An Extensive Evaluation of Portfolio Approaches for Constraint Satisfaction Problems

[1]Roberto Amadini, [2]Maurizio Gabbrielli and [3]Jacopo Mauro

[1]*University of Melbourne, Australia*
[2]*University of Bologna, Italy*
[3]*University of Oslo, Norway*

***Abstract*— In the context of Constraint Programming, a portfolio approach exploits the complementary strengths of a portfolio of different constraint solvers. The goal is to predict and run the best solver(s) of the portfolio for solving a new, unseen problem. In this work we reproduce, simulate, and evaluate the performance of different portfolio approaches on extensive benchmarks of Constraint Satisfaction Problems. Empirical results clearly show the benefits of portfolio solvers in terms of both solved instances and solving time.**

***Keywords* — Algorithm Selection, Algorithm Portfolios, Constraint Programming, Constraint Satisfaction Problems.**

I. Introduction

In the context of *Constraint Programming* (CP) [40] a portfolio approach [13,17] combines $m > 1$ different solvers $S_1, ..., S_m$ to get a globally better solver, dubbed a *portfolio solver*. When a new, unseen problem p comes, the portfolio solver seeks to predict and run the best constituent solver(s) $S_{i_1}, ..., S_{i_k}$ (with $1 \leq i \leq m$ for $j = 1, ..., k$) for solving p. Portfolio approaches can be seen as instances of the *Algorithm Selection* problem [39] where, as reported by [25], the algorithm selection is performed case-by-case according to the problem to solve.

Portfolio solvers have proven to be very efficient, especially for solving the Boolean satisfiability (SAT) problem. For instance, the SAT portfolio solvers 3S [21] and CSHC [30] won gold medals in the SAT Competition 2011 and 2013, while SATZilla [53] won the SAT Challenge 2012. Unfortunately, in the CP field fewer portfolio solvers have been proposed. In this regard, worth mentioning are CPHydra [35] that won the International Constraint Solver Competition 2008 [49] and `sunny-cp` [5] that won the MiniZinc Challenge 2015 [48]. This witnesses that portfolio approaches can be effective also in the CP domain [6], and that the research in this field is not merely theoretical: many real life applications might take advantage of portfolio solvers for solving daily life problems such as, for example, task scheduling or resource allocation problems [1, 36].

With the aim of deepening the study of portfolio solving in the CP field, in this paper we extend the research initiated by [2] by presenting a more recent and exhaustive evaluation of portfolio approaches for solving Constraint Satisfaction Problems (CSPs). Improvements are manifold: we evaluate more recent solvers, more features, we fully support the MiniZinc language [33], and we use a larger dataset of CSP instances. The obtained results are encouraging and confirm the effectiveness of portfolio solvers in terms of both solved instances and solving time.

Unfortunately, due to the difficulties in using and adapting to the CSP domain some approaches originally designed for SAT, most of the portfolio solvers we tested have been reimplemented as faithfully as possible. Hence, the goal of this paper is not to present a (possibly unfair) competition between portfolio solvers. We want instead to shed further light on CSP portfolio approaches by means of empirical evaluations. In this regard, we submitted the data and the results we computed to the *Algorithm Selection library* [9], an open-access library providing a standardized format for representing, evaluating, and comparing different portfolio approaches without the effort of rebuilding all the experimental environment.

Paper structure. Section 2 gives some background notions on CSP portfolio solvers. Section 3 explains the experimental methodology, while Section 4 describes the obtained results. In Section 5 we report the related literature and the concluding remarks.

II. Background

A *Constraint Satisfaction Problem* (CSP) is a triple consisting of a set of variables each of which associated with a domainof values thatcould take, and a set of constraints defining all the admissible assignments of values to variables [27]. The goal is normally to find a solution, i.e., a variable assignment satisfying all the constraints of the problem, by using a suitable constraint solver. In this context, a *portfolio solver* can be seen as a meta-solver consisting of $m > 1$ different solvers $S_1, ..., S_m$. *When a new, unseen CSP instance p comes, the portfolio solver seeks to predict and run the best constituent solver(s) for solving p. In the rest of the section we give a brief overview of the main ingredients characterizing a CSP portfolio solver, namely: the dataset of CSPs used to make (and test) predictions, the solvers of the portfolio, the features characterizing each CSP, and the selection algorithms used for deciding the solver(s) to run on a given CSP.*

A. Dataset, Solvers and Features

In order to build and test a good portfolio approach it is fundamental to gather an adequate *dataset* of CSPs. The data sample should capture a significant variety of problems encoded in the same language. Although nowadays the CP community has not yet agreed on a standard modelling language, *MiniZinc* [33] is probably the most used and supported language to model CP problems. However, the biggest existing dataset of CSPs we aware is the one used in the 2008 International Constraint Solver Competition (ICSC) [49]. These instances are encoded in the XML-based language XCSP [41]. In [2] an empirical evaluation on such a dataset was conducted. Here we take a step forward by exploiting the `xcsp2mzn` [3] compiler we developed for converting XCSP to MiniZinc. This allowed us to use a bigger benchmark of 8600 CSPs: 6944 instances of ICSC converted by `xcsp2mzn`, and 1656 native MiniZinc instances coming from the MiniZinc 1.6 benchmarks and the MiniZinc Challenge 2012.

