Sat4j 2.3.2: on the fly solver configuration System Description

Daniel Le Berre and Stéphanie Roussel*

Université Lille Nord de France, F-59000 Lille, France
Université d'Artois, CRIL, F-62300 Lens, France
CNRS, UMR8188, F-62300 Lens, France

Abstract. Taking the best of SAT (at large) technology for a given class of problems requires usually an expert knowledge of the solver used or to rely on an automatic configuration tool. The former condition is usually not satisfied as soon as the user is not a member of the SAT community, and the latter solution does not help the user to understand what is going on inside the solver. We propose an approach that allows the end users of Sat4j to configure a solver for a specific instance. Such an approach is based on both the ability to change dynamically the main settings of the solver when it is running and to display to the user several metrics that inform her about the state of the search in the solver. While the latter has been already explored in the SAT community, and the former has been explored in the constraint programming community, we believe that the combination of that two features is quite unique for SAT solvers. We believe that such a tool could also be used in the classroom to help students to understand how CDCL solvers work.

1 Introduction

The SAT competition has been a good way to promote the design of fast, reliable and general purpose SAT solvers during the last decade. In recent years, the use of portfolios (i.e. a system composed of several heterogeneous SAT engines) is considered by some authors [1] as the best way to tackle SAT because they show great performances during the competitions. However, when it comes to use SAT solvers in a company, the results of the SAT competition may not be that relevant. A company is likely to invest on a few SAT engines for both intellectual property and maintainability concerns. During its invited talk at Pragmatics of SAT 2011, Alexander Nadel discussed the way Intel was moving from a specific solver (Eureka) to a more modular one, to adapt the specificities of the various problems to be solved there. From the beginning, Sat4j has been designed in that spirit. However, when it comes to select which features to enable to solve a particular problem, an expert knowledge is usually required (that's

* Part of this work was supported by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the 'Contrat de Projets Etat Region (CPER) 2007-2013', and ANR TUPLES.

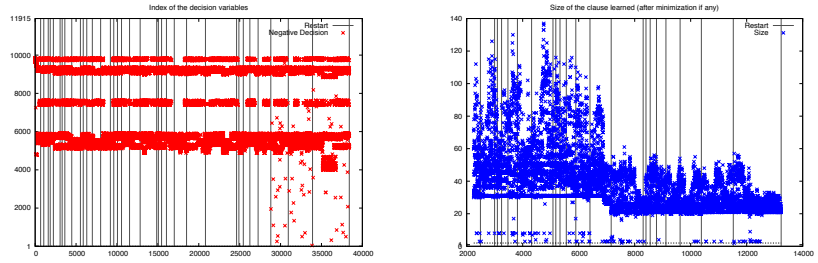
the approach used at Intel for instance). Another approach is to use automatic configuration tools such as ParamILS [2], which works fine when one has to maximize the number of problems to be solved (or to minimize the runtime of the solver) for a given set of benchmarks. Here we would like to report the approach that we have taken to allow the end users of Sat4j to configure a solver for their specific problem. Such an approach is based on both the ability to change dynamically the main strategies of the solver when it is running and to display to the user several metrics that inform her about the state of the search in the solver. While the latter has been already explored in the SAT community [3], but oriented toward the state of the instance, and the former has been explored in the constraint programming community [4], we believe that the combination of that two features is quite unique for SAT solvers.

2 Monitoring a CDCL solver behavior

Advanced SAT solver users usually use various metrics (number of decisions, conflicts, restarts, etc.) to evaluate the best settings or solver for solving a specific SAT instance (or class of instances). However, the wide success of SAT technology in solving combinatorial problems put SAT solvers in the hands of users not familiar with the way SAT solvers work. As such, they are considered as black boxes, taking a CNF as input and outputting either a model or that such a model does not exist. The challenge is thus to allow those users to get the best from SAT technology, i.e. to let them setup their SAT solver without any deep knowledge of its internals. In order to address that issue, we need to provide some feedback to the end user about what's going on inside the solver, and to provide her some hints to escape known traps, to let her "drive" the solver on a particular SAT instance. We decided to display in real time several metrics from the solver that are important to understand the behavior of the solver. The idea is to allow the user to adapt the configuration of the solver at the light of those metrics. The various metrics are logged in text files and displayed using Gnuplot version 4.6 for an efficient but platform dependent solution or displayed directly in Java thanks to the jchart2d library for a more integrated but slower solution.

