Monday, October 18th 2010

Getting started with SAT4J

BY NC ND

Foreword

This document is about using SAT4J. This document is currently very incomplete but will be updated as often as possible.

This documentation is under the **CC-BY-NC-ND** *Creative Commons license. That means you are free to copy, distribute and transmit this work under the following conditions :*

- *you must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work),*
- *you may not use this work for commercial purposes,*
- *you may not alter, transform, or build upon this work.*

Contents

Chapter 1

Introduction

The aim of the SAT4J library is to provide an efficient library of SAT solvers in Java. Compared to the OpenSAT project, the SAT4J library targets first users of SAT "black boxes", willing to embed SAT technologies into their application without worrying about the details. Since we use the library for our own research, it is also possible for SAT researchers to use it as a basis for their work.

SAT4J includes our implementation in Java of Niklas Een and Niklas Sorenson's MiniSAT specification: An extensible SAT solver. Niklas Eén and Niklas Sörensson. Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing, LNCS 2919, pp 502-518, 2003

Whereas the overall algorithmic of the solver is respected, the design has been adapted to Java practices. Furthermore, the initial design has been extended to allow testing several heuristics and learning schemes.

Chapter 2

SAT4J as a standalone solver

SAT4J can be used as as standalone tool for solving SAT, CSP and pseudo-boolean problems. MAX-SAT problems can be treated as well. Its use as an embedded library will be treated in chapter 3, page 6.

2.1 SAT4J as a standalone SAT solver

The SAT solver uses input files using the common CNF Dimacs format.

```
java -jar org.sat4j.core.jar cnffile
```

If you want to improve the efficiency of the library, the best thing to do is to increase the amount of memory available to the JVM and to use java hotspot server compiler.

```
java -server -XmsMAXRAM -XmxMAXRAM -jar org.sat4j.core.jar cnffile
```

2.2 SAT4J as a standalone CSP solver

CSP capabilities are available since release 1.5 of the library.

Note that the input format was textual for release 1.5 (using the First CSP competition table format) and is now XML because of the new XML CSP format 2.0 designed for the Second CSP competition.

Note that SAT4J does not contain a real CSP solver: it translates CSP problems given in extension into SAT problems to solve them.

To solve a CSP problem in textual format using the default encoding, use

```
java -jar sat4j-csp.jar cspfile.txt
```

To solve a CSP problem in textual format using the direct encoding, use

```
java -jar sat4j-csp.jar CSP:cspfile.txt
```

To solve a CSP problem in textual format using the binary support encoding and direct encoding for n-ary clauses, use

```
java -jar sat4j-csp.jar CSP2:cspfile.txt
```

To solve a CSP problem in textual format using the generalized support encoding, use

```
java -jar sat4j-csp.jar CSP3:cspfile.txt
```

To solve a CSP problem in XML format using the binary support encoding and direct encoding for n-ary clauses, use

```
java -jar sat4j-csp.jar cspfile.xml
```

2.3 SAT4J as a standalone pseudo-boolean solver

Pseudo Poolean capabilities is available since release 1.5 of the library. Note that the library is keeping up-to-date its input format with the one of the latest PB Evaluation, which can be found at http://www.cril.univ-artois.fr/PB07/.

To use SAT4J with a pseudo-boolean solver, use

```
java java -jar org.sat4j.pb.jar pbfile.opb
```

2.4 SAT4J as a standalone MAXSAT solver

MaxSAT capability is available since release 1.6 of the library. For these problems, use

```
java java -jar sat4j-maxsat.jar file.cnf
```

Weighted Partial MAX SAT problems are supported since release 1.7 of the library. For these problems, use the MAXSAT'06 evaluation format available at http://www.maxsat07.udl.es/ and use the following syntax :

```
java java -jar sat4j-maxsat.jar file.wcnf
```

Chapter 3

SAT4J as a java library

SAT4J can be embedded as a library in any Java program who needs to solve SAT problems. The complete documentation is available at http://sat4j.org/doc.php.

