Tree decomposition width  : 227
Tree decomposition height : 230
Number of clusters        : 890
Tree decomposition time: 0.047 seconds.
New solution: 865165767 energy: 298.395 prob: 2.564e-130 (72 backtracks, 140 nodes,␣
↪depth 3)
New solution: 844685630 energy: 296.347 prob: 1.987e-129 (128 backtracks, 254 nodes,
↪ depth 3)
New solution: 822713386 energy: 294.149 prob: 1.789e-128 (188 backtracks, 373 nodes,
↪ depth 3)
New solution: 809800912 energy: 292.858 prob: 6.506e-128 (327 backtracks, 665 nodes,
↪ depth 3)
New solution: 769281277 energy: 288.806 prob: 3.742e-126 (383 backtracks, 771 nodes,
↪ depth 3)
New solution: 755317979 energy: 287.410 prob: 1.512e-125 (714 backtracks, 1549␣
↪nodes, depth 3)
New solution: 755129381 energy: 287.391 prob: 1.540e-125 (927 backtracks, 2038␣
↪nodes, depth 3)
New solution: 711184893 energy: 282.997 prob: 1.248e-123 (1249 backtracks, 2685␣
↪nodes, depth 3)
HBFS open list restarts: 0.000 % and reuse: 39.620 % of 1474
Node redundancy during HBFS: 22.653 %
Optimum: 711184893 energy: 282.997 prob: 1.248e-123 in 21719 backtracks and 56124␣
↪nodes ( 72435 removals by DEE) and 4.310 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.xz.evid
No evidence file.
Cost function decomposition time : 0.001227 seconds.
Reverse DAC dual bound: 5879065363 energy: 1074.088 (+0.082%)
VAC dual bound: 5906374927 energy: 1076.819 (iter:486)
Number of VAC iterations: 726
Number of times is VAC: 240 Number of times isvac and itThreshold > 1: 234
Preprocessing time: 1.872 seconds.
729 unassigned variables, 4819 values in all current domains (med. size:7, max␣
↪size:7) and 3128 non-unary cost functions (med. arity:2, med. degree:6)
Initial lower and upper bounds: [5906374927, 111615200815] 94.708%
c 2097152 Bytes allocated for long long stack.
New solution: 5968997522 energy: 1083.081 prob: 4.204e-471 (0 backtracks, 19 nodes,␣
↪depth 21)
Optimality gap: [5920086558, 5968997522] 0.819 % (17 backtracks, 36 nodes)
```

(continues on next page)

(continued from previous page)

