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".

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 [], and Maximum A Posteriori (MAP) on Markov random field []. 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 [] 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 autoinstalling 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 toulbar/toulbar2
mkdir build
cd build
cmake ..
make
```

Fnally, TOULBAR2 should be soon 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 Funtion Library.

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 `.wcsp`, `.wcnf`, `.cnf`, `.qpbo`, `.uai`, `.LG`, `.pre` or `.bep`) is used to determine the nature of the file (see section 5). 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 Command line options

If you just execute:

```
toulbar2
```

Toulbar2 will give you its (long) list of optional parameter which we now dscribe in more detail. If you don't known much about Constraint and Cost Function Programming, section **??** describes some of the inner working of Toulbar2 to help you tune it to your requirements.

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

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

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

4.1 General control

-a finds all solutions (or count 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.

4.2 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 of cost functions with arity ¿=3 into smaller arity cost functions (default option)

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

-mst maximum spanning tree DAC ordering

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

4.3 Initial upper bounding

-l=[integer] limited discrepancy search, use a negative value to stop the search after the given absolute number of discrepancies has been explored (discrepancy bound = 4 by default)

-L=[integer] randomized (quasi-random variable ordering) search with restart (maximum number of nodes = 10000 by default)

-i=["string"] initial upperbound found by INCOP local search solver.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 value heuristic – read from default file "sol" or given as input filename with ".sol" extension)

4.4 Tree search algorithm and tree decomposition selection

-hbfs=[integer] hybrid best-first search, restarting from the root after a given number of backtracks (default value is 10000)

-open=[integer] hybrid best-first search limit on the number of open nodes (default value is -1)

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

-O=[filename] reads a variable elimination order from a file in order to build a tree decomposition (if BTD-like and/or variable elimination methods are used) and also a compatible 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 maximum cluster separator size (merge cluster with its father otherwise, use a negative value for no limit) (default value is -1)

-X=[integer] limit on minimum number of proper variables in a cluster (merge cluster with its father otherwise, use a zero for no limit) (default value is 0)

-E 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)

4.5 Node processing & bounding options

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

-k=[integer] soft local consistency level (NC with Strong NIC for global cost functions=0, (G)AC=1, D(G)AC=2, FD(G)AC=3, (weak) ED(G)AC=4) (default value is 4)

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

-dee=[integer] restricted dead-end elimination (value pruning by dominance rule from EAC value (dee¿=1 and dee¡=3)) and soft neighborhood substitutability (in preprocessing (dee=2 or dee=4) or during search (dee=3)) (default value is 1)

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

4.6 Branching, variable and value ordering

-svo searches using a static variable ordering heuristic. The variable order value usd will be the same order as the DAC order (see section **??**).

-b searches using only binary branching instead of the default that uses binary branching for interval domains and small domains and dichotomic branching for large enumerated domains.

-c searches using binary branching with last conflict backjumping variable ordering heuristic and dichotomic branching for large domains (default option).

-q=[integer] use weighted degree variable ordering heuristic 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 (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 sorted unary costs.

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

4.7 Console output

-help shows the default help message that TOULBAR2 prints when it gets no argument.

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

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

4.8 File output

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

-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

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

-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)

4.9 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 $\varepsilon = \frac{1}{1000}$)

4.10 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 num 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 num of binary cost functions to include
  - p3 is the percentage % of submodular cost functions among p2 cost functions (plus 10 permutations of two randomly-chosen values for each domain)

  tern-n-d-t1-p2-p3-seed

  - p3 is the num of ternary cost functions

  nary-n-d-t1-p2-p3...-pn-seed

  - pn is the num of n-ary cost functions

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

  - pn is the num of salldiff global cost functions (p2 and p3 still being used for the number of random binary and ternary cost functions)

5 Input File formats

5.1 wcsp format

5.1.1 CPD final stanza

5.2 UAI/LG formats

5.3 (Partial Weighted) MaxSAT format

5.4 QPBO format

5.5 Linkage format

5.6 BPE format

6 Using it as a library

7 Using it from Python/Numberjack

8 How does it work

8.1 Preprocessing

8.2 Upper bounding

8.3 Search algorithms