Solving Constraint Satisfaction Problems by a SAT Solver

Naoyuki Tamura, Tomoya Tanjo, and Mutsunori Banbara

Kobe University, JAPAN

CICLOPS-WLPE-2010

Contents

Contents

Contents

SAT problems and SAT solvers

SAT problems

SAT

SAT (Boolean satisfiability testing) is a problem to decide whether a given Boolean formula has any satisfying truth assignment.

- SAT is a central problem in Computer Science both theoretically and practically.
- SAT was the first NP-complete problem [Cook 1971].
- SAT has very efficient implementation (MiniSat, etc.)
- SAT-based approach is becoming popular in many areas.

SAT instances

SAT instances are given in the conjunctive normal form (CNF).

CNF formula

- A CNF formula is a conjunction of clauses.
- A clause is a disjunction of literals.
- A literal is either a Boolean variable or its negation.

DIMACS CNF is used as the standard format for CNF files.

p cnf 9 7	; Number of variables and clauses
1 2 0	; $a \vee b$
9 3 0	; $c \vee d$
1 8 4 0	; $a \vee e \vee f$
-2 -4 5 0	; $\neg b \vee \neg f \vee g$
-4 6 0	; $\neg f \vee h$
-2 -6 7 0	; $\neg b \vee \neg h \vee i$
-5 -7 0	; $\neg g \vee \neg i$

SAT solvers

SAT Solver

SAT solver is a program to decide whether a given SAT instance is satisfiable (`SAT`) or unsatisfiable (`UNSAT`).

Usually, it also returns a truth assignment as a solution when the instance is SAT.

- Systematic (complete) SAT solver answers `SAT` or `UNSAT`.
  - Most of them are based on the DPLL algorithm.
- Stochastic (incomplete) SAT solver only answers `SAT` (no answers for `UNSAT`).
  - Local search algorithms are used.

DPLL Algorithm

[Davis & Putnam 1960], [Davis, Logemann & Loveland 1962]

function DPLL(S: a CNF formula, σ: a variable assignment)
  $\sigma := \mathrm{UP}(S, \sigma)$; /* unit propagation */
  if S is satisfied by σ then return true;
  if S is falsified by σ then return false;
  choose an unassigned variable x from $S\sigma$;
  return DPLL(S, $\sigma \cup \{x \mapsto 0\}$) or DPLL($S$, $\sigma \cup \{x \mapsto 1\}$);

function UP(S: a CNF formula, σ: a variable assignment)
  while $S\sigma$ contains a unit clause $\{l\}$ do
    if l is positive then $\sigma := \sigma \cup \{l \mapsto 1\}$;
    else $\sigma := \sigma \cup \{\bar{l} \mapsto 0\}$;
  return σ;

- $S\sigma$ represents a CNF formula obtained by applying σ to S.

DPLL

1. Choose a and decide $a \mapsto 0$

$$
\begin{gathered}
C_1 : a \lor b \\
C_2 : c \lor d \\
C_3 : a \lor e \lor f \\
C_4 : \neg b \lor \neg f \lor g \\
C_5 : \neg f \lor h \\
C_6 : \neg b \lor \neg h \lor i \\
C_7 : \neg g \lor \neg i
\end{gathered}
$$

DPLL

1. Choose a and decide $a \mapsto 0$
   - Propagate $b \mapsto 1$ from C_1

$$
\begin{gathered}
C_1 : a \vee b \\
C_2 : c \vee d \\
C_3 : a \vee e \vee f \\
C_4 : \neg b \vee \neg f \vee g \\
C_5 : \neg f \vee h \\
C_6 : \neg b \vee \neg h \vee i \\
C_7 : \neg g \vee \neg i
\end{gathered}
$$

DPLL

$$C_1 : a \vee b$$
$$C_2 : c \vee d$$
$$C_3 : a \vee e \vee f$$
$$C_4 : \neg b \vee \neg f \vee g$$
$$C_5 : \neg f \vee h$$
$$C_6 : \neg b \vee \neg h \vee i$$
$$C_7 : \neg g \vee \neg i$$

1. Choose a and decide $a \mapsto 0$
   - Propagate $b \mapsto 1$ from C_1
2. Choose c and decide $c \mapsto 0$

DPLL

$C_1 : a \vee b$
$C_2 : c \vee d$
$C_3 : a \vee e \vee f$
$C_4 : \neg b \vee \neg f \vee g$
$C_5 : \neg f \vee h$
$C_6 : \neg b \vee \neg h \vee i$
$C_7 : \neg g \vee \neg i$

1. Choose a and decide $a \mapsto 0$
   - Propagate $b \mapsto 1$ from C_1
2. Choose c and decide $c \mapsto 0$
   - Propagate $d \mapsto 1$ from C_2

DPLL

$C_1 : a \vee b$

$C_2 : c \vee d$

$C_3 : a \vee e \vee f$

$C_4 : \neg b \vee \neg f \vee g$

$C_5 : \neg f \vee h$

$C_6 : \neg b \vee \neg h \vee i$

$C_7 : \neg g \vee \neg i$

1. Choose a and decide $a \mapsto 0$
   - Propagate $b \mapsto 1$ from C_1
2. Choose c and decide $c \mapsto 0$
   - Propagate $d \mapsto 1$ from C_2
3. Choose e and decide $e \mapsto 0$

DPLL

$$C_1 : a \lor b$$
$$C_2 : c \lor d$$
$$C_3 : a \lor e \lor f$$
$$C_4 : \neg b \lor \neg f \lor g$$
$$C_5 : \neg f \lor h$$
$$C_6 : \neg b \lor \neg h \lor i$$
$$C_7 : \neg g \lor \neg i$$

