The 33rd Annual Conference of the Japanese Society for Artificial Intelligence, 2019

A SAT-based CSP Solver sCOP and its Results on 2018 XCSP3 Competition

Takehide Soh[*1] Daniel Le Berre[*2] Mutsunori Banbara[*3] Naoyuki Tamura[*1]

[*1]Kobe University [*2]CRIL-CNRS, UMR 8188, Université d'Artois [*3]Nagoya University

Constraint Satisfaction Problem (CSP) is the combinatorial problem of finding a variable assignment which satisfies all given constraints over finite domains. CSP has a wide range of applications in the research domains of Artificial Intelligence and Operations Research. XCSP3 is one of major constraint languages that can describe CSPs. More than 23,000 instances over 105 series are available in the XCSP3 database. In 2018, the international XCSP3 competition was held and 18 solvers participated.

This paper describes the under development CSP solver sCOP and its results on the 2018 XCSP3 competition. sCOP is a SAT-based solver which encodes CSPs into SAT problems and finds a solution using SAT solvers. Currently, sCOP equips the order and log encodings, and uses off-the-shelves backend SAT solvers. We registered sCOP to two competition tracks—CSP-Standard-Sequential and CSP-Standard-Parallel—of the 2018 XCSP3 competition and won both tracks.

1. Input of sCOP—XCSP3 Language

This section explains the XCSP3 language [Boussemart 17], an input of the CSP solver sCOP. XCSP3 is an XML based constraint language that can describe CSPs. Let's start with an easy example of XCSP3 and then explain its constraints.

1.1 Example

The graph coloring problem (GCP) is a problem whose goal is to assign a color to each node in a given graph such that there is no two same-colored nodes connected by an edge. Figure 1 shows an instance of GCP.

This GCP can be represented by a CSP. We introduce five variables n_0, n_1, n_2, n_3, n_4. Each variable has the same domain $\{0, 1, 2\}$ which represents different colors like red, green, and blue. We also introduce the conjunction of the six inequalities representing the constraints of the GCP instance.

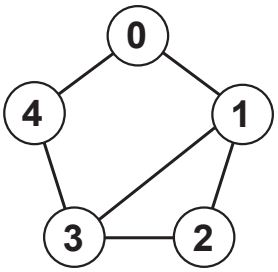
$$n_0 \neq n_1 \qquad n_0 \neq n_4 \qquad n_1 \neq n_2$$
$$n_1 \neq n_3 \qquad n_2 \neq n_3 \qquad n_3 \neq n_4$$

This CSP can be represented by the XCSP3 instance in Fig. 1. Line 1 describes the format name "XCSP3" and the type "CSP" of the given problem. Line 2 to 4 describe the five integer variable by using array. The domain of those array variables is set to the range from 0 to 2, and their identifier is set to "n". Line 5 to 14 describe the conjunction of the six inequalities.

1.2 Constraints in XCSP3

In the previous example, constraints are represented by using *intentional* constraints (ne). Except intensional constraints, we can use *extensional* and *global* constraints in XCSP3. This section explains each kind of constraints.

Intentional constraints are constraints using the arithmetic, logical, and comparison operators. As arithmetic operators, `add` (addition), `sub` (subtraction), `mul` (multiplication), `div` (integer division), `abs` (absolute value), etc. can be used. As logical operators, `not` (¬), `and` (∧), `or` (∨), `xor` (⊕), `iff` (⇔), `imp` (⇒), etc. can be used. As comparison operators, `le` (≤), `lt` (<), `ge` (≥), `gt` (>), `ne` (≠), `eq` (=), etc. can be used. For instance, a constraint $(2x + 3 < y) \vee z \geq 1$ can be represented in the following intensional constraints of XCSP3.




```
<instance format="XCSP3" type="CSP">
  <variables>
    <array id="n" size="[5]"> 0..2 </array>
  </variables>
  <constraints>
    <intension>
      and(ne(n[0],n[1]),
          ne(n[0],n[4]),
          ne(n[1],n[2]),
          ne(n[1],n[3]),
          ne(n[2],n[3]),
          ne(n[3],n[4]))
    </intension>
  </constraints>
</instance>
```

