Ann Math Artif Intell (2016) 76:229–246
DOI 10.1007/s10472-015-9459-5

Portfolio approaches for constraint optimization problems

Roberto Amadini[1] · Maurizio Gabbrielli[1] · Jacopo Mauro[1]

Published online: 21 May 2015

Abstract Within the Constraint Satisfaction Problems (CSP) context, a methodology that has proven to be particularly performant consists of using a portfolio of different constraint solvers. Nevertheless, comparatively few studies and investigations have been done in the world of Constraint Optimization Problems (COP). In this work, we provide a generalization to COP as well as an empirical evaluation of different state of the art existing CSP portfolio approaches properly adapted to deal with COP. The results obtained by measuring several evaluation metrics confirm the effectiveness of portfolios even in the optimization field, and could give rise to some interesting future research.

Keywords Algorithm portfolio · Artificial intelligence · Combinatorial optimization · Constraint programming · Machine learning

Mathematics Subject Classifications (2010) 90C27

1 Introduction

Constraint Programming (CP) is a declarative paradigm that allows to express relations between different entities in form of constraints that must be satisfied. One of the main goals of CP is to model and solve *Constraint Satisfaction Problems* (CSP) [29]. Several techniques

✉ Jacopo Mauro
jmauro@cs.unibo.it

Roberto Amadini
amadini@cs.unibo.it

Maurizio Gabbrielli
gabbri@cs.unibo.it

[1] Department of Computer Science and Engineering/Lab. Focus INRIA,
University of Bologna, Bologna, Italy

and constraint solvers were developed for solving CSPs and "simplified" CSPs such as the well-known Boolean satisfiability problem (SAT), Satisfiability Modulo Theories (SMT) [10], and Answer-Set Programming (ASP) [8]. One of the more recent trends in this research area —especially in the SAT field— is trying to solve a given problem by using a portfolio approach [15, 36]. An algorithm portfolio is a general methodology that exploits a number of different algorithms in order to get an overall better algorithm. A portfolio of CP solvers can therefore be seen as a particular solver, called *portfolio solver*, that exploits a collection of $m > 1$ different constituent solvers $s_1, \ldots, s_m$ to get a globally better CP solver. When a new unseen instance p comes, the portfolio solver tries to predict which are the best constituent solvers $s_1, \ldots, s_k$ ($k \leq m$) for solving p and then runs such solver(s) on p. This solver selection process is clearly a fundamental part for the success of the approach and it is usually performed by using Machine Learning (ML) techniques.

Exploiting the fact that different solvers are better at solving different problems, portfolios have proven to be particularly effective. For example, the overall winners of international solving competitions like [14, 37] are often portfolio solvers. Despite the proven effectiveness of portfolio approaches in the CSP domain, and in particular in the SAT field, only few studies have tried to apply portfolio techniques to *Constraint Optimization Problems* (COPs). In these kind of problems the goal is to find a consistent solution that minimizes (maximizes) a specific objective function. The search is performed by means of suitable constraint solvers provided with techniques for comparing different solutions. Clearly, a COP is by definition more general than a CSP. Moreover, when considering portfolio approaches, some issues which are obvious for CSPs are less clear for COPs. For example, as we discuss later, defining a suitable metric which allows to compare different solvers is not immediate. These difficulties may explain in part the lack of exhaustive studies on portfolios consisting of different COP solvers. Indeed, to the best of our knowledge, only few works deal with portfolios of COP solvers and most of them refer only to a specific problem (e.g., the *Traveling Salesman Problem* [21]) or to a specific solver (e.g., by using runtime prediction techniques for tuning the parameters of a single solver [43]). Nevertheless, this area is of particular interest since in many real-life applications we are not interested in finding just "a" solution for a given problem but "the" optimal solution, or at least a good one.

In this work we tackle this problem and we perform a first step toward the definition of COP portfolios. We first formalized a suitable model for adapting the "classical" satisfaction-based portfolios to address COPs, providing also different metrics to measure portfolio performances. Then, by making use of an exhaustive benchmark of 2670 COP instances, we tested the performances of several portfolio approaches using portfolios of different size (from 2 up to 12 constituent solvers). In particular, we adapted two among the best effective SAT portfolios, namely SATzilla [42] and 3S [22], to the optimization field. We compared their performances against some off the shelf approaches —built on top of the widely used ML classifiers— and versus SUNNY, a lazy portfolio approach recently introduced in [4] that (unlike those mentioned above) does not require an explicit offline training phase.

The empirical results indicate that, also in the case of COPs, the portfolio approaches almost always significantly outperform the Single Best Solver available. The performances of the SATzilla and 3S inspired approaches turn out to be better than the ones obtained by exploiting off the shelf classifiers, even though in this case the performance difference does not seem as pronounced as in the case of CSPs [1]. Finally, we observe that the generalization of SUNNY to COPs appears to be particularly effective: this algorithm has indeed reached the best performances in our experiments.

Paper structure In Section 2 we introduce the metrics we adopted to evaluate the COP solvers, and in particular the `score` function. Section 3 presents the methodology and the portfolio algorithms we used to conduct the tests. The obtained results are detailed in Section 4 while related work is discussed in Section 5. Concluding remarks and future works are finally contained in Section 6.

This paper is an extended version of our previous work [3]. In particular, we improve on it by introducing and examining new evaluation metrics. Moreover, in this work we measure the solvers performance by considering as a timeout the one used in the MiniZinc Challenge [34], the only surviving international competition for CP solvers (and, in particular, for COP solvers).

2 Solution quality evaluation

When satisfaction problems are considered, the definition and the evaluation of a portfolio solver is straightforward. Indeed, the outcome of a solver run for a given time on a given instance can be either *'solved'* (i.e., a solution is found or the unsatisfiability is proven) or *'not solved'* (i.e., the solver does not say anything about the problem). Building and evaluating a CSP solver is then conceptually easy: the goal is to maximize the number of solved instances, solving them as fast as possible. Unfortunately, in the COP world the dichotomy solved/not solved is no longer suitable. A COP solver in fact can provide sub-optimal solutions or even give the optimal one without being able to prove its optimality. Moreover, in order to speed up the search COP solvers could be executed in a non-independent way. Indeed, the knowledge of a sub-optimal solution can be used by a solver to further prune its search space, and therefore to speed up the search process. Thus, the independent (even parallel) execution of a sequence of solvers may differ from a "cooperative" execution where the best solution found by a given solver is used as a lower bound by the solvers that are launched afterwards.

Analogously to the metric typically used in the CSP field for measuring the number of solved instances, it is possible to define a metric which considers the number of *proven optima*. However, the significance of such a metric is rather limited since it does not take into account the time taken by a solver to prove the optimality and it excessively penalizes a solver that finds the optimal value (even instantaneously) without being able to prove its optimality. Another simple metric that can be used is the *optimization time*, i.e., the time needed to find an optimal solution and to prove its optimality. Unfortunately, even this metric is rather poor at discriminating COP solver performance. Indeed, for most of the non-trivial optimization problems no solver may be able to prove optimality within a reasonable timeout. The main issue here is that, although the ideal goal of a COP solver is to prove the optimality as soon as possible, for many real life applications it is far better to get a good solution in a relatively short time rather than consume too much time finding the optimal value (or proving its optimality).