1. Choose a and decide $a \mapsto 0$
   - Propagate $b \mapsto 1$ from C_1
2. Choose c and decide $c \mapsto 0$
   - Propagate $d \mapsto 1$ from C_2
3. Choose e and decide $e \mapsto 0$
   - Propagate $f \mapsto 1$ from C_3

DPLL

$$C_1 : a \vee b$$
$$C_2 : c \vee d$$
$$C_3 : a \vee e \vee f$$
$$C_4 : \neg b \vee \neg f \vee g$$
$$C_5 : \neg f \vee h$$
$$C_6 : \neg b \vee \neg h \vee i$$
$$C_7 : \neg g \vee \neg i$$

1. Choose a and decide $a \mapsto 0$
   - Propagate $b \mapsto 1$ from C_1
2. Choose c and decide $c \mapsto 0$
   - Propagate $d \mapsto 1$ from C_2
3. Choose e and decide $e \mapsto 0$
   - Propagate $f \mapsto 1$ from C_3
   - Propagate $g \mapsto 1$ from C_4

DPLL

$$C_1 : a \lor b$$
$$C_2 : c \lor d$$
$$C_3 : a \lor e \lor f$$
$$C_4 : \neg b \lor \neg f \lor g$$
$$C_5 : \neg f \lor h$$
$$C_6 : \neg b \lor \neg h \lor i$$
$$C_7 : \neg g \lor \neg i$$

1. Choose a and decide $a \mapsto 0$
   - Propagate $b \mapsto 1$ from C_1
2. Choose c and decide $c \mapsto 0$
   - Propagate $d \mapsto 1$ from C_2
3. Choose e and decide $e \mapsto 0$
   - Propagate $f \mapsto 1$ from C_3
   - Propagate $g \mapsto 1$ from C_4
   - Propagate $i \mapsto 0$ from C_7

DPLL

$C_1 : a \vee b$
$C_2 : c \vee d$
$C_3 : a \vee e \vee f$
$C_4 : \neg b \vee \neg f \vee g$
$C_5 : \neg f \vee h$
$C_6 : \neg b \vee \neg h \vee i$
$C_7 : \neg g \vee \neg i$

1. Choose a and decide $a \mapsto 0$
   - Propagate $b \mapsto 1$ from C_1
2. Choose c and decide $c \mapsto 0$
   - Propagate $d \mapsto 1$ from C_2
3. Choose e and decide $e \mapsto 0$
   - Propagate $f \mapsto 1$ from C_3
   - Propagate $g \mapsto 1$ from C_4
   - Propagate $i \mapsto 0$ from C_7
   - Propagate $h \mapsto 1$ from C_5

DPLL

$C_1 : a \lor b$
$C_2 : c \lor d$
$C_3 : a \lor e \lor f$
$C_4 : \neg b \lor \neg f \lor g$
$C_5 : \neg f \lor h$
$C_6 : \neg b \lor \neg h \lor i$
$C_7 : \neg g \lor \neg i$

1. Choose a and decide $a \mapsto 0$
   - Propagate $b \mapsto 1$ from C_1
2. Choose c and decide $c \mapsto 0$
   - Propagate $d \mapsto 1$ from C_2
3. Choose e and decide $e \mapsto 0$
   - Propagate $f \mapsto 1$ from C_3
   - Propagate $g \mapsto 1$ from C_4
   - Propagate $i \mapsto 0$ from C_7
   - Propagate $h \mapsto 1$ from C_5
   - Conflict occurred at C_6

DPLL

$$C_1 : a \lor b$$
$$C_2 : c \lor d$$
$$C_3 : a \lor e \lor f$$
$$C_4 : \neg b \lor \neg f \lor g$$
$$C_5 : \neg f \lor h$$
$$C_6 : \neg b \lor \neg h \lor i$$
$$C_7 : \neg g \lor \neg i$$

1. Choose a and decide $a \mapsto 0$
   - Propagate $b \mapsto 1$ from C_1
2. Choose c and decide $c \mapsto 0$
   - Propagate $d \mapsto 1$ from C_2
3. Choose e and decide $e \mapsto 0$
   - Propagate $f \mapsto 1$ from C_3
   - Propagate $g \mapsto 1$ from C_4
   - Propagate $i \mapsto 0$ from C_7
   - Propagate $h \mapsto 1$ from C_5
   - Conflict occurred at C_6
4. Backtrack and decide $e \mapsto 1$

Modern SAT solvers

- The following techniques have been introduced to DPLL and they drastically improved the performance of modern SAT solvers.
  - **CDCL** (Conflict Driven Clause Learning) [Silva 1996]
  - Non-chronological Backtracking [Silva 1996]
  - Random Restarts [Gomes 1998]
  - Watched Literals [Moskewicz & Zhang 2001]
  - Variable Selection Heuristics [Moskewicz & Zhang 2001]
- Chaff and zChaff solvers made one to two orders magnitude improvement [2001].
- SAT competitions and SAT races since 2002 contribute to the progress of SAT solver implementation techniques.
- MiniSat solver showed its good performance in the 2005 SAT competition with about 2000 lines of code in C++.
- Modern SAT solvers can handle instances with more than 10^6 variables and 10^7 clauses.

CDCL (Conflict Driven Clause Learning)

- At conflict, a reason of the conflict is extracted as a clause and it is remembered as a learnt clause.
- Learnt clauses significantly prunes the search space in the further search.
- Learnt clause is generated by resolution in backward direction.
- The resolution is stopped at First UIP (Unique Implication Point) [Moskewicz & Zhang 2001].