A portfolio solver contains a number of different constituent solvers that clearly should be as effective as possible. However, the individual performance of a solver is not the only key to success: what really matters is the contribution of a solver to the portfolio performance [51]. For this reason, increasing the number of constituent solvers does not necessarily mean increasing the performance of a portfolio. Conversely,

DOI: 10.9781/ijimai.2016.3712

having too many candidates solvers can make the solvers prediction inefficient and especially inaccurate. In this work we consider (subsets of) a collection of 11 different solvers that attended the MiniZinc Challenge, namely: BProlog, Fzn2smt, CPX, G12/FD, G12/LazyFD, G12/CBC, Gecode, iZplus, MinisatID, Mistral, and OR-Tools.

Usually portfolio solvers decide the solver(s) to run according to a set of *features* extracted from the instance to solve. Features are specific attributes characterizing a given problem instance, and are clearly of paramount importance for the success of a portfolio approach [39]. Features can be divided in *static* (computed off-line according to the problem specification) and *dynamic* (computed at runtime by monitoring the problem resolution). In this paper we used `mzn2feat` [3] to extract a set of 155 features (144 static, 11 dynamic) from a MiniZinc instance. For more details about such features we refer the interested reader to [3].

B. Algorithm Selection

There are several ways to select one or more constituent solver(s) for solving a given instance. A primary distinction can be done between the approaches that require training and the so-called *lazy* approaches [25] that do not need it. For the former, the training phase is usually performed off-line and empirical evidences prove that a good training can lead to very good performance (e.g., see [50,21,31,30]). However, avoiding the training phase can be clearly advantageous in terms of simplicity and flexibility: new information can be used to improve the predictions without rebuilding the prediction model. For this reasons some lazy approaches have been proposed in the literature (e.g., see [35,37,34,11,43,4]). A further distinction can be made between algorithms that run just one solver and those that schedule more solvers. These may have some practical advantages since they reduce the risk of choosing a wrong solver. Furthermore, scheduling more solvers enables the communication of potentially relevant information such as bounds [6] or nogoods [23].

In this work we considered different selectors disparate in their nature. We implemented and adapted them to the CSP domain trying to be as faithful as possible to their original concept. In particular, we compared the performance of off-the-shelf *Machine Learning* (shortly, ML) classifiers against some well-known portfolio approaches, namely: CPHydra [35], ISAC [22], 3S [21], SATzilla [50], and SUNNY [4]. In the following we provide a brief overview of such approaches.

Off-the-shelf (OTS) are selectors that rely on off-the-shelf ML classification algorithms to predict the best solver to run for a given instance. Thanks to WEKA [14] we implemented a number of well-known OTS selectors based on well-know classifiers, namely: *IBk* (*k*-Nearest Neighbours), *J48* (4.5 decision trees), *PART* (PART decision lists), *RF* (Random Forests), and *SMO* (Support Vector Machines).

CPHydra [35] is the first general CSP portfolio solver proposed in the literature. It uses a *k*-Nearest Neighbour (*k*-NN) algorithm for computing a schedule of its constituent solvers according to the *k*-neighbours runtimes. The schedule is computed by solving a generalization of a knapsack problem. CPHydra won the ICSC 2008.

ISAC [22] is a configuration tool that aims at optimally configuring a highly parametrized algorithm. In this work we use the ISAC ``*Pure Solver Portfolio*'' approach following what done by [31] in the SAT field. The training instances are clustered and the solver that solves the most instances in the cluster closer to the instance to be solved is selected.

3S [21] is a SAT solver conjugating a fixed-time static solver schedule (computed off-line) with the dynamic selection of one long-running solver. This solver is chosen with a *k*-NN algorithm and is eventually executed after the static schedule. 3S was the best dynamic portfolio in the SAT Competition 2011.

SATzilla [52] is a SAT solver relying on runtime prediction models. Its last version [51] uses a weighted random forest approach provided with a cost-sensitive loss function for punishing misclassifications in direct proportion to their performance impact. SATzilla won the SAT Challenge 2012.