```
New solution: 5922481881 energy: 1078.430 prob: 4.404e-469 (17 backtracks, 48 nodes,
↪ depth 8)
Optimality gap: [5922481881, 5922481881] 0.000 % (21 backtracks, 52 nodes)
Number of VAC iterations: 846
Number of times is VAC: 360 Number of times isvac and itThreshold > 1: 351
Node redundancy during HBFS: 11.538 %
Optimum: 5922481881 energy: 1078.430 prob: 4.404e-469 in 21 backtracks and 52 nodes␣
↪( 2749 removals by DEE) and 1.980 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.xz.evid
No evidence file.
Cost function decomposition time : 0.000679 seconds.
Reverse DAC dual bound: 103988236701 energy: -12486.138 (+0.000%)
VAC dual bound: 103988236701 energy: -12486.138 (iter:4068)
Number of VAC iterations: 4389
Number of times is VAC: 189 Number of times isvac and itThreshold > 1: 186
Threshold: 2326139858 NbAssignedVar: 0 Ratio: 0.0000000
Threshold: 2320178814 NbAssignedVar: 0 Ratio: 0.0000000
Threshold: 21288438 NbAssignedVar: 19 Ratio: 0.0000000
Threshold: 11823689 NbAssignedVar: 20 Ratio: 0.0000000
Threshold: 8187968 NbAssignedVar: 21 Ratio: 0.0000001
Threshold: 6858739 NbAssignedVar: 22 Ratio: 0.0000001
Threshold: 6058812 NbAssignedVar: 22 Ratio: 0.0000001
Threshold: 5504560 NbAssignedVar: 22 Ratio: 0.0000001
Threshold: 3972336 NbAssignedVar: 23 Ratio: 0.0000002
Threshold: 3655432 NbAssignedVar: 23 Ratio: 0.0000002
Threshold: 3067825 NbAssignedVar: 23 Ratio: 0.0000002
Threshold: 2174446 NbAssignedVar: 24 Ratio: 0.0000003
Threshold: 1641827 NbAssignedVar: 24 Ratio: 0.0000004
Threshold: 1376213 NbAssignedVar: 24 Ratio: 0.0000005
Threshold: 208082 NbAssignedVar: 24 Ratio: 0.0000031
Threshold: 104041 NbAssignedVar: 26 Ratio: 0.0000068
Threshold: 52020 NbAssignedVar: 27 Ratio: 0.0000140
Threshold: 26010 NbAssignedVar: 27 Ratio: 0.0000281
Threshold: 13005 NbAssignedVar: 27 Ratio: 0.0000561
Threshold: 6502 NbAssignedVar: 27 Ratio: 0.0001122
Threshold: 3251 NbAssignedVar: 27 Ratio: 0.0002245
Threshold: 1625 NbAssignedVar: 27 Ratio: 0.0004491
Threshold: 812 NbAssignedVar: 27 Ratio: 0.0008987
Threshold: 406 NbAssignedVar: 27 Ratio: 0.0017974
Threshold: 203 NbAssignedVar: 27 Ratio: 0.0035947
Threshold: 101 NbAssignedVar: 27 Ratio: 0.0072250
Threshold: 50 NbAssignedVar: 27 Ratio: 0.0145946
Threshold: 25 NbAssignedVar: 27 Ratio: 0.0291892
```

(continues on next page)

(continued from previous page)

```
Threshold: 12 NbAssignedVar: 27 Ratio: 0.0608108
Threshold: 6 NbAssignedVar: 27 Ratio: 0.1216216
Threshold: 3 NbAssignedVar: 27 Ratio: 0.2432432
Threshold: 1 NbAssignedVar: 27 Ratio: 0.7297297
RASPS/VAC threshold: 203
Preprocessing time: 41.340 seconds.
37 unassigned variables, 3366 values in all current domains (med. size:38, max␣
↪size:331) and 626 non-unary cost functions (med. arity:2, med. degree:35)
Initial lower and upper bounds: [103988236701, 239074057808] 56.504%
New solution: 104206588216 energy: -12464.303 prob: inf (0 backtracks, 3 nodes,␣
↪depth 6)
RASPS done in preprocessing (backtrack: 4 nodes: 8)
New solution: 104174014744 energy: -12467.560 prob: inf (4 backtracks, 12 nodes,␣
↪depth 6)
Optimality gap: [104174014744, 104174014744] 0.000 % (7 backtracks, 15 nodes)
Number of VAC iterations: 4695
Number of times is VAC: 458 Number of times isvac and itThreshold > 1: 451
Node redundancy during HBFS: 0.000 %
Optimum: 104174014744 energy: -12467.560 prob: inf in 7 backtracks and 15 nodes (␣
↪937 removals by DEE) and 41.354 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.
Cost function decomposition time : 0.000485 seconds.
Reverse DAC dual bound: 91 (+86.813%)
Reverse DAC dual bound: 92 (+1.087%)
Preprocessing time: 0.040 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)
New solution: 188 (45 backtracks, 143 nodes, depth 37)
New solution: 187 (155 backtracks, 473 nodes, depth 47)
New solution: 186 (892 backtracks, 2247 nodes, depth 19)
New solution: 185 (3874 backtracks, 8393 nodes, depth 70)
New solution: 184 (29475 backtracks, 62393 nodes, depth 40)
New solution: 183 (221446 backtracks, 522724 nodes, depth 11)
Node redundancy during HBFS: 37.221 %
Optimum: 183 in 281307 backtracks and 896184 nodes ( 9478 removals by DEE) and 25.
↪977 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.
Cost function decomposition time : 2e-06 seconds.
Reverse DAC dual bound: 48 (+2.083%)
Preprocessing time: 0.006 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)
Optimality gap: [58, 1000] 94.200 % (355 backtracks, 812 nodes)
Optimality gap: [72, 1000] 92.800 % (575 backtracks, 1309 nodes)
Optimality gap: [1000, 1000] 0.000 % (575 backtracks, 1318 nodes)
Number of solutions    : =  576
Time                   :    0.306 seconds
... 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.
Cost function decomposition time : 3e-06 seconds.
Reverse DAC dual bound: 48 (+2.083%)
Preprocessing time: 0.009 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)
New solution: 48 (2 backtracks, 11 nodes, depth 3)
Node redundancy during HBFS: 18.182 %
Optimum: 48 in 2 backtracks and 11 nodes ( 0 removals by DEE) and 0.017 seconds.
+++++++++ Search for solution 2 +++++++++
New solution: 52 (2 backtracks, 879 nodes, depth 871)
Optimality gap: [50, 49] -2.000 % (5 backtracks, 882 nodes)
New solution: 51 (5 backtracks, 1748 nodes, depth 868)
Optimality gap: [51, 49] -3.922 % (6 backtracks, 1749 nodes)
Node redundancy during HBFS: 0.172 %
Optimum: 51 in 6 backtracks and 1749 nodes ( 0 removals by DEE) and 0.046 seconds.
+++++++++ Search for solution 3 +++++++++
New solution: 74 (6 backtracks, 2569 nodes, depth 823)
New solution: 62 (14 backtracks, 3407 nodes, depth 824)
New solution: 58 (21 backtracks, 4245 nodes, depth 821)
Optimality gap: [53, 49] -7.547 % (29 backtracks, 4270 nodes)
Optimality gap: [56, 49] -12.500 % (30 backtracks, 4276 nodes)
Optimality gap: [57, 49] -14.035 % (31 backtracks, 4292 nodes)
New solution: 57 (31 backtracks, 5114 nodes, depth 819)
Node redundancy during HBFS: 1.017 %
Optimum: 57 in 31 backtracks and 5114 nodes ( 0 removals by DEE) and 0.146 seconds.
+++++++++ Search for solution 4 +++++++++
```

(continues on next page)

(continued from previous page)

