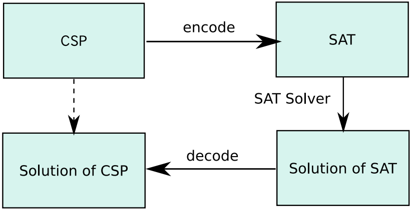
Solving Constraint Satisfaction Problems with SAT Technology

Naoyuki Tamura, Tomoya Tanjo, and Mutsunori Banbara

Kobe University, JAPAN

Contents



A SAT-based CSP Solver

1. SAT problems and SAT solvers
2. SAT encodings of Constraint Satisfaction Problems (CSP)
3. A SAT-based Constraint Solver Sugar
4. Solving Nonogram Puzzles by Example
5. Solving Open-Shop Scheduling (OSS) Problems by Example

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]

(1) function DPLL(S: a CNF formula, σ: a variable assignment)
(2) begin
(3) while $\emptyset \notin S\sigma$ and $\exists\{l\} \in S\sigma$ do /* unit propagation */
(4) if l is positive then $\sigma := \sigma \cup \{l \mapsto 1\}$;
(5) else $\sigma := \sigma \cup \{\bar{l} \mapsto 0\}$;
(6) if S is satisfied by σ then return true;
(7) if $\emptyset \in S\sigma$ then return false;
(8) choose an unassigned variable x from $S\sigma$;
(9) return DPLL(S, $\sigma \cup \{x \mapsto 0\}$) or DPLL($S$, $\sigma \cup \{x \mapsto 1\}$);
(10) end

- $S\sigma$ represents a CNF formula obtained by applying σ to S.
- $\emptyset$ means an empty clause (i.e. contradiction).

DPLL

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

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

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

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

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

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

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

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

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

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

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
   - 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 less than 1000 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.

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

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.
  - Minimum house-keeping informations are kept 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.

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

- 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

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

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

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

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

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

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

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.

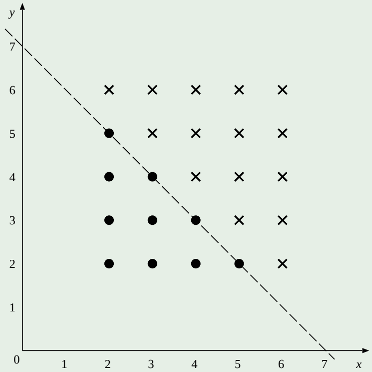
$$\neg p(x_1 = i_1) \vee \cdots \vee \neg p(x_k = i_k)$$

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 $x + y \le 7$

$$\neg p(x=2) \vee \neg p(y=6)$$
$$\neg p(x=3) \vee \neg p(y=5)$$
$$\neg p(x=3) \vee \neg p(y=6)$$
$$\neg p(x=4) \vee \neg p(y=4)$$
$$\neg p(x=4) \vee \neg p(y=5)$$
$$\neg p(x=4) \vee \neg p(y=6)$$
$$\neg p(x=5) \vee \neg p(y=3)$$
$$\neg p(x=5) \vee \neg p(y=4)$$
$$\neg p(x=5) \vee \neg p(y=5)$$
$$\neg p(x=5) \vee \neg p(y=6)$$
$$\neg p(x=6) \vee \neg p(y=2)$$
$$\neg p(x=6) \vee \neg p(y=3)$$
$$\neg p(x=6) \vee \neg p(y=4)$$
$$\neg p(x=6) \vee \neg p(y=5)$$
$$\neg p(x=6) \vee \neg p(y=6)$$



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.

Order encoding (cont.)

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

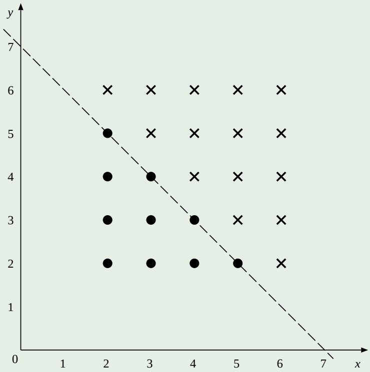
Constraint clauses

- 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.)

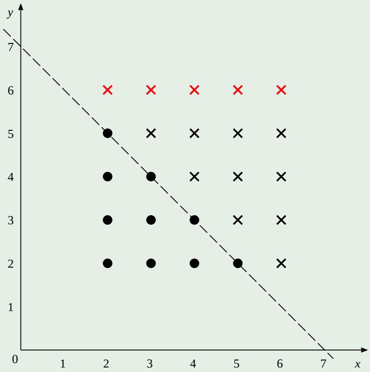
Encoding a constraint $x + y \le 7$



Order encoding (cont.)