SUNNY [4] is a lazy algorithm portfolio using a *k*-NN algorithm for selecting a sub-portfolio of solvers to run. Solvers are scheduled according to their performance in the neighbourhood. `sunny-cp` [5], a parallel portfolio solver built on top of the SUNNY algorithm [4], won the MiniZinc Challenge 2015.

III. Methodology

In this section we explain the methodology used for conducting the experiments. Following what is usually done by most of the approaches, we first removed all the constant features and we scaled all the non-constant ones in the range [-1, 1], ending up with a reduced set of 114 features. Fixed a timeout of $T = 1800$ seconds[1], we then filtered the dataset of the 8600 CSPs mentioned in Section II-A by removing the ``easiest'' instances (i.e., those solved when computing the dynamic features) and the ``hardest'' ones (i.e., those for which the feature extraction required more than $T/2 = 900$ seconds). We discarded the easiest since if an instance is already solved during the feature extraction, then no solver prediction is needed. The hardest ones were instead discarded since if the extraction takes more than $T/2$ seconds, then recompiling the MiniZinc model into FlatZinc (a step needed to run the solvers) would take at least other $T/2$ seconds, therefore consuming all the time slot available. The final dataset Δ on which we conducted the experiments was constituted by 4642 MiniZinc instances (3538 from ICSC, 6 from MiniZinc Challenge 2012, and 1098 from MiniZinc 1.6 benchmarks).

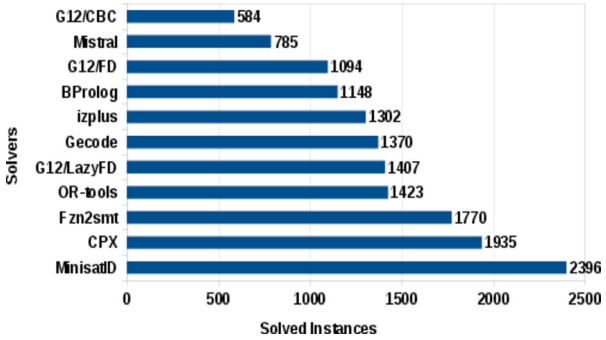


Fig. 1. Total number of solved instances for each solver of the portfolio.

We ran all the 11 solvers listed in Section II-A on each of the 4642 instances of Δ, thus solving 51062 CSPs[2]. We ran all of the solvers with their default parameters, their specific FlatZinc redefinitions, and keeping track of their performance within the timeout T. We then built ten portfolios $\prod_m$ of different size $m = 2, \ldots, 11$ where $\prod_m$ is the portfolio with cardinality m maximizing the number of solved instances in Δ (we used the average solving time for breaking ties). Unlike other approaches, following [2], we decided to keep in Δ the 944 CSPs not solvable by any solver. We took this decision since these instances could affect the behavior of a portfolio approach. For example, SUNNY allocates to a predesignated backup solver an amount of time proportional to the instances of the *k*-neighborhood that no solver can solve.

The single solvers performance are listed in Fig. 1. The *Single Best Solver* (SBS) of the portfolio is MinisatID [10] since it solves

1 The same timeout used in the ICSC (the timeout of MiniZinc Challenge is lower).
2 We used Intel Dual-Core 2.93GHz computers with 2 GB of RAM and Ubuntu operating system.

the greatest number of instances. Each of the portfolio approaches described in Section II-B has been simulated and evaluated using a 5-repeated 5-fold cross validation [8]. We evaluated the performance of each approach in terms of *Average Solving Time* (AST)[3] and *Percentage of Solved Instances* (PSI) within *T* seconds.

IV. Results

This section presents the obtained results. In addition to the SBS and the portfolio approaches, we add to the evaluation the *Virtual Best Solver* (VBS) baseline. The VBS is an "oracle" solver always selecting the best solver of the portfolio for any given instance. For all the reimplemented approaches (i.e., ISAC, 3S, and SATzilla) we use the *'-like'* suffix. For the OTS approaches we tried different techniques like oversampling, parameters tuning, meta-classifiers, and feature selection. The best results were obtained by RF (with 250 decision trees) and SMO (with a RBF kernel and the C, γ parameters set to 2^9 and 2^{-8} respectively). In the rest of the section, for better viewing, we report only their performance among all the OTS variants we experimented. For all the approaches relying on k-NN algorithm we fixed $k = 10$ and used the Euclidean distance metric.

