About Benchmarking and Competitions of Solvers in Constraint Programming

Frédéric Boussemart[1] and **Christophe Lecoutre**[1] and **Arnaud Malapert**[2] and **Cédric Piette**[1]
[1] Université d'artois, CNRS, CRIL, UMR 8188, rue de l'université, 62307 Lens cedex, France
[2] Université Nice Sophia Antipolis, CNRS, I3S, UMR 7271, 06900 Sophia Antipolis, France

Abstract

CSP (Constraint Satisfaction Problem) is the problem of finding an assignment of values to a set of variables such that a set of constraints is satisfied. In this short paper, we briefly describe how past CSP competitions were conducted, present the main features of the new format XCSP3, targeted to become a CP (Constraint Programming) standard, and discuss about classification of instances as well as ranking of solvers. In the context of benchmarking and solver competitions, this position paper aims at encouraging the cross-fertilization of ideas and methods in both planning and constraint reasoning domains.

Constraint programming (CP) is a general technology providing simple, general and efficient models and algorithms for solving combinatorial constrained problems. At the heart of CP, we find the framework CSP (Constraint Satisfaction Problem) that consists in solving problem instances represented by Constraint Networks (CNs). A solution to a CN is obtained by instantiating its set of variables so that its set of constraints is satisfied. This framework has many derivatives, mainly extensions, as indicated in Figure 1: temporal CSP (TCSP), weighted CSP (WCSP), valued CSP (VCSP), quantified CSP (QCSP), constraint optimization problem (COP), Max-CSP, distributed CSP (DisCSP), and so on.

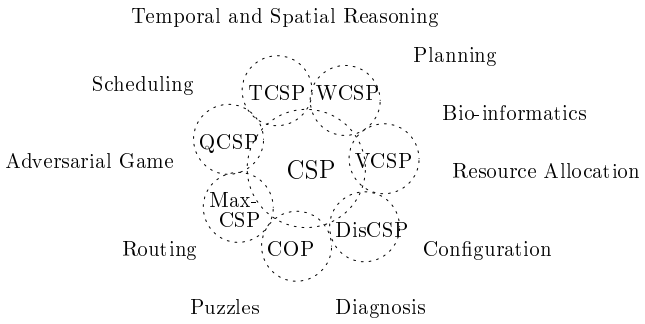

Figure 1: The framework CSP and some of its extensions.

Systems for solving CP instances are called constraint solvers. Several competitions have been organized over the years to assess the different solvers or algorithms developed in the CP community. A first series of competitions has focused on generic, black-box solvers that do not require human intervention to perform well. These competitions have been organized in this spirit, using XCSP 2.1 as input format, in 2005, 2006, 2008 and 2009. They have contributed to identify interesting techniques and to draw a fair picture of state-of-the-art algorithms or heuristics (Lecoutre, Roussel, and van Dongen 2010). On the other hand, every year since 2008, another series of competitions has been run: the MiniZinc challenge. Different solvers are compared on common Minizinc/Flatzinc benchmarks, in particular so that solver implementers can examine the detailed results and determine the strengths and weaknesses of their system, and problem modelers can judge which system might be preferable for their problem (Stuckey, Becket, and Fischer 2010). In the rest of the paper, we briefly describe how past CP competitions were conducted, present the main features of the new format XCSP3, and discuss about classification of instances as well as ranking of solvers.

Constraint Solver Competitions

In this section, we succinctly present how the fourth international constraint solver competition (CSC), held in 2009, was organized. There were three problems:

- CSP
- Max-CSP
- WCSP

The objective for CSP is finding a solution or proving that no solution exists. The objective for Max-CSP is finding an assignment of variables that violates as few constraints as possible. The objective for WCSP is finding an assignment of minimal violation cost (cost functions, instead of hard constraints, are considered). CSP is a decision problem whereas Max-CSP and WCSP are optimization ones.

There were two solver categories:

- complete
- incomplete

Complete solvers can determine if an instance is satisfiable or not (or can find and prove optimality for Max-CSP

and WCSP) whereas incomplete solvers cannot prove the unsatisfiability or the optimum.