```
New solution: 73 (44 backtracks, 5923 nodes, depth 773)
New solution: 72 (46 backtracks, 6702 nodes, depth 778)
New solution: 58 (53 backtracks, 7485 nodes, depth 773)
Optimality gap: [58, 49] -15.517 % (70 backtracks, 7584 nodes)
Node redundancy during HBFS: 1.846 %
Optimum: 58 in 70 backtracks and 7584 nodes ( 0 removals by DEE) and 0.256 seconds.
+++++++++ Search for solution 5 +++++++++
New solution: 80 (70 backtracks, 8307 nodes, depth 726)
New solution: 74 (100 backtracks, 9139 nodes, depth 728)
New solution: 66 (112 backtracks, 9896 nodes, depth 724)
New solution: 64 (116 backtracks, 10636 nodes, depth 725)
New solution: 58 (171 backtracks, 11654 nodes, depth 725)
Node redundancy during HBFS: 3.484 %
Optimum: 58 in 171 backtracks and 11654 nodes ( 0 removals by DEE) and 0.474␣
↪seconds.
+++++++++ Search for solution 6 +++++++++
New solution: 79 (178 backtracks, 12347 nodes, depth 677)
New solution: 76 (207 backtracks, 13102 nodes, depth 677)
New solution: 65 (212 backtracks, 13804 nodes, depth 680)
Optimality gap: [59, 49] -16.949 % (251 backtracks, 14053 nodes)
Optimality gap: [60, 49] -18.333 % (256 backtracks, 14093 nodes)
Optimality gap: [61, 49] -19.672 % (259 backtracks, 14126 nodes)
Optimality gap: [62, 49] -20.968 % (260 backtracks, 14165 nodes)
New solution: 62 (260 backtracks, 14849 nodes, depth 675)
Node redundancy during HBFS: 4.936 %
Optimum: 62 in 260 backtracks and 14849 nodes ( 0 removals by DEE) and 0.688␣
↪seconds.
+++++++++ Search for solution 7 +++++++++
c 2097152 Bytes allocated for long long stack.
New solution: 77 (267 backtracks, 15495 nodes, depth 630)
New solution: 76 (283 backtracks, 16160 nodes, depth 629)
New solution: 75 (334 backtracks, 16982 nodes, depth 628)
New solution: 68 (335 backtracks, 17615 nodes, depth 628)
Optimality gap: [64, 49] -23.438 % (383 backtracks, 17946 nodes)
New solution: 65 (383 backtracks, 18577 nodes, depth 627)
Optimality gap: [65, 49] -24.615 % (383 backtracks, 18581 nodes)
Node redundancy during HBFS: 5.915 %
Optimum: 65 in 383 backtracks and 18581 nodes ( 0 removals by DEE) and 0.963␣
↪seconds.
+++++++++ Search for solution 8 +++++++++
New solution: 81 (383 backtracks, 19161 nodes, depth 583)
New solution: 80 (425 backtracks, 19865 nodes, depth 583)
New solution: 69 (471 backtracks, 20646 nodes, depth 585)
New solution: 68 (479 backtracks, 21273 nodes, depth 581)
New solution: 65 (483 backtracks, 21881 nodes, depth 580)
Node redundancy during HBFS: 6.014 %
Optimum: 65 in 483 backtracks and 21881 nodes ( 0 removals by DEE) and 1.175␣
↪seconds.
+++++++++ Search for solution 9 +++++++++
New solution: 68 (483 backtracks, 22413 nodes, depth 535)
Optimality gap: [66, 49] -25.758 % (581 backtracks, 22902 nodes)
New solution: 66 (581 backtracks, 23434 nodes, depth 531)
```

(continues on next page)

(continued from previous page)