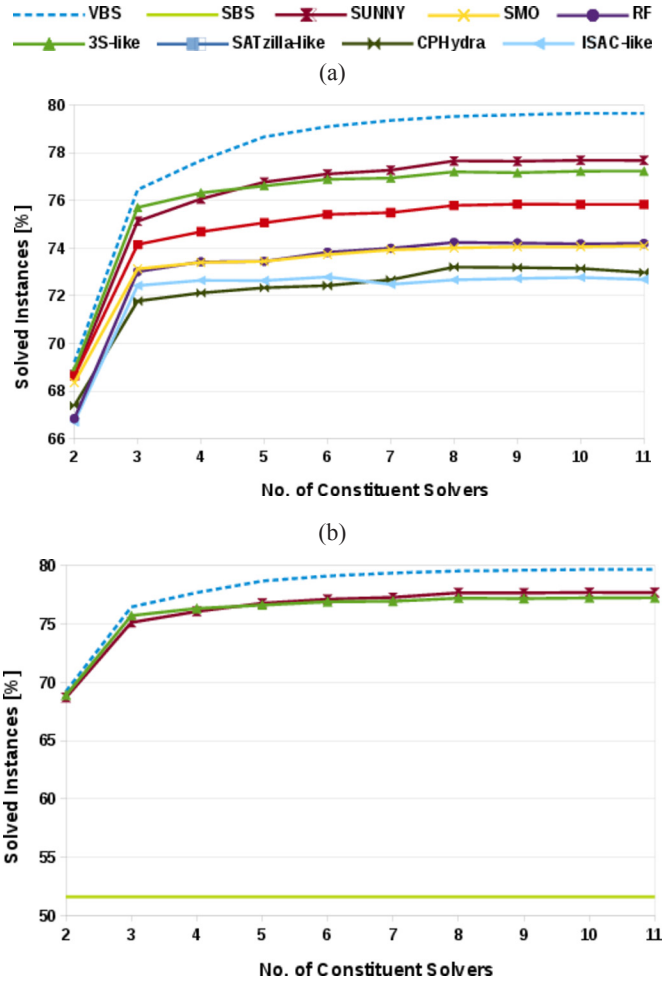


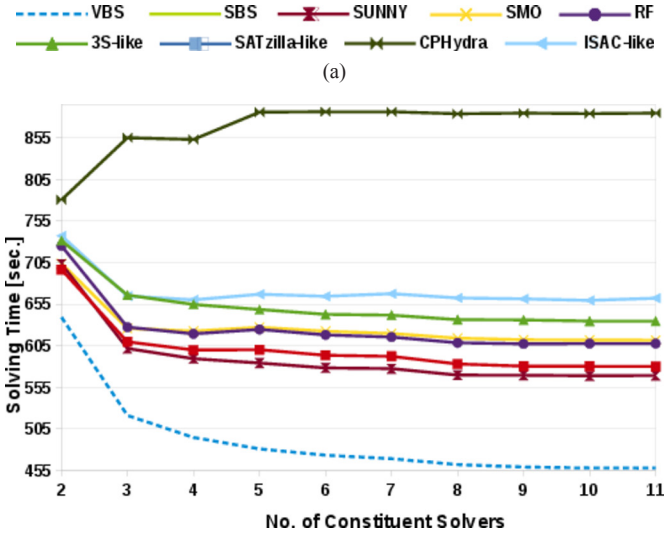
Fig. 2: PSI performance.

Fig. 2a shows the Percentage of Solved Instances for the aforementioned approaches. All of them have good performance. As already observed by [2], 3S-like and SATzilla-like are better than the best OTS approaches, which in turn solve more instances than ISAC-like and CPHydra. We do not notice the performance deterioration observed by [2] when increasing the portfolio size: the addition of a new solver is almost always beneficial, or at least not so harmful. Being the methodology of the experiments basically the same of [2], we deem that such a behavior is due to the different nature of the dataset, the features, and the solvers we used in this evaluation.

The peak performances are reached by 3S-like (77.23% with 11 solvers) and SUNNY (77.69% with 10 solvers) while in this case SATzilla-like is slightly worse (75.85% with 9 solvers). Fig. 2b depicts the performance of 3S-like and SUNNY only, together with the SBS and the VBS. It is immediately visible the performance difference between the best portfolio approaches and the SBS, which solves just 51.62% of the instances. In particular, SUNNY is able to close up to 92.95% of the gap between the SBS and the VBS.

Fig. 3a shows the Average Solving Time for each approach. As also noted by [2] the AST is highly anti-correlated with the PSI for all the approaches except CPHydra. 3S-like however is slower if compared to its performance in the work by [2]. A plausible explanation is that CPHydra and 3S-like do not employ any heuristic for sorting the selected solvers. Let us explain this with a simple example. Let us suppose that a solver S_1 solves a given CSP in 10 seconds, while another solver S_2 fails to solve it. Now consider two portfolio approaches P_1 and P_2. P_1 schedules S_1 for the first 900 seconds, and then S_2 for the remaining 900 seconds. Symmetrically, P_2 schedules S_2 for 900 seconds and then S_1 for the remaining time. Despite both P_1 and P_2 solves the CSP —so the different schedules do not influence the PSI— the solving time of P_1 will be 10 seconds, while the one of P_2 will be 910 seconds. Clearly, this difference might have a great influence on the AST.