Figure 1: (top) GCP instance (bottom) its XCSP3 representation

```
<intension>
  or(lt(add(mul(2,x),3),y),ge(z,1))
</intension>
```

Contact: Takehide Soh, Information Information Science and Technology Center, Kobe University, 1-1 Rokkodai, Nada, Kobe, 657-8501, soh@lion.kobe-u.ac.jp

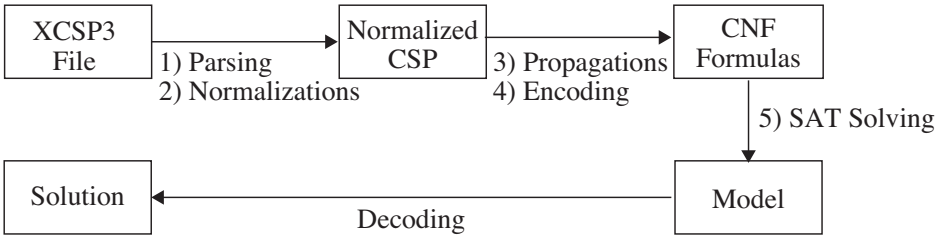


Figure 2: Framework of sCOP

Extensional constraints represent feasible (or infeasible) value combinations of variables extensionally. It is also called *table* constraints. As the name suggests, constraints are often represented by tables. For instance, one of the inequalities in the previous GCP example, $n_0 \neq n_1$ is represented by the following tables.

(Support Table for $n_0 \neq n_1$)

n_0	n_1
0	1
0	2
1	0
1	2
2	0
2	1

(Conflict Table for $n_0 \neq n_1$)

n_0	n_1
0	0
1	1
2	2

There are two types of extensional constraints: *support* and *conflict*. The former represents feasible value combinations of variables and the latter represents infeasible ones. Obviously, they complement each other and one of them is enough to represent constraints. The conflict table above can be represented in the following extensional constraints of XCSP3.

```
<extension>
  <list> n[0] n[1] </list>
  <conflicts> (0,0)(1,1)(2,2) </conflicts>
</extension>
```

Global constraints represent comparatively complex but useful constraints to model applications. In XCSP3, 19 global constraints are available. For instance, *allDifferent* is such a global constraint ensuring mutually different values must be assigned to given variables. An implicit allDifferent constraints between nodes 1, 2, and 3 in the previous GCP instance can be represented in the following constraints of XCSP3.

```
<allDifferent>
  n[1] n[2] n[3]
</allDifferent>
```

Other than above, XCSP3 provides more advanced constructs for symbolic/set/real/graph/stochastic/qualitative variables, soft constraints, mono and multi objective optimization, etc. In XCSP3 competitions, a set of basic constructs named XCSP3-Core is used. Interested reader is referred to the full specification [*1].

2. SAT-based CSP Solver sCOP

sCOP [Soh 18] is a SAT-based CSP Solver written in Scala. Given a XCSP3 instance file, sCOP finds a solution using a SAT solver. Figure 2 shows the framework of sCOP.

Following Sugar [Tamura 08] and Diet-Sugar [Soh 17], sCOP encodes CSPs (XCSP3 instances) into SAT problem in conjunctive normal form (CNF) using (i) the order encoding [Tamura 09, Tamura 13] and (ii) the log encoding for Pseudo-Boolean (PB) constraints [Soh 17]. Then, sCOP invokes a SAT solver that would find a model if any. The obtained model is decoded into a solution of the XCSP3 instance. sCOP is publicly available in its web page [*2].

1) Parsing. Parsing of XCSP3 formatted file is done by using an XCSP3 official tool XCSP3-Java-Tools [*3]. Currently, sCOP accepts all constraints in the XCSP3-core language[*1].

