TOULBAR2 User documentation

The TOULBAR2 developer team

November 22, 2019

1 What is toulbar2

TOULBAR2 is an exact black box discrete optimization solver targeted at solving cost function networks (CFN), thus solving the so-called "weighted Constraint Satisfaction Problem" or WCSP. Cost function networks can be simply described by a set of discrete variables each having a specific finite domain and a set of integer cost functions, each involving some of the variables. The WCSP is to find an assignment of all variables such that the sum of all cost functions is minimum and lest than a given upper bound often denoted as k or $\top$. Functions can be typically specified by sparse or full tables but also more concisely as specific functions called "global cost functions" [3].

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 [16], and Maximum A Posteriori (MAP) on Markov random field [16]. 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 [2] 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 [25]. 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 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.

2 How do I install it ?

TOULBAR2 is an open source solver distributed under the MIT license as a set of C++ sources managed with git at `http://github.com/toulbar2/toulbar2`. If you want to use a released version, then you can download there source archives of a specific release that should be easy to compile on most Linux systems.

If you want to compile the latest sources yourself, you will need a modern C++ compiler, CMake, Gnu MP Bignum library, a recent version of boost libraries and optionally the jemalloc memory management and OpenMPI libraries. 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 [15].

3 How do I test it ?

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

```
make test
```

For debugging TOULBAR2 (compile with flag `CMAKE_BUILD_TYPE="Debug"`), more test examples are available at Cost Function Library. The following commands run TOULBAR2 (executable must be found on your system path) on every

problems with a 1-hour time limit and compare their optimum with known optima (in .ub files).

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

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

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

4 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`, `.uai`, `.uai.gz`, `.uai.xz`, `.LG`, `.LG.gz`, `.LG.xz`, `.pre` or `.bep`) is used to determine the nature of the file (see section 7). 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.

5 Quick start

1. Download a binary weighted constraint satisfaction problem (WCSP) file *example.wcsp* from the toulbar2's Documentation Web page. Solve it with default options:

```
toulbar2 EXAMPLES/example.wcsp


Read 25 variables, with 5 values at most, and 63 cost functions, with maximum arity 2.
Cost function decomposition time : 7e-06 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. degree:5)
```

```
Initial lower and upper bounds: [20, 64] 68.750%
New solution: 28 (0 backtracks, 6 nodes, depth 7)
New solution: 27 (5 backtracks, 15 nodes, depth 4)
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 % (51 backtracks, 126 nodes)
Optimality gap: [24, 27] 11.111 % (71 backtracks, 169 nodes)
Optimality gap: [25, 27] 7.407 % (90 backtracks, 225 nodes)
Optimality gap: [27, 27] 0.000 % (103 backtracks, 258 nodes)
Node redundancy during HBFS: 19.767 %
Optimum: 27 in 103 backtracks and 258 nodes ( 271 removals by DEE) and 0.007 seconds.
end.
```

2. Solve a WCSP using INCOP, a local search method [24] applied just after preprocessing, in order to find a good upper bound before a complete search:

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

```
Read 25 variables, with 5 values at most, and 63 cost functions, with maximum arity 2.
Cost function decomposition time : 1.2e-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: 0.242 seconds.
24 unassigned variables, 116 values in all current domains (med. size:5, max size:5) and 62 non-unary cost functions (med. degree:5)
Initial lower and upper bounds: [20, 27] 25.926%
Optimality gap: [21, 27] 22.222 % (4 backtracks, 8 nodes)
Optimality gap: [22, 27] 18.519 % (17 backtracks, 40 nodes)
Optimality gap: [23, 27] 14.815 % (109 backtracks, 238 nodes)
Optimality gap: [24, 27] 11.111 % (112 backtracks, 249 nodes)
Optimality gap: [27, 27] 0.000 % (119 backtracks, 270 nodes)
Node redundancy during HBFS: 11.852 %
Optimum: 27 in 119 backtracks and 270 nodes ( 264 removals by DEE) and 0.250 seconds.
end.
```

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

```
toulbar2 EXAMPLES/example.wcsp -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 : 8e-06 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. degree:5)
Initial lower and upper bounds: [20, 28] 28.571%
New solution: 27 (0 backtracks, 4 nodes, depth 5)
Optimality gap: [21, 27] 22.222 % (6 backtracks, 14 nodes)
Optimality gap: [22, 27] 18.519 % (22 backtracks, 55 nodes)
Optimality gap: [23, 27] 14.815 % (54 backtracks, 129 nodes)
Optimality gap: [24, 27] 11.111 % (58 backtracks, 144 nodes)
Optimality gap: [25, 27] 7.407 % (74 backtracks, 210 nodes)
Optimality gap: [27, 27] 0.000 % (83 backtracks, 243 nodes)
Node redundancy during HBFS: 31.276 %
Optimum: 27 in 83 backtracks and 243 nodes ( 289 removals by DEE) and 0.006 seconds.
end.
```

```
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
```

4. Download a larger WCSP file *scen06.wcsp* from the toulbar2's Documentation Web page. Solve it using a limited discrepancy search strategy [14] in order to speed-up the search for finding good upper bounds first[1]:

[1]By default, toulbar2 uses another diversification strategy based on hybrid best-first search [2].