Decisions index Displays the id of the decision variables over time (see Figure 1(a)). That information is logged at each branching decision. It allows to check if the decisions are limited to a group of variables or if they are spread across all variables. In the former case, adding some randomization to the heuristics may help (see Figure 1(a) on the right the effect of activating random walks). We noted several interesting patterns using that metric. Note that it is sometimes interesting to distinguish between positive and negative decisions (on which phase the solver first branches on). As such, we display the indexes on two graphs, one for positive literals and one for negative literals.

Activity value Displays for each variable the value of its activity. That information is logged at each restart, because it needs to be done for all the variables at



(a) Negative decisions index (b) Learned clauses size

Fig. 1. Examples of metrics displayed at runtime

once. As such, the time needed to create the data file is not negligible. That information is mainly useful to check that the heuristics work as expected (rescale every now and then, decision ids are taken among the variables with highest value). No real action can be decided from that metric.

Size of learned clauses CDCL solvers learn one clause for each conflicting assignment. That information is logged every time a conflict is found. There exist several minimization strategies that can be used to reduce those clauses, especially the one introduced in Minisat 1.13[5]. That metric can be used to check whether such techniques are effective or not. In Figure 1(b) for instance, one can see the minimization strategy at work since step 7000.

Evaluation of learned clauses There are several ways to evaluate learned clauses (activity, LBD[6]). That metric provides the evaluation of the learned clauses available in the solver. Since "worst" learned clauses are removed periodically, one should observe learned clauses of increasing quality.

Speed Number of propagations per second. We compute every two seconds how many propagations have been performed. The solver usually has a decreasing velocity because it learns new clauses over time. Aggressive learned clauses deletion strategies such as glucose's one allow to keep the solver in a good velocity. The objective of that metric is to allow the end user to remove some learned clauses when the solver slows down significantly. Note however that such metric is highly dependent of which other metrics are logged, because they affect the running time of the solver. We noticed that such metric is currently unstable, i.e. varies a lot during the search, thus making it difficult to use in practice. We are currently looking for a better way to measure velocity.

Decision level Denotes the number of decisions taken before reaching a conflict. That metric is logged at each conflict found. It corresponds to the depth in the

search tree. The depth should decrease over time. If that measure is stalled, then performing a restart can help escaping that plateau.

Trail level Denotes the number of variables assigned when reaching a conflict. That metric is logged at each conflict found. A pathological case that can be noticed using that measure is when the solver assigns almost all the variables before reaching a conflict. In that case, the solver is in a "generate and test" situation, i.e. conflicts arise on full assignments, with few chances to take advantage of backjumping and learning to cut the search space.

All those metrics can easily be logged in any CDCL solver to monitor the behavior of that solver. However, for sake of efficiency, most solvers do not allow to change its settings at runtime (they are usually decided at compilation time). Sat4j was designed from the beginning to be highly flexible, and allowing on the fly solver configuration could be added without much efforts because the main changes were limited to allow concurrent access and modifications of the main solver components.

3 On-the-fly solver configuration

Many components are configurable in Sat4j. See the previous system description [7] for details. We focus here on features that can now be changed when the solver is running.

Phase selection When the heuristic selects a decision variable, there are several strategies possible to select the phase (or polarity) of the variable to branch on first. It can be fixed (negative or positive first, user defined), random, or based on previous assignment (RSAT phase saving[8]), etc. The application of each strategy can be checked by monitoring the indexes of decision variables on the dedicated graphs, which allows to easily check their implementation.