In the previous example, $\neg b \vee \neg f$ is generated as a learnt clause.

$$\dfrac{\dfrac{\dfrac{C_6 : \neg b \vee \neg h \vee i \quad C_5 : \neg f \vee h}{\neg b \vee \neg f \vee i} \quad C_7 : \neg g \vee \neg i}{\neg b \vee \neg f \vee \neg g} \quad C_4 : \neg b \vee \neg f \vee g}{\neg b \vee \neg f}$$

SAT-based Approach

SAT-based approach is becoming popular for solving hard combinatorial problems.

- Planning (SATPLAN, Blackbox) [Kautz & Selman 1992]
- Automatic Test Pattern Generation [Larrabee 1992]
- Job-shop Scheduling [Crawford & Baker 1994]
- Software Specification (Alloy) [1998]
- Bounded Model Checking [Biere 1999]
- Software Package Dependency Analysis (SATURN)
  - SAT4J is used in Eclipse 3.4.
- Rewriting Systems (AProVE, Jambox)
- Answer Set Programming (clasp, Cmodels-2)
- FOL Theorem Prover (iProver, Darwin)
- First Order Model Finder (Paradox)
- Constraint Satisfaction Problems (Sugar) [Tamura et al. 2006]

Why SAT-based? (personal opinions)

SAT solvers are very fast.

- Clever implementation techniques, such as two literal watching.
  - It minimizes house-keeping informations for backtracking.
- Cache-aware implementation [Zhang & Malik 2003]
  - For example, a SAT-encoded Open-shop Scheduling problem instance gp10−10 is solved within 4 seconds with more than 99% cache hit rate by MiniSat.

```
$ valgrind --tool=cachegrind minisat gp10-10-1091.cnf
L2 refs:      42,842,531  ( 31,633,380 rd +11,209,151 wr)
L2 misses:    25,674,308  ( 19,729,255 rd + 5,945,053 wr)
L2 miss rate:        0.4% (        0.4%   +       1.0%  )
```

Why SAT-based? (personal opinions)

SAT-based approach is similar to RISC approach in '80s by Patterson.

- **RISC**: Reduced Instruction Set Computer
- Patterson claimed a computer of a "reduced" and fast instruction set with an efficient optimizing compiler can be faster than a "complex" computer (**CISC**).

SAT Solver	$\Longleftrightarrow$	**RISC**
SAT Encoder	$\Longleftrightarrow$	**Optimizing Compiler**

- In that sense, study of both SAT solvers and SAT encodings are important and interesting topics.

SAT encodings
of
Constraint Satisfaction Problems

Finite linear CSP

Finite linear CSP

- **Integer variables** with finite domains
  - $\ell(x)$: the lower bound of x
  - $u(x)$: the upper bound of x
- **Boolean variables**
- **Arithmetic operators**: $+$, $-$, constant multiplication, etc.
- **Comparison operators**: $=$, $\neq$, $\geq$, $>$, $\leq$, $<$
- **Logical operators**: $\neg$, $\wedge$, $\vee$, $\Rightarrow$

- We can restrict the comparison to $\sum a_i x_i \leq c$ without loss of generality where x_i's are integer variables and a_i's and c are integer constants.
- We also use the followings in further descriptions.
  - n : number of integer variables
  - d : maximum domain size of integer expressions

SAT encodings

There have been several methods proposed to encode CSP into SAT.

- *Direct encoding* is the most widely used one [de Kleer 1989].
- Order encoding is a new encoding showing a good performance for a wide variety of problems [Tamura et al. 2006].
  - It is first used to encode job-shop scheduling problems by [Crawford & Baker 1994].
  - It succeeded to solve previously undecided problems in open-shop scheduling, job-shop scheduling, and two-dimensional strip packing.
- Other encodings:
  - *Multivalued encoding* [Selman 1992]
  - *Support encoding* [Kasif 1990]
  - *Log encoding* [Iwama 1994]
  - *Log-support encoding* [Gavanelli 2007]

Direct encoding

In direct encoding [de Kleer 1989], a Boolean variable $p(x = i)$ is defined as true iff the integer variable x has the domain value i, that is, $x = i$.

Boolean variables for each integer variable x

$$p(x = i) \qquad (\ell(x) \leq i \leq u(x))$$

For example, the following five Boolean variables are used to encode an integer variable $x \in \{2, 3, 4, 5, 6\}$,

5 Boolean variables for $x \in \{2, 3, 4, 5, 6\}$

$$p(x = 2) \quad p(x = 3) \quad p(x = 4) \quad p(x = 5) \quad p(x = 6)$$

Direct encoding (cont.)

The following at-least-one and at-most-one clauses are required to make $p(x = i)$ be true iff $x = i$.

Clauses for each integer variable x

$$p(x = \ell(x)) \lor \cdots \lor p(x = u(x))$$
$$\neg p(x = i) \lor \neg p(x = j) \qquad (\ell(x) \le i < j \le u(x))$$

For example, 11 clauses are required for $x \in \{2, 3, 4, 5, 6\}$.

11 clauses for $x \in \{2, 3, 4, 5, 6\}$

$$p(x = 2) \lor p(x = 3) \lor p(x = 4) \lor p(x = 5) \lor p(x = 6)$$
$$\neg p(x = 2) \lor \neg p(x = 3) \qquad \neg p(x = 2) \lor \neg p(x = 4) \qquad \neg p(x = 2) \lor \neg p(x = 5)$$
$$\neg p(x = 2) \lor \neg p(x = 6) \qquad \neg p(x = 3) \lor \neg p(x = 4) \qquad \neg p(x = 3) \lor \neg p(x = 5)$$
$$\neg p(x = 3) \lor \neg p(x = 6) \qquad \neg p(x = 4) \lor \neg p(x = 5)$$
$$\neg p(x = 4) \lor \neg p(x = 6) \qquad \neg p(x = 5) \lor \neg p(x = 6)$$

Direct encoding (cont.)

