SUNNY-CP and the MiniZinc challenge

Roberto Amadini, Maurizio Gabbrielli, Jacopo Mauro

Published in Theory and Practice of Logic Programming 11(4–5), 611–627 (2011)
(27th Int'l. Conference on Logic Programming (ICLP'11) Special Issue)

SUNNY-CP and the MiniZinc Challenge

Roberto Amadini
Department of Computing and Information Systems, The University of Melbourne, Australia.

Maurizio Gabbrielli
DISI, University of Bologna, Italy / FOCUS Research Team, INRIA, France.

Jacopo Mauro
Department of Informatics, University of Oslo, Norway.

Abstract

In Constraint Programming (CP) a portfolio solver combines a variety of different constraint solvers for solving a given problem. This fairly recent approach enables to significantly boost the performance of single solvers, especially when multicore architectures are exploited. In this work we give a brief overview of the portfolio solver `sunny-cp`, and we discuss its performance in the MiniZinc Challenge—the annual international competition for CP solvers—where it won two gold medals in 2015 and 2016.

Under consideration in Theory and Practice of Logic Programming (TPLP)

1 Introduction

In *Constraint Programming* (CP) the goal is to model and solve Constraint Satisfaction Problems (CSPs) and Constraint Optimisation Problems (COPs) (Rossi et al. 2006). Solving a CSP means finding a solution that satisfies all the constraints of the problem, while for COPs the goal is to find an optimal solution, which minimises (or maximises) an objective function.

A fairly recent trend to solve combinatorial problems, based on the well-known *Algorithm Selection* problem (Rice 1976), consists of building portfolio solvers (Gomes and Selman 2001). A *portfolio solver* is a meta-solver that exploits a collection of $n > 1$ constituent solvers $s_1, \dots, s_n$. When a new, unseen problem comes, the portfolio solver seeks to predict and run its best solver(s) $s_{i_1}, \dots, s_{i_k}$ (with $1 \leq k \leq n$) for solving the problem.

Despite that plenty of Algorithm Selection approaches have been proposed (Kotthoff 2014; Smith-Miles 2008; Hutter et al. 2014), a relatively small number of portfolio solvers have been practically adopted (Amadini et al. 2015c). In particular, only few portfolio solvers participated in CP solvers competitions. The first one (for solving CSPs only) was CPHydra (O'Mahony et al. 2008) that in 2008 won the International CSP Solver Competition.[1] In 2013 a portfolio solver based on Numberjack (Hebrard et al. 2010) attended the *MiniZinc Challenge* (MZNC) (Stuckey et al. 2014), nowadays the only surviving international competition for CP solvers.

[1] The International CSP Solver Competition ended in 2009.

Between 2014 and 2016, `sunny-cp` was the only portfolio solver that joined the MZNC. Its first, sequential version had appreciable results in the MZNC 2014 but remained off the podium. In MZNC 2015 and 2016 its enhanced, parallel version (Amadini et al. 2015a) demonstrated its effectiveness by winning the gold medal in the Open category of the challenge.

In this paper, after a brief overview of `sunny-cp`, we discuss the performance it achieved in the MiniZinc Challenges 2014—2016 and we propose directions for future works. The lessons we learned are:

- a portfolio solver is robust even in prohibitive scenarios, like the MiniZinc Challenge, characterised by small-size test sets and unreliable solvers;
- in a multicore setting, a parallel portfolio of sequential solvers appears to be more fruitful than a single, parallel solver;
- `sunny-cp` can be a useful baseline to improve the state-of-the-art for (not only) the CP field, where dealing with non-reliable solvers must be properly addressed.

2 SUNNY and SUNNY-CP

In this section we provide a high-level description of `sunny-cp`, referring the interested reader to Amadini et al. (2015b; 2015a) for a more detailed presentation.

`sunny-cp` is an open-source CP portfolio solver. Its first implementation was sequential (Amadini et al. 2015b), while the current version exploits multicore architectures to run more solvers in parallel and to enable their cooperation via bounds sharing and restarting policies. To the best of our knowledge, it is currently the only parallel portfolio solver able to solve generic CP problems encoded in MiniZinc language (Nethercote et al. 2007).

`sunny-cp` is built on top of SUNNY algorithm (Amadini et al. 2014b). Given a set of known problems, a solving timeout T and a portfolio Π, SUNNY uses the *k-Nearest Neighbours* (k-NN) algorithm to produce a sequential schedule $\sigma = [(s_1, t_1), \ldots, (s_k, t_n)]$ where solver $s_i \in \Pi$ has to be run for t_i seconds and $\sum_{i=1}^{n} t_i = T$. The time slots t_i and the ordering of solvers s_i are defined according to the average performance of the solvers of Π on the k training instances closer to the problem to be solved.