The format used to represent problem instances was XCSP 2.1; it is described in (Roussel and Lecoutre 2009). Some tools were also provided:

- C and C++ parsers for XCSP 2.1
- A Java parser for XCSP 2.1
- A program to check the validity of problem instances in format XCSP 2.1
- A program to check solutions

Execution Environment

Solvers were run on a cluster of computers using the Linux operating system. They were run under the control of another program that enforces some limits on memory and CPU time. Solvers were run inside a sandbox that prevents unauthorized use of the system (network connections, file creation outside the allowed directory, among others). It was required that two executions of a solver with the same parameters and system resources should output the same result in approximately the same time (so that the experiments can be repeated).

Of course, contestants were asked to provide the organizers with a command line for running their solvers. In this command line, the following placeholders were replaced by the actual information given by the evaluation environment:

- BENCHNAME replaced by the name of the file containing the instance to solve.
- RANDOMSEED replaced by a random seed which is a number between 0 and $4,294,967,295$. This parameter had to be used for initializing the random number generator (when the solver uses random numbers). It is recorded by the evaluation environment, which allows to solve a given instance under the same conditions if necessary.
- TIMEOUT represents the total CPU time (in seconds) that the solver may use before being killed. May be used to adapt the solver strategy.
- MEMLIMIT represents the total amount of memory (in MiB) that the solver may use before being killed. May be used to adapt the solver strategy.
- TMPDIR is the name of the only directory where the solver is allowed to read/write temporary files.
- DIR is the name of the directory where the solver files are stored.

After TIMEOUT seconds have elapsed, the solver first receives a SIGTERM to give it a chance to output the best solution found so far (in the case of an optimizing solver). One second later, the program receives a SIGKILL signal from the controlling program to terminate the solver. Similarly, a solver that used more memory than the limit defined by MEMLIMIT was sent a SIGTERM followed one second later by a SIGKILL.

The solver could not write to any file except standard output, standard error and files in the TMPDIR directory. A solver was not allowed to open any network connection or launch external commands which are not related to the solving process.

Output Rules

The evaluation environment recorded everything that is output by a solver on stdout/stderr (up to a limit of 1MiB) and put a timestamp on each line. This can be very informative to check how solvers behave on some instances.

Of course, solvers had to output special messages to the standard output in order to check results. The output format was inspired by the DIMACS output specification of the SAT competitions. Lines output by the solver had to be prefixed by "c ","s ","v ", "d " or "o ". Other lines were ignored.

- solution ("s " line) These lines are mandatory, start with lower case s followed by space (ASCII code 32), and are followed by one of the following answers:
  - UNSUPPORTED
  - SATISFIABLE
  - UNSATISFIABLE (only used for CSP)
  - UNKNOWN
  - OPTIMUM FOUND

  Note that UNSUPPORTED is used when the solver recognizes a constraint that is not implemented.
- values ("v " line) These lines are mandatory and contain a solution, i.e., a sequence of numbers, with whitespace as separator.
- diagnostic ("d " line) These lines are optional. A keyword followed by a value must be given on such lines. For example, a possible keyword is ASSIGNMENTS.
- comment ("c " line) Such lines are optional and may appear anywhere in the solver output. They contain any information that authors want to output. They are recorded by the evaluation environment for later viewing.
- objective cost ("o " line) (for Max-CSP and WCSP, only) These lines contain an integer value.

On the one hand, a CSP solver had to output exactly one "s " line and in addition, when the instance is found SATISFIABLE, exactly one "v " line. On the other hand, since a MAX-CSP or WCSP solver does not stop as soon as it finds a solution but instead tries to find a better solution, it must be given a way to output the best found solution even when it reaches the time limit. There are two options depending on the ability of the solver to intercept signals. The first option can be used if the solver is able to catch the signal SIGTERM : when interrupted, it must output the best found solution. The second option must be used otherwise: a certificate "v " line is output each time the solver finds a solution which is better than the previous ones.

Ranking

For each problem (CSP, Max-CSP and WCSP) and solver category, a ranking was computed for different categories of instances, based on constraint arity (binary and non-binary

constraints) and constraint representation (extensional, intensional and global constraints). Solvers were ranked as follows.

CSP: Solvers claiming incorrect results in a given category were disqualified from this category. Of the remaining solvers, the solver solving the most problems were declared the winner. Ties were broken by considering the minimum total solution time.

Max-CSP/WCSP: Solvers with no incorrect results in a given category were ranked as follows:

Incomplete solvers were ranked in increasing order of their average relative error (i.e. the average normalized difference between the cost of solutions found by the solver and the cost of the best known solutions).