A constraint is encoded by enumerating its conflict points.

Constraint clauses

- When $x_1 = i_1, \ldots, x_k = i_k$ violates the constraint, the following clause is added.

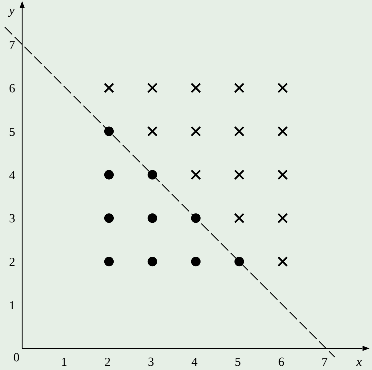
$$\neg p(x_1 = i_1) \vee \cdots \vee \neg p(x_k = i_k)$$

Direct encoding (cont.)

A constraint $x + y \le 7$ is encoded into the following 15 clauses by enumerating conflict points (crossed points).

15 clauses for $x + y \le 7$

$$
\begin{array}{l}
\neg p(x=2) \lor \neg p(y=6) \\
\neg p(x=3) \lor \neg p(y=5) \\
\neg p(x=3) \lor \neg p(y=6) \\
\neg p(x=4) \lor \neg p(y=4) \\
\neg p(x=4) \lor \neg p(y=5) \\
\neg p(x=4) \lor \neg p(y=6) \\
\neg p(x=5) \lor \neg p(y=3) \\
\neg p(x=5) \lor \neg p(y=4) \\
\neg p(x=5) \lor \neg p(y=5) \\
\neg p(x=5) \lor \neg p(y=6) \\
\neg p(x=6) \lor \neg p(y=2) \\
\neg p(x=6) \lor \neg p(y=3) \\
\neg p(x=6) \lor \neg p(y=4) \\
\neg p(x=6) \lor \neg p(y=5) \\
\neg p(x=6) \lor \neg p(y=6)
\end{array}
$$



Order encoding

In order encoding [Tamura et al. 2006], a Boolean variable $p(x \le i)$ is defined as true iff the integer variable x is less than or equal to the domain value i, that is, $x \le i$.

Boolean variables for each integer variable x

$$p(x \le i) \qquad (\ell(x) \le i < u(x))$$

For example, the following four Boolean variables are used to encode an integer variable $x \in \{2, 3, 4, 5, 6\}$,

4 Boolean variables for $x \in \{2, 3, 4, 5, 6\}$

$$p(x \le 2) \quad p(x \le 3) \quad p(x \le 4) \quad p(x \le 5)$$

Boolean variable $p(x \le 6)$ is unnecessary since $x \le 6$ is always true.

Order encoding (cont.)

The following clauses are required to make $p(x \le i)$ be true iff $x \le i$.

Clauses for each integer variable x

$$\neg p(x \le i-1) \lor p(x \le i) \quad (\ell(x) < i < u(x))$$

For example, 3 clauses are required for $x \in \{2,3,4,5,6\}$.

3 clauses for $x \in \{2,3,4,5,6\}$

$$\neg p(x \le 2) \lor p(x \le 3)$$
$$\neg p(x \le 3) \lor p(x \le 4)$$
$$\neg p(x \le 4) \lor p(x \le 5)$$

Order encoding (cont.)

The following table shows possible satisfiable assignments for the given clauses.

$$\neg p(x \le 2) \lor p(x \le 3)$$
$$\neg p(x \le 3) \lor p(x \le 4)$$
$$\neg p(x \le 4) \lor p(x \le 5)$$

Satisifiable assignments

$p(x \le 2)$	$p(x \le 3)$	$p(x \le 4)$	$p(x \le 5)$	Intepretation
1	1	1	1	$x = 2$
0	1	1	1	$x = 3$
0	0	1	1	$x = 4$
0	0	0	1	$x = 5$
0	0	0	0	$x = 6$

Order encoding (cont.)

Satisfiable partial assignments

$p(x \le 2)$	$p(x \le 3)$	$p(x \le 4)$	$p(x \le 5)$	Intepretation
–	–	–	–	$x = 2..6$
–	–	–	1	$x = 2..5$
–	–	1	1	$x = 2..4$
–	1	1	1	$x = 2..3$
0	–	–	–	$x = 3..6$
0	0	–	–	$x = 4..6$
0	0	0	–	$x = 5..6$
0	–	–	1	$x = 3..5$
0	–	1	1	$x = 3..4$
0	0	–	1	$x = 4..5$

"–" means undefined.

- Partial assignments on Boolean variables represent bounds of integer variables.

Order encoding (cont.)

A constraint is encoded by enumerating its conflict regions instead of conflict points.

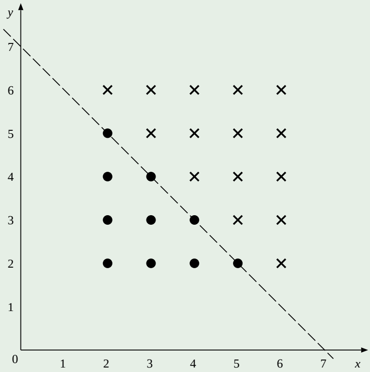
Constraint clauses

- When all points $(x_1, \ldots, x_k)$ in the region $i_1 < x_1 \leq j_1$, $\ldots$, $i_k < x_k \leq j_k$ violate the constraint, the following clause is added.

$$p(x_1 \leq i_1) \lor \neg p(x_1 \leq j_1) \lor \cdots \lor p(x_k \leq i_k) \lor \neg p(x_k \leq j_k)$$

Order encoding (cont.)