Encoding a constraint $x + y \leq 7$

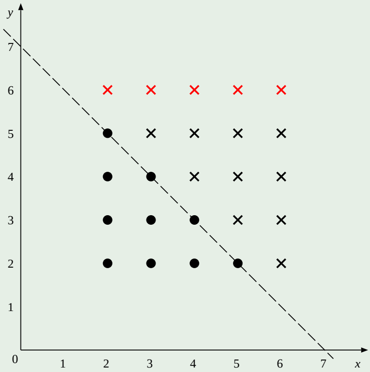
$\neg(y \geq 6)$



Order encoding (cont.)

Encoding a constraint $x + y \leq 7$

$p(y \leq 5)$

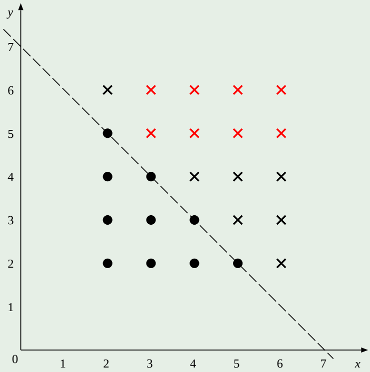


Order encoding (cont.)

Encoding a constraint $x + y \leq 7$

$p(y \leq 5)$

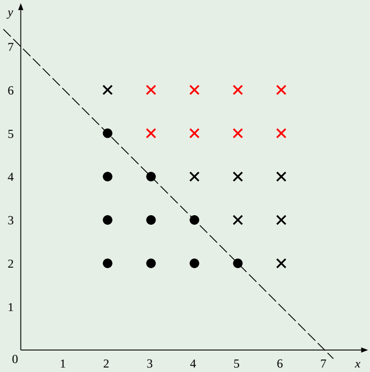
$\neg(x \geq 3 \land y \geq 5)$



Order encoding (cont.)

Encoding a constraint $x + y \leq 7$

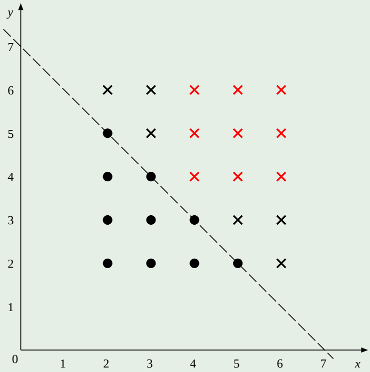
$p(y \leq 5)$
$p(x \leq 2) \vee p(y \leq 4)$



Order encoding (cont.)

Encoding a constraint $x + y \le 7$

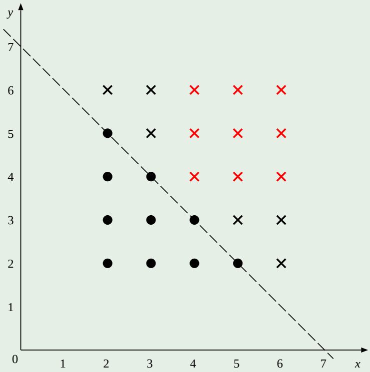
$p(y \le 5)$
$p(x \le 2) \vee p(y \le 4)$
$\neg(x \ge 4 \wedge y \ge 4)$



Order encoding (cont.)

Encoding a constraint $x + y \le 7$

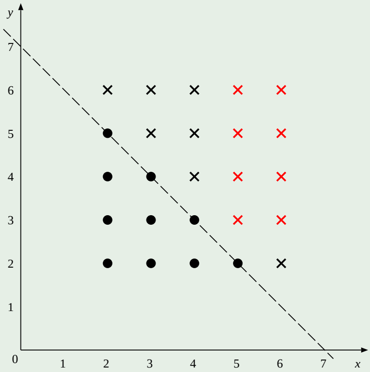
$p(y \le 5)$
$p(x \le 2) \lor p(y \le 4)$
$p(x \le 3) \lor p(y \le 3)$



Order encoding (cont.)

Encoding a constraint $x + y \leq 7$

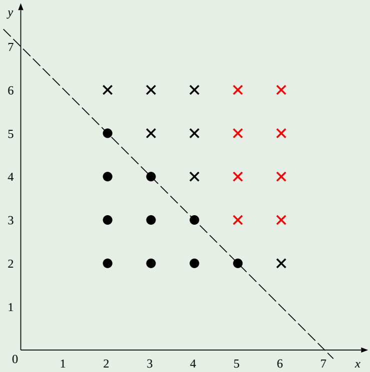
$p(y \leq 5)$
$p(x \leq 2) \vee p(y \leq 4)$
$p(x \leq 3) \vee p(y \leq 3)$
$\neg(x \geq 5 \wedge y \geq 3)$



Order encoding (cont.)