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

```
Read 100 variables, with 44 values at most, and 1222 cost functions, with maximum arity 2.
Cost function decomposition time : 5.6e-05 seconds.
Preprocessing time: 0.171626 seconds.
82 unassigned variables, 3273 values in all current domains (med. size:44, max size:44) and 327 non-unary cost functions (med. degree:6)
Initial lower and upper bounds: [0, 248338] 100.000%
--- [0] LDS 0 --- (0 nodes)
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: 8451 (0 backtracks, 110 nodes, depth 1)
--- [0] LDS 1 --- (110 nodes)
New solution: 6134 (2 backtracks, 386 nodes, depth 2)
New solution: 5795 (4 backtracks, 527 nodes, depth 2)
New solution: 5711 (5 backtracks, 590 nodes, depth 2)
New solution: 5444 (6 backtracks, 676 nodes, depth 2)
New solution: 4828 (7 backtracks, 747 nodes, depth 2)
New solution: 4507 (9 backtracks, 853 nodes, depth 2)
New solution: 4408 (10 backtracks, 910 nodes, depth 2)
New solution: 4145 (15 backtracks, 1047 nodes, depth 2)
New solution: 3730 (18 backtracks, 1112 nodes, depth 2)
--- [0] LDS 2 --- (1132 nodes)
New solution: 3635 (64 backtracks, 1606 nodes, depth 3)
New solution: 3585 (150 backtracks, 2169 nodes, depth 3)
New solution: 3493 (168 backtracks, 2283 nodes, depth 3)
New solution: 3472 (171 backtracks, 2312 nodes, depth 3)
--- [0] LDS 4 --- (2420 nodes)
New solution: 3463 (988 backtracks, 5683 nodes, depth 5)
New solution: 3441 (990 backtracks, 5711 nodes, depth 5)
New solution: 3401 (1101 backtracks, 6178 nodes, depth 5)
--- [0] Search with no discrepancy limit --- (8111 nodes)
Optimality gap: [226, 3401] 93.355 % (44869 backtracks, 94455 nodes)
New solution: 3400 (66078 backtracks, 136900 nodes, depth 27)
New solution: 3399 (66079 backtracks, 136902 nodes, depth 26)
New solution: 3389 (66091 backtracks, 136933 nodes, depth 30)
Optimality gap: [813, 3389] 76.011 % (93403 backtracks, 191535 nodes)
Optimality gap: [1088, 3389] 67.896 % (94456 backtracks, 193641 nodes)
Optimality gap: [1862, 3389] 45.058 % (96296 backtracks, 197321 nodes)
Optimality gap: [2036, 3389] 39.923 % (96395 backtracks, 197519 nodes)
Optimality gap: [2754, 3389] 18.737 % (96442 backtracks, 197613 nodes)
Optimum: 3389 in 96443 backtracks and 197615 nodes ( 384523 removals by DEE) and 53.398 seconds.
end.
```