3S-like is better than CPHydra since it solves more instances and schedules the solvers in a reduced time window ($T/10 = 180$ seconds). Conversely, the heuristic used by SUNNY (which sorts the selected solvers by increasing solving time in the k-neighbourhood) is fruitful in this context. SATzilla-like is not far from SUNNY, confirming that it can minimize the AST more than 3S-like, even if it solves less instances. Also in this case the difference with the SBS is remarkable (see Fig. 3b). The best AST performance is reached by SUNNY (568.84 seconds) which by using 10 solvers is able to close the 77.52% of the gap between the SBS and the VBS. The strong anti-correlation between AST and PSI is confirmed by the low Pearson coefficient (about –0.79). There is instead a linear correlation between the PSI and the AST of CPHydra. Nonetheless, its worst performance (884.81 seconds) is however better than the one of the SBS. For better viewing, Table 1 and Table 2 report the actual values of PSI and AST respectively.



3. If a (portfolio) solver can not solve an instance in T seconds, its solving time is set to T. This choice is also adopted in the MiniZinc Challenge, while in other contexts (e.g., SAT competitions) a penalization of $10 \times T$ seconds is given (PAR10 score).

TABLE 1: PSI VALUES

No. Solvers	SBS	VBS	RF	SMO	SATzilla	3S	ISAC	CPHydra	SUNNY
2	51.62	69.22	66.84	68.35	68.63	68.9	66.7	67.38	68.69
3	51.62	76.45	73.01	73.13	74.15	75.71	72.43	71.78	75.13
4	51.62	77.68	73.43	73.4	74.69	76.32	72.65	72.12	76.07
5	51.62	78.67	73.45	73.45	75.07	76.61	72.64	72.34	76.77
6	51.62	79.1	73.83	73.73	75.42	76.89	**72.79**	72.43	77.11
7	51.62	79.36	73.99	73.93	75.49	76.95	72.48	72.67	77.28
8	51.62	79.53	**74.24**	74.01	75.79	77.21	72.67	**73.21**	77.66
9	51.62	79.6	74.21	74.05	**75.85**	77.17	72.73	73.18	77.65
10	51.62	79.66	74.18	74.06	75.84	**77.23**	72.77	73.15	**77.69**
11	51.62	79.66	74.2	**74.09**	75.84	**77.23**	72.68	72.98	**77.69**
MIN	*51.62*	*69.22*	*66.84*	*68.35*	*68.63*	*68.9*	*66.7*	*67.38*	*68.69*
MAX	*51.62*	*79.66*	*74.24*	*74.09*	*75.85*	*77.23*	*72.79*	*73.21*	***77.69***
AVG	*51.62*	*77.9*	*73.14*	*73.22*	*74.68*	*76.02*	*72.05*	*72.12*	*76.17*

TABLE 2: AST VALUES

No. Solvers	SBS	VBS	RF	SMO	SATzilla	3S	ISAC	CPHydra	SUNNY
2	950.91	639.36	725.12	703.85	696.53	731.54	737.21	**780.5**	703.39
3	950.91	521.23	627.35	625.96	609.81	666.06	664.79	855.24	601.78
4	950.91	494.77	619.35	622.61	599.94	654.8	660.3	853.04	589.41
5	950.91	481.04	624.61	627.17	600.32	648.65	666.9	885.84	584.2
6	950.91	473.47	618.08	622.54	593.55	642.83	664.55	886.39	578.4
7	950.91	469.18	615.43	619.72	592.3	641.8	667.57	886.35	577.54
8	950.91	462.17	608.54	614.11	583.1	636.47	662.57	883.89	569.7
9	950.91	459.25	**607.28**	612.15	580.43	636.09	661.39	884.75	569.51
10	950.91	458.12	607.63	611.96	580.35	**634.72**	**659.54**	884.19	**568.84**
11	950.91	458.03	607.68	**611.68**	**580.3**	634.83	662.27	884.81	569.3
MIN	*950.91*	*458.03*	*607.28*	*611.68*	*580.3*	*634.72*	*659.54*	*780.5*	***568.84***
MAX	*950.91*	*639.36*	*725.12*	*703.85*	*696.53*	*731.54*	*737.21*	*886.39*	*703.39*
AVG	*950.91*	*491.66*	*626.11*	*627.17*	*601.66*	*652.78*	*670.71*	*868.5*	*591.21*