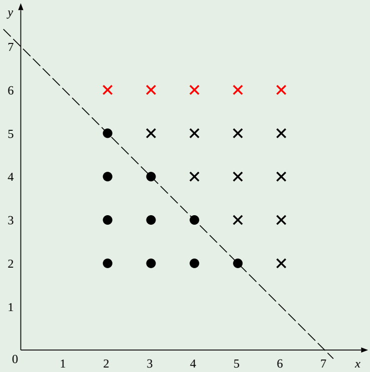
Encoding a constraint $x + y \leq 7$



Order encoding (cont.)

Encoding a constraint $x + y \le 7$

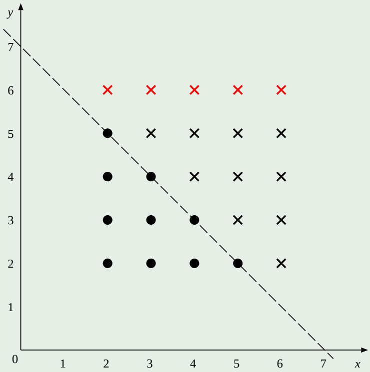
$\neg(y \ge 6)$



Order encoding (cont.)

Encoding a constraint $x + y \leq 7$

$p(y \leq 5)$

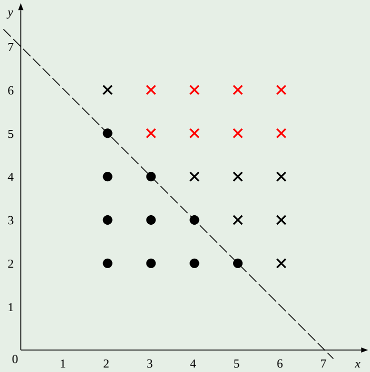


Order encoding (cont.)

Encoding a constraint $x + y \leq 7$

$p(y \leq 5)$

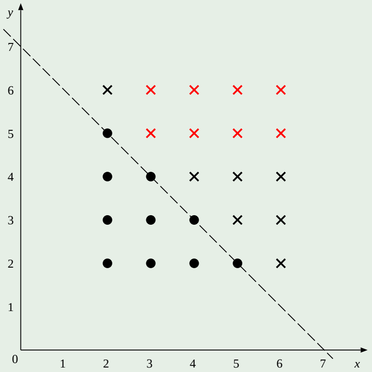
$\neg(x \geq 3 \wedge y \geq 5)$



Order encoding (cont.)

Encoding a constraint $x + y \leq 7$

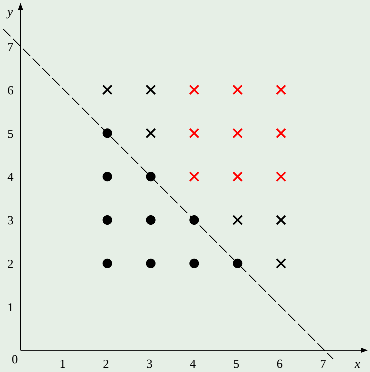
$p(y \leq 5)$
$p(x \leq 2) \vee p(y \leq 4)$



Order encoding (cont.)

Encoding a constraint $x + y \leq 7$

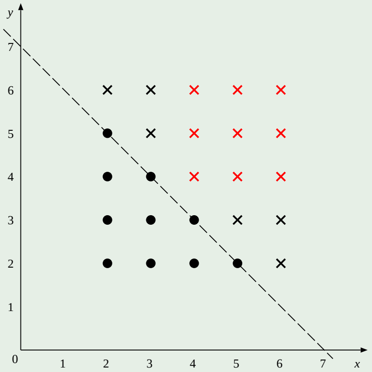
$p(y \leq 5)$
$p(x \leq 2) \vee p(y \leq 4)$
$\neg(x \geq 4 \wedge y \geq 4)$



Order encoding (cont.)

Encoding a constraint $x + y \le 7$

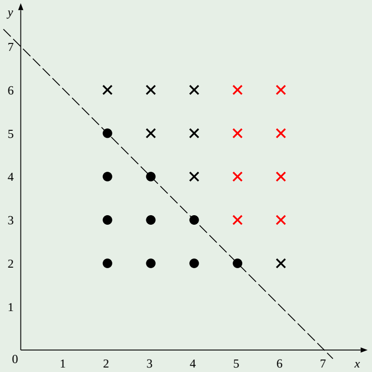
$p(y \le 5)$
$p(x \le 2) \vee p(y \le 4)$
$p(x \le 3) \vee p(y \le 3)$



Order encoding (cont.)

Encoding a constraint $x + y \leq 7$

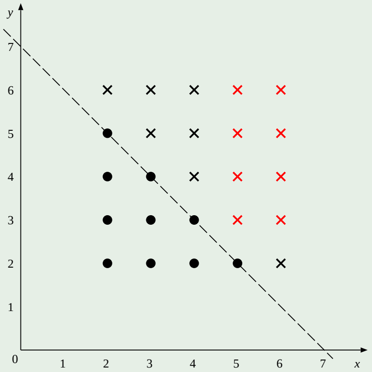
$p(y \leq 5)$
$p(x \leq 2) \vee p(y \leq 4)$
$p(x \leq 3) \vee p(y \leq 3)$
$\neg(x \geq 5 \wedge y \geq 3)$



Order encoding (cont.)