Encoding a constraint $x + y \le 7$

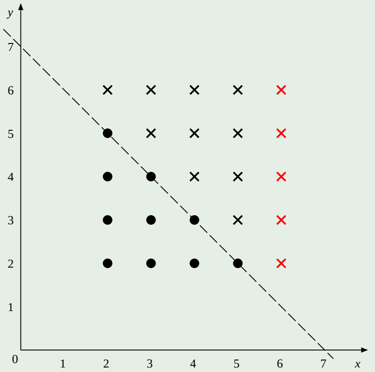
$p(y \le 5)$
$p(x \le 2) \vee p(y \le 4)$
$p(x \le 3) \vee p(y \le 3)$
$p(x \le 4) \vee p(y \le 2)$



Order encoding (cont.)

Encoding a constraint $x + y \leq 7$

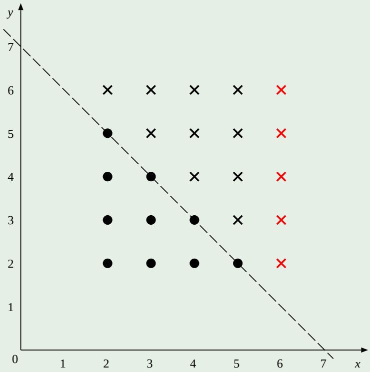
$p(y \leq 5)$
$p(x \leq 2) \vee p(y \leq 4)$
$p(x \leq 3) \vee p(y \leq 3)$
$p(x \leq 4) \vee p(y \leq 2)$
$\neg(x \geq 6)$



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 $x + y \leq 7$

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

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

Summary of the order encoding

The following shows the numbers of Boolean variables and clauses for encoding CSP with (maximum) domain size d.

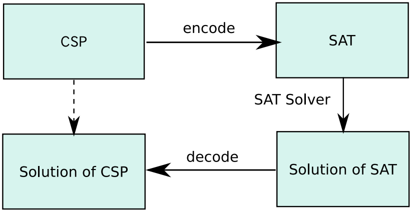
Number of Boolean variables and clauses

	Numbers
Boolean variables for x	$O(d)$
Clauses for x	$O(d)$
Clauses for $\sum_{i=1}^{m} a_i x_i \le c$	$O(d^{m-1})$

- $O(d^{m-1})$ for an m-ary constraint can be reduced to $O(md^2)$ by introducing new integer variables.
- Unit propagation on the order encoding establishes bound propagation in the original CSP.

A SAT-based Constraint Solver Sugar

Sugar: a SAT-based Constraint Solver



- **Sugar** is a constraint solver based on the **order encoding**.
- External SAT solvers (such as MiniSat and PicoSAT) are used as the backend solver.
- 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.
- In the 2009 CSP solver competition, Sugar became the winner in three categories out of seven categories.

Components of Sugar

- **Java program**
  - Parser of CSP files in XML and Lisp-like format
  - Converter of general CSP into linear CSP
    - Multiplications of variables are not supported.
  - 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

Converting general CSP

Expressions other than $\sum a_i x_i \le c$ can be converted in linear expressions 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 extra condition $(x \ge E) \land (x \ge F) \land ((x \le E) \lor (x \le F))$
$\min(E, F)$	x with extra condition $(x \le E) \land (x \le F) \land ((x \ge E) \lor (x \ge F))$
$\mathrm{abs}(E)$	$\max(E, -E)$
E div c	q with extra condition $(E = c\,q + r) \land (0 \le r) \land (r < c)$
E mod c	r with extra condition $(E = c\,q + r) \land (0 \le r) \land (r < c)$

- Global constraint (e.g. alldifferent) can be encoded by using its definition.

Encoding linear CSP into SAT

- Converting linear CSP into CNF by Tseitin transformation

$$A \lor (B \land C) \overset{\text{equi-sat}}{\Longleftrightarrow} (A \lor p) \land (\neg p \lor B) \land (\neg p \lor C)$$

  - Literal is either a Boolean variable, its negation, or a linear comparison $\sum a_i x_i \le c$
- Apply Tseitin transformation to linear comparisons if necessary

$$\sum a_i x_i \le c \lor \sum b_j y_j \le d$$

$$\overset{\text{equi-sat}}{\Longleftrightarrow} \quad (p \lor \sum b_j y_j \le d) \land (\neg p \lor \sum a_i x_i \le c)$$

- Encode linear comparisons by the order encoding, and Boolean literals are distributed to the encoded formula

CSC'2009

- The fourth CSP solver competition (CSC'2009) was held in 2009 with 9 teams and 14 solvers.
- In CSC'2009, the rankings were made in 7 categories.

- Solvers should answer whether the given CSP is `SAT` or `UNSAT`.
- Solvers are ranked with the number of solved instances under specified CPU time and memory limits. In case of tie, ranking is made with the cumulated CPU time on solved instances.
- Solvers giving a wrong answer in a category is disqualified in that category.

