TOULBAR2 User documentation

The TOULBAR2 developer team

June 20, 2018

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.

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 Gnu Public Library (GPL) as a set of C++ sources managed with git at `http://mulcyber.toulouse.inra.fr/projects/toulbar2`. If you want to use a released version, then you can download there binary archives as a shell archive, an rpm or a Debian package that should be easy to use on most Linux systems as well as an auto-installing executable for Windows.

If you want to compile it 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 library. You can then clone TOULBAR2 on your machine and compile it by executing:

```
git clone http://mulcyber.toulouse.inra.fr/anonscm/git/toulbar2/toulbar2.git
cd toulbar2/toulbar2
mkdir build
cd build
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 Library. Other benchmarks coming from various discrete optimization languages are available at Genotoul EvalGM [15].

3 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`, `.wcsp`, `.wcnf`, `.cnf`, `.qpbo`, `.uai`, `.LG`, `.pre` or `.bep`) is used to determine the nature of the file (see section 6). 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.

4 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 : 1.1e-05 seconds.
Reverse DAC lower bound: 20 (+10%)
Reverse DAC lower bound: 22 (+9.09091%)
Preprocessing time: 0.000893 seconds.
24 unassigned variables, 117 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: [22,64[ 65.625%
New solution: 29 (0 backtracks, 8 nodes, depth 9)
Optimality gap: [ 23 , 29 ] 20.6897 % (8 backtracks, 16 nodes)
New solution: 27 (8 backtracks, 24 nodes, depth 8)
Optimality gap: [ 24 , 27 ] 11.1111 % (14 backtracks, 30 nodes)
Optimality gap: [ 25 , 27 ] 7.40741 % (44 backtracks, 113 nodes)
Optimality gap: [ 26 , 27 ] 3.7037 % (54 backtracks, 151 nodes)
Optimality gap: [ 27 , 27 ] 0 % (54 backtracks, 157 nodes)
Node redundancy during HBFS: 30.5732 %
Optimum: 27 in 54 backtracks and 157 nodes ( 205 removals by DEE) and 0.004785 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 : 1.4e-05 seconds.
Reverse DAC lower bound: 20 (+10%)
Reverse DAC lower bound: 22 (+9.09091%)
Preprocessing time: 0.001324 seconds.
New solution: 27 (0 backtracks, 0 nodes, depth 1)
INCOP solving time: 0.24782 seconds.
24 unassigned variables, 117 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: [22,27[ 18.5185%
Optimality gap: [ 23 , 27 ] 14.8148 % (19 backtracks, 48 nodes)
Optimality gap: [ 24 , 27 ] 11.1111 % (23 backtracks, 60 nodes)
Optimality gap: [ 25 , 27 ] 7.40741 % (53 backtracks, 130 nodes)
Optimality gap: [ 27 , 27 ] 0 % (65 backtracks, 167 nodes)
Node redundancy during HBFS: 22.1557 %
Optimum: 27 in 65 backtracks and 167 nodes ( 137 removals by DEE) and 0.25257 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 : 0 seconds.
Reverse DAC lower bound: 20 (+10%)
Reverse DAC lower bound: 22 (+9.09091%)
Preprocessing time: 0.000889 seconds.
24 unassigned variables, 117 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: [22,28[ 21.4286%
Optimality gap: [ 23 , 28 ] 17.8571 % (7 backtracks, 14 nodes)
New solution: 27 (7 backtracks, 20 nodes, depth 6)
```

```
Optimality gap: [ 23 , 27 ] 14.8148 % (12 backtracks, 25 nodes)
Optimality gap: [ 24 , 27 ] 11.1111 % (26 backtracks, 67 nodes)
Optimality gap: [ 25 , 27 ] 7.40741 % (58 backtracks, 163 nodes)
Optimality gap: [ 27 , 27 ] 0 % (70 backtracks, 217 nodes)
Node redundancy during HBFS: 35.4839 %
Optimum: 27 in 70 backtracks and 217 nodes ( 158 removals by DEE) and 0.005221 seconds.
end.


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


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

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]:

```
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 : 7.3e-05 seconds.
Preprocessing time: 0.16806 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%
--- [0] LDS 0 --- (0 nodes)
c 2097152 Bytes allocated for x stack.
c 4194304 Bytes allocated for x stack.
c 8388608 Bytes allocated for x 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: [ 349 , 3401 ] 89.7383 % (44869 backtracks, 94455 nodes)
New solution: 3400 (56746 backtracks, 118233 nodes, depth 24)
New solution: 3399 (56747 backtracks, 118235 nodes, depth 23)
New solution: 3389 (56749 backtracks, 118248 nodes, depth 29)
Optimality gap: [ 1023 , 3389 ] 69.8141 % (90007 backtracks, 184743 nodes)
Optimality gap: [ 1306 , 3389 ] 61.4636 % (91295 backtracks, 187319 nodes)
Optimality gap: [ 1818 , 3389 ] 46.3559 % (93973 backtracks, 192675 nodes)
Optimality gap: [ 2261 , 3389 ] 33.2842 % (94054 backtracks, 192837 nodes)
Optimality gap: [ 2777 , 3389 ] 18.0584 % (94060 backtracks, 192849 nodes)
Optimum: 3389 in 94062 backtracks and 192853 nodes ( 375927 removals by DEE) and 59.6263 seconds.
end.
```

5. Download file *404.wcsp*. Solve it using Depth-First Brand and Bound with Tree Decomposition (BTD) [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 : 7.9e-05 seconds.
Reverse DAC lower bound: 63 (+28.5714%)
Reverse DAC lower bound: 65 (+3.07692%)
Reverse DAC lower bound: 66 (+1.51515%)
Preprocessing time: 0.007746 seconds.
```

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

```
88 unassigned variables, 226 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: [66,155[ 57.4194%
Tree decomposition width  : 19
Tree decomposition height : 43
Number of clusters        : 47
Tree decomposition time: 0.002612 seconds.
New solution: 124 (22 backtracks, 40 nodes, depth 2)
Optimality gap: [ 69 , 124 ] 44.3548 % (22 backtracks, 40 nodes)
New solution: 123 (36 backtracks, 69 nodes, depth 2)
Optimality gap: [ 76 , 123 ] 38.2114 % (36 backtracks, 69 nodes)
New solution: 117 (166 backtracks, 333 nodes, depth 2)
Optimality gap: [ 87 , 117 ] 25.641 % (166 backtracks, 333 nodes)
Optimality gap: [ 89 , 117 ] 23.9316 % (237 backtracks, 521 nodes)
Optimality gap: [ 92 , 117 ] 21.3675 % (270 backtracks, 638 nodes)
New solution: 114 (375 backtracks, 897 nodes, depth 2)
Optimality gap: [ 97 , 114 ] 14.9123 % (375 backtracks, 897 nodes)
Optimality gap: [ 99 , 114 ] 13.1579 % (437 backtracks, 1108 nodes)
Optimality gap: [ 100 , 114 ] 12.2807 % (439 backtracks, 1130 nodes)
Optimality gap: [ 101 , 114 ] 11.4035 % (446 backtracks, 1198 nodes)
Optimality gap: [ 102 , 114 ] 10.5263 % (450 backtracks, 1207 nodes)
Optimality gap: [ 104 , 114 ] 8.77193 % (470 backtracks, 1265 nodes)
Optimality gap: [ 105 , 114 ] 7.89474 % (528 backtracks, 1457 nodes)
Optimality gap: [ 106 , 114 ] 7.01754 % (528 backtracks, 1464 nodes)
Optimality gap: [ 107 , 114 ] 6.14035 % (534 backtracks, 1490 nodes)
Optimality gap: [ 109 , 114 ] 4.38596 % (545 backtracks, 1533 nodes)
Optimality gap: [ 110 , 114 ] 3.50877 % (545 backtracks, 1549 nodes)
Optimality gap: [ 112 , 114 ] 1.75439 % (556 backtracks, 1585 nodes)
Optimality gap: [ 113 , 114 ] 0.877193 % (556 backtracks, 1601 nodes)
Optimality gap: [ 114 , 114 ] 0 % (556 backtracks, 1618 nodes)
HBFS open list restarts: 0 % and reuse: 10.8635 % of 359
Node redundancy during HBFS: 33.1891 %
Optimum: 114 in 556 backtracks and 1618 nodes ( 22 removals by DEE) and 0.032429 seconds.
end.
```