Encoding a constraint $x + y \leq 7$

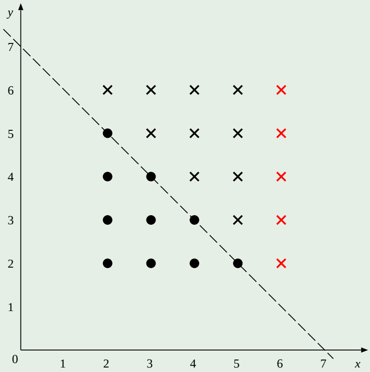
$p(y \leq 5)$
$p(x \leq 2) \vee p(y \leq 4)$
$p(x \leq 3) \vee p(y \leq 3)$
$p(x \leq 4) \vee p(y \leq 2)$



Order encoding (cont.)

Encoding a constraint $x + y \le 7$

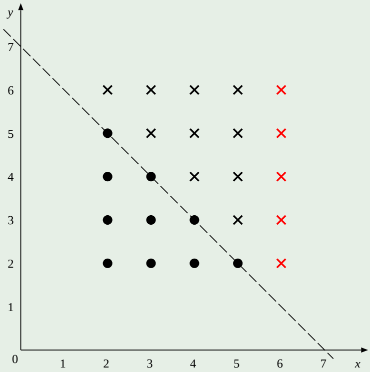
$p(y \le 5)$
$p(x \le 2) \lor p(y \le 4)$
$p(x \le 3) \lor p(y \le 3)$
$p(x \le 4) \lor p(y \le 2)$
$\neg(x \ge 6)$



Order encoding (cont.)

Encoding a constraint $x + y \le 7$

$p(y \le 5)$
$p(x \le 2) \lor p(y \le 4)$
$p(x \le 3) \lor p(y \le 3)$
$p(x \le 4) \lor p(y \le 2)$
$p(x \le 5)$



Bound propagation in order encoding

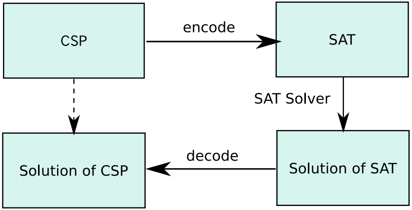
Encoding a constraint $x + y \le 7$

$$
\begin{aligned}
C_1 &: & p(y \le 5) \\
C_2 &: & p(x \le 2) \lor p(y \le 4) \\
C_3 &: & p(x \le 3) \lor p(y \le 3) \\
C_4 &: & p(x \le 4) \lor p(y \le 2) \\
C_5 &: & p(x \le 5)
\end{aligned}
$$

- When $p(x \le 3)$ becomes false (i.e. $x \ge 4$), $p(y \le 3)$ becomes true (i.e. $y \le 3$) by unit propagation on C_3.
- This corresponds to the bound propagation in CSP solvers.

A SAT-based Constraint Solver Sugar

Sugar: a SAT-based Constraint Solver



- **Sugar** is a constraint solver based on the **order encoding**.
- In the **2008 CSP solver competition**, Sugar became the winner in GLOBAL category.
- In the **2008 Max-CSP solver competition**, Sugar became the winner in three categories of INTENSIONAL and GLOBAL constraints.
- In the **2009 CSP solver competition**, Sugar became the winner in three categories of GLOBAL constraints.

Components of Sugar

- **Java program**
  - Parser
  - Linearizer
  - Simplifier of eliminating variable domains by General Arc Consistency algorithm
  - Encoder based on the order encoding
  - Decoder
- **External SAT solver**
  - MiniSat (default), PicoSAT, and any other SAT solvers
- **Perl script**
  - Command line script

Translation of constraints

- Linear constraints are translated by the order encoding.
- Non-linear constraints are translated in linear forms as follows:

Expression	Conversion
$E = F$	$(E \le F) \land (E \ge F)$
$E \ne F$	$(E < F) \lor (E > F)$
$\max(E, F)$	x
	with $(x \ge E) \land (x \ge F) \land ((x \le E) \lor (x \le F))$
$\min(E, F)$	x
	with $(x \le E) \land (x \le F) \land ((x \ge E) \lor (x \ge F))$
$\mathrm{abs}(E)$	$\max(E, -E)$
$E \text{ div } c$	q
	with $(E = c\,q + r) \land (0 \le r) \land (r < c)$
$E \bmod c$	r
	with $(E = c\,q + r) \land (0 \le r) \land (r < c)$

Translation of global constraints

- alldifferent$(x_1, x_2, \ldots, x_n)$ constraint is translated as follows:

$$\bigwedge_{i<j} (x_i \neq x_j)$$
$$\bigvee_{i} (x_i \geq lb + n - 1)$$
$$\bigvee_{i} (x_i \leq ub - n + 1)$$

  where the last two are extra pigeon hole clauses, and lb and ub are the lower and upper bounds of $\{x_1, x_2, \ldots, x_n\}$.

- Other global constraints (element, weightedsum, cumulative, etc.) are translated in a straightforward way.

Solving CSP by Examples

- Open-Shop Scheduling (OSS) Problems
- Latin Square Problems

Open-Shop Scheduling (OSS) Problems

- An OSS problem consists of n jobs and n machines.
  - $J_0, J_1, \ldots, J_{n-1}$
  - $M_0, M_1, \ldots, M_{n-1}$
- Each job J_i consists of n operations.
  - $O_{i0}, O_{i1}, \ldots, O_{i(n-1)}$
- An operation O_{ij} of job J_i is processed at machine M_j, and has a positive processing time p_{ij}.
- Operations of the same job J_i must be processed sequentially but can be processed in any order.
- Each machine M_j can handle one operation at a time.

Objective of OSS

- Minimize the completion time (*makespan*) of finishing all jobs.

