Reduction of finite linear CSPs to SAT using different encodings

Mirko Stojadinović
mirkos@matf.bg.ac.rs

Department of Computer Science
Faculty of Mathematics
University of Belgrade

Fifth Workshop on Formal and Automated Theorem Proving and Applications
Outline

1 Introduction

2 Background

3 System description and experimental results

4 Conclusions and further work
Motivation

- To build a new system that solves Constraint Satisfaction Problems (CSP) and Constraint Optimization Problems (COP) efficiently
- Several tools exist that reduce these problems to SAT, and each is using one of several encodings
- No encoding is suitable for all kinds of problems
- Our system should support different encodings as well as solving by using SMT solvers
Finite Linear CSP

- $V$ is finite set of integer variables
- Comparisons: $a_0x_0 + \ldots + a_{m-1}x_{m-1} \# c$,
  $\# \in \{\leq, <, \geq, >, =, ! =\}$, $x_i \in V$, $a_i, c \in \mathbb{Z}$.
- $B$ is set of Boolean variables
- Clauses are formed as disjunctions of literals where literals are the elements of $B \cup \{\neg p \mid p \in B\} \cup \{\text{comparisons}\}$.
- $S$ is a finite set of clauses (over $V$ and $B$).
Examples

- Scheduling, timetabling, sequencing, routing, rostering, planning.
- Games and puzzles: sudoku, magic square, 8 queens, golomb ruler

Simple example

```
(int x_1 1 2)
(int x_2 1 4)
(int x_3 2 3)
(and (! = x_1 x_2) (< x_3 (+ x_1 x_2)))
```

One of the solutions to this problem is assignment

\[ x_1 = 1, x_2 = 2, x_3 = 2. \]
Reductions of CSP and COP to satisfiability problems

Reduction to SMT

- One approach is solving these problems by reduction to SMT and using SMT solvers (fzn2smt)

Reduction to SAT

- Other approach is reduction to SAT and several tools for this purpose have been made (spec2sat, sugar, URSA, FznTini)
- Each tool uses one of several encodings (direct, support, log, order)
Direct encoding

- For each integer variable $x_i$ and every value $v$ in its domain (i.e., between $l_i$ and $u_i$), a Boolean variable $p_{i,v}$ is created.
- Exactly one of these variables needs to be true, and this is achieved by imposing cardinality constraint
  $$p_{i,l_i} + \ldots + p_{i,u_i} = 1$$
- Example: if $x_1 \in \{3, 4, 5\}$ then variables $p_{1,3}, p_{1,4}, p_{1,5}$ are introduced, and exactly one of this variables has to be true,
  $$p_{1,3} + p_{1,4} + p_{1,5} = 1$$
Support encoding

- The same Boolean variables are introduced as in direct encoding.
- The difference is that direct encoding uses *conflict* clauses and support encoding uses *support* clauses.
- Example: for integer variables $x_1 \in \{3, 4, 5\}$, $x_2 \in \{4, 5, 6\}$
  Boolean variables $p_{1,3}, p_{1,4}, p_{1,5}$ ($p_{1,3} + p_{1,4} + p_{1,5} = 1$) and $p_{2,4}, p_{2,5}, p_{2,6}$ ($p_{2,4} + p_{2,5} + p_{2,6} = 1$) are introduced.
  Constraint $x_1 < x_2$ can be expressed with clauses.

<table>
<thead>
<tr>
<th>Conflict clauses</th>
<th>Support clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg p_{1,4} \lor \neg p_{2,4}$</td>
<td>$\neg p_{1,3} \lor p_{2,4} \lor p_{2,5} \lor p_{2,6}$</td>
</tr>
<tr>
<td>$\neg p_{1,5} \lor \neg p_{2,4}$</td>
<td>$\neg p_{1,4} \lor p_{2,5} \lor p_{2,6}$</td>
</tr>
<tr>
<td>$\neg p_{1,5} \lor \neg p_{2,5}$</td>
<td>$\neg p_{1,5} \lor p_{2,6}$</td>
</tr>
</tbody>
</table>
Log encoding

- Each integer variable is encoded with the same number $n$ of Boolean variables (i.e., bits). Integer variable $x_i$ is represented with $p_{i,0}, \ldots, p_{i,n-1}$ and its value is calculated using $\bigvee_{k=0}^{n-1} 2^k p_{i,k}$.
- For each value $v$ not in the domain of $x_i$ a constraint that forbids $x_i = v$ is imposed.
- Example: integer variable $x_1 \in \{1, 2\}$ can be represented with two Boolean variables, $p_{1,0}$ and $p_{1,1}$. Clause forbiding $x_1 = 0$ is $(p_{1,0} \oplus 0) \lor (p_{1,1} \oplus 0)$ and clause forbiding $x_1 = 3$ is $(p_{1,0} \oplus 1) \lor (p_{1,1} \oplus 1)$. 