An interesting method to rank COP solvers is the one used in the MiniZinc Challenge [34]. The evaluation metric is based on a *Borda* count voting system [13], where each problem is treated like a voter who ranks the solvers. Each solver gets a score proportional to the number of solvers it beats. A solver s scores points on problem p by comparing its performance with each other solver s' on problem p. 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 then the scoring is based on the optimization time with a timeout of 900 seconds. In particular, s scores 0 if it gives no answer (or an incorrect answer), or 0.5 if both s and s' complete the search in 0 seconds.[1] Otherwise the score assigned to the solver s is $\texttt{time}(p, s')/(\texttt{time}(p, s) + \texttt{time}(p, s'))$ where $\texttt{time}(p, s)$ is the optimization time of solver s on problem p. This metric takes into account both the best solution found and the time needed for completing the search process. It has also some disadvantages. In case of indistinguishable answers, it overestimates small time differences for easy instances, as well as underrate big time differences in case of medium and hard instances. Indeed, suppose that two solvers s and s' solve a problem p in 1 and 2 seconds respectively. The score assigned to s will be 2/3 while s' scores 1/3. However, the same score would be reached by the two solvers if $\texttt{time}(p, s', T) = 2\,\texttt{time}(p, s, T)$. Hence, if for example $\texttt{time}(p, s, T) = 400$ and $\texttt{time}(p, s, T) = 800$, the difference between the scores of s and s' would be the same even if the absolute time difference is 1 second in the first case and 400 seconds in the second. Moreover, the Borda comparisons of MiniZinc Challenge metric do not take into account of the difference in quality between distinguishable answers: the solver that gives the worse solution always scores 0 points, regardless of whether such solution is very close or very far from the best one.

Given these problems, in order to study the effectiveness of COP solvers (and consequently the performance of COP portfolio solvers) we need different and more sophisticated evaluation metrics.

In this work we propose a scoring system that takes into account the quality of the solutions without relying on cross comparisons between solvers. The idea is to evaluate the solution quality at the stroke of the solving timeout T, by giving to each COP solver (portfolio based or not) a reward proportional to the distance between the best solution it finds and the best known solution. An additional reward is assigned if the optimality is proven, while a punishment is given if no solution is found without proving unsatisfiability. In particular, given a COP instance i, we assign to a solver s a score of 1 if it proves optimality for i, 0 if s does not find solutions. Otherwise, we give to s a score corresponding to the value of its best solution scaled into the range $[\alpha, \beta]$, where $0 \leq \alpha \leq \beta \leq 1$. Intuitively, the parameters α and β map respectively the value of the worst and the best known solution of the known COP solvers at the timeout T.

In order to formally define this scoring function and to evaluate the quality of a solver, we denote with U the universe of the available solvers (possibly including portfolio solvers) and with T the solving timeout in seconds that we are willing to wait at most. We use the function $\texttt{sol}$ to define the solver outcomes. In particular we associate to $\texttt{sol}(s, i, t)$ the outcome of the solver s for the instance i at time t. The value $\texttt{sol}(s, i, t)$ can be either $\texttt{unk}$, if s gives no answer about i; $\texttt{sat}$, if s finds at least a solution for i but does not prove the optimality; $\texttt{opt}$ or $\texttt{uns}$ if s proves optimality or unsatisfiability. Similarly, we use the function $\texttt{val}$ to define the values of the objective function. In particular, with $\texttt{val}(s, i, t)$ we indicate the best value of the objective function found by solver s for instance i at time t. If s does not find any solution for i at time t, the value $\texttt{val}(s, i, t)$ is undefined. We assume the solvers behave monotonically, i.e., as time goes the solution quality gradually improves and never degrades.

[1] In the challenge the execution times are quantized to seconds.

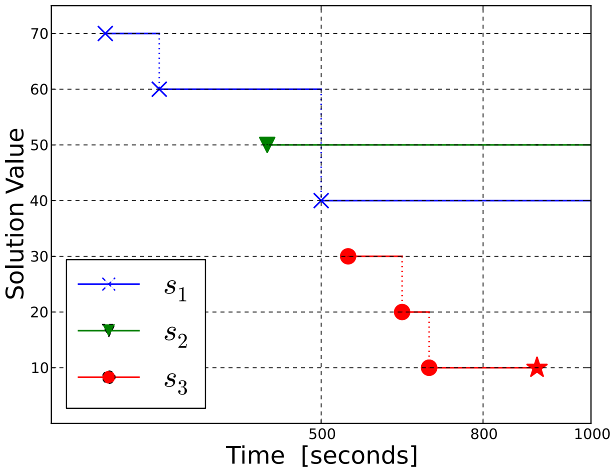


Fig. 1 Solver performances example

We are now ready to associate to every instance i and solver s a weight that quantitatively represents how good is s when solving i over time t. We define the *scoring value* of s (shortly, score) on the instance i at a given time t as a function $\texttt{score}_{\alpha,\beta}$ defined as follows:

$$\texttt{score}_{\alpha,\beta}(s,i,t) = \begin{cases} 0 & \textit{if}\ \texttt{sol}(s,i,t) = \texttt{unk} \\ 1 & \text{if } \texttt{sol}(s,i,t) \in \{\texttt{opt},\texttt{uns}\} \\ \beta & \text{if } \texttt{sol}(s,i,t) = \texttt{sat} \text{ and } \texttt{MIN}(i) = \texttt{MAX}(i) \\ \max\left\{0,\ \beta - (\beta-\alpha)\cdot\dfrac{\texttt{val}(s,i,t)-\texttt{MIN}(i)}{\texttt{MAX}(i)-\texttt{MIN}(i)}\right\} & \\ \qquad \text{if } \texttt{sol}(s,i,t) = \texttt{sat} \text{ and } i \text{ is a } \textit{minimization} \text{ problem} & \\ \max\left\{0,\ \alpha + (\beta-\alpha)\cdot\dfrac{\texttt{val}(s,i,t)-\texttt{MIN}(i)}{\texttt{MAX}(i)-\texttt{MIN}(i)}\right\} & \\ \qquad \text{if } \texttt{sol}(s,i,t) = \texttt{sat} \text{ and } i \text{ is a } \textit{maximization} \text{ problem} & \end{cases}$$

where $\texttt{MIN}(i)$ and $\texttt{MAX}(i)$ are the minimal and maximal objective function values found by any solver s at the time limit T.[2]

As an example, let us consider the scenario in Fig. 1 depicting the performances of three different solvers run on the same minimization problem. By choosing $T = 500$ as time limit, $\alpha = 0.25$, $\beta = 0.75$, the score assigned to s_1 is 0.75 because it finds the solution with minimal value (40), the score of s_2 is 0.25 since it finds the solution with maximal value (50), and the score of s_3 is 0 because it does not find a solution in T seconds. If instead $T = 800$, the score assigned to s_1 becomes $0.75 - (40-10)\cdot 0.5/(50-10) = 0.375$ while the score of s_2 is 0.25 and the score of s_3 is 0.75. If instead $T = 1000$, since s_3 proves the optimality of the value 10 at time 900 (see the point marked with a star in Fig. 1) it receives a corresponding reward reaching then the score 1.