5. 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) [25] during 5 seconds:

```
toulbar2 EXAMPLES/scen06.wcsp EXAMPLES/scen06.dec -vns -time=5
```

```
Read 100 variables, with 44 values at most, and 1222 cost functions, with maximum arity 2.
Cost function decomposition time : 7.8e-05 seconds.
Preprocessing time: 0.171271 seconds.
82 unassigned variables, 3273 values in all current domains (med. size:44, max size:44) and 327 non-unary cost functions (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: 8451 (0 backtracks, 110 nodes, depth 111)
Problem decomposition in 55 clusters with size distribution: min: 1 median: 5 mean: 4.782 max: 12
****** Restart 1 with 1 discrepancies and UB=8451 ****** (110 nodes)
New solution: 8440 (0 backtracks, 110 nodes, depth 1)
New solution: 7474 (0 backtracks, 112 nodes, depth 2)
New solution: 6481 (0 backtracks, 117 nodes, depth 2)
New solution: 6358 (0 backtracks, 117 nodes, depth 1)
New solution: 6252 (0 backtracks, 120 nodes, depth 2)
New solution: 6250 (0 backtracks, 120 nodes, depth 1)
New solution: 6248 (0 backtracks, 120 nodes, depth 1)
New solution: 6223 (0 backtracks, 120 nodes, depth 1)
New solution: 6131 (0 backtracks, 123 nodes, depth 2)
New solution: 6121 (0 backtracks, 123 nodes, depth 1)
New solution: 6022 (0 backtracks, 123 nodes, depth 1)
New solution: 5098 (2 backtracks, 141 nodes, depth 2)
New solution: 5096 (2 backtracks, 141 nodes, depth 1)
New solution: 5095 (2 backtracks, 141 nodes, depth 1)
New solution: 5094 (2 backtracks, 141 nodes, depth 1)
New solution: 4887 (5 backtracks, 205 nodes, depth 2)
New solution: 4630 (10 backtracks, 232 nodes, depth 2)
New solution: 4629 (10 backtracks, 234 nodes, depth 2)
New solution: 4557 (30 backtracks, 366 nodes, depth 2)
New solution: 4556 (35 backtracks, 383 nodes, depth 1)
```

```
New solution: 4544 (37 backtracks, 395 nodes, depth 2)
New solution: 4388 (48 backtracks, 549 nodes, depth 2)
New solution: 4220 (68 backtracks, 706 nodes, depth 2)
New solution: 4218 (68 backtracks, 706 nodes, depth 1)
New solution: 4216 (68 backtracks, 706 nodes, depth 1)
New solution: 4208 (70 backtracks, 713 nodes, depth 1)
New solution: 4133 (71 backtracks, 715 nodes, depth 1)
New solution: 4127 (72 backtracks, 724 nodes, depth 2)
New solution: 4125 (75 backtracks, 733 nodes, depth 1)
New solution: 3709 (97 backtracks, 898 nodes, depth 2)
New solution: 3703 (240 backtracks, 2276 nodes, depth 2)
New solution: 3700 (240 backtracks, 2276 nodes, depth 1)
New solution: 3598 (278 backtracks, 2540 nodes, depth 2)
New solution: 3588 (279 backtracks, 2543 nodes, depth 1)
New solution: 3562 (280 backtracks, 2546 nodes, depth 2)
New solution: 3542 (280 backtracks, 2549 nodes, depth 2)
New solution: 3523 (283 backtracks, 2564 nodes, depth 2)
New solution: 3473 (384 backtracks, 3243 nodes, depth 2)
New solution: 3472 (386 backtracks, 3251 nodes, depth 2)
New solution: 3471 (388 backtracks, 3256 nodes, depth 1)
New solution: 3470 (388 backtracks, 3256 nodes, depth 1)
New solution: 3469 (391 backtracks, 3264 nodes, depth 1)
New solution: 3447 (409 backtracks, 3344 nodes, depth 2)
New solution: 3437 (414 backtracks, 3377 nodes, depth 2)
New solution: 3430 (439 backtracks, 3517 nodes, depth 2)
New solution: 3420 (439 backtracks, 3517 nodes, depth 1)
New solution: 3412 (468 backtracks, 3664 nodes, depth 2)
New solution: 3389 (517 backtracks, 4099 nodes, depth 2)
****** Restart 2 with 2 discrepancies and UB=3389 ****** (5528 nodes)

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

6. Download another difficult instance *scen07.wcsp*. Solve it using a variable neighborhood search algorithm (UDGVNS) with maximum cardinality search cluster decomposition and absorption [25] during 5 seconds:

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


Read 200 variables, with 44 values at most, and 2665 cost functions, with maximum arity 2.
Cost function decomposition time : 0.000209 seconds.
Reverse DAC dual bound: 10001 (+0.010%)
Preprocessing time: 0.415 seconds.
162 unassigned variables, 6481 values in all current domains (med. size:44, max size:44) and 764 non-unary cost functions (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: 1504924 (0 backtracks, 233 nodes, depth 234)
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=1504924 ****** (233 nodes)
New solution: 1485526 (0 backtracks, 233 nodes, depth 1)
New solution: 1485524 (0 backtracks, 233 nodes, depth 1)
New solution: 1485324 (0 backtracks, 233 nodes, depth 1)
New solution: 1445622 (0 backtracks, 233 nodes, depth 1)
New solution: 1445621 (0 backtracks, 233 nodes, depth 1)
New solution: 1445520 (0 backtracks, 233 nodes, depth 1)
New solution: 1425518 (0 backtracks, 236 nodes, depth 2)
New solution: 1425120 (0 backtracks, 244 nodes, depth 2)
New solution: 1425118 (0 backtracks, 244 nodes, depth 1)
New solution: 1425117 (0 backtracks, 244 nodes, depth 1)
New solution: 1415116 (0 backtracks, 244 nodes, depth 1)
New solution: 1405119 (0 backtracks, 244 nodes, depth 1)
New solution: 1404917 (4 backtracks, 258 nodes, depth 1)
New solution: 1394914 (4 backtracks, 258 nodes, depth 1)
New solution: 1385310 (5 backtracks, 269 nodes, depth 2)
New solution: 1385217 (6 backtracks, 274 nodes, depth 2)
New solution: 1375216 (6 backtracks, 274 nodes, depth 1)
New solution: 1375118 (6 backtracks, 274 nodes, depth 1)
New solution: 1365317 (6 backtracks, 278 nodes, depth 2)
New solution: 1365313 (10 backtracks, 291 nodes, depth 1)
New solution: 1365114 (10 backtracks, 294 nodes, depth 2)
New solution: 1365112 (12 backtracks, 300 nodes, depth 1)
New solution: 1365111 (12 backtracks, 300 nodes, depth 1)
New solution: 1364913 (12 backtracks, 301 nodes, depth 2)
New solution: 1364911 (16 backtracks, 313 nodes, depth 1)
New solution: 1344915 (22 backtracks, 361 nodes, depth 2)
New solution: 1344914 (26 backtracks, 374 nodes, depth 1)
New solution: 1344913 (26 backtracks, 374 nodes, depth 1)
New solution: 1344815 (26 backtracks, 374 nodes, depth 1)
New solution: 1344411 (26 backtracks, 380 nodes, depth 2)
New solution: 1344409 (26 backtracks, 380 nodes, depth 1)
New solution: 1344407 (26 backtracks, 380 nodes, depth 1)
New solution: 1344307 (30 backtracks, 394 nodes, depth 2)
```

```
New solution: 1344202 (30 backtracks, 394 nodes, depth 1)
New solution: 1343799 (40 backtracks, 452 nodes, depth 2)
New solution: 1343797 (49 backtracks, 483 nodes, depth 2)
New solution: 1343795 (71 backtracks, 565 nodes, depth 1)
New solution: 484210 (121 backtracks, 865 nodes, depth 2)
New solution: 484208 (121 backtracks, 865 nodes, depth 1)
New solution: 484207 (121 backtracks, 865 nodes, depth 1)
New solution: 474106 (123 backtracks, 873 nodes, depth 2)
New solution: 444404 (127 backtracks, 892 nodes, depth 2)
New solution: 435004 (127 backtracks, 892 nodes, depth 1)
New solution: 394609 (142 backtracks, 980 nodes, depth 2)
New solution: 394511 (142 backtracks, 980 nodes, depth 1)
New solution: 384514 (158 backtracks, 1053 nodes, depth 2)
New solution: 384512 (158 backtracks, 1053 nodes, depth 1)
New solution: 384412 (163 backtracks, 1075 nodes, depth 2)
New solution: 384411 (163 backtracks, 1075 nodes, depth 1)
New solution: 384409 (163 backtracks, 1076 nodes, depth 2)
New solution: 374409 (168 backtracks, 1091 nodes, depth 2)
New solution: 374408 (168 backtracks, 1091 nodes, depth 1)
New solution: 364406 (168 backtracks, 1091 nodes, depth 1)
New solution: 364405 (168 backtracks, 1093 nodes, depth 2)
New solution: 364308 (217 backtracks, 1375 nodes, depth 2)
New solution: 364108 (220 backtracks, 1391 nodes, depth 2)
New solution: 364107 (258 backtracks, 1625 nodes, depth 1)
New solution: 364106 (272 backtracks, 1678 nodes, depth 1)
New solution: 364105 (383 backtracks, 2504 nodes, depth 1)
****** Restart 2 with 2 discrepancies and UB=364105 ****** (4399 nodes)
New solution: 364104 (771 backtracks, 5441 nodes, depth 3)
****** Restart 3 with 2 discrepancies and UB=364104 ****** (6823 nodes)

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

7. Download file *404.wcsp*. Solve it using Depth-First Brand and Bound with Tree Decomposition and HBFS (BTD-HBFS) [10] based on a min-fill variable ordering:

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


Read 100 variables, with 4 values at most, and 710 cost functions, with maximum arity 3.
Cost function decomposition time : 4.7e-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: 0.007 seconds.
88 unassigned variables, 228 values in all current domains (med. size:2, max size:4) and 591 non-unary cost functions (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 2)
Optimality gap: [70, 124] 43.548 % (20 backtracks, 35 nodes)
New solution: 123 (34 backtracks, 64 nodes, depth 2)
Optimality gap: [77, 123] 37.398 % (34 backtracks, 64 nodes)
New solution: 119 (173 backtracks, 348 nodes, depth 2)
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 2)
Optimality gap: [97, 117] 17.094 % (261 backtracks, 609 nodes)
New solution: 114 (342 backtracks, 858 nodes, depth 2)
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 ( 23 removals by DEE) and 0.030 seconds.
end.
```