For each problem p, a *feature vector* is computed and the Euclidean distance is used to retrieve the k instances in the training set closer to p. In a nutshell, a feature vector is a collection of numerical attributes that characterise a given problem instance. `sunny-cp` uses several features, e.g., statistics over the variables, the (global) constraints, the objective function (when applicable), the search heuristics. In total, `sunny-cp` uses 95 features.[2]

The sequential schedule σ is then parallelised on the $c \geq 1$ available cores by running the first and most promising $c - 1$ solvers in the k-neighbourhood on the

[2] The first version of `sunny-cp` also used graph features and dynamic features, afterwards removed for the sake of efficiency and portability. For more details about `sunny-cp` features, please see Amadini et al. (2014a) and `https://github.com/CP-Unibo/mzn2feat`.

Solver(s)	Description
Choco*, G12/FD, Gecode, JaCoP**, Mistral**, OR-Tools*	Finite Domain (FD) solvers
Chuffed, CPX, G12/LazyFD, Opturion CPX*	Lazy Clause Generation solvers
G12/CBC, MZN/Gurobi	MIP solvers
HaifaCSP*	Proof-producing CSP Solver
iZplus*	CP solver using local search and no-good techniques
MinisatID*	Combines techniques from SAT, SMT, CP, and ASP
Picat-SAT**	Encodes CP problems into SAT

Table 1: Constituent solvers of `sunny-cp`. The * symbol indicates the solvers introduced in MZNC 2015, while ** indicates those introduced in MZNC 2016.

first $c-1$ cores, while the remaining solvers (if any) are assigned to the last available core by linearly widening their allocated times to cover the time window $[0, T]$.

The notion of "promising solver" depends on the context. For CSPs, the performance is measured only in terms of number of solved instances and average solving time. For COPs, also the quality of the solutions is taken into account (Amadini and Stuckey 2014). We might say that `sunny-cp` uses a conservative policy: first, it skims the original portfolio by selecting a promising subset of its solvers; second, it allocates to each of these solvers an amount of time proportional to their supposed effectiveness.

Solvers are run in parallel and a *"bound-and-restart"* mechanism is used for enabling the bounds sharing between the running COP solvers (Amadini et al. 2015a). This allows one to use the (sub-optimal) solutions found by a solver to narrow the search space of the other scheduled solvers. If there are fewer solvers than cores, `sunny-cp` simply allocates a solver per core.

Since `sunny-cp` treats solvers as black boxes, it can not support the sharing of the bounds knowledge without the solvers interruption. For this reason, a restarting threshold T_r is used to decide when to stop a solver and restart it with a new bound. A running solver is stopped and restarted when: *(i)* it has not found a solution in the last T_r seconds; *(ii)* its current best bound is obsolete w.r.t. the overall best bound found by another scheduled solver.

Table 1 summarises the solvers used by `sunny-cp` in the MZNCs 2014–2016. For more details about these solvers, see Prud'homme et al. (2016), de Cat et al. (2013), Zhou and Kjellerstrand (2016), Veksler and Strichman (2016), MiniZinc (2016), JaCoP (2016), Mistral (2016), OR-Tools (2016), Chuffed (2016), Opturion CPX (2016), iZplus (2016).

Solver	Score
OR-Tools *	1098.85
Chuffed	*1034.81*
Choco *	973.27
Opturion CPX	929.76
Gecode *	858.24
iZplus	758.47
G12/LazyFD	*664.44*
Mistral	614.62
MZN/Gurobi	*589.38*
JaCoP	577.08
Fzn2smt	556.94
Gecoxicals	512.73
MZN/CPLEX	*447*
G12/FD	*426.53*
Numberjack *	**383.18**
Picat	363.02
G12/CBC	*118.69*

(a) MZNC 2013, Open category.

Solver	Score
Chuffed	*1326.02*
OR-Tools *	1086.97
Opturion CPX	1081.02
sunny-cp-seq-pre	***1066.46***
Choco *	1007.61
iZplus *	996.32
sunny-cp-seq	***968.64***
G12/LazyFD	784.28
HaifaCSP	781.72
Gecode *	*721.48*
SICStus Prolog	710.51
Mistral	705.56
MinisatID	588.74
Picat SAT	588.06
JaCoP	550.74
G12/FD	*528.26*
Picat CP	404.88
Concrete	353.74

(b) MZNC 2014, Open category.

Solver	Score
sunny-cp-seq-pre	***835.44***
Chuffed	*831.32*
sunny-cp-seq	***763.55***
Opturion CPX	621.73
OR-Tools	620.34
SICStus Prolog	555.61
Choco	503.29
MinisatID	472.90
Gecode	*482.61*
G12/LazyFD	*434.81*
JaCoP	405.66
G12/FD	*293.21*
Picat CP	291.53

(c) MZNC 2014, Fixed category with `sunny-cp` and MinisatID.

Table 2: Results of MZNCs 2013–2014. Portfolio solvers are in bold font, parallel solvers are marked with *, not eligible solvers are in italics.

3 SUNNY-CP and the MiniZinc Challenge

The MiniZinc Challenge (MZNC) (Stuckey et al. 2014) is the annual international competition for CP solvers. Portfolio solvers compete in the "Open" class of MZNC, where all solvers are free to use multiple threads or cores, and no search strategy is imposed.