2) Normalization. Before pre-processing, all constraints are translated into intensional constraints. This normalization is processed as follows. Global constraints are translated into intensional constraints by a straightforward way but we use extra pigeon hole constraints for allDifferent constraints as in Sugar [Tamura 08]. Extensional constraints are translated into intensional constraints by using a variant of multi-valued decision diagrams. This is a difference to ones in Sugar. All intentional constraints are normalized to be in the form of CNF using Tseitin transformation. Literals of this CNF-CSP are linear comparisons $\sum_i a_i x_i \geq k$ where a_i's are integer coefficients, x_i's are integer variables and k is an integer constant.

3) Pre-processing: propagation. Constraint propagations are executed to the normalized CSP (clausal CSP, i.e., in the form of CNF over linear comparisons $\sum_i a_i x_i \geq k$) to remove redundant values, variables, and linear comparisons. Currently, it is done by using an AC3 like algorithm.

4) Encoding into SAT. In sCOP, the order encoding [Tamura 09, Tamura 13] and the log encoding are used. The order encoding uses propositional variables $p_{x \geq d}$'s meaning $x \geq d$ for each domain value d of each integer variable x. To encode linear comparisons, Algorithm 1 of the literature [Tamura 13] is used in sCOP. The log encoding uses a binary representation of integer variables. There are several ways to encode linear comparisons by using those propositional variables. In sCOP, we replace all integer variables with its binary representation—it gives us a set of PB constraints. We then encode those PB constraints into CNF formulas by using the BDD encoding [Eén 06]. sCOP basically uses the order encoding but uses the log encoding in case that the huge number of clauses is expected to be encoded. For this expectation, the idea of domain product criteria [Soh 17] is used.

5) SAT Solving. By using DIMACS CNF files, sCOP's backend

*1 http://www.xcsp.org/specifications

*2 https://tsoh.org/sCOP/

*3 https://github.com/xcsp3team/XCSP3-Java-Tools

SAT solver is switchable. In the 2018 XCSP3 competition, sCOP uses MapleCOMSPS and glucose-syrup. A SAT solver MapleCOMSPS [*4] is used for sequential CSP solving. It is the winning solver on the main track of the SAT competition 2016. It also shows a good performance for solving CSP instances encoded by sCOP. A SAT solver glucose-syrup [*5] is used for parallel CSP solving. It is the winning solver on the parallel track of the SAT competition 2017.

Example. After saving the XCSP3 instances of Figure 1 as a plain text file `gcp.xml`, the execution of sCOP returns the following results using a default SAT solver.

```
$ java -jar scop.jar gcp.xml
<... encoding information ...>
s SATISFIABLE
v <instantiation>
v     <list>n[0] n[1] n[2] n[3] n[4]</list>
v     <values>2 0 2 1 0</values>
v </instantiation>
```

3. Results on 2018 XCSP3 Competition

International competitions of CSP solvers using the XCSP language have been held since 2005. The first series were held with earlier versions of XCSP languages and called International CSP Solver Competition (CSC). CSCs were held in 2005, 2006, 2008 and 2009. After some break period, it is re-started in 2017 with XCSP3. In those competitions, solvers are submitted from research institutions over Europe and North America.

3.1 Overview of 2018 XCSP3 Competition

In 2018 XCSP3 Competition, there are the following 8 tracks.

	Solver	Seq/Par	Timeout	#Ins.	#Sol.
CSP	Standard	Sequential	40 min.	236	14
		Parallel	40 min.	236	4
COP		Sequential	40 min.	346	8
		Parallel	40 min.	346	1
		Sequential	4 min.	346	7
		Parallel	4 min.	346	1
CSP	Mini	Sequential	40 min.	176	11
COP		Sequential	40 min.	188	7

CSP and *COP* are categories for Constraint Satisfaction Problem and Constraint Optimization Problem. There are two solver categories: *Mini* is a category for "mini solvers" which is comparatively small and simple solvers. Also, those mini solvers must be open source software. *Standard* is a category for other solvers. *Sequential* and *Parallel* are categories for sequential solvers and parallel solvers that can use eight CPU cores. For COP category, there is a "Fast" solver track whose goal is to evaluate solvers computing a good quality solution within a comparatively short time—4 minutes. In the other tracks, 40 minutes for each instance are given to solvers. About benchmark, 236 instances are selected for CSP categories and 346 instances are selected for COP categories. In case for "mini solvers", benchmark instances are limited to contain only intensional constraints or some basic global constraints.