6. Solve the same problem using Russian Doll Search exploiting BTD [25]:

```
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 : 7.8e-05 seconds.
Reverse DAC lower bound: 63 (+28.5714%)
Reverse DAC lower bound: 65 (+3.07692%)
Reverse DAC lower bound: 66 (+1.51515%)
Preprocessing time: 0.008334 seconds.
88 unassigned variables, 226 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: [66,155[ 57.4194%
Tree decomposition width  : 19
Tree decomposition height : 43
Number of clusters        : 47
Tree decomposition time: 0.00235 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: 41 (398 backtracks, 692 nodes, depth 7)
New solution: 40 (409 backtracks, 708 nodes, depth 7)
New solution: 36 (419 backtracks, 738 nodes, depth 21)
---  done  cost = [36,36] (507 backtracks, 878 nodes, depth 1)

--- Solving cluster subtree 46 ...
New solution: 114 (507 backtracks, 878 nodes, depth 1)
---  done  cost = [114,114] (507 backtracks, 878 nodes, depth 1)

Optimum: 114 in 507 backtracks and 878 nodes ( 13 removals by DEE) and 0.026724 seconds.
end.
```

7. Solve a WCSP using the original Russian Doll Search method [27] 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 : 0.000854 seconds.
Preprocessing time: 0.016547 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[ 99.9942%
Tree decomposition width  : 59
Tree decomposition height : 233
Number of clusters        : 239
Tree decomposition time: 0.017876 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 6.83524 seconds.
end.
```

8. 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 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.
Generic variable elimination of degree 4
Maximum degree of generic variable elimination: 4
Cost function decomposition time : 0.008698 seconds.
Preprocessing time: 0.110693 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[ 95.8185%
Tree decomposition width  : 227
Tree decomposition height : 230
Number of clusters        : 890
Tree decomposition time: 0.054128 seconds.
New solution: 865165767 log10like: -129.591 prob: 2.56352e-130 (72 backtracks, 140 nodes, depth 2)
New solution: 844300454 log10like: -128.685 prob: 2.06541e-129 (128 backtracks, 254 nodes, depth 2)
New solution: 819061410 log10like: -127.589 prob: 2.57706e-128 (185 backtracks, 368 nodes, depth 2)
New solution: 812515216 log10like: -127.305 prob: 4.95931e-128 (361 backtracks, 756 nodes, depth 2)
New solution: 806836620 log10like: -127.058 prob: 8.75064e-128 (399 backtracks, 834 nodes, depth 2)
New solution: 784864376 log10like: -126.104 prob: 7.87558e-127 (541 backtracks, 1147 nodes, depth 2)
New solution: 734383216 log10like: -123.911 prob: 1.22645e-124 (871 backtracks, 1935 nodes, depth 2)
New solution: 733157137 log10like: -123.858 prob: 1.38643e-124 (1011 backtracks, 2221 nodes, depth 2)
New solution: 727478541 log10like: -123.611 prob: 2.44634e-124 (1278 backtracks, 2773 nodes, depth 2)
New solution: 711184893 log10like: -122.904 prob: 1.24779e-123 (1672 backtracks, 3583 nodes, depth 2)
HBFS open list restarts: 0 % and reuse: 40.6146 % of 1497
Node redundancy during HBFS: 33.2613 %
Optimum: 711184893 log10like: -122.904 prob: 1.24779e-123 in 15757 backtracks and 47187 nodes ( 15233 removals by DEE) and 3.65787 seconds.
end.
```