[2] Formally, $\texttt{MIN}(i) = \min V_i$ and $\texttt{MAX}(i) = \max V_i$ where $V_i = \{\texttt{val}(s,i,T)\ .\ s \in U\}$. Note that a portfolio solver executing more than one solver for $t < T$ seconds could produce a solution that is outside the range $[\texttt{MIN}(i), \texttt{MAX}(i)]$, potentially generating a score lower than α or even lower than zero. The latter case however is very uncommon: in our experiments we noticed that the 0 score was assigned only to the solvers that did not find any solution.

The score of a solver is therefore a measure in the range [0, 1] that is linearly dependent on the distance between the best solution it finds and the best solutions found by every other available solver. We decided to scale the values of the objective function in a linear way essentially for the sake of simplicity. Other choices, like for instance using a logarithmic scaling or considering the subtended area $\int_0^T \texttt{val}(s, i, t)\, dt$ might also be useful and justifiable in a real scenario. The exploration of the impact of such alternative choices is however outside the scope of this paper, and left as a future work. Moreover, since in this work the portfolio evaluation is based on simulations, we will not consider portfolio approaches that exploit the collaboration between different solvers since the side effects of bounds communication are unpredictable in advance.

In order to build and compare different COP solvers, we will then use the score evaluation function. Obviously, different settings for α and β parameters could be considered. Among all the possible ones, we found it reasonable to set $\alpha = 0.25$ and $\beta = 0.75$. In this way, we divided into halves the codomain: one half (i.e., the range [0.25, 0.75]) is intended to map the quality of the sub-optimal solutions, while the other half consists of two "gaps" (i.e., the intervals [0, 0.25] and [0.75, 1]) that are meant to either penalize the lack of solutions or to reward the proven optimality. In the following, unless otherwise specified, we will simply use the notation score to indicate the function $\texttt{score}_{0.25,0.75}$. As one can imagine, other thresholds would have been equally justifiable. Even though a systematic study of the impact of α and β parameters is outside the scope of this paper, in this work we will also report the results obtained by using the $\texttt{score}_{0,1}$ function (see Section 4.1). We have chosen this alternative setting since 0 and 1 are the minimum and maximum values that α and β can take. On the basis of such results we conjecture that, when reasonable values of α and β are considered, the choice of different parameters does not affect significantly the overall ranking of the solvers.

3 Methodology

Taking as baseline the methodology and the results of [1], in this section we present the main ingredients and the procedure that we used for conducting our experiments and for evaluating the portfolio approaches.

3.1 Solvers, dataset, and features

In order to build our portfolios we considered all the publicly available and directly usable solvers of the MiniZinc Challenge 2012.[3] The universe U was composed by 12 solvers, namely: BProlog, Fzn2smt, CPX, G12/FD, G12/LazyFD, G12/MIP, Gecode, izplus, JaCoP, MinisatID, Mistral, and OR-Tools. We used all of them with their default parameters, their global constraint redefinitions when available, and keeping track of each solution found by every solver within the timeout of the MiniZinc Challenge, i.e., $T = 900$ seconds.

To conduct our experiments on a dataset of instances as realistic and large as possible, we have considered all the COPs of the MiniZinc 1.6 benchmark [33]. In addition, we have also added all the instances of the MiniZinc Challenge 2012, thus obtaining an initial dataset of 4977 instances in MiniZinc format.

[3] Among all the solvers of the challenge we did not use Chuffed, G12/CPLEX, and G12/Gurobi because they were not publicly available, and Choco because of problems with its FlatZinc interpreter.

In order to reproduce the portfolio approaches, we have extracted for each instance a set of 155 features by exploiting the features extractor `mzn2feat` [2]. We preprocessed these features by scaling their values in the range [-1, 1] and by removing all the constant features. In this way, we ended up with a reduced set of 130 features on which we conducted our experiments. We have also filtered the initial dataset by removing, on one hand, the "easiest" instances (i.e., those for which the optimality was proven during the feature extraction) and, on the other, the "hardest" (i.e., those for which the features extraction has required more than $T = 900$ seconds). These instances were discarded essentially for two reasons. First, if an instance is already optimized during the features extraction, no solver prediction is needed. Second, if the extraction time exceeds the timeout, then there is no more time for solving the problem.[4]

The final dataset Δ on which we conducted our experiments thus consisted of 2670 instances.

3.2 Portfolio composition

After running every solver of the universe U on each instance of the dataset Δ keeping track of all the solutions found, we built portfolios of different size $m = 2, \dots, 12$. While in the case of CSPs the ideal choice is typically to select the portfolio of solvers that maximizes the number of (potentially) solved instances, in our case such a metric is no longer appropriate since we have to take into account the quality of the solutions. We decided to select for each portfolio size $m = 2, \dots, 12$ the portfolio P_m that maximizes the total score (possible ties have been broken by minimizing the solving time). Formally:

$$P_m = \underset{P \in \{S \subseteq U \,.\, |S|=m\}}{\arg\max} \sum_{i \in \Delta} \max\{\texttt{score}(s, i, T) \,.\, s \in P\}$$

Let us explain the portfolio composition with a simple example where the universe of solvers is $U = \{s_1, \dots, s_4\}$, the dataset of problems is $\Delta = \{i_1, i_2, i_3\}$ and the scores/optimization times of each $s \in U$ for each problem $i \in \Delta$ are defined as listed in Table 1. The solver with highest score $(1 + 0.25 + 0.75 = 2)$ is s_3 and therefore $P_1 = \{s_3\}$. The selected portfolio of size 2 is instead $P_2 = \{s_1, s_2\}$, since it is the one that maximizes the sum $\sum_{i \in \Delta} \max\{\texttt{score}(s, i, T) \,.\, s \in S\}$ for each $S \subseteq U$ such that $|S| = 2$. Indeed, it is the only portfolio that reaches the score of $1 + 1 + 0.75 = 2.75$. The maximum score with three solvers (still 2.75) is achieved by $\{s_1, s_2, s_3\}$ and $\{s_1, s_2, s_4\}$ which have an average optimization time of 695.56 and 795.56 seconds respectively. Thus, $P_3 = \{s_1, s_2, s_3\}$. Trivially, $P_4 = \{s_1, \dots, s_4\}$.

We then elected a *backup solver*, that is a solver designated to handle exceptional circumstances like the premature failure of a constituent solver. After simulating different voting scenarios, the choice fell on CPX, a solver of the MiniZinc suite [33].[5] that in the following we will refer as the *Single Best Solver* (SBS) of the portfolio.

[4] We noticed that, especially for huge instances, the time needed for extracting features was strongly dominated by the FlatZinc conversion. FlatZinc [9] is the low level language that each solver uses for solving a given MiniZinc instance. A key feature of FlatZinc is that, starting from a general MiniZinc model, every solver can produce a specialized FlatZinc by redefining the global constraints definitions. For the instances of the final dataset the feature extraction time was in average 10.36 seconds, with a maximum value of 504 seconds and a median value of 3.17 seconds.

[5] To select the backup solver we used the methodology adopted in [1]. CPX was thus selected because it is the winner of all the elections we simulated using *Borda*, *Approval*, and *Plurality* criteria.