Conflict clause minimization Minisat 1.13 introduced two conflict minimization procedures to reduce further the clause derived by conflict analysis. Sat4j implements both, called simple and expensive simplifications. Such feature can also be deactivated. While clause minimization works usually pretty well on CNF, there are some cases with instances containing long pseudo-boolean constraints where clause minimization is ineffective. The successful application of such features should decrease the size of learned clauses. The amount of reduction could be used to measure the effectiveness of the technique. Figure 1(b) shows an example of successful use of clause minimization.

Random walk Sometimes the heuristics keep the solver in a part of the search tree with few chances to escape. This can be seen by looking at the index of the variables chosen by the heuristics over time. Sometimes, some patterns may occur (see Figure 1(a) left part). In order to escape from the bad choices of the

heuristics, a common practice is to pick a variable at random, which is often referred to as making a random walk. Sat4j allows to add random walks with a given probability to the heuristics. Its action can be checked by looking at the distribution of the decision variable indexes over time (see Figure 1(a) right part).

Learned constraints deletion strategy Recent works suggest that aggressive deletion strategy is important. Several measures of the importance of the learned clause can be used: activity, as in the original Minisat, i.e. clauses that contribute often to a conflict, or Literals Blocks Distance (LBD), as defined in Glucose[6], that partition clauses by the number of different decision levels involved during its creation. Those measures are used to remove periodically the worst half learned clauses from the solver. We allow the user to choose one of those evaluation schemes, and to cleanup the database periodically (every x conflicts) or on demand (the user asks directly the solver to perform the cleanup).

Restart strategy Restarts strategies have received also a lot of attention those recent years, especially dynamic restart strategies, i.e. strategies that adapt themselves to the instance. Sat4j provides only static restart strategies at the moment (the one inherited from Minisat, the one used in Picosat and a Luby-style one). However, we also allow the end user to decide when to restart. Restarts are represented as vertical bars when they occur on most graphs (see Figure 1).

Prebuilt solvers Sat4j comes with a wide range of prebuilt solvers for both decision and optimization problems. The configuration that performs the best on a wide range of benchmarks from the application category of the SAT competition is the default one. However, a different configuration can perform much better on a specific class of benchmarks. Prebuilt solvers are thus useful for us to record combination of features that proved to be useful.

4 Conclusion

We introduced in Sat4j 2.3.2 a new feature to allow our end users to get a better understanding of the effects of the various parameters available in our CDCL solver[1]. That feature is based on the possibility to log various information during the search, to display that information live using Gnuplot, and more importantly to allow the end user to change at runtime those parameters to drive the solver during the search. It is to the best of our knowledge the first time that such feature is implemented in a SAT solver. While SAT solvers have been thought as black boxes, push-button technology by many users, delivering search monitoring and control to the end user gives her the power to gain a better understanding of the behavior of the solver on her particular problem. Thanks to the visualization of the literals returned by the heuristics, we found a bug in our implementation

[1] The tool is available for download from Sat4j usual web page: `http://download.forge.objectweb.org/sat4j/sat4j-pos12.tgz`.

of the RSAT phase strategy that was introduced between Sat4j 1.7 and Sat4j 2.0, when Sat4j was decomposed into modules to avoid dependencies on third party code in the core sat and pseudo boolean components. The phase was no longer properly recorded and we could notice it by looking at the phase of the decision variables (only negative literals showed up). Since that bug was not making the solver incorrect, and that many changes that prevented the solver to behave exactly as the 1.7 release happened during the release of Sat4j 2.0, that problem remained unnoticed for four years. Thanks to the "remote control", i.e. the GUI that allows to easily set the various parameters of a CDCL solver in Sat4j, one end user reported that she could quickly find a configuration that provided solutions of better quality than any of the pre-built solvers.As suggested by an anonymous referee, we plan to allow recording and replaying the dynamic setup of the solver to build a fully customized solver. We believe that such a tool can be useful in many contexts. First, it helps us to understand why Sat4j is behaving poorly on particular benchmarks. The situation of "generate and test" described earlier is a real issue that we met on a specific class of benchmarks. Second, users of Sat4j that would like to find the most appropriate settings for their problem can use that interactive tool to do it: we developed that tool for them, and from their early feedback, the tool suits them. Finally, such a tool is also useful for teaching CDCL architecture to students: many strategies are available in Sat4j and the newcomer to CDCL architecture can see live the effect of changing a single parameter. While all SAT solvers have the ability to log information about their state, few have the capability to allow on the fly configuration: this is only possible if some kind of runtime configuration exists in the solver, which is rarely the case for SAT solvers designed for speed (where most parameters are fixed at compile time). As such, we believe that Sat4j is currently a really nice framework for teaching boolean satisfaction and optimization courses.