CSC'2009: Class of constraints and categories

- Extensional constraints: either tuples of support points or conflict points are explicitly given for each constraint.
- Intensional constraints: constructed from arithmetic, comparison, and logical operators.
- Global constraints: `alldifferent`, `element`, `weightedsum`. `cumulative`

	2-ary		N-ary					
Category	Ext.	Int.	Ext.	Int.	Alldiff	Elt	WSum	Cumul
2-ARY-EXT	√							
2-ARY-INT	√	√						
N-ARY-EXT	√		√					
N-ARY-INT	√	√	√	√				
GLOBAL1	√	√	√	√	√			
GLOBAL2	√	√	√	√	√	√	√	
GLOBAL3	√	√	√	√	√	√	√	√

CSC'2009: Benchmark instances

- Benchmark instances are written in XML format (XCSP 2.1), and classified into the following seven categories.

- 2-ARY-EXT: instances of 2-ary extensional constraints. The most of them are random CSPs.
- 2-ARY-INT: instances of 2-ary intensional and extensional constraints such as shop scheduling, frequency assignment, graph coloring, N-queens problems.
- N-ARY-EXT: instances of N-ary extensional constraints such as random CSPs and crossword puzzles.
- N-ARY-INT: instances of N-ary intensional and extensional constraints such as bounded model checking, real-time mutual-exclusion protocol verification, multi knapsack, pseudo Boolean algebra, Golomb ruler, social golfer problems.
- GLOBAL1–3: instances including global constraints such as Latin squares and timetabling problems.

Benchmarks in XCSP 2.1 ▸ Web

CSC'2009: Competition environment

- Cluster of bi-Xeon 3 GHz, 2MB cache, 2GB RAM kindly provided by the CRIL, University of Artois, France
- All solvers were run in 32 bits mode
- Each solver was imposed a memory limit of 900 MB (to avoid swapping and to allow two jobs to run concurrently on a node)
- CSP solvers were given a time limit of 30 minutes (1800s).

CSC'2009: List of Solvers

Solver's name	Categories								
	2E	2I	NE	NI	G1	G2	G3		
Abscon AC	√	√	√	√	√	√		Java	CRIL, Univ. d'Artois
Abscon ESAC	√	√	√	√	√	√		Java	CRIL, Univ. d'Artois
bpsolver	√	√	√	√	√	√	√	Prolog	CUNY
Choco 2.1.1	√	√	√	√	√	√	√	Java	École des Mines de Nantes
Choco 2.1.1b	√	√	√	√	√	√	√	Java	École des Mines de Nantes
Concrete	√	√	√	√	√			Java	École des Mines de Nantes
Concrete DC	√	√	√	√	√			Java	École des Mines de Nantes
Conquer	√		√					Java	University College Cork
Mistral	√	√	√	√	√	√	√	C++	4C, University College Cork
pcs	√	√	√	√				SAT?	Israel Institute of Technology?
pcs-restart	√	√	√	√				SAT?	Israel Institute of Technology?
SAT4J CSP	√	√	√	√	√			SAT	CRIL, Univ. d'Artois
Sugar+minisat	√	√	√	√	√	√	√	SAT	Kobe Univ.
Sugar+picosat	√	√	√	√	√	√	√	SAT	Kobe Univ.

CSC'2009: Teams and Solvers

CSC'2009: Teams and Solvers

Categories	Instances	Teams	Solvers
2-ARY-EXT	635	9	14
2-ARY-INT	686	8	13
N-ARY-EXT	699	9	14
N-ARY-INT	709	8	13
GLOBAL1	118	7	11
GLOBAL2	276	5	8
GLOBAL3	162	4	6

GLOBAL1: alldifferent constraint
GLOBAL2: alldifferent+element+weightedsum constraints
GLOBAL3: alldifferent+cumulative+element+weightedsum constraints

Results of 2-ARY-EXT (635 instances)

2-ARY-EXT: SAT+UNSAT Answers

Rank	Solver	#Solved	% of VBS	CPU time
1	Mistral	570	94%	46856.82
2	Choco 2.1.1b	556	91%	55414.54
3	Abscon AC	551	90%	58656.18
4	Abscon ESAC	547	90%	50388.31
5	Choco 2.1.1	547	90%	57385.70
6	Cocrete DC	504	83%	60964.15
7	Concrete	503	83%	49380.89
8	Sugar+minisat	466	77%	71234.79
9	Sugar+picosat	438	72%	53442.74
10	SAT4J CSP	421	69%	51843.43
11	bpsolver	416	68%	56052.25
12	pcs-restart	394	65%	46361.44
13	pcs	393	65%	56915.05