(b)

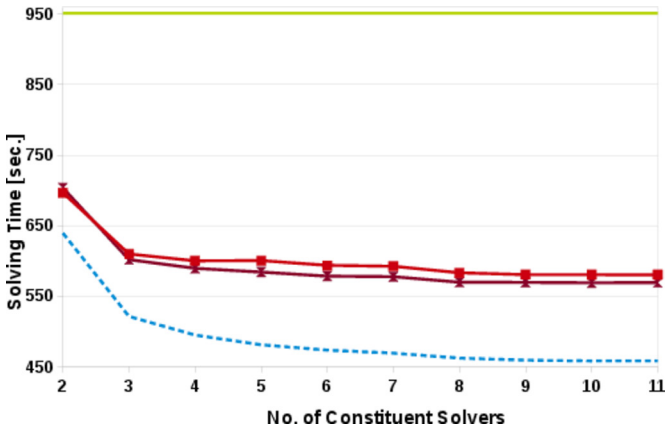


Fig. 3: AST performance.

Reproducibility The problem of effectively reproducing and comparing different approaches is a well-known issue that also affected this work. Indeed, some of the approaches we tested were not publicly available or extremely hard to use and adapt when available. There are several different ways to adapt an approach to CSP, and many other solver selectors exists. Clearly, comparing them all is a daunting task. To address this problem, the Algorithm Selection Library (ASlib) [9] has been recently introduced. ASlib provides a standardized format for representing very heterogeneous portfolio scenarios with the aim of effectively sharing and comparing different approaches. Unfortunately, at the time we conducted the experiments the ASlib had not been developed yet. We then submitted to ASlib the data and the results described in this paper, hoping that this will foster the creation of further and better portfolio approaches for the CSP field. Furthermore, the source code we developed for conducting the experiments is available at: http://www.cs.unibo.it/~amadini/csp_portfolio.zip

V. Conclusions

In this paper we presented an empirical analysis of different portfolio approaches for solving Constraint Satisfaction Problems (CSPs). We simulated and evaluated different approaches on extensive benchmarks of CSPs encoded in MiniZinc language. The obtained results are encouraging and confirm the effectiveness of CSP portfolio solvers in terms of both solved instances and solving time.

Since the impossibility of using the original code, most of the approaches have been reimplemented trying to be as faithful as possible. However, for making our experiments reproducible and comparable, we submitted the evaluation scenario to the Algorithm Selection library [9]. Indeed, in addition to the approaches evaluated in this paper, a plethora of other CSP portfolio approaches have been proposed in the literature [32,12,46,18,2]. For more comprehensive surveys about algorithm selection and runtime prediction we refer the interested reader to [25,45,20].

The possible extensions of this work are manifold. From the CSP point of view, the gap with SAT portfolio solvers is still pronounced. An immediate research direction is therefore to encourage the construction, the experimentation, and the dissemination of effective and portable CSP portfolio solvers by devising new techniques and strategies. Moreover, even if in this work we focused only on sequential approaches, the multi-solver nature of portfolios naturally leads to the parallelization of the solvers execution [29,24,42,16,5].

A well-known problem concerns the selection of the most informative features for removing redundant information and improving the prediction accuracy [19,26]. Reducing the training times [47] and exploiting incoming knowledge [28] are also promising directions for having more dynamic portfolios. Finally, we remark that portfolio approaches can be successfully applied in the most disparate domains. Besides SAT and CSP fields, successful portfolio solvers have been developed also for Answer-Set Programming (ASP) [15], Quantified Boolean Formula (QBF) [38], Planning [44], Constraint Optimization Problems (COPs) [6].

VI. References

[1] Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. A Constraint-Based Model for Fast Post-Disaster Emergency Vehicle Routing. *IJIMAI*, vol. 2, num. 4, pp. 67-75, 2013.

[2] Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. An Empirical Evaluation of Portfolios Approaches for Solving CSPs. In *CPAIOR*, vol. 7874 of LNCS, pp. 316-324. Springer, 2013.

[3] Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. An Enhanced Features Extractor for a Portfolio of Constraint Solvers. In *SAC*, pp. 1357-1359. ACM, 2014.

[4] Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. SUNNY: a lazy portfolio approach for constraint solving. *TPLP*, vol. 14, num. 4-5, pp. 509-524, 2014.

[5] Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. A Multicore Tool for Constraint Solving. In *IJCAI*, pp. 232-238. AAAI Press, 2015.

[6] Roberto Amadini and Peter J. Stuckey. Sequential Time Splitting and Bounds Communication for a Portfolio of Optimization Solvers. In *CP*, vol. 8656 of *LNCS*, pp. 108-124. Springer, 2014.