- OSS is highly non-deterministic. OSS with n jobs and n machines has $(n!)^{2n}$ feasibile schedulings.

OSS instance `gp03-01`

$$(p_{ij}) = \begin{pmatrix} 661 & 6 & 333 \\ 168 & 489 & 343 \\ 171 & 505 & 324 \end{pmatrix}$$

An optimal solution of `gp03-01` (makespan=1168)

Constraint Modeling of gp03-01

Defining integer variables

- m: makespan
- s_{ij}: start time of the operation O_{ij}

Defining constraints

- For each s_{ij},
$$s_{ij} + p_{ij} \le m$$
- For each pair of operations O_{ij} and O_{il} of the same job J_i,
$$(s_{ij} + p_{ij} \le s_{il}) \lor (s_{il} + p_{il} \le s_{ij})$$
- For each pair of operations O_{ij} and O_{kj} of the same machine M_j,
$$(s_{ij} + p_{ij} \le s_{kj}) \lor (s_{kj} + p_{kj} \le s_{ij})$$

Constraint Modeling of gp03-01

CSP representation of gp03-01

$$s_{00} + 661 \le m$$
$$s_{01} + 6 \le m$$
$$s_{02} + 333 \le m$$
$$\cdots\cdots\cdots$$
$$s_{22} + 324 \le m$$
$$(s_{00} + 661 \le s_{01}) \vee (s_{01} + 6 \le s_{00})$$
$$(s_{00} + 661 \le s_{02}) \vee (s_{02} + 333 \le s_{00})$$
$$(s_{01} + 6 \le s_{02}) \vee (s_{02} + 333 \le s_{01})$$
$$\cdots\cdots\cdots$$
$$(s_{02} + 333 \le s_{12}) \vee (s_{12} + 343 \le s_{02})$$
$$(s_{02} + 333 \le s_{22}) \vee (s_{22} + 324 \le s_{02})$$
$$(s_{12} + 343 \le s_{22}) \vee (s_{22} + 324 \le s_{12})$$

Solving gp03-01 by Sugar

Satisfiable case ($m \leq 1168$)

Solving gp03-01 by Sugar

Satisfiable case ($m \leq 1168$)

- MiniSat finds a solution by performing
  - 12 decisions and
  - 1 conflict (backtrack).

Solving gp03-01 by Sugar

Satisfiable case ($m \leq 1168$)

- MiniSat finds a solution by performing
  - 12 decisions and
  - 1 conflict (backtrack).

Unsatisfiable case ($m \leq 1167$)

Solving gp03-01 by Sugar

Satisfiable case ($m \leq 1168$)

- MiniSat finds a solution by performing
  - 12 decisions and
  - 1 conflict (backtrack).

Unsatisfiable case ($m \leq 1167$)

- MiniSat proves the unsatisfiability by performing
  - 6 decisions and
  - 5 conflicts (backtracks).

- Efficient bound propagations were realized by unit propagations of MiniSat solver.

Latin Square Problems

Latin Square Problem of size 5

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$

Latin Square Problems

Latin Square Problem of size 5

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldifferent in each row (5 rows)

Latin Square Problems

Latin Square Problem of size 5

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldifferent in each row (5 rows)

Latin Square Problems

Latin Square Problem of size 5

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldifferent in each row (5 rows)
- alldifferent in each column (5 columns)

Latin Square Problems

Latin Square Problem of size 5

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldifferent in each row (5 rows)
- alldifferent in each column (5 columns)

Latin Square Problems

Latin Square Problem of size 5

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldifferent in each row (5 rows)
- alldifferent in each column (5 columns)
- alldifferent in each diagonal (10 diagonals)

Latin Square Problems

Latin Square Problem of size 5

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldifferent in each row (5 rows)
- alldifferent in each column (5 columns)
- alldifferent in each diagonal (10 diagonals)

Latin Square Problems

Latin Square Problem of size 5

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldifferent in each row (5 rows)
- alldifferent in each column (5 columns)
- alldifferent in each diagonal (10 diagonals)

Latin Square Problems

Latin Square Problem of size 5

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldifferent in each row (5 rows)
- alldifferent in each column (5 columns)
- alldifferent in each diagonal (10 diagonals)

Latin Square Problems

Latin Square Problem of size 5

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

1	2	3	4	5
3	4	5	1	2
5	1	2	3	4
2	3	4	5	1
4	5	1	2	3

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldifferent in each row (5 rows)
- alldifferent in each column (5 columns)
- alldifferent in each diagonal (10 diagonals)
- Latin Square of size 5 is satisfiable.

Performance comparison of Latin Square Problems

Size	SAT/UNSAT	Sugar+m	Abscon	Mistral	bpsolver	Choco
3	UNSAT	0.57	0.66	0.01	0.03	0.41
4	UNSAT	0.59	0.63	0.01	0.03	0.41
5	SAT	0.73	0.68	0.01	0.03	0.58
6	UNSAT	0.79	0.82	0.03	0.17	0.73
7	SAT	0.94	0.78	0.01	0.04	0.77
8	UNSAT	1.01	676.75	-	-	-
9	UNSAT	1.08	-	-	-	-
10	UNSAT	1.16	-	-	-	-
11	SAT	1.35	-	-	-	-
12	UNSAT	1.66	-	-	-	-

- The table shows the CPU times (in seconds) of Latin Square Problems at 2009 CSP Solver Competition ("-" means timeout).