The scoring system of the MZNC is based on a *Borda* count (Chevaleyre et al. 2007) where a solver s is compared against each other solver s' over 100 problem instances—belonging to different classes—defined in the MiniZinc language. If s gives a better answer than s' then it scores 1 point, if it gives a worse solution it scores 0 points. If s and s' give indistinguishable answers the scoring is based on the solving time.[3]

Until MZNC 2014, the solving timeout was 15 minutes and did not include the MiniZinc-to-FlatZinc conversion time. Starting from the MZNC 2015 this time has been included, and the timeout has been extended to 20 minutes.

3.1 MiniZinc Challenges 2013–2014

Table 2 summarizes the Open class results in the MZNCs 2013–2014. The first

[3] Please refer to `http://www.minizinc.org/challenge.html` for further details.

portfolio solver that attended a MiniZinc Challenge in 2013 (see Table 2a) was based on Numberjack platform (Hebrard et al. 2010). In the following years, `sunny-cp` was unfortunately the only portfolio solver that entered the competition.

In 2014, `sunny-cp` was a sequential solver running just one solver at time. We will denote it with `sunny-cp-seq` to distinguish such version from the current parallel one. `sunny-cp-seq` came with two versions: the default one and a version with pre-solving denoted in Table 2b as `sunny-cp-seq-pre`. In the latter a static selection of solvers was run for a short time, before executing the default version in the remaining time. Both of the two versions used the same portfolio of 7 solvers, viz. Chuffed, CPX, G12/FD, G12/LazyFD, Gecode, MinisatID, MZN/Gurobi. For more details, we refer the reader to Amadini et al. (2015b).

`sunny-cp-seq` improved on Numberjack and obtained respectable results: the two variants ranked 4^{th} and 7^{th} out of 18. `sunny-cp-seq` had to compete also with parallel solvers and all its solvers except MinisatID and MZN/Gurobi adopted the “fixed” strategy, i.e., they used the search heuristic defined in the problems instead of their preferred strategy. As described by Amadini et al. (2016a), we realised afterward that this choice is often not rewarding.

To give a measure of comparison, as shown in Table 2c, `sunny-cp-seq` in the “Fixed”[4] category—where sequential solvers must follow the search heuristic defined in the model—would have been ranked 1^{st} and 3^{rd}. Moreover, unlike other competitors, the results of `sunny-cp-seq` were computed by including also the MiniZinc-to-FlatZinc[5] conversion time since, by its nature, `sunny-cp` can not be a FlatZinc solver (see Amadini et al. (2015b) for more details). This penalised `sunny-cp-seq` especially for the easier instances.

3.2 MiniZinc Challenge 2015

Several enhancements of `sunny-cp-seq` were implemented after the MZNC 2014: *(i)* `sunny-cp` became parallel, enabling the simultaneous execution of its solvers while retaining the bounds communication for COPs; *(ii)* new state-of-the-art solvers were incorporated in its portfolio; *(iii)* `sunny-cp` became more stable, usable, configurable and flexible. These improvements, detailed by Amadini et al. (2015a) where `sunny-cp` has been tested on large benchmarks, have been reflected in its performance in the MZNC 2015.

`sunny-cp` participated in the competition with two versions: a default one and an “eligible” one, denoted `sunny-cp`$^-$ in Table 3. The difference is that `sunny-cp`$^-$ did not include solvers developed by the organisers of the challenge, and therefore was eligible for prizes. `sunny-cp`$^-$ used Choco, Gecode, HaifaCSP, iZplus, MinisatID, Opturion CPX and OR-Tools solvers, while `sunny-cp` used also Chuffed, MZN/Gurobi, G12/FD and G12/LazyFD. Since the availability of eight logical

[4] According to MZNC rules, each solver in the Fixed category that has not a Free version is automatically promoted in the Free category (analogously, solvers in the Free category can be entered in the Parallel category, and then in turn in the Open category).

[5] MZNC uses the MiniZinc language to specify the problems, while the solvers use the lower level specification language FlatZinc, which is obtained by compilation from MiniZinc models.

Solver	Score	Incomplete
sunny-cp *	***1351.13***	***1175.2***
Chuffed	*1342.37*	*1118.16*
sunny-cp⁻ *	**1221.88**	**1156.25**
OR-Tools *	1111.83	1071.67
Opturion CPX	1094.09	1036.65
Gecode *	*1049.49*	*979.05*
Choco *	1027.65	989
iZplus *	1021.13	1082.92
JaCoP	914.97	865.64
Mistral *	872.35	878.53
MinisatID	835.01	793.74
MZN/CPLEX *	*799.92*	*686.64*
MZN/Gurobi *	*774.3*	*697.12*
Picat SAT	744.53	626.61
MinisatID-MP	637.14	700.35
G12/FD	*629.94*	*664.79*
Picat CP	617.22	654.81
Concrete	533.42	657.2
YACS *	404.01	553.51
OscaR/CBLS	403.61	536.17

(a) Open category.