Complete solvers were ranked in decreasing order of the number of instances for which a solution was proved to be OPTIMUM (using average relative error as tie breaker).

Minizinc Competitions

In this section, we succinctly present how the last Minizinc competition, held in 2014, was organized. The language used for representing problem instances in that competition was MiniZinc (Nethercote et al. 2007) (version 1.6), and its related low-level format FlatZinc. Entrants to the challenge provide a FlatZinc solver as well as global constraint definitions specialized for their solver. A translator, called mzn2fzn, is run on MiniZinc models using the provided global constraint definitions to create FlatZinc files, which are given as input to the contestant solvers.

An entrant in the challenge is a constraint solver that is installed in a virtual machine (VM) provided by the organizers. The provided VM with an installed solver is run by an executable file called fzn-exec that invokes a FlatZinc solver handling FlatZinc version 1.6. The syntax is:

```
fzn-exec [<options>] file.fzn
```

where:

- the argument file.fzn is the name of a FlatZinc 1.6 model instance to evaluate.
- the options are:

  -a For satisfaction problems, the solver must output all solutions, and for optimization problems, the solver must output the first optimal solution and all found intermediate solutions.

  -f The solver is free to ignore any specified search strategy.

  -p <n> The solver is free to use multiple threads and/or cores during search. The argument n specifies the number of cores that are available.

There have been four competition CLASSES:

- FD search: solvers must follow the search strategy given in each input file, and are disqualified when they do not follow it.
- Free search: solvers are free to ignore the search strategy.
- Parallel search: solvers are free to use multiple threads or cores.
- Open class: this class allows the usage of portfolio solvers. Solvers are free to use multiple threads or cores to solve the problem.

Output of solvers must conform to the FlatZinc 1.6 specification. For optimization problems, when the time limit is exceeded before the final solution is printed then the last complete approximate solution printed is considered to be the solution for that instance. Each solver s is run on problem instance p and the following information is collected:

- $wck(p, s)$ - the wall clock time used by s.
- $solved(p, s)$ - true if p is solved by s
- $quality(p, s)$ - the objective value of the best solution found by s.
- $optimal(p, s)$ - true if the objective value is proved optimal by s.

Scoring Procedure The scoring procedure is based on the Borda count voting system. Each benchmark instance is treated like a voter who ranks the solvers. Each solver scores points related to the number of solvers that it beats. More precisely, a solver s scores points on problem p by comparing its performance with each other solver s' as follows:

- if s gives a better answer than s' it scores 1 point,
- else if $\neg solved(p, s)$ or s gives a worse answer than s' it scores 0 point.
- else (s and s' gives indistinguishable answers) scoring is based on execution time comparison: s scores $wck(p, s')$ / $(wck(p, s') + wck(p, s))$, or 0.5 if both finished in 0s.

A solver s is said to be better than a solver s' iff:

- for satisfaction problems, $solved(p, s) \wedge \neg solved(p, s')$
- for optimization problems,

$$solved(p, s) \wedge \neg solved(p, s') \vee optimal(p, s) \wedge \neg optimal(p, s') \vee quality(p, s) > quality(p, s')$$

From XCSP 2.1 to XCSP3

Although significant efforts were performed these recent years, in the context of competitions of solvers, as shown in previous sections, the Constraint Programming (CP) community still suffers from the lack of a standard low-level format for representing various forms of combinatorial problems subject to constraints and optimization. It is important to note that we specifically refer here to the possibility of generating and exchanging files containing precise descriptions of problem instances (no model/data separation), so that fair comparisons of problem solving approaches can be made in good conditions, and experiments can be reproduced easily. The two current proposals, XCSP 2.1 and FlatZinc, have some drawbacks that certainly prevent them from becoming such a "universal" format. For example, it was not possible to deal with objective functions in XCSP

2.1, and not possible to deal with weighted constraint networks in MiniZinc (and so, in FlatZinc), although a proposal was made in (Ansotegui et al. 2011), but to the best of our knowledge, never incorporated in official specifications.

In (Boussemart et al. 2015), the specifications for a major revision/extension of the basic format XCSP 2.1 are introduced. This new format, XCSP3, is a rather light language that allows us to get integrated representations of combinatorial constrained problems, by enumerating variables, constraints and objectives in a very simple and unambiguous way.