```
Node redundancy during HBFS: 6.900 %
Optimum: 66 in 581 backtracks and 23434 nodes ( 0 removals by DEE) and 1.379␣
↪seconds.
+++++++++ Search for solution 10 +++++++++
New solution: 68 (619 backtracks, 24035 nodes, depth 484)
Optimality gap: [67, 49] -26.866 % (686 backtracks, 24436 nodes)
Optimality gap: [68, 49] -27.941 % (686 backtracks, 24444 nodes)
Node redundancy during HBFS: 7.924 %
Optimum: 68 in 686 backtracks and 24444 nodes ( 0 removals by DEE) and 1.597␣
↪seconds.
+++++++++ Search for solution 11 +++++++++
New solution: 72 (714 backtracks, 24958 nodes, depth 436)
New solution: 68 (739 backtracks, 25534 nodes, depth 436)
Node redundancy during HBFS: 8.052 %
Optimum: 68 in 739 backtracks and 25534 nodes ( 0 removals by DEE) and 1.712␣
↪seconds.
+++++++++ Search for solution 12 +++++++++
c 4194304 Bytes allocated for long long stack.
New solution: 81 (770 backtracks, 26006 nodes, depth 389)
New solution: 78 (772 backtracks, 26399 nodes, depth 389)
New solution: 77 (779 backtracks, 26818 nodes, depth 389)
New solution: 76 (809 backtracks, 27354 nodes, depth 390)
New solution: 72 (858 backtracks, 28065 nodes, depth 389)
Optimality gap: [69, 49] -28.986 % (863 backtracks, 28122 nodes)
Optimality gap: [70, 49] -30.000 % (864 backtracks, 28130 nodes)
Optimality gap: [71, 49] -30.986 % (864 backtracks, 28140 nodes)
New solution: 71 (864 backtracks, 28532 nodes, depth 387)
Node redundancy during HBFS: 8.762 %
Optimum: 71 in 864 backtracks and 28532 nodes ( 0 removals by DEE) and 1.981␣
↪seconds.
+++++++++ Search for solution 13 +++++++++
New solution: 76 (898 backtracks, 28974 nodes, depth 343)
New solution: 72 (906 backtracks, 29334 nodes, depth 340)
Optimality gap: [72, 49] -31.944 % (979 backtracks, 29782 nodes)
Node redundancy during HBFS: 9.563 %
Optimum: 72 in 979 backtracks and 29782 nodes ( 0 removals by DEE) and 2.212␣
↪seconds.
+++++++++ Search for solution 14 +++++++++
New solution: 86 (1062 backtracks, 30429 nodes, depth 292)
New solution: 80 (1078 backtracks, 30768 nodes, depth 292)
New solution: 74 (1085 backtracks, 31080 nodes, depth 292)
Optimality gap: [74, 49] -33.784 % (1102 backtracks, 31203 nodes)
Node redundancy during HBFS: 10.124 %
Optimum: 74 in 1102 backtracks and 31203 nodes ( 0 removals by DEE) and 2.441␣
↪seconds.
+++++++++ Search for solution 15 +++++++++
New solution: 79 (1103 backtracks, 31448 nodes, depth 246)
New solution: 78 (1122 backtracks, 31726 nodes, depth 246)
New solution: 76 (1183 backtracks, 32087 nodes, depth 245)
Optimality gap: [76, 49] -35.526 % (1231 backtracks, 32181 nodes)
Node redundancy during HBFS: 9.816 %
Optimum: 76 in 1231 backtracks and 32181 nodes ( 0 removals by DEE) and 2.603␣
↪seconds.
```

(continues on next page)

(continued from previous page)