9. 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.
Lb before VAC: 4506326046
Cost function decomposition time : 0.001407 seconds.
```

```
Reverse DAC lower bound: 5853428901 (+0.0694528%)
Preprocessing VAC mean lb/incr: 2.8643e+06     total increments: 463     cyclesize: 22.2419     k: 1.29806 (mean), 4 (max)
Lb after VAC: 5905811708
Preprocessing time: 2.70276 seconds.
730 unassigned variables, 4828 values in all current domains (med. size:7, max size:7) and 3131 non-unary cost functions (med. degree:6)
Initial lower and upper bounds: [5905811708,111615200815[ 94.7088%
c 2097152 Bytes allocated for x stack.
c 4194304 Bytes allocated for x stack.
New solution: 6135086360 log10like: -477.589 prob: 2.57374e-478 (0 backtracks, 116 nodes, depth 117)
Optimality gap: [ 5908760056 , 6135086360 ] 3.68905 % (104 backtracks, 220 nodes)
New solution: 5922481881 log10like: -468.356 prob: 4.40413e-469 (104 backtracks, 331 nodes, depth 109)
Optimality gap: [ 5909173184 , 5922481881 ] 0.224715 % (191 backtracks, 418 nodes)
Optimality gap: [ 5922481881 , 5922481881 ] 0 % (191 backtracks, 420 nodes)
VAC mean lb/incr: -nan     total increments: 0     cyclesize: -nan     k: -nan (mean), 0 (max)
Node redundancy during HBFS: 1.19048 %
Optimum: 5922481881 log10like: -468.356 prob: 4.40413e-469 in 191 backtracks and 420 nodes ( 18060 removals by DEE) and 3.08068 seconds.
end.
```

10. 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.
Cost function decomposition time : 0.000352 seconds.
Reverse DAC lower bound: 102868791154 (+0.010277%)
Preprocessing time: 11.6439 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: [102868791154,239074057808[ 56.972%
c 2097152 Bytes allocated for x stack.
c 4194304 Bytes allocated for x stack.
c 8388608 Bytes allocated for x stack.
c 16777216 Bytes allocated for x stack.
c 33554432 Bytes allocated for x stack.
c 67108864 Bytes allocated for x stack.
New solution: 104206588216 log10like: 5413.18 prob: inf (0 backtracks, 48 nodes, depth 49)
Optimality gap: [ 103905905484 , 104206588216 ] 0.288545 % (22 backtracks, 70 nodes)
New solution: 104174014744 log10like: 5414.59 prob: inf (22 backtracks, 82 nodes, depth 3)
Optimality gap: [ 103963607615 , 104174014744 ] 0.201977 % (23 backtracks, 83 nodes)
Optimality gap: [ 104010471712 , 104174014744 ] 0.15699 % (24 backtracks, 93 nodes)
Optimality gap: [ 104174014744 , 104174014744 ] 0 % (25 backtracks, 96 nodes)
Node redundancy during HBFS: 19.7917 %
Optimum: 104174014744 log10like: 5414.59 prob: inf in 25 backtracks and 96 nodes ( 3803 removals by DEE) and 12.2549 seconds.
end.
```