8. Solve the same problem using Russian Doll Search exploiting BTD [26]:

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

```
Read 100 variables, with 4 values at most, and 710 cost functions, with maximum arity 3.
Cost function decomposition time : 4.7e-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: 0.007 seconds.
88 unassigned variables, 228 values in all current domains (med. size:2, max size:4) and 591 non-unary cost functions (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 1)
---  done  cost = [0,0] (0 backtracks, 0 nodes, depth 1)

--- Solving cluster subtree 6 ...
New solution: 0 (0 backtracks, 0 nodes, depth 1)
---  done  cost = [0,0] (0 backtracks, 0 nodes, depth 1)

--- Solving cluster subtree 7 ...

...

--- Solving cluster subtree 44 ...
New solution: 42 (417 backtracks, 720 nodes, depth 5)
New solution: 39 (428 backtracks, 741 nodes, depth 8)
New solution: 35 (444 backtracks, 783 nodes, depth 21)
---  done  cost = [35,35] (554 backtracks, 958 nodes, depth 1)

--- Solving cluster subtree 46 ...
New solution: 114 (554 backtracks, 958 nodes, depth 1)
---  done  cost = [114,114] (554 backtracks, 958 nodes, depth 1)

Optimum: 114 in 554 backtracks and 958 nodes ( 50 removals by DEE) and 0.026 seconds.
end.
```

