Automatically Reformulating SAT-Encoded CSPs

Lyndon Drake, Alan Frisch, Ian Gent, and Toby Walsh

lyndon@cs.york.ac.uk

AI Group, Computer Science Department,
University of York, United Kingdom

Introduction

- It is possible to solve a number of problems efficiently by mapping them to SAT
- Mapping to SAT flattens some structural information
- The choice of encoding can greatly affect SAT solver performance
- Sometimes it is possible to infer extra clauses that enhance an encoding

Outline

- Encoding CSPs as SAT instances
  - Variables
  - Domains
  - Constraints
- Hyper binary resolution
- Automatically inferring support clauses
- Results and future work

CSPs

- Binary constraints
- Example CSP:
  - Three variables: a, b, and c
  - All three variables have the same domain, $d = \{1, 2, 3\}$
  - Three constraints: $a < b$, $b < c$, and $c < a$

Propositional satisfiability

- Conjunctive normal form:

$$a \vee \neg b$$
$$b \vee c$$
$$\neg c \vee d$$

Encoding into SAT

Three steps:

- CSP variables and values to SAT variables
- CSP domains to SAT clauses
- CSP constraints to SAT clauses

Step 1: variables to variables

- SAT variable a_v is true when CSP variable a takes the value v
- $n \times d$ SAT variables, where n is the number of variables and d is the domain size

Step 2: domains to clauses

- **at-least-one** clauses

$$(a_1 \vee a_2 \vee a_3)$$

- **at-most-one** clauses

$$\neg a_1 \vee \neg a_2$$

$$\neg a_2 \vee \neg a_3$$

$$\neg a_1 \vee \neg a_3$$

Step 3: constraints to clauses – direct

- $a < b$

$$\neg a_1 \vee \neg b_1$$
$$\neg a_2 \vee \neg b_2$$
$$\neg a_2 \vee \neg b_1$$
$$\neg a_3 \vee \neg b_3$$
$$\neg a_3 \vee \neg b_2$$
$$\neg a_3 \vee \neg b_1$$

Step 3: constraints to clauses – support

- $a < b$

$$\neg a_3$$
$$\neg a_2 \vee b_3$$
$$\neg a_1 \vee b_2 \vee b_3$$
$$\neg b_1$$
$$\neg b_2 \vee a_1$$
$$\neg b_3 \vee a_2 \vee a_1$$

Comparing the two encodings

- Neither is particularly complicated
- Both require polynomial space
- SAT solvers generally perform better on instances encoded using support

Resolution

- Given the clauses:

$$x_1 \vee x_2$$

$$x_1 \vee \neg x_2 \vee \neg x_3$$

- we can infer the clause:

$$x_1 \vee \neg x_3$$

HypBinRes (Bacchus 2002)

- Given the clauses:

$$x_1 \vee x_2 \vee \ldots \vee x_n$$
$$\neg x_1 \vee h$$
$$\neg x_2 \vee h$$
$$\ldots$$
$$\neg x_{n-1} \vee h$$

- we can infer the clause:

$$x_n \vee h$$

Inferring some support clauses

- Take the at-least-one clause for a:

$$a_1 \vee a_2 \vee a_3$$

- add the conflict clauses $b = 2$:

$$\neg b_2 \vee \neg a_2$$

$$\neg b_2 \vee \neg a_3$$

- and infer the support clause for $b = 2$:

$$\neg b_2 \vee a_1$$

Inferring more support clauses

- Only clauses for functional constraints will be inferred by HypBinRes
- Applying HypBinRes during search infers the “useful” support clauses
- Relaxing the conditions on the HypBinRes rule allows us to infer all the support clauses

Results

- Experiments carried out using 2CLS+EQ
  - Solver written by Fahiem Bacchus
  - Efficient implementation of HypBinRes during DPLL search
- HypBinRes during search significantly outperforms HypBinRes as a preprocessor
  - Inferring support clauses for the non-functional constraints is worthwhile
- 2CLS+EQ on the direct encoding outperforms 2CLS+EQ on the support encoding

Questions in search of answers

- Does the generalised HypBinRes rule infer useful clauses on other problems?
- Are the conflict clauses useful when all the support clauses exist?
- Is HypBinRes useful as a preprocessor?
- Can we explain the relationship between different SAT encodings of other problems in terms of inference?

Conclusions