Solver	Score	Incomplete
sunny-cp *	***1423.58***	***1256.65***
Chuffed	*1387.95*	*1166.56*
sunny-cp⁻ *	**1304.39**	**1240.88**
Opturion CPX	1146.18	1091.76
iZplus	1070.15	1093.26
OR-Tools	994.41	917.17
Mistral	960.16	937.01
JaCoP-fd	912.1	838.77
Gecode	*908.32*	*867.82*
Choco	864.39	887.08
MinisatID	828.2	791.23
MZN/CPLEX	*786.11*	*698.77*
Picat SAT	780.13	709.62
MZN/Gurobi	*724.27*	*654.7*
MinisatID-MP	623.58	688.47
Picat CP	618.78	633.61
G12/FD	*589.65*	*607.02*
Concrete	560.16	676.08
YACS	458.81	601.81
OscaR/CBLS	418.67	539.73

(b) Free Category with `sunny-cp`.

Table 3: Results of MZNC 2015. Portfolio solvers are in bold font, parallel solvers are marked with *, not eligible solvers are in italics.

cores, `sunny-cp` performed algorithm selection for computing and distributing the SUNNY sequential schedule, while `sunny-cp`$^-$ launched all its solvers in parallel.

Table 3 shows that `sunny-cp` is the overall best solver while `sunny-cp`$^-$ won the gold medal since Chuffed—the best sequential solver—was not eligible for prizes. The column "Incomplete" refers to the MZNC score computed without giving any point for proving optimality or infeasibility. This score, meant to evaluate local search solvers, only takes into account the quality of a solution. Note that with this metric also `sunny-cp`$^-$ overcomes Chuffed, without having it in the portfolio.

Several reasons justify the success of `sunny-cp` in MZNC 2015. Surely the parallelisation on multiple cores of state-of-the-art solvers was decisive, especially because it was cooperative thanks to bounds sharing mechanism. Moreover, differently from MZNC 2014, all the solvers were run with their free version instead of the fixed one. Furthermore, the MZNC rules were less penalising for portfolio solvers since for the first time in the history of the MZNCs the total solving time included also the MiniZinc-to-FlatZinc conversion time.

We underline that the constituent solvers of `sunny-cp` do not exploit multi-threading. Hence, the parallel solvers marked with * in Table 3a are not the constituent solvers of `sunny-cp` but their (new) parallel variants.

The overall best single solver is Chuffed, which is sequential. Having it in the portfolio is clearly a benefit for `sunny-cp`. However, even without Chuffed, `sunny-cp`$^-$ is able to provide solutions of high quality (see "Incomplete" column of Table 3) proving that also the other solvers are important for the success of `sunny-cp`. We remark that—as pointed out also by Amadini et al. (2015b)—when compared to the best solver for a given problem, a portfolio solver always has additional overheads (e.g., due to feature extraction or memory contention issues) that penalise its score.

The 100 problems of MZNC 2015 are divided into 20 different problem classes, each of which consisting of 5 instances: in total, 10 CSPs and 90 COPs. `sunny-cp` was the best solver for only two classes: `cvrp` and `freepizza`. Interestingly, for the whole `radiation` problem class, `sunny-cp`$^-$ scored 0 points because it always provided an unsound answer due to a buggy solver. This is a sensitive issue that should not be overlooked. On the one hand, a buggy solver inevitably affects the whole portfolio making it buggy as well. On the other hand, not using an unstable solver may penalize the global performance since experimental solvers like Chuffed and iZplus can be very efficient even if not yet in a stable stage.

As we shall see also in Section 3.3, unlike SAT but similarly to SMT field, most CP solvers are not fully reliable (e.g., in MZNC 2014 one third of the solvers provided at least an unsound answer). When unreliable solvers are used, a possible way to mitigate the problem is to verify *a posteriori* the solution. For instance, another constituent solver can be used for double-checking a solution. Obviously, checking all the solutions of all the solvers implies a slowdown in the solving time. Note that the biggest problems arise when the solver does not produce a solution or when it declares a sub-optimal solution as optimal. In the first case, since solvers usually do not present a proof of the unsatisfiability, checking the correctness of the answer requires solving the same problem from scratch. In the second case, the presence of a solution may simplify the check of the answer, but checking if a solution is optimal is still an NP-hard problem.

In MZNC 2015 `sunny-cp` checked HaifaCSP, since its author warned us about its unreliability. This allowed `sunny-cp` to detect 21 incorrect answers. Without this check its performance would have been dramatically worse: `sunny-cp` would have scored 87.5 points less—thus resulting worse than Chuffed—while `sunny-cp`$^-$ would have scored 206.84 points less, passing from the gold medal to no medal. However, this check was not enough: due to bugs in other constituent solvers `sunny-cp` provided 5 wrong answers, while `sunny-cp`$^-$ provided 7 wrong answers.