3.1 Embedding a SAT solver in Java code

```
public class Example {

    public static void main(String[] args) {
        ISolver solver = SolverFactory.newDefault();
        solver.setTimeout(3600); // 1 hour timeout
        Reader reader = new DimacsReader(solver);
        // CNF filename is given on the command line
        try {
            IProblem problem = reader.parseInstance(args[0]);
            if (problem.isSatisfiable()) {
                System.out.println("Satisfiable␣!");
                System.out.println(reader.decode(problem.model()));
            } else {
                System.out.println("Unsatisfiable␣!");
            }
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
        } catch (ParseFormatException e) {
            // TODO Auto-generated catch block
        } catch (IOException e) {
            // TODO Auto-generated catch block
        } catch (ContradictionException e) {
            System.out.println("Unsatisfiable␣(trivial)!");
        } catch (TimeoutException e) {
            System.out.println("Timeout,␣sorry!");
        }
    }
}
```

3.2 Feeding a SAT solver without a Reader

```
final int MAXVAR = 1000000;
final int NBCLAUSES = 500000;

ISolver solver = SolverFactory.newDefault();

// prepare the solver to accept MAXVAR variables. MANDATORY
solver.newVar(MAXVAR);
// not mandatory for SAT solving. MANDATORY for MAXSAT solving
solver.setExpectedNumberOfClauses(NBCLAUSES);
// Feed the solver using Dimacs format, using arrays of int
// (best option to avoid dependencies on SAT4J IVecInt)
for (int i=0;i<NBCLAUSES;i++) {
  int [] clause = // get the clause from somewhere
  // the clause should not contain a 0,
  // only integer (positive or negative)
  // with absolute values less or equal to MAXVAR
  // e.g. int [] clause = {1, -3, 7}; is fine
  // while int [] clause = {1, -3, 7, 0}; is not fine
  solver.addClause(new VecInt(clause)); // adapt Array to IVecInt
}

// we are done. Working now on the IProblem interface
IProblem problem = solver;
if (problem.isSatisfiable()) {
   ....
} else {
 ...
}
```

3.3 Iterating over all models

```
ISolver solver = SolverFactory.newDefault();
        ModelIterator mi = new ModelIterator(solver);
        solver.setTimeout(3600); // 1 hour timeout
        Reader reader = new InstanceReader(mi);

        // filename is given on the command line
        try {
            boolean unsat = true;
            IProblem problem = reader.parseInstance(args[0]);
            while (problem.isSatisfiable()) {
               unsat = false;
               int [] model = problem.model();
                // do something with each model
            }
            if (unsat)
                // do something for unsat case
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (ParseFormatException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ContradictionException e) {
            System.out.println("Unsatisfiable␣(trivial)!");
        } catch (TimeoutException e) {
            System.out.println("Timeout,␣sorry!");
        }
```

3.4 Which solver is right for me?

To make things easier for the end user, SAT4J provides in `org.sat4j.minisat.SolverFactory` two convenience methods that provide you a SAT solver depending of your needs:

- `defaultSolver()` is well suited for huge and difficult SAT benchmarks. It is basically the best solver available in the library.
- `lightSolver()` is useful for people using a SAT solver for hundreds or thousands small or easy SAT problems within their application.

3.5 Feeding a SAT solver using logical gates

The class `GateTranslator` is meant to easily feed a SAT solver using logical gates. It can be found in the `org.sat4j.tools` package.

It can be useful for those not wanting to transform logical expressions into the CNF format by themselves.

3.6 Dependency helper

The dependency helper is a helper class intended to make life easier to people to feed a sat solver programmatically. It can be found in `org.sat4j.pb.tools.DependencyHelper<T,C>`.

Chapter 4

Building SAT4J from the source

SAT4J source code can be found on OW2 SVN at the address http://forge.objectweb.org/plugins/scmsvn/index.php?group_id=228.

We provide both an ant and a Maven build file that easily allow any user to build SAT4J from source.

First checkout the source code available on HEAD (development version!) :

```
svn checkout svn://svn.forge.objectweb.org/svnroot/sat4j/maven/trunk
```

As an alternative, you can take the source code of one of the official releases, here 2.0.5 (this is the preferred solution) :

```
svn checkout svn://svn.forge.objectweb.org/svnroot/sat4j/maven/tags/2_0_5
```

To build the library for Java 1.4 and newer, just run the ant command with no argument :

```
ant
```

To build a CSP solver just type:

```
ant csp
```

To get the list of available options, just type:

```
ant -p
```

As for Maven users, they will use the classical command to install the library:

```
mvn install
```