Table 1 (score, time) of each solver $s \in U$ for every problem $i \in \Delta$

	i_1	i_2	i_3	Average time (sec.)
s_1	(1, 150)	(0.25, 1000)	(0.75, 1000)	716.67
s_2	(0, 1000)	(1, 10)	(0, 1000)	670
s_3	(1, 100)	(0.75, 1000)	(0.7, 1000)	700
s_4	(0.75, 1000)	(0.75, 1000)	(0.25, 1000)	1000

3.3 Portfolio approaches

We tested different portfolio techniques. In particular, we have considered two state of the art SAT approaches (SATzilla and 3S) as well as some relatively simple off-the-shelf ML classifiers used as solver selectors. Moreover, we have also implemented a generalization of the recently introduced CSP portfolio solver SUNNY [4] in order to deal with optimization problems.

We would like to underline that in the case of 3S and SATzilla approaches we did not use the original methods which are tailored for the SAT domain. As later detailed, we have instead adapted these two approaches for the optimization world trying to modify them as little as possible. For simplicity, in the following, we refer to these adapted versions with their original names, 3S and SATzilla. A study of alternative adaptations is outside the scope of this paper.

In the following we then provide an overview of these algorithms.

Off the shelf Following the methodology of [1], off the shelf approaches were implemented by simulating the execution of a solver predicted by a ML classifier. We built 5 different simple baselines by using 5 well-known ML classifiers, viz.: IBk, J48, PART, Random Forest, and SMO, and exploiting their implementation in WEKA [18] with default parameters. In order to train the models we added for each instance of the dataset a label corresponding to the best constituent solver for such instance, where with "best solver" we mean the one that has the highest score, using the minimum optimization time for breaking ties.

For all the instances not solvable by any solver of the portfolio we used a special label *no solver*. In the cases where the solver predicted by a classifier was labeled no solver, we directly simulated the execution of the backup solver.

Note that the choice of predicting directly the best solver was done for simplicity. More complex choices, e.g., predicting the score or the optimization time for solver selection, are possible but outside the scope of this paper.

3S (SAT Solver Selector) [22] is a SAT portfolio solver that combines a fixed-time solver schedule with the dynamic selection of one long-running component solver: first, it executes for 10 % of the time limit short runs of solvers; then, if a given instance is not yet solved after such time, a designated solver is selected for the remaining time by using a k-NN algorithm. 3S was the best-performing dynamic portfolio at the International SAT Competition 2011.

The major issue when adapting 3S for optimization problems is to compute the fixed-time schedule since, differently from SAT problems, in this case the schedule should also take into account the quality of the solutions. We then tested different minimal modifications, trying to be as little invasive as possible and mainly changing the objective metric of the original Integer Programming (IP) problem used to compute the schedule. The performances of the different versions we tried were similar. Among those considered, the

IP formulation that has achieved the best performance is the one that: first, tries to maximize the solved instances; then, tries to maximize the sum of the score of the solved instances; finally, tries to minimize the solving time. Formally, the objective function of the best approach considered was obtained by replacing that of the IP problem defined in [22] (we use the very same notation) with:

$$\max \left[C_1 \sum_{y} y_i + C_2 \sum_{i,S,t} \texttt{score}(S,i,t) \cdot x_{S,t} + C_3 \sum_{S,t} t \cdot x_{S,t} \right]$$

where $C_1 = -C^2$, $C_2 = C$, $C_3 = -\frac{1}{C}$, and adding the constraint $\sum_t x_{S,t} \leq 1, \forall S$.

SATzilla [42] is a SAT solver that relies on runtime prediction models to select the solver that (hopefully) has the fastest running time on a given problem instance. Its last version [41] uses a weighted random forest approach provided with an explicit cost-sensitive loss function punishing misclassifications in direct proportion to their impact on portfolio performance. SATzilla won the 2012 SAT Challenge in the Sequential Portfolio Track.

Unlike 3S, reproducing this approach turned out to be more straightforward. The only substantial difference concerns the construction of the runtimes matrix that is exploited by SATzilla to construct its selector, which is based on $m(m-1)/2$ pairwise cost-sensitive decision forests.[6] Since our goal is to maximize the score rather than to minimize the runtime, instead of using such a matrix we have defined a matrix of "anti-scores" P in which every element $P_{i,j}$ corresponds to the score of solver j on instance i subtracted to 1, that is $P_{i,j} = 1 - \texttt{score}(j,i,T)$.

SUNNY Amadini et al. [4] is a lazy algorithm portfolio that, differently from previously mentioned approaches, does not need an offline training phase. For a given instance i and a given portfolio P, SUNNY uses a k-NN algorithm to select from the training set a subset $N(i,k)$ of the k instances closer to i. Then, on-the-fly, it computes a schedule of solvers by considering the smallest *sub-portfolio* $S \subseteq P$ able to solve the maximum number of instances in the neighbourhood $N(i,k)$ and by allocating to each solver of S a time proportional to the number of solved instances in $N(i,k)$. Finally, solvers are sorted by increasing solving time in $N(i,k)$ and then executed in such order.

We faced some design choices to tailor the algorithm for optimization problems. In particular, we decided to select the sub-portfolio $S \subseteq P$ that maximizes the score in the neighbourhood and to allocate to each solver a time proportional to its total score in $N(i,k)$. In particular, while in the CSP version SUNNY allocates to the backup solver an amount of time proportional to the number of instances not solved in $N(i,k)$, here we assign to it a slot of time proportional to $k-h$ where h is the maximum score achieved by the sub-portfolio S. Finally, solvers are sorted by increasing optimization time in $N(i,k)$.

To better understand how SUNNY works on a given COP, let us consider an example analogous to that reported in Section 3.2. Let us suppose that we want to solve a COP i by means of a portfolio $\Pi = \{s_1, s_2, s_3, s_4\}$ where s_3 is the backup solver, $T = 1000$ seconds the timeout, $k = 3$, $N(i,k) = \{i_1, i_2, i_3\}$, and the scores/optimization times as listed in Table 1. The minimum size sub-portfolio that allows to reach the highest score $h = 1 + 1 + 0.75 = 2.75$ is $\{s_1, s_2\}$. On the basis of the sum of the scores reached by s_1 and s_2 in $N(i,k)$ (resp. 2 and 1) the slot size is $t = T/(2 + 1 + (k-h)) = 307.69$ seconds. The

[6] For more details, we refer the interested reader to [41].

time assigned to s_1 is $2 * t = 615.38$, while for s_2 is $1 * t = 307.69$. The remaining 76.93 seconds are finally allocated to the backup solver s_3. After sorting the solvers by increasing optimization time, SUNNY executes first s_2 for 615.38 seconds, then s_3 for 76.93 seconds, and finally s_1 for 307.69 seconds.