3.3 MiniZinc Challenge 2016

In the MiniZinc Challenge 2016 we enrolled three versions, namely: `sunny-cp`, `sunny-cp`$^-$, and `sunny-cp`$^{--}$.

`sunny-cp` was not eligible for prizes and added to the portfolio of MZNC 2015 three new solvers: JaCoP, Mistral, and Picat-SAT.

Solver	Score	Incomplete
LCG-Glucose	*1899.23*	*1548.2*
***sunny-cp* ***	***1877.79***	***1616.19***
Chuffed	*1795.57*	*1486.8*
LCG-Glucose-UC	*1671.52*	*1306.26*
sunny-cp^{--} *	**1620.82**	**1486.11**
*MZN/Gurobi **	*1499.04*	*1308.18*
HaifaCSP	1448.35	1343.54
*MZN/CPLEX **	*1436.05*	*1287.09*
Picat SAT	1423.81	1336.36
iZplus *	1374.12	1446.36
sunny-cp^{-} *	**1365.31**	**1205.73**
Choco *	1342.41	1390.21
OR-Tools *	1115.8	1258.51
*Gecode **	*1110.19*	*1137.21*
MinisatID *	992.12	1002.17
MZN/SCIP	*985.37*	*1011.25*
JaCoP	923.78	1041.03
Mistral *	826.61	935.8
MZN/CBC	*754.77*	*827.06*
SICStus Prolog	754.33	837.57
G12/FD	703.14	829.39
Concrete	583.9	627.36
Picat CP	475.47	651.63
OscaR/CBLS	468.5	708
Yuck *	316	412

(a) Open category.

Solver	Score	Incomplete
***sunny-cp* ***	***1054.83***	***928.95***
LCG-Glucose	*1029.43*	*876.56*
Chuffed	*993.79*	*844.42*
sunny-cp^{-} *	**982.7**	**893.39**
LCG-Glucose-UC	*929.28*	*748.17*
sunny-cp^{--} *	**899.47**	**875.6**
*MZN/Gurobi **	*862.26*	*705.18*
*MZN/CPLEX **	*829.12*	*704.59*
iZplus *	779.88	778.98
HaifaCSP	777.91	775.48
Picat SAT	735.82	713.71
Choco *	700.46	765.13
Gecode *	633	639.35
OR-Tools *	560.5	659.38
MZN/SCIP	*545.85*	*535.75*
MinisatID *	498.33	539.69
SICStus Prolog	437.33	510.66
JaCoP	433.76	555.49
MZN/CBC	*421.32*	*453.06*
Mistral *	382.68	470.87
G12/FD	*374.56*	*430.27*
Concrete	291.42	327.7
Picat CP	260.79	334.13
OscaR/CBLS	216.5	286.5
Yuck *	171	181

(b) Open Category without the instances on which `sunny-cp`$^{-}$ failed.

Table 4: Open class results of MZNC 2016. Portfolio solvers are in bold font, parallel solvers are marked with *, not eligible solvers are in italics.

`sunny-cp`$^{-}$ contained only the eligible solvers of `sunny-cp`, i.e., Choco, Gecode, HaifaCSP, JaCoP, iZplus, MinisatID, Mistral, Opturion CPX, OR-Tools, Picat.[6]

`sunny-cp`$^{--}$ contained only the solvers of `sunny-cp`$^{-}$ that never won a medal in the Free category of the last three challenges, i.e., Gecode, HaifaCSP, JaCoP, MinisatID, Mistral, Picat.

Ideally, we aimed to measure the contribution of the supposedly best solvers of `sunny-cp`$^{-}$. Conversely, to `sunny-cp` and `sunny-cp`$^{-}$, `sunny-cp`$^{--}$ does not need to schedule its solvers, having fewer solvers than available cores. It just launches all its solvers in parallel.

Table 4a shows the Open category ranking of the MZNC 2016. These results

[6] We did not have an updated version of Choco and Opturion solvers, so we used their 2015 version.

are somehow unexpected if compared with those of the previous editions. For the first time, Chuffed has been outperformed by a sequential solver, i.e., the new, experimental LCG-Glucose—a lazy clause generation solver based on Glucose SAT solver. Surprisingly, solvers like OR-Tools, iZplus, Choco had subdued performance. Conversely, HaifaCSP and Picat-SAT performed very well. The sharp improvement of the solvers based on Gurobi and CPLEX is also clear, arguably due to a better linearisation of the MiniZinc models (Belov et al. 2016). Local search solvers still appear immature.

The results of `sunny-cp` are definitely unexpected. In particular, it appears quite strange that `sunny-cp`$^-$ performed far worse than `sunny-cp`$^{--}$ although having more, and ideally better, solvers. We then thoroughly investigated this anomaly since, as also shown in Amadini et al. (2015a; 2016a), the dynamic scheduling of the available solvers is normally more fruitful than statically running an arbitrarily good subset of them over the available cores.