*4 https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/

*5 http://www.labri.fr/perso/lsimon/glucose/

Table 1: Ranking of CSP-Standard-Sequential Track

Rank	Solver	#Solved	S/U	%VBS
—	VBS (Virtual Best Solver)	163	103/60	100
1	scop-order+maple	146	92/54	90
2	scop-both+maple	140	87/53	86
3	PicatSAT	138	85/53	85
4	Mistral-2.0	116	80/36	71
5	Choco-solver-4.0.7b seq	115	77/38	71
6	Concrete-3.9.2	92	64/28	56
7	OscaR - Conf. Ord. Res.	90	62/28	55
8	Concrete-3.9.2-SuperNG	84	55/29	52
9	Sat4j-CSP	83	40/43	51
10	OscaR - Conf. Order.	81	51/30	50
11	cosoco-1.12	79	53/26	48
12	BTD	76	31/45	47
13	BTD_12	76	32/44	47
14	macht	66	33/33	40

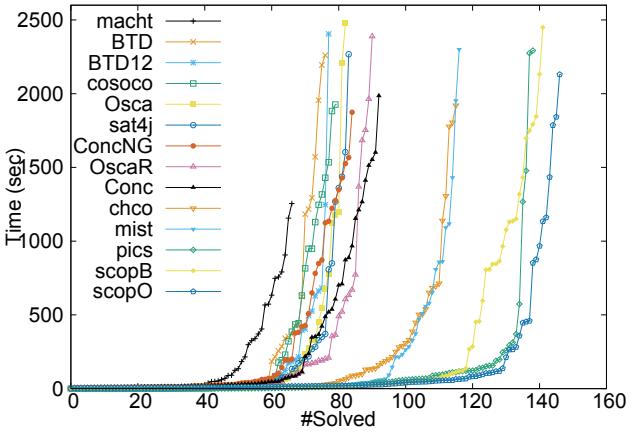


Figure 3: Cactus Plot of CSP-Standard-Sequential Track

Among those eight tracks, two tracks were canceled because there was only one solver registered.

sCOP registered to CSP-Standard-Sequential and CSP-Standard-Parallel tracks. The former is popular and the most competitive track in a sense of the number of participated solvers.

3.2 Results of sCOP

Table 1 shows the solver ranking of the CSP-Standard-Sequential track with regards to the number of solved instances. The first row shows the abstract VBS (virtual best solver) which is the collection of the best solver for each instance. The third column shows the number of instances solved and the fourth column shows the number of satisfiable and unsatisfiable instances solved. The fifth column shows the percentages of the number of solved instances w.r.t. VBS. Among all 14 solvers, sCOP using the order encoding solved the most number of instances. sCOP using the both of the order and log encodings takes the second place.

Figure 3 shows the cactus plot whose x-axis is the number of instances solved and y-axis is the CPU time in seconds. The meaning of the cactus plot is that "each of x instances were solved within y CPU seconds". Note that "which x instances" are different for each solver. For each plot, being righter means solvers solve more instances, being lower means solvers solve instances faster.

Table 2: Number of Instances Solved: Figures are organized by instance series.

Series Name	#Ins.	Choco-solver-4.0.7b	Mistral-2.0	PicatSAT	scop-both+maple	scop-order+maple	Ctrs.
Bibd	12	4	**8**	**8**	**8**	**8**	pure int
CarSequencing	17	11	6	**17**	12	**17**	sup.
ColouredQueens	12	3	**4**	**4**	3	3	pure int
Crossword	13	**5**	3	2	3	3	sup.
Dubois	12	7	7	**12**	**12**	**12**	sup.
Eternity	15	**7**	**7**	6	6	6	sup.
Frb	16	3	3	3	**5**	**5**	conf./sup.
GracefulGraph	11	**6**	**6**	**6**	**6**	**6**	pure int
Haystacks	10	4	2	**10**	**10**	**10**	conf./sup.
Langford	11	7	**9**	**9**	**9**	**9**	pure int
MagicHexagon	11	4	**5**	3	3	3	pure int
MisteryShopper	10	**10**	**10**	**10**	**10**	**10**	sup.
PseudoBoolean-dec	13	4	6	4	7	**8**	pure int
Quasigroups	16	6	6	7	**8**	**8**	pure int
Rlfap-dec-scens11	12	11	10	**12**	**12**	**12**	pure int
SocialGolfers	12	6	6	**8**	**8**	**8**	pure int
SportsScheduling	10	3	**4**	**4**	**4**	**4**	sup.
StripPacking	12	7	8	**11**	**11**	**11**	sup.
Subisomorphism	11	**7**	6	2	3	3	conf./sup.
Total	236	115	116	138	140	**146**	