3.4 Validation

In order to validate and test each of the above approaches we used a 5-repeated 5-fold cross validation [6]. The dataset Δ was randomly partitioned in 5 disjoint folds $\Delta_1, \dots, \Delta_5$ treating in turn one fold Δ_i, for $i = 1, \dots, 5$, as the test set and the union of the remaining folds $\bigcup_{j \neq i} \Delta_j$ as the training set. In order to avoid possible *overfitting* problems we repeated the random generation of the folds for 5 times, thus obtaining 25 different training sets (consisting of 534 instances each) and 25 different training sets (consisting of 2136 instances). For every instance of every test set we then computed the solving strategy proposed by the particular portfolio approach and we simulated it using a time cap of 900 seconds. For estimating the scores time we have taken into account both the time needed for converting a MiniZinc model to FlatZinc and the time needed for extracting the features. In order to evaluate the performances, we then computed the evaluation metrics introduced in the previous section.

4 Results

In this section we present the experimental results achieved by the different approaches when varying the portfolio size.[7] The performances are measured in terms of the `score` metric as well as the the optima proven, the optimization time and the MiniZinc Challenge score as described in Section 2.

For ease of reading, in all the plots we report only the two best approaches among all the off the shelf classifiers we evaluated, namely Random Forest (RF) and SMO. As a baseline for our experiments, in addition to the Single Best Solver (SBS) CPX, we have also introduced an additional solver called *Virtual Best Solver* (VBS), i.e., an oracle solver that for every instance always selects and runs the best solver of the portfolio according to the given metric. The source code developed to conduct and replicate the experiments is publicly available at http://www.cs.unibo.it/~amadini/amai_2014.zip.

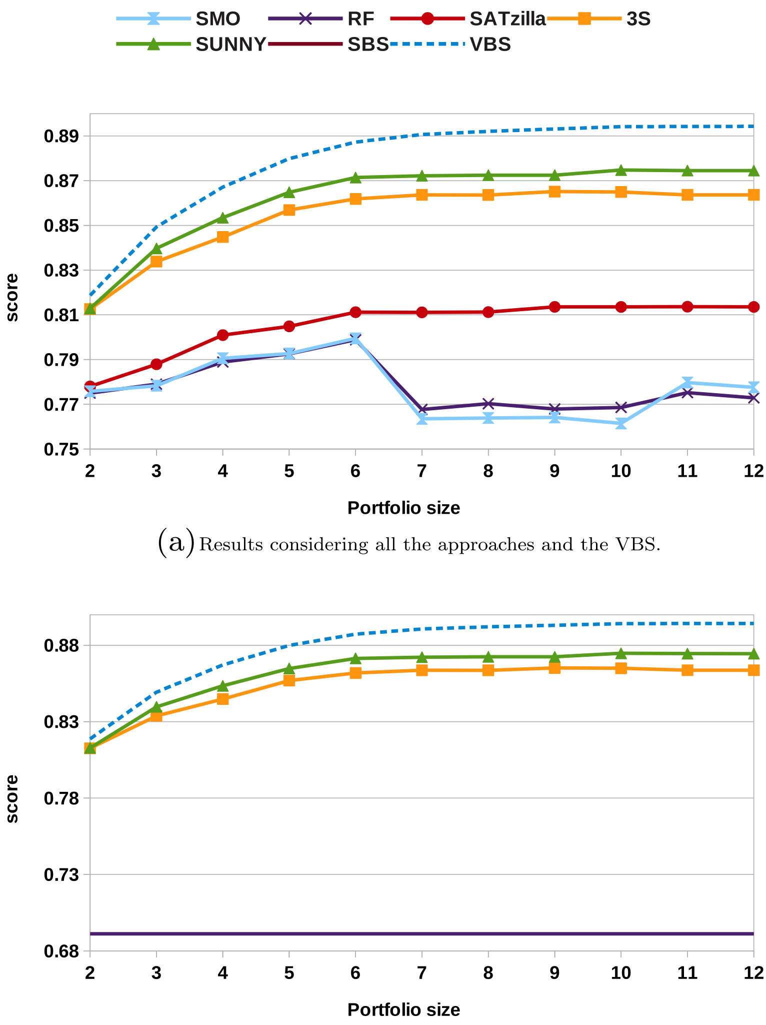
4.1 `score`

Figure 2a shows the average results of `score` by considering all the portfolio approaches and the VBS. For the sake of readability, Fig. 2b visualizes instead the same results by considering the VBS baseline, the SBS, and the two best approaches only.

As expected, all the considered approaches have good performances and greatly outperform the SBS (even with portfolios of small size).

Similarly to what happen for the Percentage of Solved Instances for CSP portfolios (see the results in [1, 2]), it is possible to notice that the off the shelf approaches have usually

[7] To conduct the experiments we used Intel Dual-Core 2.93GHz computers with 3 MB of CPU cache, 2 GB of RAM, and Ubuntu 12.04 operating system. For keeping track of the optimization times we considered the CPU time by exploiting Unix `time` command. All the information collected has been submitted to the Algorithm Selection Library [7].



(a) Results considering all the approaches and the VBS.

(b) Results considering SBS, VBS, and the best two approaches.

Fig. 2 Average Score

lower performances, even though the gap between the best approaches and them is not so pronounced here.

The best portfolio approach is SUNNY, that reaches a peak performance of 0.8747 by using a portfolio of 10 solvers and is able to close the 90.38 % of the gap between the SBS and VBS. 3S is however very close to SUNNY, and in particular the difference between the best performance of 3S (0.8651 with 9 solvers) and the peak performance of SUNNY is minimal (about 0.96 %).

It is interesting to notice that, unlike the others, the off the shelf approaches have non monotonic performances when the portfolio sizes increases. This is particularly evident looking at their performance decrease when a portfolio of size 7 is used instead of one with just 6 solvers. This instability is obviously a bad property for a portfolio approach and it is probably due to the fact that the classifiers on which such approaches rely become inaccurate when they have to chose between too many candidate solvers (a similar behaviour was noticed also in [1, 2]).

The performance of SATzilla lies in the middle between off the shelf approaches and the scheduling-based approaches 3S and SUNNY. The performance difference is probably due to the fact that SATzilla selects only one solver to run. Many COP solvers (especially the FD solvers that don't rely on lazy clause generation or MIP techniques) are indeed able to quickly find good sub-optimal solutions, even though they may fail to further improve them later. Therefore, although at a first glance it may seem counterintuitive, the choice of scheduling more than one solver turns out to be effective also for COPs. In this way the risk of predicting a bad solver is reduced, while the overall quality of the solving process is often preserved.

4.1.1 `score`*parametrization*

We conjecture that varying the α and β parameters has not a big impact on the solvers ranking. Figure 3 reports the average score by setting the score parameters to $\alpha = 0$ and $\beta = 1$. As previously underlined, 0 and 1 were chosen because these values are the extremes of α and β parameters. The resulting function can be seen as a metric that only measures the quality of a solution, without discriminating when the optimality is proven or not. This might make sense for applications in which it is very difficult to prove the optimality with one major drawback: the $\texttt{score}_{0,1}$ metric does not discriminate between the absence of solutions (perhaps due to a buggy solver) and a solution of low quality (but still a sound solution).