Firstly, we note that for the easier instances `sunny-cp`$^{--}$ is inherently faster than `sunny-cp` and `sunny-cp`$^-$ because it does not need to schedule its solvers, and therefore it skips the feature extraction and the algorithm selection phases of the SUNNY algorithm (Amadini et al. 2014b). Nevertheless, most of the MZNC 2016 instances are not easy to solve.

Another reason is that `sunny-cp`$^{--}$ always runs HaifaCSP and Picat-SAT, two solvers that performed better than expected, while `sunny-cp`$^-$ executes Picat-SAT only for 37 problems. Nonetheless, `sunny-cp`$^-$ always executes HaifaCSP so also this explanation can not fully explain the performance difference.

The actual reason behind the performance gap relies on some *buggy solvers* which belongs to `sunny-cp`$^-$ but not to `sunny-cp`$^{--}$.[7] In our pre-challenge tests we did not notice inconsistencies in any of the solvers, except for Choco. So we decided to check the solutions only for Choco and HaifaCSP (the latter because of the unreliability shown in the MZNC 2015, see Section 3.2). However, none of these solvers gave an unsound outcome in the MZNC 2016. Conversely, Opturion and OR-Tools solvers provided a lot of incorrect, and unfortunately unchecked, answers. We also noticed that for some instances our version of Mistral failed when restarted with a new bound, while on the same instances the Free version of Mistral provided a sound outcome.

In total, `sunny-cp`$^-$ gave 24 wrong answers,[8] meaning that it competed only on the 76% of the problems of MZNC 2016. `sunny-cp`$^{--}$ failed instead on 5 instances.

Table 4b shows the results without the 24 instances for which `sunny-cp`$^-$ gave an incorrect answer. We underline that this table has a purely indicative value: for a more comprehensive comparison, also the instances where other solvers provided an incorrect answer should be removed. On these 76 problems `sunny-cp` overcomes LCG-Glucose, while `sunny-cp`$^-$ impressively gains 7 positions and becomes gold

[7] With the term "buggy solver" we not necessarily mean that the solver itself is actually buggy. The problems may arise due to a misinterpretation of the FlatZinc instances or to the wrong decomposition of global constraints (Rossi et al. 2006).

[8] Namely, all the 5 instances of `depot-placement`, `gfd-schedule`, and `nfc` classes; 4 instances of `tpp` class; 1 instance of `cryptanalysis`, `filter`, `gbac`, `java-auto-gen`, `mapping` classes.

medallist being the first of the eligible solvers. sunny-cp^{--} however behaves well (silver medallist), being overtaken by sunny-cp^{-} only.

Note that the results of sunny-cp are good also in the original ranking of Table 4a since, being this version not eligible for prizes, the organisers enabled the solutions checking of G12/LazyFD, HaifaCSP, Mistral, Opturion, OR-Tools. This allowed sunny-cp to detect 19 incorrect answers.

An interesting insight is given by the Incomplete score, which does not give any benefit when a solver concludes the search (i.e., when optimality or unsatisfiability is proven). As observed also in Section 3.2, with this metric sunny-cp can significantly overcome a solver that has a greater score (e.g., see the Incomplete sunny-cp in Table 4a). This confirms the attitude of sunny-cp in finding good solutions even when it does not conclude the search.

4 Conclusions

We presented an overview of sunny-cp, a fairly recent CP portfolio solver relying on the SUNNY algorithm, and we discussed its performance in the MiniZinc Challenge—the annual international competition for CP solvers.

In the MiniZinc Challenge 2014 sunny-cp received an honourable mention, in 2015 it has been the first portfolio solver to win a (gold) medal, and in 2016—despite several issues with buggy solvers—it confirmed the first position.

For the future of CP portfolio solvers, it would be interesting having more portfolio competitors to improve the state of the art in this field. Different portfolio approaches have been already compared w.r.t. sunny-cp and its versions (Amadini et al. 2014b; Amadini et al. 2016b; Amadini et al. 2015; Lindauer et al. 2016).

The Algorithm Selection approaches of the ICON Challenge 2015 (Kotthoff 2015) might be adapted to deal with generic CP problems. The SUNNY algorithm itself, which is competitive in the CP scenarios of [Amadini et al.; Amadini et al. 2014b; 2016b],[9] provided very poor performance in the SAT scenarios of the ICON Challenge and Lindauer et al. (2016) show that it can be strongly improved with a proper training phase.

sunny-cp runs in parallel different single-threaded solvers. This choice so far has proved to be more fruitful than parallelising the search of a single solver. However, the possibility of using multi-threaded solvers may have some benefits when solving hard problems as shown by Malitsky et al. (2012) for SAT problems.

The multi-threaded execution also enables search splitting strategies. It is not clear to us if the use of all the available cores, as done by sunny-cp, is the best possible strategy. As shown by Sabharwal and Samulowitz (2014) it is possible that running in parallel all the solvers on the same multicore machine slows down the execution of the individual solvers. Therefore, it may be more convenient to leave free one or more cores and run just the most promising solvers. Unfortunately, it

[9] We submitted such scenarios, namely CSP-MZN-2013 and COP-MZN-2013, to the Algorithm Selection Library (coseal 2014).

is hard to extrapolate a general pattern to understand the interplay between the solvers and their resource consumption.

