Sugar++: A SAT-Based MAX-CSP/COP Solver

Tomoya Tanjo Naoyuki Tamura Mutsunori Banbara

Kobe University, Japan

Sep. 14, 2008
CP'08 Workshop of CSP Solver Competition

Background

- Enormous progress in performance of SAT solvers has been made in the last decade.
- Such progress has enabled to solve problems by encoding them to SAT problems: hardware verification, planning, and scheduling.
- Sugar is a CSP solver based on a new SAT-encoding method named "order encoding".
- In the order encoding, a comparison $x \leq a$ is encoded by a different Boolean variable $P_{x,a}$ for each integer variable x and integer value a.
- In Sugar, a SAT-encoded CSP is solved by the MiniSat (Eén and Sörensson) solver.

Main features of Sugar++

- Sugar++ is an enhancement of Sugar by using an incremental version of MiniSat.
- In Sugar++, a MAX-CSP is translated into a COP (Constraint Optimization Problem), and then it is encoded into a SAT problem except an optimization condition.
- Sugar++ solves the COP by invoking only one MiniSat process for a single SAT problem with varying the bound condition of the objective variable.
- Therefore the learnt clauses generated during the search can be reused.

Translating MAX-CSP into COP

- Given a CSP, the Max-CSP is to find an assignment that minimizes the number of violated constraints.
- The Max-CSP for a CSP$(V, Dom, \{C_1, C_2, \ldots, C_n\})$ can be translated into a COP: minimize *cost* s.t. CSP$(V^*, Dom^*, \mathcal{C}^*)$ as follows:
  - $V^* = V \cup \{c_1, \ldots, c_n, cost\}$
    - c_i: The penalty of the constraint C_i.
    - *cost*: The objective variable to be minimized.
  - $Dom^*(x) = \begin{cases} \{0,1\} & \text{if } x = c_i \quad (1 \le i \le n) \\ \{0,\ldots,n\} & \text{if } x = cost \\ Dom(x) & \text{otherwise} \end{cases}$
  - $\begin{aligned} \mathcal{C}^* \quad = \quad & \{(c_1 = 0) \Rightarrow C_1, \\ & (c_2 = 0) \Rightarrow C_2, \\ & \ldots \\ & (c_n = 0) \Rightarrow C_n, \\ & cost \ge \sum c_i\} \end{aligned}$

Example: Translating MAX-CSP into COP

MAX-CSP

`(int x 0 2) (int y 1 3)`	$x \in \{0, 1, 2\}$, $y \in \{0, 1, 2\}$
`(= (+ x (* 2 y)) 5)`	$x + 2y = 5$
`(= (+ (* 2 x) (* 3 y)) 8)`	$2x + 3y = 8$

Example: Translating MAX-CSP into COP

MAX-CSP

```
(int x 0 2)    (int y 1 3)
(= (+ x (* 2 y)) 5)
(= (+ (* 2 x) (* 3 y)) 8)
```

$x \in \{0,1,2\}$, $y \in \{0,1,2\}$
$x + 2y = 5$
$2x + 3y = 8$

COP

```
(int x 0 2)       (int y 1 3)
(int c1 0 1)      (int c2 0 1)
 (imp (= c1 0) (= (+ x (* 2 y)) 5))
 (imp (= c2 0) (= (+ (* 2 x) (* 3 y)) 8))
(int cost 0 2)    (>= cost (+ c1 c2))
(objective minimize cost)
```

- The symbol imp means an implication.

Example: Translating MAX-CSP into COP

MAX-CSP

```
(int x 0 2)   (int y 1 3)
(= (+ x (* 2 y)) 5)
(= (+ (* 2 x) (* 3 y)) 8)
```

$x \in \{0, 1, 2\}$, $y \in \{0, 1, 2\}$
$x + 2y = 5$
$2x + 3y = 8$

COP

```
(int x 0 2)      (int y 1 3)
(int c1 0 1)     (int c2 0 1)
 (imp (= c1 0) (= (+ x (* 2 y)) 5))
 (imp (= c2 0) (= (+ (* 2 x) (* 3 y)) 8))
(int cost 0 2)   (>= cost (+ c1 c2))
(objective minimize cost)
```

- The symbol imp means an implication.
- The integer variables c1 and c2 are the penalties of the corresponding constraints.