Actually, here are the main advantages of XCSP3:

- **Generality**. XCSP3 allows us to represent many forms of combinatorial constrained problems since it can deal with constraint satisfaction, mono and multi-objective optimization, preferences through soft constraints, variable quantification, qualitative reasoning, continuous constraint solving, distributed constraint reasoning, and probabilistic constraint reasoning.
- **Completeness**. A very large range of constraints is available, including rarely used variants and encompassing practically (i.e., to a very large extent) all constraints that can be found in major constraint solvers such as, e.g., Choco and Gecode.
- **Understanding**. We pay attention to control the number of concepts and basic constraint forms, advisedly exploiting automatic variations of these forms through lifting, restriction, sliding, combination and relaxation mechanisms, so as to facilitate global understanding.
- **Readability**. The new format is more compact, and less redundant than XCSP 2.1, making it very easy to read and understand, especially as variables and constraints can be handled under the form of arrays/groups.
- **Flexibility**. It will be very easy to extend the format, if necessary, in the future, for example by adding new kind of global constraints, or by adding a few XML attributes in order to handle new concepts.
- **Easiness of Parsing**. Thanks to the XML architecture of the format, basically, it is easy to parse instance files at a coarse-grain level. Besides, parsers written in Java and C++ are made available.
- **Dedicated Website**. A website, companion of XCSP3, will be open soon, with many models/series/instances made available. This website will allow the user to make sophisticated queries in order to select and download the instances that he finds relevant.

The main novelties of XCSP3 with respect to XCSP 2.1 are:

- **Optimization**. XCSP3 can manage both mono-objective and multi-objective optimization.
- **New Types of Variables**. It is possible to define 0/1, integer, symbolic, float, qualitative, set, and graph variables, in XCSP3.
- **Lifted and Restricted forms of Constraints**. It is natural to extend basic forms of constraints over lists (tuples), sets and multi-sets. It is simple to build restricted forms of constraints by considering some properties of lists.
- **Meta-constraints**. It is possible to exploit sliding and logical mechanisms over variables and constraints.
- **Relaxed constraints**. Relaxed cost-based constraints can be defined easily.
- **Reification**. Half and full reification is easy, and made possible by letting the user associate a 0/1 variable with any constraint of the problem through a dedicated XML attribute.
- **Views**. In XCSP3, it is possible to post constraints with arguments that are not limited to simple variables or constants, thus, avoiding in some situations the necessity of introducing auxiliary variables and constraints, and permitting solvers that can handle variable views to do it.
- **Structure**. It is possible to post variables under the form of arrays (of any dimension) and to post constraints in (semantic or syntaxic) groups, thereby, partly preserving the structure of the models.
- **Annotations**. It is possible to add annotations to the instances, for indicating search guidance and filtering preferences.

```
<instance format="XCSP3" type="CSP">
  <variables>
    <array id="M" size="[3][3]">
      1..9
    </array>
  </variables>
  <constraints>
    <allDifferent> M[][]</allDifferent>
    <group>
      <sum>
        <list> %... </list>
        <condition> (eq,15) </condition
            >
      </sum>
      <args> M[0][] </args>
      <args> M[1][] </args>
      <args> M[2][] </args>
      <args> M[][0] </args>
      <args> M[][1] </args>
      <args> M[][2] </args>
      <args>
        M[0][0] M[1][1] M[2][2]
      </args>
      <args>
        M[2][0] M[1][1] M[0][2]
      </args>
    </group>
  </constraints>
</instance>
```

As an illustration, let us consider the XCSP3 formulation for the 3-order instance of the Magic Square problem, given above. A magic square of order 3 is an arrangement of numbers $1, 2, \ldots, 9$ in a square grid, where the numbers in each row, and in each column, and the numbers in the main and secondary diagonals, all add up to the same number (15).

Classification

A recurrent issue in benchmarking is the classification of instances. We can distinguish two common practices which are not exclusive: qualitative classification and quantitative classification. A *qualitative* classification gathers instances according to some features (nature of the problems, random generation, ...). A *quantitative* classification gathers instances according to their difficulty. The latter is frequently used in other communities, but to our knowledge, only qualitative classification are publicly available for MiniZinc and XCSP.

Quantitative classification We propose a simple classification method based on the results of various solvers. For example, these results may correspond to those obtained at successive competitions.

The Virtual Best Solver (VBS) is a theoretical construction which returns the best answer provided by one of the solvers. Similarly, the Virtual Worst Solver (VWS) returns the worst answer provided by one of the solvers.