9. Solve another WCSP using the original Russian Doll Search method [28] with static variable ordering (following problem file) and soft arc consistency:

```
toulbar2 EXAMPLES/505.wcsp -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 : 0.000826 seconds.
Preprocessing time: 0.014949 seconds.
233 unassigned variables, 666 values in all current domains (med. size:2, max size:4) and 1966 non-unary cost functions (med. degree:16)
Initial lower and upper bounds: [2, 34347] 99.994%
Tree decomposition width  : 59
Tree decomposition height : 233
Number of clusters        : 239
Tree decomposition time: 0.017 seconds.
--- Solving cluster subtree 0 ...
New solution: 0 (0 backtracks, 0 nodes, depth 1)
---  done  cost = [0,0] (0 backtracks, 0 nodes, depth 1)

--- Solving cluster subtree 1 ...
New solution: 0 (0 backtracks, 0 nodes, depth 1)
---  done  cost = [0,0] (0 backtracks, 0 nodes, depth 1)

--- Solving cluster subtree 2 ...

...

--- Solving cluster subtree 3 ...
New solution: 21253 (26963 backtracks, 48851 nodes, depth 2)
New solution: 21251 (26991 backtracks, 48883 nodes, depth 3)
---  done  cost = [21251,21251] (26992 backtracks, 48883 nodes, depth 1)

--- Solving cluster subtree 238 ...
New solution: 21253 (26992 backtracks, 48883 nodes, depth 1)
---  done  cost = [21253,21253] (26992 backtracks, 48883 nodes, depth 1)

Optimum: 21253 in 26992 backtracks and 48883 nodes ( 0 removals by DEE) and 5.931 seconds.
end.
```

10. Solve the same WCSP using a parallel variable neighborhood search algorithm (UPDGVNS) with min-fill cluster decomposition [25] using 4 cores during 5 seconds:

```
mpirun -n 4 toulbar2 EXAMPLES/505.wcsp -vns -0=-3 -time=5


Read 240 variables, with 4 values at most, and 2242 cost functions, with maximum arity 3.
Cost function decomposition time : 0.001761 seconds.
Reverse DAC dual bound: 11120 (+81.403%)
Reverse DAC dual bound: 11128 (+0.072%)
Preprocessing time: 0.065 seconds.
233 unassigned variables, 666 values in all current domains (med. size:2, max size:4) and 1966 non-unary cost functions (med. degree:16)
Initial lower and upper bounds: [11128, 34354] 67.608%
Tree decomposition time: 0.018 seconds.
Problem decomposition in 89 clusters with size distribution: min: 4 median: 11 mean: 11.831 max: 23
New solution: 26266 (0 backtracks, 59 nodes, depth 60)
New solution: 26265 in 0.037 seconds.
New solution: 26264 in 0.041 seconds.
New solution: 26260 in 0.043 seconds.
New solution: 26258 in 0.046 seconds.
New solution: 25259 in 0.048 seconds.
New solution: 24260 in 0.079 seconds.
New solution: 24259 in 0.082 seconds.
New solution: 23260 in 0.091 seconds.
New solution: 23259 in 0.104 seconds.
New solution: 23258 in 0.114 seconds.
New solution: 23257 in 0.124 seconds.
New solution: 23256 in 0.148 seconds.
New solution: 23255 in 0.152 seconds.
New solution: 23254 in 0.179 seconds.
New solution: 23253 in 0.226 seconds.
New solution: 23252 in 0.240 seconds.
New solution: 23251 in 0.247 seconds.
New solution: 23250 in 0.253 seconds.
New solution: 23249 in 0.264 seconds.
New solution: 23247 in 0.383 seconds.
New solution: 22257 in 0.394 seconds.
New solution: 22256 in 0.399 seconds.
New solution: 22254 in 0.403 seconds.
New solution: 22253 in 0.414 seconds.
New solution: 21254 in 0.420 seconds.
New solution: 21253 in 0.429 seconds.
--------------------------------------------------------------------------
MPI_ABORT was invoked on rank 2 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...
```

11. 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 EXAMPLES/example.dec -vns


Read 25 variables, with 5 values at most, and 63 cost functions, with maximum arity 2.
Cost function decomposition time : 8e-06 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. 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 ****** (51 nodes)
****** Restart 3 with 4 discrepancies and UB=27 ****** (127 nodes)
****** Restart 4 with 8 discrepancies and UB=27 ****** (347 nodes)
****** Restart 5 with 16 discrepancies and UB=27 ****** (917 nodes)
Optimum: 27 in 601 backtracks and 1307 nodes ( 1775 removals by DEE) and 0.039 seconds.
end.
```

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

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

```
Read 25 variables, with 5 values at most, and 63 cost functions, with maximum arity 2.
Cost function decomposition time : 1.4e-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. 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 0.057 seconds.
Total CPU time = 0.267 seconds
Solving real-time = 0.068 seconds (not including preprocessing time)
end.
```

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