Results of 2-ARY-INT (686 instances)

2-ARY-INT: SAT+UNSAT Answers

Rank	Solver	#Solved	% of VBS	CPU time
1	Abscon ESAC	517	84%	17096.32
2	Abscon AC	513	83%	21759.06
3	Mistral	511	83%	27455.40
4	Choco 2.1.1b	510	83%	35843.62
5	Choco 2.1.1	508	82%	32542.78
6	Sugar+picosat	479	78%	36242.99
7	Sugar+minisat	470	76%	27807.61
8	Concrete	428	69%	34105.01
9	pcs-restart	419	68%	21837.86
10	pcs	419	68%	23992.67
11	Cocrete DC	372	60%	66687.23
12	bpsolver	349	57%	36187.51
13	SAT4J CSP	306	50%	17934.90

Results of N-ARY-EXT (699 instances)

N-ARY-EXT: SAT+UNSAT Answers

Rank	Solver	#Solved	% of VBS	CPU time
1	Mistral	585	96%	50114.03
2	Abscon AC	545	89%	41302.37
3	Concrete	544	89%	114700.63
4	Abscon ESAC	543	89%	42093.60
5	Conquer	532	87%	46955.70
6	Choco 2.1.1	532	87%	84300.32
7	Cocrete DC	532	87%	169470.05
8	Choco 2.1.1b	528	86%	75926.12
9	pcs	497	81%	82946.74
10	pcs-restart	496	81%	81860.66
11	bpsolver	394	64%	42707.44
12	Sugar+minisat	374	61%	72646.69
13	Sugar+picosat	350	57%	60281.82
14	SAT4J CSP	209	34%	30160.19

Results of N-ARY-INT (709 instances)

N-ARY-INT: SAT+UNSAT Answers

Rank	Solver	#Solved	% of VBS	CPU time
1	Mistral	572	91%	17837.81
2	Choco 2.1.1	560	89%	27734.86
3	Choco 2.1.1b	548	87%	31917.26
4	pcs-restart	546	87%	22451.19
5	pcs	542	86%	21390.68
6	bpsolver	513	81%	92578.86
7	Abscon ESAC	489	78%	38474.89
8	Abscon AC	481	76%	28646.82
9	Sugar+minisat	481	76%	33430.46
10	Sugar+picosat	478	76%	24531.14
11	Concrete	439	70%	82222.21
12	Cocrete DC	342	54%	85478.60
13	SAT4J CSP	171	27%	18993.60

Results of GLOBAL1 (118 instances)

GLOBAL1: SAT+UNSAT Answers

Rank	Solver	#Solved	% of VBS	CPU time
1	Sugar+picosat	104	97%	6050.38
2	Mistral	98	92%	5337.83
3	Sugar+minisat	88	82%	11507.92
4	Abscon ESAC	78	73%	10820.98
5	Abscon AC	77	72%	8404.10
6	Concrete	73	68%	5389.29
7	Choco 2.1.1	72	67%	6341.37
8	Choco 2.1.1b	71	66%	3690.13
9	Cocrete DC	68	64%	9825.10
10	SAT4J CSP	65	61%	11822.21
11	bpsolver	60	56%	1767.53

Results of GLOBAL2 (276 instances)

GLOBAL2: SAT+UNSAT Answers

Rank	Solver	#Solved	% of VBS	CPU time
1	Sugar+picosat	229	92%	10863.44
2	Sugar+minisat	222	89%	13704.74
3	Mistral	217	87%	10112.06
4	Choco 2.1.1	194	78%	10375.69
5	Choco 2.1.1b	193	78%	18774.49
6	bpsolver	186	75%	20642.14
7	Abscon AC	113	45%	31535.22
8	Abscon ESAC	95	38%	27897.30

Results of GLOBAL3 (162 instances)

GLOBAL3: SAT+UNSAT Answers

Rank	Solver	#Solved	% of VBS	CPU time
1	Sugar+minisat	136	94%	4196.23
2	Sugar+picosat	135	94%	3035.81
3	Choco 2.1.1	125	87%	10938.94
4	Choco 2.1.1b	122	85%	7647.70
5	Mistral	120	83%	2549.04
6	bpsolver	111	77%	27888.36

CSC'2009: Results of Sugar

- Sugar was bad for extensional constraints.
- It did OK for intensional constraints.
- It did quite well for global constraints.

CSC'2009: Results of Sugar

- Sugar was bad for extensional constraints.
- It did OK for intensional constraints.
- It did quite well for global constraints.

- Some 2-ARY-INT and N-ARY-INT instances contain extensional constraints (for example, crossword problems).
- To evaluate the performance purely on intensional constraints, 1797 instances with only intensional and global constraints are selected for comparison (PUREINT).