References

1. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: Portfolio-based algorithm selection for sat. J. Artif. Intell. Res. (JAIR) **32** (2008) 565–606
2. Hutter, F., Hoos, H.H., Stützle, T.: Automatic algorithm configuration based on local search. In: AAAI, AAAI Press (2007) 1152–1157
3. Sinz, C., Dieringer, E.M.: Dpvis - a tool to visualize the structure of sat instances. In Bacchus, F., Walsh, T., eds.: SAT. Volume 3569 of LNCS., Springer (2005) 257–268
4. Martinez, D.: Résolution interactive de problèmes de satisfaction de contraintes. PhD thesis, Ecole nationale supérieure de l'aéronautique et de l'espace, Toulouse, FRANCE (1998)
5. Sörensson, N., Biere, A.: Minimizing learned clauses. In Kullmann, O., ed.: Proceedings of SAT'09. Volume 5584 of LNCS., Springer (2009) 237–243
6. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solver. In: Proceedings of IJCAI'09. (jul 2009) 399–404
7. Le Berre, D., Parrain, A.: The sat4j library, release 2.2 system description. Journal on Satisfiability, Boolean Modeling and Computation **7** (2010) 59–64
8. Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In: Proceedings of SAT'07. Volume 4501 of LNCS., Springer (2007) 294–299

Appendix

```
==============================
  About Sat4j launcher with on the fly configuration
==============================

The on the fly feature has been developed to allow interactive resolution.
For a specific instance, non-specialists users can have an idea of
what configuration is the most adapted.
This tool is still in development and might not be totally robust.
Please report bugs to daniel.leberre@cril.fr or stephanie.roussel@cril.fr.

==============================
  Running Sat4j with on the fly configuration
==============================

To run sat4j with on the fly configuration:

   java -jar sat4j-sat.jar -remote

These instructions should open a java window named Remote Control.  We
assume that the 1.5 version of the java command is in your path.  If
it isn't, then you should either specify the complete path to the java
command or update your PATH environment variable as described in the
installation instructions for the Java 2 SDK.

In order to visualize the different metrics, gnuplot 4.6 is
required. Gnuplot's version can be checked by running it in a
terminal. On top of that, gnuplot should be compatible with launching
a X11 terminal. To see all available terminals of gnuplot, type "set
terminal" in a gnuplot console.


==============================
  Features
==============================

Features depend on the version of sat4j (this version is displayed at
the top right corner of the remote window). Documentation on solvers and
strategies can be found on http://www.sat4j.org/doc.php

--------------
1. Main window

The main window contains 5 tabs: Solver, Restart, Heuristics, Learned
Constraints and About Solver. It also contains a console where information
```

about configurations and solver runs are displayed. If no solver is running,
only options on tab Solver should be editable.

---

2. Menu bar

The menu bar contains two components : File and Preferences.
In the file menu, it is possible to activate or deactivate tracing. It is also possible to
Activate gnuplot opens a x11 terminal as soon as a solver is running.
Deactivate gnuplot closes all gnuplot windows and does not open any one
when starting a solver.

Tracing preferences can be customized in the Preferences menu.
- background color (black by default)
- foreground color (white by default)
- restart color (dark gray by default)
  allow to change the color restarts are displayed with.