11. 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 : 0.001202 seconds.
Reverse DAC lower bound: 91 (+86.8132%)
Reverse DAC lower bound: 92 (+1.08696%)
Preprocessing time: 0.041114 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%
New solution: 189 (0 backtracks, 9 nodes, depth 10)
New solution: 188 (20 backtracks, 55 nodes, depth 10)
New solution: 187 (113 backtracks, 326 nodes, depth 20)
New solution: 186 (428 backtracks, 1013 nodes, depth 21)
New solution: 185 (8011 backtracks, 17396 nodes, depth 14)
New solution: 184 (13807 backtracks, 29658 nodes, depth 11)
New solution: 183 (13821 backtracks, 29682 nodes, depth 7)
Node redundancy during HBFS: 26.313 %
Optimum: 183 in 299378 backtracks and 812567 nodes ( 3362 removals by DEE) and 24.7018 seconds.
end.
```

12. 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 : 0 seconds.
Reverse DAC lower bound: 48 (+2.08333%)
Preprocessing time: 0.010578 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[ 95.2%
Optimality gap: [ 49 , 1000 ] 95.1 % (12 backtracks, 26 nodes)
Optimality gap: [ 58 , 1000 ] 94.2 % (353 backtracks, 813 nodes)
Optimality gap: [ 72 , 1000 ] 92.8 % (575 backtracks, 1360 nodes)
Optimality gap: [ 1000 , 1000 ] 0 % (575 backtracks, 1369 nodes)
Number of solutions    : =  576
Time                   :    0.465197 seconds
... in 575 backtracks and 1369 nodes
end.
```

13. 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]:

```
toulbar2 EXAMPLES/GEOM40_6.wcsp -O=-3 -a -B=1
```

```
Read 40 variables, with 6 values at most, and 78 cost functions, with maximum arity 2.
Cost function decomposition time : 0 seconds.
Preprocessing time: 0.001249 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%
Tree decomposition width  : 5
Tree decomposition height : 20
Number of clusters        : 29
Tree decomposition time: 0.000792 seconds.
Number of solutions    : =  4.11111e+23
Number of #goods       :    3993
Number of used #goods  :    17190
Size of sep            :    4
Time                   :    0.078109 seconds
... in 13689 backtracks and 27378 nodes
end.
```

14. 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
```

```
Read 40 variables, with 6 values at most, and 78 cost functions, with maximum arity 2.
Cost function decomposition time : 0 seconds.
Preprocessing time: 0.001978 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%

part 1 : 40 variables and 71 constraints (really added)
part 2 : 10 variables and 7 constraints (really added)
--> number of parts : 2
--> time : 0.000549 seconds.

Tree decomposition width  : 5
Tree decomposition height : 17
Number of clusters        : 33
Tree decomposition time: 0.000967 seconds.

Cartesian product     :    1.33675e+31
Upper bound of number of solutions : <= 1.71993e+24
Number of solutions    : ~= 4.8e+23
Number of #goods       :    468
Number of used #goods  :    4788
Size of sep            :    3
Time                   :    0.017849 seconds
... in 3738 backtracks and 7476 nodes
end.
```

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

5 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.

5.1 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.

5.2 Preprocessing

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

-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].

5.3 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 = 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)*] assigns variable of index i to value a (multiple assignments 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" or given directly as an additional input filename with ".sol" extension and without **-x**)

5.4 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 [25], (3) RDS-BTD with path decomposition instead of tree decomposition [25] (default value is 0)

-O=[filename] reads a variable elimination order from a file in order to build a tree decomposition (if BTD-like and/or variable elimination methods are used). 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

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 merges leaf clusters with their fathers if small local treewidth (in conjunction with option "-e")

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

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

5.5 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 [26, 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)

-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)

5.6 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 10000).

-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).

5.7 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 shows each solution found during search. The solution is printed on one line, giving the value (integer) of each variable successively in increasing file order.

5.8 File output

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

-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)*] assigns variable of index i to value a (multiple assignments 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)

5.9 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 (default value is 1000 representing $\varepsilon = \frac{1}{1000}$)

5.10 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*).

6 Input File formats

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

6.1 cfn format (.cfn and .cfn.gz 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-compressed files directly as input (.cfn.gz).
- 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.

6.2 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 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 hard 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
```

- 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
```

6.3 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
```

6.4 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
```

6.5 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 $W_{ij} \neq 0$

6.6 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.

7 Using it as a library

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

8 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] 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.

[26] 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.

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