[7] Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. Why CP Portfolio Solvers are (under)Utilized? Issues and Challenges. In *LOPSTR*, vol. 9527 of *LNCS*, pp. 349-364. *Springer*, 2015.

[8] S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, vol. 4, pp. 40-79, 2010.

[9] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Thomas Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger H. Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. Aslib: A benchmark library for algorithm selection. *CoRR*, abs/1506.02465, 2015.

[10] Broes de Cat, Bart Bogaerts, Jo Devriendt, and Marc Denecker. Model Expansion in the Presence of Function Symbols Using Constraint Programming. In *ICTAI*, pp. 1068-1075, 2013.

[11] Cormac Gebruers, Alessio Guerri, Brahim Hnich, and Michela Milano. Making Choices Using Structure at the Instance Level within a Case Based Reasoning Framework. In *CPAIOR*, vol. 3011 of *LNCS*, pp. 380-386. Springer, 2004.

[12] Ian P. Gent, Christopher Jefferson, Lars Kotthoff, Ian Miguel, Neil C. A. Moore, Peter Nightingale, and Karen E. Petrie. Learning when to use lazy learning in constraint solving. In *ECAI*, vol. 215 of *FAIA*, pp. 873-878. IOS Press, 2010.

[13] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artif. Intell.*, vol. 126, num. 1-2, pp. 43-62, 2001.

[14] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, vol 11, num. 1, November 2009.

[15] Holger Hoos, Marius Thomas Lindauer, and Torsten Schaub. claspfolio 2: Advances in algorithm selection for answer set programming. *TPLP*, vol. 14, num. 4-5, pp. 569-585, 2014.

[16] Holger H. Hoos, Marius Thomas Lindauer, and Frank Hutter. From Sequential Algorithm Selection to Parallel Portfolio Selection. In *LION*, vol. 8426 of *LNCS, pp. 21-35,* 2015.

[17] Bernardo A Huberman, Rajan M Lukose, and Tad Hogg. An economics approach to hard computational problems. *Science*, vol. 275, num. 5296, pp. 51-54, 1997.

[18] Barry Hurley, Lars Kotthoff, Yuri Malitsky, and Barry O'Sullivan. Proteus: A Hierarchical Portfolio of Solvers and Transformations. In *CPAIOR*, vol. 8451 of *LNCS*, pp. 301-317. Springer, 2014.

[19] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Identifying Key Algorithm Parameters and Instance Features Using Forward Selection. In *LION*, vol. 7997 of *LNCS*, pp. 364-381. Springer, 2013.

[20] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artif. Intell.*, vol. 206, num. 79-111, 2014.

[21] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm Selection and Scheduling. In *CP*, vol. 6876 of *LNCS*. Springer, 2011.

[22] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC - Instance-Specific Algorithm Configuration. In *ECAI*, vol. 215 of *FAIA*. IOS Press, 2010.

[23] George Katsirelos and Fahiem Bacchus. Generalized NoGoods in CSPs. In *AAAI*, pp. 390-396, 2005.

[24] George Katsirelos, Ashish Sabharwal, Horst Samulowitz, and Laurent Simon. Resolution and parallelizability: Barriers to the efficient parallelization of SAT solvers. In *AAAI*. AAAI Press, 2013.

[25] Lars Kotthoff. Algorithm Selection for Combinatorial Search Problems: A Survey. *AI Magazine*, vol. 35, num. 3, pp. 48-60, 2014.

[26] Christian Kroer and Yuri Malitsky. Feature Filtering for Instance-Specific Algorithm Configuration. In *ICTAI*, pp. 849-855. IEEE, 2011.

[27] Alan K. Mackworth. Consistency in Networks of Relations. *Artif. Intell.*, vol. 8, num. 1, pp. 99-118, 1977.

[28] Yuri Malitsky, Deepak Mehta, and Barry O'Sullivan. Evolving instance specific algorithm configuration. In *SOCS*. AAAI Press, 2013.

[29] Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Parallel SAT solver selection and scheduling. In *CP*, vol. 7514 of *LNCS*, pp. 512-526. Springer, 2012.

[30] Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm Portfolios Based on Cost-Sensitive Hierarchical Clustering. In *IJCAI*. IJCAI/AAAI, 2013.

[31] Yuri Malitsky and Meinolf Sellmann. Instance-Specific Algorithm Configuration as a Method for Non-Model-Based Portfolio Generation. In *CPAIOR*, vol. 7298 of *LNCS*. Springer, 2012.

[32] Steven Minton. Automatically configuring constraint satisfaction programs: A case study. *Constraints*, vol. 1, num. 1/2, pp. 7-43, 1996.