Table 2 shows the number of instances solved. We picked up the top five solvers and all figures are organized by 19 instance series. The second column "#Ins." denotes the number of instances included in each series. The last column "Ctr." denotes types of constraints: "pure int" means instances consist of only intensional and global constraints, "sup." means instances containing extensional constraints of supports, "conf." means instances containing extensional constraints of conflicts. According to the results, sCOP is particularly better than others in Pseudo-Boolean series and Quasigroups. Oppositely, it is worse than others in Subisomorphism.

In CSP-Standard-Parallel track, four solvers are registered. sCOP using the order encoding is also better than other solvers.

All information contains, i) how instances are selected, ii) descriptions of instance series and solvers, are available in the competition proceedings [Lecoutre 18].

4. Conclusion

This paper describes the under development SAT-based CSP solver sCOP and its results during the 2018 XCSP3 Competition. The sCOP solver is written in Scala and currently uses the order and log encodings to solve XCSP3 instances. The results of 2018 XCSP3 Competition showed that sCOP is superior to the other state-of-the-art XCSP3 solvers in terms of the number of solved instances within the given time limit. Future work is as follows. Adapting COP is important. To improve performance, implementing more SAT encodings and their hybridization are necessary. To enhance usability, supporting other constraint languages such as MiniZinc or Sugar's language are also important future work.

References

[Boussemart 17] Boussemart, F., Lecoutre, C., Audemard, G., and Piette, C.: XCSP3 An Integrated Format for Benchmarking Combinatorial Constrained Problems: XCSP3 Specifications—Version 3.0.5, `http://xcsp.org/format3.pdf` (2017)

[Eén 06] Eén, N. and Sörensson, N.: Translating Pseudo-Boolean Constraints into SAT, *Journal on Satisfiability, Boolean Modeling and Computation*, Vol. 2, No. 1-4, pp. 1–26 (2006)

[Lecoutre 18] Lecoutre, C. and Roussel, O.: XCSP3 Competition 2018 Proceedings, `https://www.cril.univ-artois.fr/~lecoutre/papers/XCSP3_2018_Proceedings.pdf` (2018)

[Soh 17] Soh, T., Banbara, M., and Tamura, N.: Proposal and Evaluation of Hybrid Encoding of CSP to SAT Integrating Order and Log Encodings, *International Journal on Artificial Intelligence Tools*, Vol. 26, No. 1, pp. 1–29 (2017)

[Soh 18] Soh, T., Berre, D. L., Banbara, M., and Tamura, N.: sCOP: SAT-based Constraint Programming System, in *XCSP3 Competition 2018 Proceedings*, pp. 93–94 (2018)

[Tamura 08] Tamura, N. and Banbara, M.: Sugar: a CSP to SAT Translator Based on Order Encoding, in *Proceedings of the 2nd International CSP Solver Competition*, pp. 65–69 (2008)

[Tamura 09] Tamura, N., Taga, A., Kitagawa, S., and Banbara, M.: Compiling Finite Linear CSP into SAT, *Constraints*, Vol. 14, No. 2, pp. 254–272 (2009)

[Tamura 13] Tamura, N., Banbara, M., and Soh, T.: PBSugar: Compiling Pseudo-Boolean Constraints to SAT with Order Encoding, in *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2013)*, pp. 1020–1027 (2013)