Available gnuplot options are:

- use or not use sliding windows (checked by default)
  if use sliding windows is not checked, it is expected that the
  solver becomes very slow after running a few minutes
- number of lines to display (11000 by default)
  this feature is only available if use sliding windows is checked.
  it is recommended to display at least 2000 lines (it won't be possible
  to see anything otherwise) and not to display more than 25000 lines
  (the solver might become very slow otherwise)
- time before launching gnuplot (8000 by default)
  if gnuplot and the solver are launched exactly at the same time, then
  some of the files gnuplot is plotting might not exist yet.
  Thus, a few seconds should elapse before starting gnuplot. The default
  time is 8000 ms, it is recommended not to go under 3000 ms.
  Waiting more than 10000ms is not useful.
- displays restarts (checked by default)
  if checked, restarts are displayed by a vertical bar on the different graphs

The second set of options selects the graphs that will be displayed when
the solver is running:
- clauses evaluation (checked by default)
- size of learned clauses (checked by default)
- decision level when a conflict occurs (checked by default)
- trail level when a conflict occurs (checked by default)
- index of decision variables (checked by default)
- number of propagations per second (unchecked by default)

the number of propagations per second is highly dependent on the number of processes running at the same time. More precisely, the number of assignments per second might be lower if too many graphs are displayed.
If checked, it is recommended not to display too many other graphs.
- variables evaluation (checked by default)

---

3. Solver Tab

This tab contains two panels : instance and solver.
On the instance panel, the instance of the problem can be specified.
If an instance was given on the command line, then the path to this instance should be displayed. On the solver tab, it is possible to choose a solver among all pre-built solvers of the Sat4j library. If the chosen instance uses pseudo boolean constraints then only pb solvers should be displayed.

If a customized solver has been specified in the command line, then the "Use customize solver" option should be checked. Until unchecked, the running solver is the customized one.

If the instance specifies an objective function, then it is possible to run the solver in optimization mode.

Finally, the solver panel contains 2 buttons:
- the "Start" button runs the solver. When pushed (i.e. when the solver is running), this button becomes a "Stop" button that definitely stops the solver.
- the "Pause" button temporally pauses the solver. When pushed, this button becomes a "Resume" button. Note that pausing the solver does not change its state: it is just frozen until it is resumed.

---

4. Restart Tab

The restart panel allows to change the restart strategy of the solver.
The different strategies available to Sat4j are displayed. Among those strategies, there should be:
- ArminRestarts
- NoRestarts
- LubyStrategy
- MiniSATRestarts

To apply the selected strategy, the restart button must be pushed.

---
5. Heuristics Tab

This tab contains 3 panels.

The first panel is named "Random Walk" and allows to assign the probability
that a decision variable is picked at random, instead of following the heuristics.
The button "Apply" has to be pushed in order to take the probability into account.

The second panel is the "Phase Strategy" panel. Available strategies
are displayed. Among them:
 - NegativeLiteralSelectionStrategy
 - PositiveLiteralSelectionStrategy
 - RSATPhaseSelectionStrategy
 - UserFixedPhaseSelectionStrategy
 - RandomLiteralSelectionStrategy
 - RSATLastLearnedClausesSelectionStrategy
 - PhaseCachingAutoEraseStrategy
 - PhaseInLastLearnedClauseSelectioNStrategy
To apply the selected strategy, click the "Apply" button.

The final panel is the hot solver panel. If checked, the "Keep solver
hot" option prevents the solver to reset its heuristics when a model is
found.

---
6. Learned Constraints Tab

This tab is composed of two panels. First, the "Learned Constraint
Deletion Strategy" panel allows to control the whole deletion
strategy. Every solver is associated with a default deletion
strategy. To keep this strategy, do not uncheck the "Use solver's
original deletion strategy". When unchecked, it is possible to use a
specific strategy by push the button "Apply changes" after :
- choosing on the slider the number of clauses after which a clean will
be made
- choosing the clauses evaluation type (Activity or LBD)

The number of propagations per second is displayed just under the first
checkbox. Note that this number depends on the other processes
that run at the same time and on the parameters chosen to run the
remote.