Results of PUREINT (1797 instances)

PUREINT: SAT+UNSAT Answers

Rank	Solver	#Solved	% of VBS	CPU time
1	Mistral	1375	86%	61248.25
2	Sugar+picosat	1354	85%	71212.80
3	Choco 2.1.1	1317	82%	80035.51
4	Sugar+minisat	1314	82%	71839.86
5	Choco 2.1.1b	1303	81%	91613.23
6	bpsolver	1084	68%	169126.56
7	Abscon AC	1042	65%	87133.47
8	Abscon ESAC	1036	65%	90641.42
9	pcs-restart	827	52%	41240.90
10	pcs	822	51%	41626.70
11	Concrete	799	50%	93806.60
12	Cocrete DC	694	43%	133461.22
13	SAT4J CSP	534	33%	44914.80

CSC'2009: Results Summary

Ranking of the Sugar Solvers

Categories	Sugar+m	Sugar+p
2-ARY-EXT	8 / 13	9 / 13
2-ARY-INT	7 / 13	6 / 13
N-ARY-EXT	12 / 14	13 / 14
N-ARY-INT	9 / 13	10 / 13
GLOBAL1	3 / 11	1 / 11
GLOBAL2	2 / 8	1 / 8
GLOBAL3	1 / 6	2 / 6

Demonstration
of
Solving Nonogram Puzzles

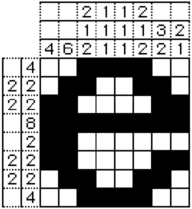
Rules of Nonogram (Paint by Pictures)

				2	1	1	2		
				1	1	1	1	3	2
		4	6	2	1	1	2	2	1
	4								
2	2								
2	2								
	8								
	2								
2	2								
2	2								
	4								

Rules of Nonogram

1. Color cells in black according to the following rules.
2. Numbers in each row are the lengths of the runs of black cells from left to right.
3. Numbers in each column are the lengths of the runs of black cells from top to bottom.
4. There must be at least one blank cell between two runs.

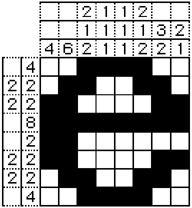
Rules of Nonogram (Paint by Pictures)



Rules of Nonogram

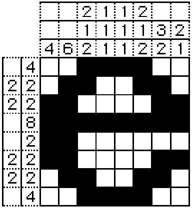
1. Color cells in black according to the following rules.
2. Numbers in each row are the lengths of the runs of black cells from left to right.
3. Numbers in each column are the lengths of the runs of black cells from top to bottom.
4. There must be at least one blank cell between two runs.

Modeling Nonogram in CSP



- Assign an integer variable $x_{ij} \in \{0, 1\}$ for each cell ($x_{ij} = 1$ means black)
  - $x_{00} = 0,\ x_{01} = 0,\ x_{02} = 1,\ x_{03} = 1,\ x_{04} = 1,\ x_{05} = 1,\ \ldots$
- Use an integer variable h_{ik} to indicate the left-most column position of the k-th run of i-th row
  - $h_{00} = 2,\ h_{10} = 1,\ h_{11} = 5,\ h_{20} = 0,\ h_{21} = 6,\ h_{30} = 0,\ \ldots$
- Use an integer variable v_{jk} to indicate the top-most row position of the k-th run of j-th column
  - $v_{00} = 2,\ v_{10} = 1,\ v_{20} = 0,\ v_{21} = 3,\ v_{22} = 6,\ v_{30} = 0,\ \ldots$

Modeling Nonogram in CSP



- Numbers in each row are the lengths of the runs of black cells from left to right.

$$x_{0j} = 1 \Leftrightarrow (h_{00} \le j \wedge j < h_{00} + 4)$$
$$x_{1j} = 1 \Leftrightarrow (h_{10} \le j \wedge j < h_{10} + 2) \vee (h_{11} \le j \wedge j < h_{11} + 2)$$
$$x_{2j} = 1 \Leftrightarrow (h_{20} \le j \wedge j < h_{20} + 2) \vee (h_{21} \le j \wedge j < h_{21} + 2)$$
$$x_{3j} = 1 \Leftrightarrow (h_{30} \le j \wedge j < h_{30} + 8)$$
$$\cdots$$

Modeling Nonogram in CSP

- Numbers in each column are the lengths of the runs of black cells from top to bottom.

$$x_{i0} = 1 \Leftrightarrow v_{00} \le i < v_{00} + 4$$
$$\ldots$$

- There must be at least one blank cell between two runs.

$$h_{10} + 2 < h_{11}$$
$$\ldots$$

Solving Nonogram by Sugar

1. CSP file is generated from a 100x100 Nonogram puzzle by a Perl script.

```
$ ./nonogram.pl data/nonogram-warship.txt >x.csp
$ wc -l x.csp
31649 x.csp
```

2. Sugar encodes the CSP into SAT with 141092 variables and 259085 clauses, and MiniSat solves it within 4 seconds.

```
$ sugar -vv x.csp | tee x.log
```

3. The solution is shown by the Perl script.

```
$ ./nonogram.pl -s x.log data/nonogram-warship.txt
```

Solving Open-Shop Scheduling (OSS) Problems by Example

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 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}.