As can be seen, the plot is rather similar to the one presented in Fig. 2. The overall ranking of the solvers is still preserved. The performance difference between 3S and SUNNY becomes definitely negligible, but SUNNY still achieves the best score (0.9784 with 9 solvers, while the peak performance of 3S is 0.9781). SATzilla remains in the middle between them and the off the shelf approaches, with a peak performance of 0.9154 (11 solvers). Finally, RF and SMO reach the best value with 6 solvers (0.8949 and 0.8948

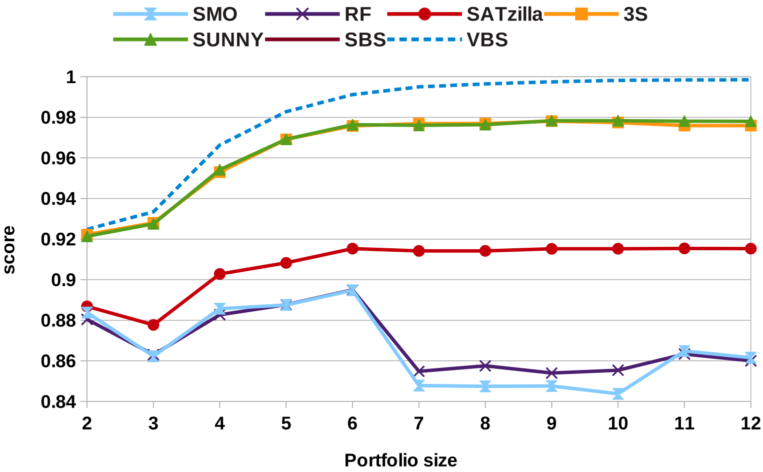


Fig. 3 Average of the $\texttt{score}_{0,1}$ metric

respectively) and still have a performance deterioration when adding more solvers. The SBS, which does not appears in the plot for the sake of readability, is pretty far away since its average $\texttt{score}_{0,1}$ is about 0.7181.

A systematic study of the sensitivity of all the portfolio approaches towards the α and β parameters of $\texttt{score}_{\alpha,\beta}$ is outside the scope of this paper. However, according to these findings, we suppose that the solvers ranking does not change significantly when reasonable values of α and β are used. The correlation between score and $\texttt{score}_{0,1}$ values is also confirmed by the Pearson product-moment coefficient, which is very high: 0.89.

4.2 Optima proven

Figure 4 shows (in percentage) the number of optima proven by the portfolio approaches, setting as additional baselines the performances of the SBS and the VBS.

Looking at the plot, it is clear the demarcation between SUNNY and the other approaches. SUNNY appears to prove far more optima w.r.t. the other techniques, reaching the maximum of 55.48 % with a portfolio of 10 solvers. We think that the performances of SUNNY are due to the fact that it properly schedules more than one solver reducing the risk of making wrong choices. Moreover, it uses this schedule for the entire time window (on the contrary, 3S uses a static schedule only for 10 % of the time window). Another interesting fact is that SUNNY mimics the behaviour of the VBS. Thus, SUNNY seems able to properly exploit the addition of a new solver by taking advantage of its contribution in terms of optima proven.

The closest approach to SUNNY is 3S, which reaches a maximum of 51.51 % with 10 solvers. The other approaches selecting just one solver per instance appear to be worse than SUNNY and 3S, and fairly close to each other. Moreover, it is evident that all the portfolio approaches greatly outperform the SBS. SUNNY in particular is able to close the 82.78 % of the gap between the SBS and VBS.

It is interesting to notice that this metric is actually not so dissimilar from score. Indeed, the Pearson coefficient between these two metrics is remarkable: about 0.72.

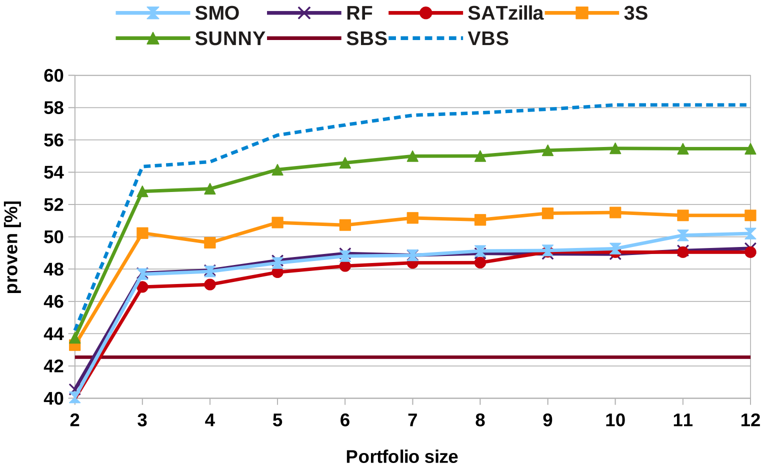


Fig. 4 Percentage of optima proven

4.3 Optimization time

Figure 5 presents the average optimization times of the various approaches, setting as additional baselines the performances of the SBS and the VBS. In case the optimality was not proven, the corresponding optimization time was set to $T = 900$ seconds.

Even in this case, the results turn out to be related to those previously reported. The only remarkable difference concerns the 3S approach, that here does not perform as well as before. We think that this is due to the fact that 3S is a portfolio that schedules more than one solver and it does not employ heuristics to decide which solver has to be executed first. SUNNY instead does not suffer from this problem since it schedules the solvers according to their performances on the already known instances. However, the performances of 3S look very close to those of SATzilla and the off the shelf approaches.

Even in this case we can observe the good performance of SUNNY that is able to close the 81.55 % of the gap between SBS and VBS reaching a peak performance of 428.95 seconds with a portfolio of 12 solvers. The second position is this time achieved by SMO, that with 12 solvers reaches a minimum of 471.05 seconds.

As one can expect, the optimization time is by definition strongly anti-correlated to the optima proven (the Pearson coefficient is -0.88) and however moderately related also to the `score` metric (the Pearson coefficient is -0.63).

4.4 MiniZinc challenge score (MZCS)

In this section we present the results using the scoring metric used in the MiniZinc Challenge. Given the nature of the score (see Section 2) it makes no sense to introduce the VBS. In fact, due to the Borda comparisons, this addition would only cause a lowering of the scores of the other "real" solvers. Conversely, it is certainly reasonable to consider the SBS against the portfolio solvers. In Fig. 6 is depicted the average score achieved by every portfolio approach and the SBS.

It interesting to note that the results are somehow different from what observed using other evaluation metrics. Even if SUNNY is still the best approach, we notice a remarkable