```
+++++++++ Search for solution 16 +++++++++
New solution: 80 (1253 backtracks, 32419 nodes, depth 197)
New solution: 79 (1315 backtracks, 32735 nodes, depth 197)
New solution: 78 (1336 backtracks, 32968 nodes, depth 196)
Optimality gap: [78, 49] -37.179 % (1349 backtracks, 32993 nodes)
Node redundancy during HBFS: 9.575 %
Optimum: 78 in 1349 backtracks and 32993 nodes ( 0 removals by DEE) and 2.760␣
↪seconds.
+++++++++ Search for solution 17 +++++++++
New solution: 80 (1349 backtracks, 33141 nodes, depth 151)
New solution: 79 (1374 backtracks, 33334 nodes, depth 149)
Optimality gap: [79, 49] -37.975 % (1474 backtracks, 33532 nodes)
Node redundancy during HBFS: 9.421 %
Optimum: 79 in 1474 backtracks and 33532 nodes ( 0 removals by DEE) and 2.924␣
↪seconds.
+++++++++ Search for solution 18 +++++++++
New solution: 80 (1546 backtracks, 33775 nodes, depth 102)
Optimality gap: [80, 49] -38.750 % (1592 backtracks, 33864 nodes)
Node redundancy during HBFS: 9.328 %
Optimum: 80 in 1592 backtracks and 33864 nodes ( 0 removals by DEE) and 3.085␣
↪seconds.
+++++++++ Search for solution 19 +++++++++
New solution: 80 (1687 backtracks, 34105 nodes, depth 54)
Node redundancy during HBFS: 9.263 %
Optimum: 80 in 1687 backtracks and 34105 nodes ( 0 removals by DEE) and 3.219␣
↪seconds.
+++++++++ Search for solution 20 +++++++++
Optimality gap: [1000, 49] -95.100 % (1809 backtracks, 34349 nodes)
Node redundancy during HBFS: 9.197 %
No solution in 1809 backtracks and 34349 nodes ( 0 removals by DEE) and 3.377␣
↪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 : 1.1e-05 seconds.
Preprocessing time: 0.001019 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%
Tree decomposition width  : 5
Tree decomposition height : 20
Number of clusters        : 29
Tree decomposition time: 0.000 seconds.
Number of solutions    : =  411110802705928379432960
Number of #goods       :    3993
```

(continues on next page)

(continued from previous page)

```
Number of used #goods  :    17190
Size of sep            :    4
Time                   :    0.055 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: 0.000997 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.

Tree decomposition width  : 5
Tree decomposition height : 17
Number of clusters        : 33
Tree decomposition time: 0.001 seconds.

Cartesian product                    :    13367494538843734031554962259968
Upper bound of number of solutions : <= 171992678400000000000000000
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

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.

6.1 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)

-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)

6.2 Preprocessing

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

-p=[integer] preprocessing only: general variable elimination of degree less than or equal to the given value (default value is -1)

-t=[integer] preprocessing only: simulates restricted path consistency by adding ternary cost functions on triangles of binary cost functions within a given maximum space limit (in MB)

-f=[integer] preprocessing only: variable elimination of functional (f=1) (resp. bijective (f=2)) variables (default value is 1)

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

-n=[integer] preprocessing only: projects n-ary cost functions on all binary cost functions if n is lower than the given value (default value is 10). See [Favier2011a].