Effect of pigeon hole clauses

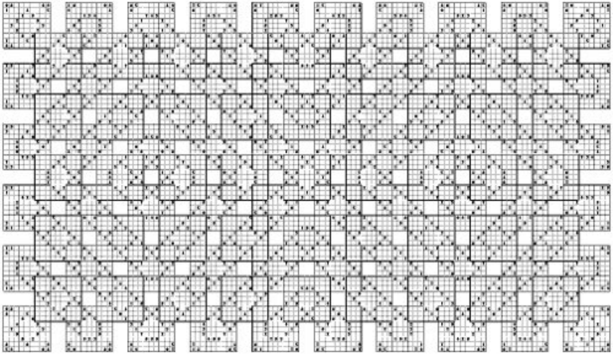
Size	SAT/UNSAT	Sugar+m with p.h. clauses	Sugar+m w/o p.h. clauses
3	UNSAT	0.44	0.35
4	UNSAT	0.47	0.41
5	SAT	0.44	0.43
6	UNSAT	0.52	0.40
7	SAT	0.80	0.69
8	UNSAT	1.08	-
9	UNSAT	0.98	-
10	UNSAT	3.12	-
11	SAT	1.59	-
12	UNSAT	3.23	-

- Pigeon hole clauses drastically improve the performance in both cases of SAT and UNSAT.
- They are well suited to the order encoding, and only two extra SAT clauses are required for each alldifferent constraint.

Demonstrations

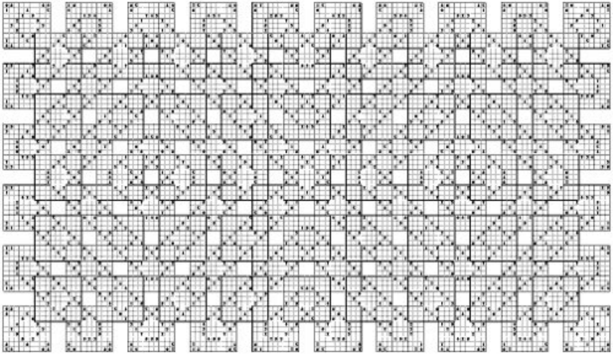
- Huge Sudoku Puzzles
- Scala Interface

Huge Sudoku Puzzle



- It consists of 105 Sudoku puzzles of 9×9 overlapping at corners (created by Hirofumi Fujiwara).
- It contains 6885 cells, 1808 hints, and 2655 alldifferent constraints.

Huge Sudoku Puzzle



- It consists of 105 Sudoku puzzles of 9×9 overlapping at corners (created by Hirofumi Fujiwara).
- It contains 6885 cells, 1808 hints, and 2655 alldifferent constraints.
- **Sugar can solve it in 30 seconds.**

Scala Interface of Sugar

Scala

- Functional OOPL on JVM (Java Virtual Machine)
- Java class libraries can be used.
- Both compiler and interpreter (REPL) are available.
- It is useful to define a DSL (Domain Specific Language).

Example of Scala Interface

Importing methods of Sugar

```
import Sugar._
```

Declaring variables

```
v('x, 0, 7)
v('y, 0, 7)
```

Adding constraints

```
c('x + 'y === 7)
c('x * 2 + 'y * 4 === 20)
```

Search a solution

```
if (search)
  println(solution)
```

n-Queens

n-Queens

```
import Sugar._
def queens(n: Int) = {
  val qs = for (i <- 0 to n-1) yield 'q(i)
  qs.foreach(v(_, 0, n-1))
  c(Alldifferent(qs: _*))
  c(Alldifferent((0 to n-1).map(i => 'q(i) + i): _*))
  c(Alldifferent((0 to n-1).map(i => 'q(i) - i): _*))
  if (search)
    do {
      println(solution)
    } while (searchNext)
}
```

Output

```
Map(q(2)->7,q(3)->5,q(7)->6,q(4)->0,q(5)->2,q(0)->3,q(6)->4,q(1)->1)
Map(q(2)->1,q(3)->5,q(7)->7,q(4)->2,q(5)->0,q(0)->4,q(6)->3,q(1)->6)
.....
```

Summary

- We presented
  - Order encoding and
  - Sugar constraint solver.
- Sugar showed a good performance for a wide variety of problems.
- The source package can be downloaded from the following web page.
  - http://bach.istc.kobe-u.ac.jp/sugar/ ▸ Web
- Sugar is developed as a software of the following project.
  - http://www.edu.kobe-u.ac.jp/istc-tamlab/cspsat/ ▸ Web

CSPSAT project (2008–2011)

Objective and Research Topics

R&D of efficient and practical SAT-based CSP solvers

- SAT encodings
  - CSP, Dynamic CSP, Temporal Logic, Distributed CSP
- Parallel SAT solvers
  - Multi-core, PC Cluster

Teams and Professors

- Kobe University (3)
- National Institute of Informatics (1)
- University of Yamanashi (3)
- Kyushu University (4)
- Waseda University (1)

Pointers

- SAT and SAT solvers
  - Handbook of Satisfiability, IOS Press, 2009.
  - International Conference on Theory and Applications of Satisfiability Testing (SAT)
  - Journal on Satisfiability, Boolean Modeling and Computation
  - "The Quest for Efficient Boolean Satisfiability Solvers", CADE 2002 [Zhang & Malik 2002]
  - "An Extensible SAT-Solver", SAT 2003 [Eén & Sörensson 2003]
- SAT and LP
  - "Logic programming with satisfiability", TPLP [Codish & Lagoon & Stuckey 2008]
  - "A Pearl on SAT Solving in Prolog", FLOPS 2010 [Howe & King 2010]
- Papers on Sugar
  - "Compiling Finite Linear CSP into SAT", Constraints, Vol.14, No.2, pp.254–272 [Tamura et al. 2009] Open Access