Example of OSS instance gp03-01

$$(p_{ij}) = \begin{array}{c} \\ \end{array} \begin{array}{ccc} M_0 & M_1 & M_2 \end{array}$$

$$(p_{ij}) = \begin{pmatrix} 661 & 6 & 333 \\ 168 & 489 & 343 \\ 171 & 505 & 324 \end{pmatrix} \begin{array}{c} J_0 \\ J_1 \\ J_2 \end{array}$$

OSS (cont.)

We use a notation $O_{ij} \longleftrightarrow O_{kl}$ to describe a constraint which means that O_{ij} and O_{kl} can not be processed at the same time.

Constraints of OSS

- Operations of the same job J_i must be processed sequentially but can be processed in any order.

$$O_{ij} \longleftrightarrow O_{il} \quad (1 \le i \le n,\ 1 \le j < l \le n)$$

- Each machine M_j can handle one operation at a time.

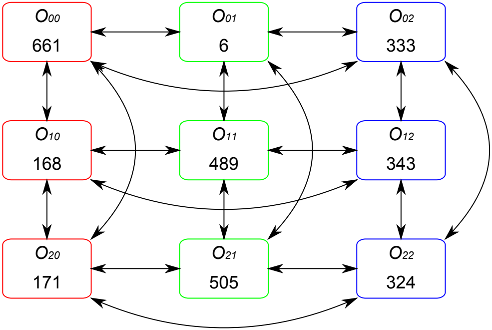
$$O_{ij} \longleftrightarrow O_{kj} \quad (1 \le j \le n,\ 1 \le i < k \le n)$$

Objective of OSS

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

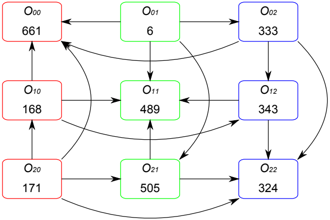
OSS instance gp03-01

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



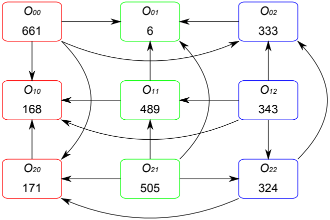
OSS instance gp03-01

A feasible solution of gp03-01 (makespan=1171)



OSS instance `gp03-01`

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



Solving Steps

1. **Modeling** the OSS instance in CSP
   - by defining integer variables, and
   - by defining constraints on them.
2. **Encoding** the CSP into a SAT instance
   - by using the order encoding.
3. **Solving** the instance by a SAT solver.
4. **Decoding** the result to obtain the solution of the original problem.

- SAT-based constraint solver (e.g. Sugar) does the last three steps for you.

Constraint Modeling of gp03-01

Defining integer variables

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

We also need to define the bounds.

Defining bounds of integer variables

- $m \in \{\ell \mathinner{..} u\}$
- $s_{ij} \in \{0 \mathinner{..} u - p_{ij}\}$

where

$$\ell = \max\left(\max_i \sum_j p_{ij},\ \max_j \sum_i p_{ij}\right) = 1000$$

$$u = \sum_k \max_{(i-j) \bmod n = k} p_{ij} = 661 + 343 + 505 = 1509$$

Constraint Modeling of gp03-01

Defining constraints

- The following constraint is added for each s_{ij}.

$$s_{ij} + p_{ij} \leq m$$

- The following constraint is added for each $O_{ij} \longleftrightarrow O_{kl}$.

$$(s_{ij} + p_{ij} \leq s_{kl}) \lor (s_{kl} + p_{kl} \leq s_{ij})$$

Constraint Modeling of gp03-01

Defining constraints

- The following constraint is added for each s_{ij}.

$$s_{ij} + p_{ij} \le m$$

- The following constraint is added for each $O_{ij} \longleftrightarrow O_{kl}$.

$$(s_{ij} + p_{ij} \le s_{kl}) \vee (s_{kl} + p_{kl} \le s_{ij})$$

However, we use the following one for further explanation.

$$\begin{array}{rcl} \neg q_{ijkl} & \vee & (s_{ij} + p_{ij} \le s_{kl}) \\ q_{ijkl} & \vee & (s_{kl} + p_{kl} \le s_{ij}) \end{array}$$

where q_{ijkl} is a new Boolean variable which means the operation O_{ij} precedes the operation O_{kl}.