-mst find a maximum spanning tree ordering for DAC

-S preprocessing only: performs singleton consistency (only in conjunction with option -A)

-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)

6.3 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*.

-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

6.4 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 10000)

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

-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

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

-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)

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

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

6.5 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".

-vnsini=[integer] initial solution for VNS-like methods found: (-1) at random, (-2) min domain values, (-3) max domain values, (-4) first solution found by a complete method, (k=0 or more) tree search with k discrepancy max (-4 by default)

-ldsmin=[integer] minimum discrepancy for VNS-like methods (1 by default)

-ldsmax=[integer] maximum discrepancy for VNS-like methods (number of problem variables multiplied by maximum domain size -1 by default)

-ldsinc=[integer] discrepancy increment strategy for VNS-like methods using (1) Add1, (2) Mult2, (3) Luby operator (2 by default)

-kmin=[integer] minimum neighborhood size for VNS-like methods (4 by default)

-kmax=[integer] maximum neighborhood size for VNS-like methods (number of problem variables by default)

-kinc=[integer] neighborhood size increment strategy for VNS-like methods using: (1) Add1, (2) Mult2, (3) Luby operator (4) Add1/Jump (4 by default)

-best=[integer] stop VNS-like methods if a better solution is found (default value is 0)

6.6 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 (default value is 1)

-P=[decimal] threshold cost value for VAC during the preprocessing phase only (default value is 1)

-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)

-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)

6.7 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).

-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) 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)

6.8 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 (default value is 0)

-mdd=[integer] maximum relaxed MDD width for diverse solution global constraint (default value is 0)

-mddh=[integer] MDD relaxation heuristic: 0: random, 1: high div, 2: small div, 3: high unary costs (default value is 0)

6.9 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.

6.10 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

-z=[integer] 1 or 3: saves original instance in 1-wcsp or 3-cfn format (1 by default), 2 or 4: saves after preprocessing in 2-wcsp or 4-cfn format (this option can be used in combination with -z=filename)

-x=[(,i[$= \# <>$]a)*] performs an elementary operation ('$=$':assign, '$\#$':remove, '$<$':decrease, '$>$':increase) with value a on variable of index i (multiple operations are separated by a comma and no space) (without any argument, a complete assignment – used as initial upper bound and as value heuristic – read from default file "sol" or given as input filename with ".sol" extension)

6.11 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)

-epsilon=[float] approximation factor for computing the partition function (greater than 1, default value is infinity)

6.12 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

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

SEVEN

INPUT FORMATS

7.1 Introduction

The available **file formats** (possibly compressed by gzip or xz, e.g., .cfn.gz, .wcsp.xz) 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)
- Weigthed Partial Max-SAT (.cnf/.wcnf file extension)
- Quadratic Unconstrained Pseudo-Boolean Optimization (*.qpbo* file extension)
- Pseudo-Boolean Optimization (.opb 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).

7.2 Formats details

7.2.1 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 separates 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 orders 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.

An 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. 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
          nb_symbols : 4
          nb_values : 2
          start : 0
```

(continues on next page)

(continued from previous page)

```
            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
          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: 2.01comparator: >
          to: 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
```

(continues on next page)

(continued from previous page)

```
                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`.

7.2.2 Weighted Constraint Satisfaction Problem file format (wcsp)

group **`wcspformat`**

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)
  - 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
- 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
```

- 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
```

- 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
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
```

(continues on next page)

(continued from previous page)

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

7.2.3 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 < ... > 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
 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
```

7.2.4 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
```

(continues on next page)

(continued from previous page)

```
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
```

7.2.5 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 $W_{ij} \neq 0$.

7.2.6 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).

7.2.7 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.

7.2.8 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 EIGHT

HOW DO I USE IT ?

8.1 Using it as a C++ library

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

8.2 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

NINE

REFERENCES

See ‘BIBLIOGRAPHY’ at the end of the document.

BIBLIOGRAPHY

[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. I, *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.