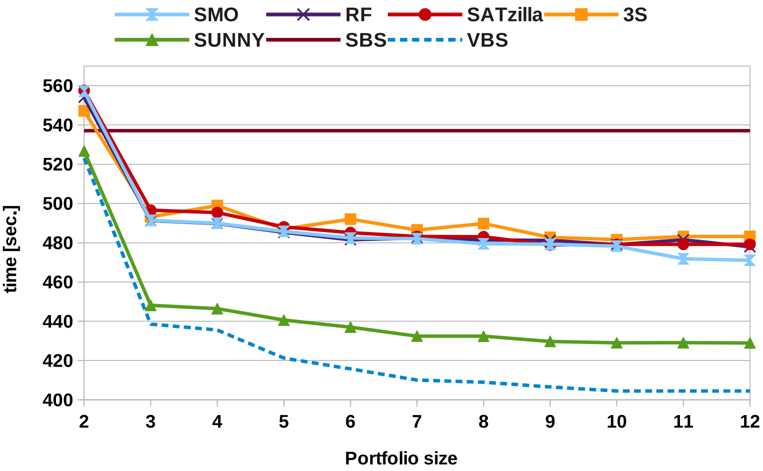


Fig. 5 Average solving time

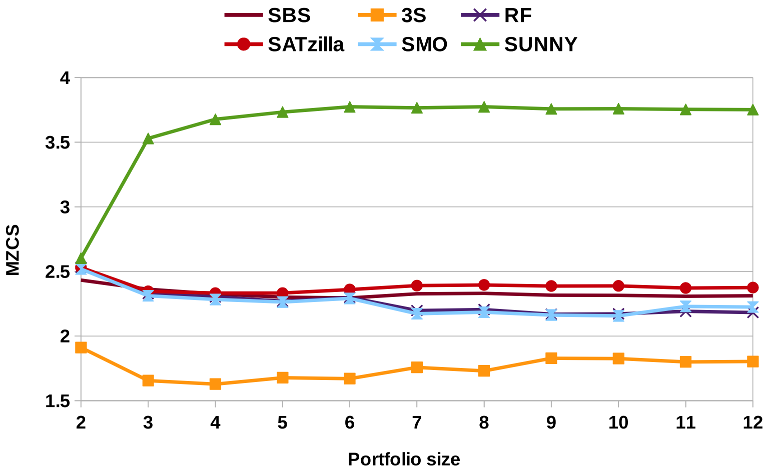


Fig. 6 MiniZinc challenge score

improvement of SATzilla as a well as a significant worsening of 3S, that turns out to be always worse than the SBS. This is due to the fact that SATzilla is able to select solvers that give better answers w.r.t. the off the shelf approaches (especially when optimality is not proven) and that, in case of indistinguishable answers, the scheduling-based approach of 3S is penalized by the Borda count used in the MiniZinc Challenge. Indeed, by considering all the conducted experiments, we notice that on average SATzilla gives a better answer than RF and SMO for the 5.83 % and 6.06 % of the instances respectively. Conversely, it gives a worst answer in the 2.29 % and 2.48 % of the cases. The better solutions quality of 3S is confirmed by the fact that on average it gives a better answer than SATzilla in the 10.24 % of the cases, while for just 6.3 % of the instances SATzilla is better than 3S. Despite this, 3S is significantly slower than SATzilla. Indeed, in case of indistinguishable answer, SATzilla is faster than 3S in the 44.08 % of the times, while it is slower in just the 1.22 %. This behaviour is somehow confirmed also by the plots in Figs. 2, 3, 4, and 5: 3S is better in finding good solutions, but it is on average slower in proving the optimality, even if it can prove far more optima.

Note that, due to the pairwise comparisons between the solvers, the performance of the SBS is not constant. Indeed, its performance depends on the performance of the other solver of the portfolio. The better a solver becomes, the more solvers it is able to beat, thus improving its score. Moreover note that, differently from `score`, here the score is not between 0 and 1 since MZCS is a value in the range $[0, m - 1]$ where m is the number of the solvers involved in the Borda count. The peak performance is achieved by SUNNY that with 6 solvers has an average score of 3.77, meaning that on average it is able to beat almost 4 out of 5 competitors per instance. All the other approaches reach instead the peak performance with 2 solvers, in correspondence with the worst SUNNY performance.

The different nature of the MZCS metric w.r.t. the other metrics is also underlined by the very low correlation. In particular, there is in practice no correlation with optima proven (0.31) and optimization time (−0.07). More pronounced, but still rather low, the correlation between MZCS and `score`: about 0.53.

5 Related work

As far as the evaluation of optimization solvers and portfolio approaches is concerned, there exists a variety of metrics used to rank them. Differently from the Borda-based MiniZinc Challenge scoring system, other well known competitions like [31, 37] rank the solvers by the overall number of solved instances, breaking the ties with the solving time.

In the previous section we have already mentioned SATZilla [42] and 3S [22] as two of the most effective portfolio approaches in the SAT and CSP domain. For a comprehensive survey on portfolio approaches applied to SAT, planning, and QBF problems we refer the interested reader to the comprehensive survey [25] and to [1] for CSPs.

With regard to the optimization problems, in the 2008 survey on algorithm selection procedures [38] the authors observe that *“there have been surprisingly few attempts to generalize the relevant meta-learning ideas developed by the machine learning community, or even to follow some of the directions of Leyton-Brown et al. in the constraint programming community”*. To the best of our knowledge, we think that the situation has not improved significantly. Indeed, in the literature, we are aware of portfolio approaches developed just for some specific instances of COP. For instance, problems like Mixed Integer Programming, Scheduling, Most Probable Explanation (MPE) and Travel Salesman Problem (TSP) are addressed by means of portfolio techniques exploiting ML methods in [17, 21].

If we exclude the solvers based on the SUNNY algorithm, the only other COP portfolio solver able to process MiniZinc language that we are aware of is based on Numberjack [19]. This solver is not considered here since it is a parallel one and it simply launches concurrently all its constituent solvers, without making any solver selection and without extracting any feature.

The SUNNY approach introduced in this work was extended in [5] for enabling the bounds communication between the scheduled solvers. Being impossible to simulate these techniques without actually running every approach, this extension was not considered here.

Other related works target the analysis of the search space of optimization problems by using techniques like landscape analysis [24], Kolmogorov complexity [11], and basins of attractions [32]. Some other approaches like [27, 39] also use ML techniques to estimate the search space of some algorithms and heuristics on optimization problems. These works look interesting because precise performance evaluations can be exploited in order to build portfolios as done, for instance, by SATzilla [42] in the SAT domain or by [28] for a class of optimization problems solved by means of branch and bound algorithms.

Another related work is [40] where ML algorithms are used to solve the Knapsack and the Set Partitioning problems by using a run-time selection of different heuristics during the search. In [23, 43] automated algorithm configurators based on AI techniques are used to boost the solving process of MIP and optimization problems. In [12] a low-knowledge approach that selects solvers for optimization problems is proposed. In this case, decisions are based only on the improvement of the solutions quality, without relying on complex prediction models or on extensive set of features.

6 Conclusions and extensions

In this paper we tackled the problem of developing a portfolio approach for solving COPs. In particular, in order to evaluate the performances of a COP solver we proposed a scoring function which takes into account the quality of the solvers solutions. We then used such a metric (and others) for comparing different portfolio techniques adapted from the satisfiability world with others based on Machine Learning classifiers and with another recently proposed lazy portfolio approach.

The results obtained clearly indicate that the portfolio approaches have remarkably better performances than a single solver. We observed that, when trying to prove optimality, the number of times a solver cannot complete the search is not negligible. Moreover, the optimization times have a heavy-tail distribution typical of complete search methods [16]. Hence, a COP setting can be considered an ideal scenario to apply portfolio approaches and obtain statistically better solvers exploiting existing ones.