Constraint Modeling of gp03-01

CSP representation of gp03-01

$$m \in \{1000 \mathinner{..} 1509\}$$
$$s_{00} \in \{0 \mathinner{..} 848\}$$
$$\cdots\cdots\cdots$$
$$s_{22} \in \{0 \mathinner{..} 1185\}$$
$$s_{00} + 661 \le m$$
$$\cdots\cdots\cdots$$
$$s_{22} + 324 \le m$$
$$\neg q_{0001} \vee s_{00} + 661 \le s_{01}$$
$$q_{0001} \vee s_{01} + 6 \le s_{00}$$
$$\cdots\cdots\cdots$$
$$\neg q_{1222} \vee s_{12} + 343 \le s_{22}$$
$$q_{1222} \vee s_{22} + 324 \le s_{12}$$

SAT Encoding of gp03-01

Encoding integer variables

- $m \in \{1000 \,..\, 1509\}$ (508 SAT clauses)

$$\neg p(m \le 1000) \vee p(m \le 1001)$$
$$\neg p(m \le 1001) \vee p(m \le 1002)$$
$$\cdots$$
$$\neg p(m \le 1507) \vee p(m \le 1508)$$

- $s_{00} \in \{0 \,..\, 848\}$ (847 SAT clauses)

$$\neg p(s_{00} \le 0) \vee p(s_{00} \le 1)$$
$$\neg p(s_{00} \le 1) \vee p(s_{00} \le 2)$$
$$\cdots$$
$$\neg p(s_{00} \le 846) \vee p(s_{00} \le 847)$$

SAT Encoding of gp03-01

Encoding constraints

- $s_{00} + 661 \le m$ (1509 SAT clauses)

$$\neg p(m \le 1000) \lor p(s_{00} \le 339)$$
$$\neg p(m \le 1001) \lor p(s_{00} \le 340)$$
$$\cdots$$
$$\neg p(m \le 1508) \lor p(s_{00} \le 847)$$

- $\neg q_{0001} \lor (s_{00} + 661 \le s_{01})$ (844 SAT clauses)

$$\neg q_{0001} \lor \neg p(s_{01} \le 660)$$
$$\neg q_{0001} \lor \neg p(s_{01} \le 661) \lor p(s_{00} \le 0)$$
$$\neg q_{0001} \lor \neg p(s_{01} \le 662) \lor p(s_{00} \le 1)$$
$$\cdots$$
$$\neg q_{0001} \lor \neg p(s_{01} \le 1502) \lor p(s_{00} \le 841)$$
$$\neg q_{0001} \lor p(s_{00} \le 842)$$

Solving the SAT instance

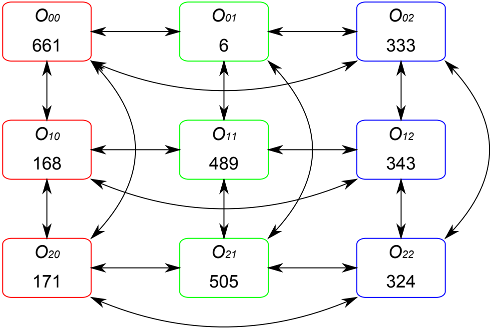
- Solving the SAT instance just gives a satisfiable assignment if there is a solution.
- To find a minimal makespan, it is needed to solve multiple SAT instances by changing the upper bound of the makespan (for example, in a binary search way).

$p(m \leq 1255)$	$p(m \leq 1127)$	$p(m \leq 1159)$	$\cdots$	$p(m \leq 1168)$
SAT	UNSAT	UNSAT	$\cdots$	SAT

- In addition, the incremental search capability of some SAT solvers can be used to avoid multiple invocations of the SAT solver.

Open-Shop Scheduling Problem

- Demonstration of solving the OSS instance gp03−01 by showing how the MiniSat solver searches a solution.
  - Satisfiable case ($m \leq 1168$)
  - Unsatisfiable case ($m \leq 1167$)



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).

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)

References

- General Readings on SAT
  - Handbook of Satisfiability, IOS Press, 2009.
  - Special Issue of “Recent Advances in SAT Techniques”, Journal of the Japanese Society for Artificial Intelligence, Vol.25, No.1, 2010. (**in Japanese**)
- Papers on SAT solvers
  - “The Quest for Efficient Boolean Satisfiability Solvers”, CADE 2002 [Zhang & Malik 2002]
  - “An Extensible SAT-Solver”, SAT 2003 [Eén & Sörensson 2003]
- Papers on Sugar
  - “Compiling Finite Linear CSP into SAT”, Constraints, Vol.14, No.2, pp.254–272 [Tamura et al. 2009] Open Access