Mirko Stojadinović mirkos@matf.bg.ac.rs

Reduction of finite linear CSPs to SAT using different encodings
Order encoding

- Integer variable $x_i$ with the domain between $l_i$ and $u_i$ is represented with Boolean variables $p_{i,l_i}, \ldots, p_{i,u_i}$, where $p_{i,v}$ represents that $x_i \leq v$ ($p_{i,u_i}$ is always true).

- For every $v \in \{l_{i+1}, \ldots, u_i\}$: $\neg p_{i,v-1} \lor p_{i,v}$ (if $x_i \leq v - 1$ then $x_i \leq v$).

- Example: if $x_1 \in \{3, 4, 5\}$ then variables $p_{1,3}, p_{1,4}, p_{1,5}$ are introduced, and following clauses are generated: $\neg p_{1,3} \lor p_{1,4}$ and $\neg p_{1,4} \lor p_{1,5}$.
System description

- System is called meSAT (Multiple Encodings to SAT) and is implemented in C++
- meSAT supports a subset of the input syntax of sugar, system that uses order encoding
- This syntax was selected since it is rather low-level and many benchmark instances can be translated to sugar syntax
- Two ways are used for solving CSP and COP: reduction to SAT and to SMT. Either only the output DIMACS or SMT-LIB file can be generated or solution can be obtained by calling SAT/SMT solver.
Experimental results

<table>
<thead>
<tr>
<th>Problem</th>
<th>#</th>
<th>Direct</th>
<th>Support</th>
<th>Log</th>
<th>Order</th>
<th>SMT</th>
<th>Sugar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph coloring</td>
<td>68</td>
<td>178.24 (52)</td>
<td>308.27 (40)</td>
<td>215.01 (49)</td>
<td>179.09 (51)</td>
<td>&gt; 500 (5)</td>
<td>164.42 (54)</td>
</tr>
<tr>
<td>Queens</td>
<td>9</td>
<td>15.51 (8)</td>
<td>15.51 (8)</td>
<td>44.3 (5)</td>
<td>31.47 (6)</td>
<td>45.59 (5)</td>
<td>30.41 (6)</td>
</tr>
<tr>
<td>Golomb ruler</td>
<td>17</td>
<td>52.07 (12)</td>
<td>62.38 (11)</td>
<td>80.17 (10)</td>
<td>53.49 (13)</td>
<td>86.17 (9)</td>
<td>40.37 (14)</td>
</tr>
<tr>
<td>Magic square</td>
<td>11</td>
<td>58.73 (6)</td>
<td>58.73 (6)</td>
<td>60.4 (5)</td>
<td>22.31 (9)</td>
<td>90 (2)</td>
<td>18.09 (10)</td>
</tr>
<tr>
<td>Knight’s tour</td>
<td>6</td>
<td>51.38 (1)</td>
<td>21.2 (4)</td>
<td>54.43 (1)</td>
<td>31.1 (3)</td>
<td>13.06 (5)</td>
<td>22.54 (4)</td>
</tr>
<tr>
<td>Sudoku</td>
<td>40</td>
<td>19.82 (40)</td>
<td>19.82 (40)</td>
<td>400 (0)</td>
<td>26.36 (40)</td>
<td>384 (4)</td>
<td>21.6 (40)</td>
</tr>
</tbody>
</table>

Results are compared to *sugar*. Different encodings perform the best on different problems.
Conclusions and further work

Conclusions

- There is no single encoding suitable for all kinds of problems
- One could benefit significantly from trying different encodings and solvers.

Further work

- Parallel solving using different encodings on multiprocessor machine
- A portfolio approach that would try to choose the best among several available encodings based only on some characteristics of the given instance.
- Solving a problem by using different encodings for different constraints