```
toulbar2 EXAMPLES/example.wcsp 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 : 9e-06 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. 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 2)
New solution: 28 (37 backtracks, 62 nodes, depth 2)
Optimality gap: [22, 28] 21.429 % (37 backtracks, 62 nodes)
Optimality gap: [23, 28] 17.857 % (285 backtracks, 581 nodes)
New solution: 27 (304 backtracks, 624 nodes, depth 2)
Optimality gap: [23, 27] 14.815 % (304 backtracks, 624 nodes)
Optimality gap: [24, 27] 11.111 % (323 backtracks, 676 nodes)
Optimality gap: [25, 27] 7.407 % (350 backtracks, 775 nodes)
Optimality gap: [26, 27] 3.704 % (350 backtracks, 785 nodes)
Optimality gap: [27, 27] 0.000 % (350 backtracks, 829 nodes)
HBFS open list restarts: 0.000 % and reuse: 10.769 % of 65
Node redundancy during HBFS: 17.612 %
Optimum: 27 in 350 backtracks and 829 nodes ( 209 removals by DEE) and 0.018 seconds.
end.
```

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

```
toulbar2 EXAMPLES/example.wcsp 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 : 9e-06 seconds.
Reverse DAC dual bound: 19 (+10.526%)
Reverse DAC dual bound: 22 (+13.636%)
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. 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 2)
New solution: 28 (32 backtracks, 46 nodes, depth 2)
Optimality gap: [23, 28] 17.857 % (37 backtracks, 58 nodes)
New solution: 27 (61 backtracks, 122 nodes, depth 2)
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 % (168 backtracks, 377 nodes)
Optimality gap: [26, 27] 3.704 % (176 backtracks, 441 nodes)
Optimality gap: [27, 27] 0.000 % (176 backtracks, 456 nodes)
HBFS open list restarts: 0.000 % and reuse: 25.926 % of 27
Node redundancy during HBFS: 26.974 %
Optimum: 27 in 176 backtracks and 456 nodes ( 47 removals by DEE) and 0.009 seconds.
end.
```

15. Download a Markov Random Field (MRF) file *pedigree9.uai* in UAI format from the toulbar2's Documentation Web page. Solve it using bounded (of degree at most 8) variable elimination enhanced by cost function decomposition in preprocessing [13] followed by BTD-HBFS exploiting only small-size (less than four variables) separators:

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


Read 1118 variables, with 7 values at most, and 1118 cost functions, with maximum arity 4.
No evidence file specified. Trying EXAMPLES/pedigree9.uai.evid
No evidence file.
Generic variable elimination of degree 4
Maximum degree of generic variable elimination: 4
Cost function decomposition time : 0.003702 seconds.
Preprocessing time: 0.071202 seconds.
232 unassigned variables, 517 values in all current domains (med. size:2, max size:7) and 415 non-unary cost functions (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: 0.046 seconds.
New solution: 865165767 energy: 298.395 prob: 2.564e-130 (72 backtracks, 140 nodes, depth 2)
New solution: 844300454 energy: 296.308 prob: 2.065e-129 (128 backtracks, 254 nodes, depth 2)
New solution: 819061410 energy: 293.784 prob: 2.577e-128 (185 backtracks, 368 nodes, depth 2)
New solution: 812515216 energy: 293.130 prob: 4.959e-128 (361 backtracks, 756 nodes, depth 2)
New solution: 806836620 energy: 292.562 prob: 8.751e-128 (399 backtracks, 834 nodes, depth 2)
New solution: 784864376 energy: 290.365 prob: 7.876e-127 (541 backtracks, 1147 nodes, depth 2)
New solution: 734383216 energy: 285.316 prob: 1.226e-124 (871 backtracks, 1935 nodes, depth 2)
New solution: 733157137 energy: 285.194 prob: 1.386e-124 (1011 backtracks, 2221 nodes, depth 2)
New solution: 727478541 energy: 284.626 prob: 2.446e-124 (1280 backtracks, 2777 nodes, depth 2)
New solution: 711570069 energy: 283.035 prob: 1.201e-123 (1418 backtracks, 3059 nodes, depth 2)
New solution: 711184893 energy: 282.997 prob: 1.248e-123 (1566 backtracks, 3371 nodes, depth 2)
HBFS open list restarts: 0.000 % and reuse: 44.123 % of 1591
Node redundancy during HBFS: 27.638 %
Optimum: 711184893 energy: 282.997 prob: 1.248e-123 in 22331 backtracks and 61690 nodes ( 19131 removals by DEE) and 4.370 seconds.
end.
```

16. Download another MRF file *GeomSurf-7-gm256.uai*. Solve it using Virtual Arc Consistency (VAC) in preprocessing [6] and exploit a VAC-based value ordering heuristic [7]:

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


Read 787 variables, with 7 values at most, and 3527 cost functions, with maximum arity 3.
No evidence file specified. Trying EXAMPLES/GeomSurf-7-gm256.uai.evid
No evidence file.
Cost function decomposition time : 0.000872 seconds.
Reverse DAC dual bound: 5875262175 energy: 1073.708 (+0.093%)
VAC dual bound: 5907774920 energy: 1076.959 (iter:519)
Preprocessing time: 2.280 seconds.
729 unassigned variables, 4819 values in all current domains (med. size:7, max size:7) and 3127 non-unary cost functions (med. degree:6)
Initial lower and upper bounds: [5907774920, 111615200815] 94.707%
c 2097152 Bytes allocated for long long stack.
c 4194304 Bytes allocated for long long stack.
New solution: 5922481881 energy: 1078.430 prob: 4.404e-469 (0 backtracks, 118 nodes, depth 119)
Optimality gap: [5922481881, 5922481881] 0.000 % (66 backtracks, 184 nodes)
Node redundancy during HBFS: 0.000 %
Optimum: 5922481881 energy: 1078.430 prob: 4.404e-469 in 66 backtracks and 184 nodes ( 2925 removals by DEE) and 2.339 seconds.
end.
```

17. Download another MRF file *1CM1.uai*. Solve it by applying first a strong dominance pruning test in preprocessing, and secondly, a modified variable ordering heuristic during search [4]:

```
toulbar2 EXAMPLES/1CM1.uai -dee=2 -m=2


Read 37 variables, with 350 values at most, and 703 cost functions, with maximum arity 2.
No evidence file specified. Trying EXAMPLES/1CM1.uai.evid
No evidence file.
Cost function decomposition time : 0.000286 seconds.
Reverse DAC dual bound: 101516542750 energy: -12733.307 (+0.961%)
```

```
Reverse DAC dual bound: 101545055767 energy: -12730.456 (+0.028%)
Preprocessing time: 9.087 seconds.
37 unassigned variables, 1679 values in all current domains (med. size:33, max size:350) and 624 non-unary cost functions (med. degree:35)
Initial lower and upper bounds: [101545055767, 239074057808] 57.526%
c 2097152 Bytes allocated for long long stack.
c 4194304 Bytes allocated for long long stack.
c 8388608 Bytes allocated for long long stack.
c 16777216 Bytes allocated for long long stack.
c 33554432 Bytes allocated for long long stack.
c 67108864 Bytes allocated for long long stack.
c 134217728 Bytes allocated for long long stack.
New solution: 104352979111 energy: -12449.664 prob: inf (0 backtracks, 52 nodes, depth 53)
Optimality gap: [103514214916, 104352979111] 0.804 % (18 backtracks, 70 nodes)
New solution: 104221186119 energy: -12462.843 prob: inf (18 backtracks, 78 nodes, depth 8)
Optimality gap: [103890459241, 104221186119] 0.317 % (22 backtracks, 82 nodes)
New solution: 104174014744 energy: -12467.560 prob: inf (22 backtracks, 87 nodes, depth 2)
Optimality gap: [103923500120, 104174014744] 0.240 % (22 backtracks, 87 nodes)
Optimality gap: [103959074242, 104174014744] 0.206 % (24 backtracks, 102 nodes)
Optimality gap: [103977982249, 104174014744] 0.188 % (25 backtracks, 109 nodes)
Optimality gap: [104174014744, 104174014744] 0.000 % (25 backtracks, 121 nodes)
Node redundancy during HBFS: 27.273 %
Optimum: 104174014744 energy: -12467.560 prob: inf in 25 backtracks and 121 nodes ( 3990 removals by DEE) and 9.995 seconds.
end.
```

18. Download a weighted Max-SAT file *brock200_4.clq.wcnf* in wcnf format from the toulbar2's Documentation Web page. Solve it using a modified variable ordering heuristic [4]:

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

```
c Read 200 variables, with 2 values at most, and 7011 clauses, with maximum arity 2.
Cost function decomposition time : 0.000375 seconds.
Reverse DAC dual bound: 91 (+86.813%)
Reverse DAC dual bound: 92 (+1.087%)
Preprocessing time: 0.036 seconds.
200 unassigned variables, 400 values in all current domains (med. size:2, max size:2) and 6811 non-unary cost functions (med. degree:68)
Initial lower and upper bounds: [92, 200] 54.000%
New solution: 189 (0 backtracks, 9 nodes, depth 10)
New solution: 188 (45 backtracks, 143 nodes, depth 36)
New solution: 187 (231 backtracks, 593 nodes, depth 34)
New solution: 185 (563 backtracks, 1305 nodes, depth 38)
New solution: 184 (27911 backtracks, 58966 nodes, depth 27)
New solution: 183 (298856 backtracks, 904243 nodes, depth 1)
Node redundancy during HBFS: 34.243 %
Optimum: 183 in 298932 backtracks and 909198 nodes ( 7535 removals by DEE) and 26.918 seconds.
end.
```

19. Download another WCSP file *latin4.wcsp*. Count the number of feasible solutions:

```
toulbar2 EXAMPLES/latin4.wcsp -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. 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                   :    0.310 seconds
... in 575 backtracks and 1318 nodes
end.
```

20. Download a crisp CSP file *GEOM40_6.wcsp* (initial upper bound equal to 1). Count the number of solutions using #BTD [12] using a min-fill variable ordering[2]:

[2]Warning, cannot use BTD to find all solutions in optimization.