Example: Translating MAX-CSP into COP

MAX-CSP

```
(int x 0 2)   (int y 1 3)
(= (+ x (* 2 y)) 5)
(= (+ (* 2 x) (* 3 y)) 8)
```

$x \in \{0,1,2\}$, $y \in \{0,1,2\}$
$x + 2y = 5$
$2x + 3y = 8$

COP

```
(int x 0 2)       (int y 1 3)
(int c1 0 1)      (int c2 0 1)
 (imp (= c1 0) (= (+ x (* 2 y)) 5))
 (imp (= c2 0) (= (+ (* 2 x) (* 3 y)) 8))
(int cost 0 2)   (>= cost (+ c1 c2))
(objective minimize cost)
```

- The symbol imp means an implication.
- The integer variables c1 and c2 are the penalties of the corresponding constraints.
- The cost is the objective variable to be minimized.

Solving COP by using SAT solver

- A solution to SAT-encoded COP can be obtained by bisection search with varying the bound condition of the objective variable.

$$\Downarrow$$

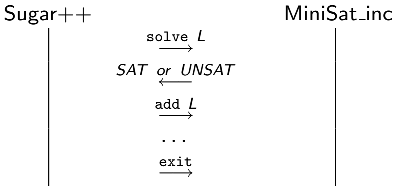
- For each search, the original Sugar invokes a different MiniSat process. This slows down the execution speed because the learnt clauses can not be reused in addition to the invocation overhead of multiple MiniSat processes.

$$\Downarrow$$

- To solve this problem, Sugar++ uses an incremental version of MiniSat called MiniSat_inc.

MiniSat_inc: an incremental version of MiniSat

- We modify the MiniSat so that it can deal with the following three commands from the standard input:
  - `add` $L_1\ L_2\ \cdots\ L_n$
    Adds a clause $\{L_1, L_2, \ldots, L_n\}$ to the SAT clause database.
  - `solve` $L_1\ L_2\ \cdots\ L_m$
    Solves the SAT problem with assumptions. These assumptions are not used for conflict analysis.
  - `exit`
    Terminates the MiniSat_inc.
- Bi-directional IO is used to perform communication between Sugar++ and MiniSat_inc.



Example: Solving COP by using MiniSat_inc

COP Example

`(int x 0 5)   (int y 0 5)`	$0 \le x \le 5, 0 \le y \le 5$
`(>= x y)      (>= (+ x y) 4)`	$x \ge y, x + y \ge 4$
`(objective minimize x)`	

- A COP is encoded into a SAT problem except the bound condition of the objective variable.
- MiniSat_inc is invoked with the SAT problem except the bound condition of the objective variable x.
- The bound condition of x are passed to MiniSat_inc during the bisection search.
  - Unknown conditions are passed as assumptions.
  - Decided conditions are added to the SAT clause database.

Example: Solving COP by using MiniSat_inc

SAT clause database		(int x 0 5)
		(int y 0 5)
		(>= x y)
		(>= (+ x y) 4)
An assumption		

- At First, the CSP part of the COP is encoded by using order encoding method.

Example: Solving COP by using MiniSat_inc

SAT clause database	$-P_{x,0} \vee P_{x,1}$ $-P_{x,1} \vee P_{x,2}$ $-P_{x,2} \vee P_{x,3}$ $-P_{x,3} \vee P_{x,4}$	`(int x 0 5)`
	$-P_{y,0} \vee P_{y,1}$ $-P_{y,1} \vee P_{y,2}$ $-P_{y,2} \vee P_{y,3}$ $-P_{y,3} \vee P_{y,4}$	`(int y 0 5)`
	$-P_{x,0} \vee P_{y,0}$ $-P_{x,1} \vee P_{y,1}$ $-P_{x,2} \vee P_{y,2}$ $-P_{x,3} \vee P_{y,3}$ $-P_{x,4} \vee P_{y,4}$	`(>= x y)`
	$-P_{x,0} \vee -P_{y,3}$ $-P_{x,1} \vee -P_{y,2}$ $-P_{x,2} \vee -P_{y,1}$ $-P_{x,3} \vee -P_{y,0}$	`(>= (+ x y) 4)`
An assumption		

- At First, the CSP part of the COP is encoded by using order encoding method.