[33] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a Standard CP Modelling Language. In *CP*, 2007.

[34] Mladen Nikolic, Filip Maric, and Predrag Janicic. Instance-Based Selection of Policies for SAT Solvers. In *SAT*, vol. 5584 of *LNCS*, pp. 326-340. Springer, 2009.

[35] Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *AICS*, 2008.

[36] Zahra Pooranian, Mohammad Shojafar, Jemal H. Abawajy and Mukesh Singhal. GLOA: A New Job Scheduling Algorithm for Grid Computing. *IJIMAI*, vol. 2, num. 1, pp. 59-64, 2013.

[37] Luca Pulina and Armando Tacchella. A Multi-engine Solver for Quantified Boolean Formulas. In *CP*, vol. 4741 of *LNCS*, pp. 574-589. Springer, 2007.

[38] Luca Pulina and Armando Tacchella. A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints*, vol. 14, num. 1, pp. 80-116, 2009.

[39] John R. Rice. The Algorithm Selection Problem. *Advances in Computers*, vol. 15, pp. 65-118, 1976.

[40] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.

[41] Olivier Roussel and Christophe Lecoutre. XML Representation of Constraint Networks: Format XCSP 2.1. *CoRR*, abs/0902.2362, 2009.

[42] Ashish Sabharwal and Horst Samulowitz. Insights into parallelism with

intensive knowledge sharing. In *CP*, vol. 8656 of *LNCS*, pp. 655-671. Springer, 2014.

[43] Horst Samulowitz, Chandra Reddy, Ashish Sabharwal, and Meinolf Sellmann. Snappy: A simple algorithm portfolio. In *SAT*, vol. 7962 of *LNCS*, pp. 422-428. Springer, 2013.

[44] Jendrik Seipp, Silvan Sievers, Malte Helmert, and Frank Hutter. Automatic configuration of sequential planning portfolios. In *AAAI*, pp. 3364-3370. AAAI Press, 2015.

[45] Kate Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, vol. 41, num. 1, 2008.

[46] Kostas Stergiou. Heuristics for dynamically adapting propagation in constraint satisfaction problems. *AI Commun.*, vol. 22, num. 3, pp. 125-141, 2009.

[47] Mirko Stojadinovic, Mladen Nikolic, and Filip Maric. Short portfolio training for CSP solving. *CoRR*, abs/1505.02070, 2015.

[48] Peter J. Stuckey, Ralph Becket, and Julien Fischer. Philosophy of the MiniZinc challenge. *Constraints*, vol. 15, num. 3, pp. 307-316, 2010.

[49] MRC van Dongen, Christophe Lecoutre, and Olivier Roussel. Third International CSP Solver Competition, 2008.

[50] L. Xu, F. Hutter, J. Shen, H. Hoos, and K. Leyton-Brown. SATzilla2012: Improved algorithm selection based on cost-sensitive classification models. Solver description, SAT Challenge 2012, 2012.

[51] Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton Brown. Evaluating Component Solver Contributions to Portfolio-Based Algorithm Selectors. In *SAT*, vol. 7317 of *LNCS*, pp. 228-241. Springer, 2012.

[52] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. The design and analysis of an algorithm portfolio for sat. In *CP*, vol. 4741 of *LNCS*, pp. 712-727. Springer, 2007.

[53] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *JAIR*, vol. 32, pp. 565-606, 2008.

Roberto Amadini received a Bachelor (2007) and Master (2011) degree in Computer Science from the University of Parma. In 2015 he received a Ph.D. in Computer Science from the University of Bologna. He is interested in Constraint Programming, Operations Research, Algorithm Selection, Software Anallysis and Verification. Currently is a Research Fellow at the Department of Computing and Information Systems of the University of Melbourne, Australia.

Maurizio Gabbrielli is professor of Computer Science at the University of Bologna, member of the INIRIA team FOCUS and Director of the EIT Digital Doctoral School. He received his Phd. in Computer Science in 1992 from the University of Pisa and worked at CWI (Amsterdam) and at the University of Pisa and of Udine. His research interests include constraint programming, formal methods for program verification and analysis, service oriented programming.

Jacopo Mauro received a bachelor and master degree in computer science from Udine University. In 2012 he receive a PhD in computer science from the University of Bologna. From 2010 to 2015 he was member of the Focus Research Group at INRIA (France). He has been involved in numerous Italian, French, and European research projects and a visiting student at CWI (Netherlands). He is currently working at the University of Oslo and interested in Concurrent Languages, Service Oriented Computing, Constraint Programming, Constraint Handling Rules, AI Planning and Distributed Application Deployment.