It is possible to force a clean by using the "Clean now" button.

The second panel deals with the simplification strategy. It is possible to choose between three strategies:
 - no reason
 - simple reason
 - expensive reason

---
7. About solver

This panel details the running solver configuration. Information are displayed only while a solver is running.

==========
   Launching customized solver
==========

When launching the remote in command line, it is possible to specify the use of a specific solver.

The usage is:

```
java -jar sat4j.jar [-C] [-d <filename>] [-f <filename>] [-H] [-k
       <number>] [-l <libname>] [-m] [-opt] [-r] [-remote] [-rw <number>]
       [-s <solvername>] [-S <solverStringDefinition>] [-T <number>] [-t
       <number>] [-y]

 -C,--conflictbased
              conflict based timeout (for
              deterministic behavior)

 -d,--dot <filename>
        creates a sat4j.dot file in current directory
        representing the search

 -f,--filename <filename>
              specifies the file to use (in conjunction with -d for
              instance)


 -H,--hot
              keep the solver hot (do not reset heuristics) when a
              model is found
```

```
 -k,--kleast <number>
             limit the search to models having at least k variables
             set to false

 -l,--library <libname>
             specifies the name of the library used (minisat by
             default)

 -m,--mute
             Set launcher in silent mode

-opt,--optimize
             uses solver in optimize mode instead of sat mode
             (default)

 -r,--trace
             traces the behavior of the solver

 -remote,--remoteControl
             launches remote control

 -rw,--randomWalk <number>
             specifies the random walk probability

 -s,--solver <solvername>
             specifies the name of a prebuilt solver from the library

 -S,--Solver <solverStringDefinition>
             setup a solver using a solver config string
     Example: -S RESTARTS=LubyRestarts/factor:512,LEARNING=MiniSATLearning
      - Available restart strategies (RESTARTS):
                    [ArminRestarts[searchParams], NoRestarts[searchParams],
                   LubyRestarts[searchParams, factor],
                   MiniSATRestarts[searchParams]]
      - Available orders (ORDERS):
                    [VarOrderHeapObjective[variableHeuristics,
                phaseSelectionStrategy, vocabulary],
                PureOrder[variableHeuristics, phaseSelectionStrategy,
                vocabulary, period],
                VarOrderHeap[variableHeuristics, phaseSelectionStrategy,
                vocabulary]]
               - Available learning (LEARNING): [ClauseOnlyLearning,
                     FixedLengthLearning[maxLength], MiniSATLearning,
                NoLearningNoHeuristics, NoLearningButHeuristics,
                ActiveLearning[limit, activityPercent],
```

```
                PercentLengthLearning[limit]]
      - Available phase strategies (PHASE):
                  [NegativeLiteralSelectionStrategy,
                 PositiveLiteralSelectionStrategy,
                 RSATPhaseSelectionStrategy,
                 UserFixedPhaseSelectionStrategy,
                 RandomLiteralSelectionStrategy,
                 RSATLastLearnedClausesPhaseSelectionStrategy,
                 PhaseCachingAutoEraseStrategy,
                 PhaseInLastLearnedClauseSelectionStrategy]
      - Available search params (PARAMS):
                    [SearchParams[claDecay, initConflictBound, varDecay,
                 conflictBoundIncFactor]]
      - Available simplifiers :
             [NO_SIMPLIFICATION, SIMPLE_SIMPLIFICATION, EXPENSIVE_SIMPLIFICATION]
      - Available building blocks:
             DSF, LEARNING, ORDERS, PHASE, RESTARTS, SIMP, PARAMS

-T,--timeoutms <number>
            specifies the timeout (in milliseconds)

-t,--timeout <number>
            specifies the timeout (in seconds)

-y,--simplify
            simplify the set of clauses if possible
```