We noticed that, even though at a first glance it can seem counterintuitive, the best performances were obtained by SUNNY, a portfolio approach that we proposed which (possibly) schedules more than one solver. This strategy reduces the risk of choosing the wrong solver and, apparently, this is more important than performing part of the computation again, as could happen when two (or more) solvers are launched on the same instance. We also noticed that the adaptation of methods deriving from SAT provides positive results but does not lead to the same gain of performance that these methods provide in the CSP and SAT field. We believe that the study of new techniques tailored to COPs may be an interesting direction in order to fill the gap with the SAT field. This is however left as a future work, as well the adaptation and test of other promising portfolio approaches like [23, 30, 35] and the use of filtering [26] or benchmark generation techniques [20] for improving the predictions accuracy.

Another direction for further research is the study of how cooperative strategies can be used among the constituent solvers, both in the sequential case and in a parallel setting, where more than one solver of the portfolio is allowed to be executed at the same time. As previously mentioned, we would also like to study the impact of using other metrics to evaluate the solution quality of the solvers.

References

1. Amadini, R., Gabbrielli, M., Mauro, J.: An empirical evaluation of portfolios approaches for solving CSPs. In: CPAIOR, Volume 7874 of Lecture Notes in Computer Science. Springer (2013)
2. Amadini, R., Gabbrielli, M., Mauro, J.: An enhanced features extractor for a portfolio of constraint solvers. In: SAC, pp. 1357–1359. ACM (2014)
3. Amadini, R., Gabbrielli, M., Mauro, J.: Portfolio approaches for constraint optimization problems. In: LION, Volume 8426 of Lecture Notes in Computer Science, pp. 21–35. Springer (2014)
4. Amadini, R., Gabbrielli, M., Mauro, J.: SUNNY: A lazy portfolio approach for constraint solving. TPLP **14**(4–5), 509–524 (2014)
5. Amadini, R., Stuckey, P.: Sequential time splitting and bounds communication for a portfolio of optimization solvers. In: CP. http://ww2.cs.mu.oz.au/pjs/papers/cp2014d.pdf (2014)
6. Arlot, S., Celisse, A.: A survey of cross-validation procedures for model selection. Stat. Surv. **4**, 40–79 (2010)
7. Algorithm Selection Library (COSEAL project). https://code.google.com/p/coseal/wiki/AlgorithmSelectionLibrary
8. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
9. Becket, R.: Specification of FlatZinc - Version 1.6. http://www.minizinc.org/downloads/doc-1.6/flatzinc-spec.pdf
10. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press (2009)
11. Borenstein, Y., Riccardo, P.: Kolmogorov complexity, optimization and hardness. In: Evolutionary Computation, pp. 112–119 (2006)
12. Carchrae, T., Beck, J.C.: Applying machine learning to low-knowledge control of optimization algorithms. Comput. Intell. **21**(4), 372–387 (2005)

13. Chevaleyre, Y., Endriss, U., Lang, J., Maudet, N.: A short introduction to computational social choice. In: SOFSEM, volume 4362 of LNCS, pp. 51–69. Springer (2007)
14. Third International CSP Solver Competition 2008. http://www.cril.univ-artois.fr/CPAI09/
15. Gomes, C.P., Selman, B.: Algorithm portfolios. Artif. Intell. **126**(1–2), 43–62 (2001)
16. Gomes, C.P., Selman, B., Crato, N.: Heavy-tailed distributions in combinatorial search. In: CP, Volume 1330 of Lecture Notes in Computer Science, pp. 121–135. Springer (1997)
17. Guo, H., Hsu, WH.: A machine learning approach to algorithm selection for NP-hard optimization problems: A case study on the MPE problem. Annals OR **156**(1), 61–82 (2007)
18. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, IH.: The WEKA data mining software: An update. SIGKDD Explor. Newsl. **11**(1) (2009)
19. Hebrard, E., O'Mahony, E., O'Sullivan, B.: Constraint programming and combinatorial optimisation in numberjack. In: CPAIOR-10, Volume 6140 of LNCS, pp. 181–185. Springer-Verlag (2010)
20. Hoos, H.H., Kaufmann, B., Schaub, T., Schneider, M.: Robust benchmark set selection for boolean constraint solvers. In: LION, Volume 7997 of Lecture Notes in Computer Science, pp. 138–152. Springer (2013)
21. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: The state of the art. CoRR, arXiv:1211.0906 (2012)
22. Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Algorithm selection and scheduling. In: CP, Volume 6876 of Lecture Notes in Computer Science. Springer (2011)
23. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC - instance-specific algorithm configuration. In: ECAI, Volume 215 of Frontiers in Artificial Intelligence and Applications. IOS Press (2010)
24. Knowles, JD., Corne, D.: Towards landscape analyses to inform the design of hybrid local search for the multiobjective quadratic assignment problem. In: HIS, Volume 87 of Frontiers in Artificial Intelligence and Applications, pp. 271–279. IOS Press (2002)
25. Kotthoff, L.: Algorithm selection for combinatorial search problems: A survey. CoRR, arXiv:1210.7959 (2012)
26. Kroer, C., Malitsky, Y.: Feature filtering for instance-specific algorithm configuration. In: ICTAI, pp. 849–855. IEEE (2011)
27. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In: CP, Volume 2470 of Lecture Notes in Computer Science, pp. 556–572. Springer (2002)
28. Lobjois, L., Lemaître, M.: Branch and bound algorithm selection by performance prediction. In: AAAI/IAAI, pp. 353–358. AAAI Press / The MIT Press (1998)
29. Mackworth, A.K.: Consistency in networks of relations. Artif. Intell. **8**(1), 99–118 (1977)
30. Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Algorithm portfolios based on cost-sensitive hierarchical clustering. In: IJCAI. IJCAI/AAAI (2013)
31. Max-SAT 2013. http://maxsat.ia.udl.cat/introduction/
32. Merz, P.: Advanced fitness landscape analysis and the performance of memetic algorithms. Evol. Comput. **12**(3), 303–325 (2004)
33. Minizinc version 1.6. http://www.minizinc.org/download.html
34. MiniZinc Challenge. http://www.minizinc.org/challenge2014/rules2014.html
35. OMahony, E., Hebrard, E., Holland, A., Nugent, C., OSullivan, B.: Using case-based reasoning in an algorithm portfolio for constraint solving. In: AICS 08 (2009)
36. Rice, J.R.: The algorithm selection problem. Adv. Comput. **15**, 65–118 (1976)
37. SAT Challenge 2012. http://baldur.iti.kit.edu/SAT-Challenge-2012/
38. Smith-Miles, K.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv. **41**(1) (2008)
39. Smith-Miles, K.A.: Towards insightful algorithm selection for optimisation using meta-learning concepts. In: IJCNN, pp. 4118–4124. IEEE (2008)
40. Telelis, O., Stamatopoulos, P.: Combinatorial optimization through statistical instance-based learning. In: ICTAI, pp. 203–209 (2001)
41. Xu, L., Hutter, F., Shen, J., Hoos, H., Leyton-Brown, K.: SATzilla2012: Improved algorithm selection based on cost-sensitive classification models. Solver description, SAT Challenge 2012 (2012)
42. Lin, X., Hutter, F., Hoos, HH., Leyton-Brown, K.: SATzilla-07: The design and analysis of an algorithm portfolio for SAT. In: CP, Volume 4741 of Lecture Notes in Computer Science. Springer (2007)
43. Lin, X., Hutter, F., Hoos, HH., Leyton-brown, K.: Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In: RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (2011)