Easy instances are classified using the **VWS**: **training** (less than 10 seconds); or **easy** (less than 1000 seconds). **Hard** instances are classified using the **VBS**: **medium** (less than 1000 seconds); **challenging** (no timeout); or **hard** (timeout or memory exception).

These categories form a partition of the instances. Indeed, an instance belongs to the first category above whose acceptance condition is met. For example, a medium instance is solved in less than 1000 seconds by the VBS, but more than 1000 seconds by the VWS (otherwise, it would be an easy instance).

Of course, such a classification attempt should also take into account the experimental protocols followed for obtaining the results (e.g., computing infrastructures)

Scoring Procedure

The main drawback of the scoring procedure in the CSC competition is that solving times are only considered for tie breaking for CSP and not considered at all for optimization. On the other hand, the scoring procedure in the Minizinc competition is based on the Borda count voting system, as explained earlier. Unfortunately, the way points are distributed in case of indistinguishable answers does not capture user's preferences very well. Indeed, if the solver s solves the first n problems in 0.1 seconds and the n last problems in 1000 seconds whereas the solver s' solves the first n problems in 0.2 seconds and the n last problems in 500 seconds, then both solvers obtain the same score (n) whereas most users would certainly prefer s'.

To partially address this issue, we propose a scoring variant. Let t and t' respectively denote $wck(p, s)$ and $wck(p, s')$, i.e., the wall clock times of solvers s and s' when solving the problem p. In case of indistinguishable answers, s scores $f(t, t') = t' \div (t+t')$ according to the Borda system. Here, we define $g(t, t') = g(t) + (1 - g(t) - g(t')) \times f(t, t')$ in which some points are given by contract $g(t) = 1 \div 2 \times (\log_a(t+1)+1)$ ($a = 10$) where $g(t)$ is a strictly decreasing function from 0.5 toward 0. The remaining points are shared between the two solvers using Function f. Using Function g in the previous example, solvers s and s' are respectively scored $0.81 \times n$ and $1.19 \times n$ points: s' is therefore preferred to s.

This idea arose because in competitions there is usually a significant number of easy problems. Only using Function f introduces a bias toward solvers that solve easy problems very quickly. Besides, one of the main drawback of the Borda count is that it does not satisfy the Condorcet criterion, i.e., a solver who wins a duel against each other candidate is not always the winner of the competition.

In future competitions, it would certainly be relevant to modify the scoring procedure, while being careful that it remains consistent with the competition rules. For instance, the Borda count does not satisfy the independence of clones criterion, stating that the winner must not change when a non-winning solver is duplicated (considered twice). So, rules could impose that each contestant submit at most one solver (contrary to previous competitions).

Conclusion

In this position paper, we have provided CP feedback on benchmarking and solver competitions. We have also presented some innovative developments in terms of problem representation format and competition rules. We hope that this material may be useful to the organizers of the international planning competitions.

Acknowledgments

This work has been supported by both CNRS and OSEO (BPI France) within the ISI project 'Pajero'

References

[Ansotegui et al. 2011] Ansotegui, C.; Bofill, M.; Palah, M.; Suy, J.; and Villaret, M. 2011. W-MiniZinc: A proposal for modeling weighted CSPs with MiniZinc. In *Proceedings of 1st International Workshop on MiniZinc*.

[Boussemart et al. 2015] Boussemart, F.; Lecoutre, C.; Perradin, V.; and Piette, C. 2015. XCSP3: An integrated format for benchmarking combinatorial constrained problems. Technical Report (preliminary), CRIL.

[Lecoutre, Roussel, and van Dongen 2010] Lecoutre, C.; Roussel, O.; and van Dongen, M. 2010. Promoting robust black-box solvers through competitions. *Constraints* 15(3):317–326.

[Nethercote et al. 2007] Nethercote, N.; Stuckey, P.; Becket, R.; Brand, S.; Duck, G.; and Tack, G. 2007. MiniZinc: Towards a standard CP modelling language. In *Proceedings of CP'07*, 529–543.

[Roussel and Lecoutre 2009] Roussel, O., and Lecoutre, C. 2009. XML representation of constraint networks: Format XCSP 2.1. Technical Report arXiv:0902.2362, CoRR.

[Stuckey, Becket, and Fischer 2010] Stuckey, P.; Becket, R.; and Fischer, J. 2010. Philosophy of the MiniZinc challenge. *Constraints* 15(3):307–316.