```
toulbar2 EXAMPLES/GEOM40_6.wcsp -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 : 5e-06 seconds.
Preprocessing time: 0.000937 seconds.
40 unassigned variables, 240 values in all current domains (med. size:6, max size:6) and 78 non-unary cost functions (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
Number of used #goods  :    17190
Size of sep            :    4
Time                   :    0.055 seconds
... in 13689 backtracks and 27378 nodes
end.
```

21. Get a quick approximation of the number of solutions of a CSP with Approx#BTD [12]:

```
toulbar2 EXAMPLES/GEOM40_6.wcsp -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 : 6e-06 seconds.
Preprocessing time: 0.000965 seconds.
40 unassigned variables, 240 values in all current domains (med. size:6, max size:6) and 78 non-unary cost functions (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     :    133674945388437340315549622599 68
Upper bound of number of solutions : <= 1719926784000000000000000
Number of solutions    : ~= 480000000000000000000000
Number of #goods       :    468
Number of used #goods  :    4788
Size of sep            :    3
Time                   :    0.012 seconds
... in 3738 backtracks and 7476 nodes
end.
```

6 Command line options

If you just execute:

```
toulbar2
```

TOULBAR2 will give you its (long) list of optional parameter which we now describe in more detail.

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 [9] (aka Soft Neighborhood Substitutability) preprocessing.

6.1 General control

-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] give 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.

-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 [13] 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 [13].

-mst find a maximum spanning tree ordering for DAC

-M=[integer] apply the Min Sum Diffusion algorithm (default is inactivated, with a number of iterations of 0). See [7].

-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 [14], 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 [24]. 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**)

6.4 Tree search algorithms and tree decomposition selection

-hbfs=[integer] hybrid best-first search [2], 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)

-B=[integer] (0) DFBB, (1) BTD [10], (2) RDS-BTD [26], (3) RDS-BTD with path decomposition instead of tree decomposition [26] (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 [25] (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 [17].

-k=[integer] soft local consistency level (NC [18] with Strong NIC for global cost functions=0 [21], (G)AC=1 [27, 18], D(G)AC=2 [8], FD(G)AC=3 [19], (weak) ED(G)AC=4 [11, 22]) (default value is 4). See also [7, 23].

-A=[integer] enforces VAC [6] 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)

-dee=[integer] restricted dead-end elimination [9] (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 [20].

-q=[integer] use weighted degree variable ordering heuristic [5] 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 [4] (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).

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

6.8 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.

6.9 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 format in filename (or "problem.wcsp" if no parameter is given) writes also the graphviz dot file and the degree distribution of the input problem

-z=[integer] 1: saves original instance (by default), 2: saves after preprocessing (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)

6.10 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)

6.11 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*).

7 Input File formats

Notice that by default TOULBAR2 distinguishes file formats based on their extension.

7.1 cfn format (.cfn, .cfn.gz, and .cfn.xz file extension)

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.

See a full description in file document CFNformat.pdf in the doc repository on GitHub or directly on the toulbar2 Web site.

7.2 wcsp format (.wcsp file extension)

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*
  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)
- 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 [1]:
  - 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

  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).
list_size1 and *list_size2* must be equal in *ssame*.
Cost functions defined in intention cannot be shared.

Note

More about network-based global cost functions can be found here `https://metivier.users.greyc.fr/decomposable/`

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
```

- 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 ==
```

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
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
```

7.3 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
```

7.4 Partial Weighted MaxSAT format

Max-SAT input format (.cnf)

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

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

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

Weighted Max-SAT input format (.wcnf)

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

Example of Weighted Max-SAT formula:

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

Partial Max-SAT input format (.wcnf)

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

Example of Partial Max-SAT formula:

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

Weighted Partial Max-SAT input format (.wcnf)

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

Example of Weighted Partial Max-SAT formula:

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

7.5 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. 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 *-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 $W_{ij} \neq 0$

7.6 Linkage format (.pre)

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

8 Using it as a library

See TOULBAR2 reference manual which describes the libtb2.so C++ library API.

9 Using it from Python/Numberjack

See `http://numberjack.ucc.ie`.

References

[1] 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).

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] A. Favier, S. de Givry, and P. Jégou. Exploiting problem structure for solution counting. In *Proc. of CP-09*, pages 335–343, Lisbon, Portugal, 2009.

[13] 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.

[14] W. D. Harvey and M. L. Ginsberg. Limited discrepency search. In *Proc. of IJCAI-95*, Montréal, Canada, 1995.

[15] 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.

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

[17] 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.

[18] J. Larrosa. On arc and node consistency in weighted CSP. In *Proc. AAAI'02*, pages 48–53, Edmondton, (CA), 2002.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] G. Verfaillie, M. Lemaître, and T. Schiex. Russian doll search. In *Proc. of AAAI-96*, pages 181–187, Portland, OR, 1996.