Example: Solving COP by using MiniSat_inc

SAT clause database	$-P_{x,0} \lor P_{x,1}$ $-P_{x,1} \lor P_{x,2}$ $-P_{x,2} \lor P_{x,3}$ $-P_{x,3} \lor P_{x,4}$	`(int x 0 5)`
	$-P_{y,0} \lor P_{y,1}$ $-P_{y,1} \lor P_{y,2}$ $-P_{y,2} \lor P_{y,3}$ $-P_{y,3} \lor P_{y,4}$	`(int y 0 5)`
	$-P_{x,0} \lor P_{y,0}$ $-P_{x,1} \lor P_{y,1}$ $-P_{x,2} \lor P_{y,2}$ $-P_{x,3} \lor P_{y,3}$ $-P_{x,4} \lor P_{y,4}$	`(>= x y)`
	$-P_{x,0} \lor -P_{y,3}$ $-P_{x,1} \lor -P_{y,2}$ $-P_{x,2} \lor -P_{y,1}$ $-P_{x,3} \lor -P_{y,0}$	`(>= (+ x y) 4)`
An assumption	$P_{x,3}$	`(<= x 3)`

- Sugar++ assumes `(<= x 3)`, that is $P_{x,3}$, since $0 \le x \le 5$.
- This CNF under the assumption `(<= x 3)` is satisfiable.
- The learnt clauses are generated and added to the SAT clause database during the search.

Example: Solving COP by using MiniSat_inc

SAT clause database	$-P_{x,0} \vee P_{x,1}$ $-P_{x,1} \vee P_{x,2}$ $-P_{x,2} \vee P_{x,3}$ $-P_{x,3} \vee P_{x,4}$	`(int x 0 5)`
	$-P_{y,0} \vee P_{y,1}$ $-P_{y,1} \vee P_{y,2}$ $-P_{y,2} \vee P_{y,3}$ $-P_{y,3} \vee P_{y,4}$	`(int y 0 5)`
	$-P_{x,0} \vee P_{y,0}$ $-P_{x,1} \vee P_{y,1}$ $-P_{x,2} \vee P_{y,2}$ $-P_{x,3} \vee P_{y,3}$ $-P_{x,4} \vee P_{y,4}$	`(>= x y)`
	$-P_{x,0} \vee -P_{y,3}$ $-P_{x,1} \vee -P_{y,2}$ $-P_{x,2} \vee -P_{y,1}$ $-P_{x,3} \vee -P_{y,0}$	`(>= (+ x y) 4)`
	$P_{x,3}$	`(<= x 3)`
An assumption		

- Sugar++ adds a unit clause $\{P_{x,3}\}$ to the SAT clause database since this CNF under the assumption `(<= x 3)` is satisfiable.

Example: Solving COP by using MiniSat_inc

SAT clause database	~~$P_{x,0} \vee P_{x,1}$~~	$\neg P_{x,1} \vee P_{x,2}$	`(int x 0 5)`
	~~$P_{x,2} \vee P_{x,3}$~~	~~$P_{x,3} \vee P_{x,4}$~~	
	~~$P_{y,0} \vee P_{y,1}$~~	$\neg P_{y,1} \vee P_{y,2}$	`(int y 0 5)`
	~~$P_{y,2} \vee P_{y,3}$~~	~~$P_{y,3} \vee P_{y,4}$~~	
	~~$P_{x,0} \vee P_{y,0}$~~	$\neg P_{x,1} \vee P_{y,1}$	`(>= x y)`
	$\neg P_{x,2} \vee P_{y,2}$	~~$P_{x,3} \vee P_{y,3}$~~	
	~~$P_{x,4} \vee P_{y,4}$~~		
	~~$P_{x,0} \vee \neg P_{y,3}$~~	$\neg P_{x,1} \vee \neg P_{y,2}$	`(>= (+ x y) 4)`
	$\neg P_{x,2} \vee \neg P_{y,1}$	~~$P_{x,3} \vee \neg P_{y,0}$~~	
	~~$P_{x,3}$~~		`(<= x 3)`
An assumption			

- Sugar++ adds a unit clause $\{P_{x,3}\}$ to the SAT clause database since this CNF under the assumption `(<= x 3)` is satisfiable.
- The unit propagations eliminate some clauses.