One direction for further investigations, clearly emerged from the challenge outcomes, concerns how to deal with unstable solvers. Under these circumstances it is important to find a trade-off between reliability and performance. Developing an automated way of checking a CP solver outcome when the answer is "unsatisfiable problem" or "optimal solution" is not a trivial challenge: we can not merely do a solution check, but we have to know and check the actual *explanation* for which the solver provided such an outcome.

A major advancement for CP portfolio solvers would be having API for injecting constraints at runtime, without stopping a running solver. Indeed, interrupting a solver means losing all the knowledge it has collected so far. This is particularly bad for Lazy Clause Generation solvers, and in general for every solver relying on no-good learning.

Another interesting direction for further studies is to consider the impact of the global constraints (Rossi et al. 2006) on the performances of the portfolio solver. It is well-known that the propagation algorithms and the decompositions used for global constraints are the keys of solvers effectiveness. We believe that the use of solvers supporting different global constraint decompositions may be beneficial.

We underline that—even if focused on Constraint Programming—this work can be extended to other fields, e.g., Constraint Logic Programming, Answer-Set Programming or Planning, where portfolio solving has been used only marginally.

To conclude, in order to follow the good practice of making the tools publicly available and easy to install and use, we stress that `sunny-cp` is publicly available at `https://github.com/CP-Unibo/sunny-cp` and can be easily installed, possibly relying on the Docker container technology for avoiding the installation of its constituent solvers. All the results of this paper can be reproduced and verified by using the web interface of `http://www.minizinc.org/challenge.html`.

Acknowledgements

We are grateful to all the authors and developers of the constituent solvers of `sunny-cp`, for providing us the tools and the instructions to use the solvers.

We thank all the MiniZinc Challenge staff, and in particular Andreas Schutt, for the availability and technical support.

This work was supported by the EU project FP7-644298 HyVar: Scalable Hybrid Variability for Distributed, Evolving Software Systems.

References

Amadini, R., Biselli, F., Gabbrielli, M., Liu, T., and Mauro, J. 2015. SUNNY for algorithm selection: a preliminary study. In *Proceedings of the 30th Italian Conference on Computational Logic, Genova, Italy, July 1-3, 2015.*, D. Ancona, M. Maratea, and V. Mascardi, Eds. CEUR Workshop Proceedings, vol. 1459. CEUR-WS.org, 202–206.

Amadini, R., Gabbrielli, M., and Mauro, J. 2014a. An enhanced features extractor for a portfolio of constraint solvers. In *Symposium on Applied Computing, SAC 2014,* *Gyeongju, Republic of Korea - March 24 - 28, 2014*, Y. Cho, S. Y. Shin, S. Kim, C. Hung, and J. Hong, Eds. ACM, 1357–1359.

Amadini, R., Gabbrielli, M., and Mauro, J. 2014b. SUNNY: a Lazy Portfolio Approach for Constraint Solving. *TPLP 14,* 4-5, 509–524.

Amadini, R., Gabbrielli, M., and Mauro, J. 2015a. A Multicore Tool for Constraint Solving. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, Q. Yang and M. Wooldridge, Eds. AAAI Press, 232–238.

Amadini, R., Gabbrielli, M., and Mauro, J. 2015b. SUNNY-CP: a sequential CP portfolio solver. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, R. L. Wainwright, J. M. Corchado, A. Bechini, and J. Hong, Eds. ACM, 1861–1867.

Amadini, R., Gabbrielli, M., and Mauro, J. 2015c. Why CP portfolio solvers are (under)utilized? Issues and challenges. In *Logic-Based Program Synthesis and Transformation - 25th International Symposium, LOPSTR 2015, Siena, Italy, July 13-15, 2015. Revised Selected Papers*, M. Falaschi, Ed. Lecture Notes in Computer Science, vol. 9527. Springer, 349–364.

Amadini, R., Gabbrielli, M., and Mauro, J. 2016a. Parallelizing Constraint Solvers for Hard RCPSP Instances. In *Learning and Intelligent Optimization - 10th International Conference, LION 10, Ischia, Italy, May 29 - June 1, 2016, Revised Selected Papers*, P. Festa, M. Sellmann, and J. Vanschoren, Eds. Lecture Notes in Computer Science, vol. 10079. Springer, 227–233.

Amadini, R., Gabbrielli, M., and Mauro, J. 2016b. Portfolio approaches for constraint optimization problems. *Annals of Mathematics and Artificial Intelligence 76,* 1-2, 229–246.

Amadini, R. and Stuckey, P. J. 2014. Sequential Time Splitting and Bounds Communication for a Portfolio of Optimization Solvers. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, B. O'Sullivan, Ed. Lecture Notes in Computer Science, vol. 8656. Springer, 108–124.