Example: Solving COP by using MiniSat_inc

SAT clause database	$\neg P_{x,1} \vee P_{x,2}$	`(int x 0 5)`
	$\neg P_{y,1} \vee P_{y,2}$	`(int y 0 5)`
	$\neg P_{x,1} \vee P_{y,1}$ $\neg P_{x,2} \vee P_{y,2}$	`(>= x y)`
	$\neg P_{x,1} \vee \neg P_{y,2}$ $\neg P_{x,2} \vee \neg P_{y,1}$	`(>= (+ x y) 4)`
		`(<= x 3)`
An assumption	$P_{x,1}$	`(<= x 1)`

- Next, Sugar++ assumes `(<= x 1)`, that is $P_{x,1}$, since $0 \leq x \leq 3$.
- The learnt clauses that are generated in previous search are reused.

Example: Solving COP by using MiniSat_inc

SAT clause database	$-P_{x,1} \lor P_{x,2}$	(int x 0 5)
	$-P_{y,1} \lor P_{y,2}$	(int y 0 5)
	$-P_{x,1} \lor P_{y,1}$ $-P_{x,2} \lor P_{y,2}$	(>= x y)
	$-P_{x,1} \lor -P_{y,2}$ $-P_{x,2} \lor -P_{y,1}$	(>= (+ x y) 4)
	$-P_{x,1}$	(<= x 3) (> x 1)
An assumption		

- Next, Sugar++ assumes (<= x 1), that is $P_{x,1}$, since $0 \leq x \leq 3$.
- The learnt clauses that are generated in previous search are reused.
- Sugar++ adds a unit clause $\{-P_{x,1}\}$ to the SAT clause database since this CNF under the assumption (<= x 1) is unsatisfiable. $-P_{x,1}$ is the negation of $P_{x,a}$.

Example: Solving COP by using MiniSat_inc

SAT clause database	~~$\neg P_{x,1} \vee P_{x,2}$~~	(int x 0 5)
	$\neg P_{y,1} \vee P_{y,2}$	(int y 0 5)
	~~$P_{x,1} \vee \neg P_{y,1}$~~ $\neg P_{x,2} \vee P_{y,2}$	(>= x y)
	~~$P_{x,1} \vee \neg P_{y,2}$~~ $\neg P_{x,2} \vee \neg P_{y,1}$	(>= (+ x y) 4)
	~~$\neg P_{x,1}$~~	(<= x 3) (> x 1)
An assumption		

- Next, Sugar++ assumes (<= x 1), that is $P_{x,1}$, since $0 \leq x \leq 3$.
- The learnt clauses that are generated in previous search are reused.
- Sugar++ adds a unit clause $\{\neg P_{x,1}\}$ to the SAT clause database since this CNF under the assumption (<= x 1) is unsatisfiable. $\neg P_{x,1}$ is the negation of $P_{x,a}$.
- The unit propagations eliminate some clauses.

Example: Solving COP by using MiniSat_inc

SAT clause database		(int x 0 5)
	$\neg P_{y,1} \lor P_{y,2}$	(int y 0 5)
	$\neg P_{x,2} \lor P_{y,2}$	(>= x y)
	$\neg P_{x,2} \lor \neg P_{y,1}$	(>= (+ x y) 4)
		(<= x 3) (> x 1)
An assumption	$P_{x,2}$	(<= x 2)

- Sugar++ assumes (<= x 2) since $2 \leq x \leq 3$.
- The learnt clauses that are generated in previous search are reused.
- Because this CNF under the assumption (<= x 2) is satisfiable, the optimal value $x = 2$ is found.

Conclusion

- We talked about Sugar++ that is an enhancement of Sugar by using an incremental version of MiniSat.
- In Sugar++, a MAX-CSP is translated into a COP, and then it is encoded into a SAT problem except an optimization condition.
- Sugar++ solves the COP by invoking only one MiniSat process for a single SAT problem with varying the bound condition of the objective variable.
- Therefore the learnt clauses generated during the search can be reused.

Benchmarks

	No. of searching	Sugar	Sugar++	Sugar/Sugar++
gcp-2-FullIns_5	4	255.27	334.64	0.76
golombRuler-8	6	15.24	18.48	0.82
jss-ft10	9	148.17	65.33	2.27
oss-gp10-01	12	116.03	34.14	3.4
Tdsp-C1-1	4	7.24	23.78	0.3
AVERAGE				1.51

- In average, Sugar++ is 50% faster than Sugar.
- Larger the number of searching are, more effective the incremental search are.