Belov, G., Stuckey, P. J., Tack, G., and Wallace, M. 2016. Improved Linearization of Constraint Programming Models. In *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proc.*, M. Rueher, Ed. Lecture Notes in Computer Science, vol. 9892. Springer, 49–65.

Chevaleyre, Y., Endriss, U., Lang, J., and Maudet, N. 2007. A Short Introduction to Computational Social Choice. In *SOFSEM 2007: Theory and Practice of Computer Science, 33rd Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 20-26, 2007, Proceedings*, J. van Leeuwen, G. F. Italiano, W. van der Hoek, C. Meinel, H. Sack, and F. Plasil, Eds. Lecture Notes in Computer Science, vol. 4362. Springer, 51–69.

Chuffed. 2016. Chuffed Solver. `https://github.com/geoffchu/chuffed`.

coseal. 2014. Algorithm Selection Library. `https://code.google.com/p/coseal/wiki/AlgorithmSelectionLibrary`.

de Cat, B., Bogaerts, B., Devriendt, J., and Denecker, M. 2013. Model Expansion in the Presence of Function Symbols Using Constraint Programming. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6, 2013.* IEEE Computer Society, 1068–1075.

Gomes, C. P. and Selman, B. 2001. Algorithm portfolios. *Artificial Intelligence 126,* 1-2, 43–62.

Hebrard, E., O'Mahony, E., and O'Sullivan, B. 2010. Constraint Programming and Combinatorial Optimisation in Numberjack. In *Integration of AI and OR Techniques* *in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proc.*, A. Lodi, M. Milano, and P. Toth, Eds. Lecture Notes in Computer Science, vol. 6140. Springer, 181–185.

Hutter, F., Xu, L., Hoos, H. H., and Leyton-Brown, K. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence 206*, 79–111.

iZplus. 2016. iZplus Solver Description. https://www.minizinc.org/challenge2016/description_izplus.txt.

JaCoP. 2016. JaCoP Solver. http://jacop.osolpro.com/.

Kotthoff, L. 2014. Algorithm Selection for Combinatorial Search Problems: A Survey. *AI Magazine 35,* 3, 48–60.

Kotthoff, L. 2015. ICON challenge on algorithm selection. *CoRR abs/1511.04326*.

Lindauer, M., Bergdoll, R., and Hutter, F. 2016. An Empirical Study of Per-instance Algorithm Scheduling. In *Learning and Intelligent Optimization - 10th International Conference, LION 10, Ischia, Italy, May 29 - June 1, 2016, Revised Selected Papers*, P. Festa, M. Sellmann, and J. Vanschoren, Eds. Lecture Notes in Computer Science, vol. 10079. Springer, 253–259.

Malitsky, Y., Sabharwal, A., Samulowitz, H., and Sellmann, M. 2012. Parallel SAT Solver Selection and Scheduling. In *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, Canada, October 8-12, 2012. Proc.*, M. Milano, Ed. Lecture Notes in Computer Science, vol. 7514. Springer, 512–526.

MiniZinc. 2016. MiniZinc Software. https://www.minizinc.org/software.html.

Mistral. 2016. Mistral Solver. https://github.com/ehebrard/Mistral-2.0.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. 2007. MiniZinc: Towards a Standard CP Modelling Language. In *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, C. Bessiere, Ed. Lecture Notes in Computer Science, vol. 4741. Springer, 529–543.

O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., and O'Sullivan, B. 2008. Using case-based reasoning in an algorithm portfolio for constraint solving. *Irish conference on artificial intelligence and cognitive science*.

Opturion CPX. 2016. Opturion CPX Solver. www.opturion.com/cpx.

OR-Tools. 2016. OR-Tools Solver. https://github.com/google/or-tools.

Prud'homme, C., Fages, J.-G., and Lorca, X. 2016. *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S.

Rice, J. R. 1976. The Algorithm Selection Problem. *Advances in Computers 15*, 65–118.

Rossi, F., Beek, P. v., and Walsh, T. 2006. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA.

Sabharwal, A. and Samulowitz, H. 2014. Insights into Parallelism with Intensive Knowledge Sharing. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, B. O'Sullivan, Ed. Lecture Notes in Computer Science, vol. 8656. Springer, 655–671.

Smith-Miles, K. 2008. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys 41,* 1, 6:1–6:25.

Stuckey, P. J., Feydy, T., Schutt, A., Tack, G., and Fischer, J. 2014. The MiniZinc Challenge 2008-2013. *AI Magazine 35,* 2, 55–60.

Veksler, M. and Strichman, O. 2016. Learning general constraints in CSP. *Artificial Intelligence 238*, 135–153.

Zhou, N. and Kjellerstrand, H. 2016. The Picat-SAT Compiler. In *Practical Aspects of Declarative Languages - 18th International Symposium, PADL 2016, St. Petersburg,* *FL, USA, January 18-19, 2016. Proceedings*, M. Gavanelli and J. H. Reppy, Eds. Lecture Notes in Computer Science, vol. 9585. Springer, 48–62.