Portfolio Approaches for Constraint Optimization Problems [⋆]

Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro

Department of Computer Science and Engineering/Lab. Focus INRIA,
University of Bologna, Italy.
{amadini, gabbri, jmauro}@cs.unibo.it

Abstract. Within the Constraints Satisfaction Problems (CSP) context, a methodology that has proven to be particularly performant consists in 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. Experimental results confirm the effectiveness of portfolios even in the optimization field, and could give rise to some interesting future research.

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) [25]. Several techniques and constraint solvers were developed for solving CSPs and simplified CSPs problems such as the well-known Boolean satisfiability problem (SAT), Satisfiability Modulo Theories (SMT) [7], and Answer Set Programming (ASP) [5]. 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 [12, 32]. 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, dubbed *portfolio solver*, that exploits a collection of $m > 1$ different constituent solvers $s_1, \dots, s_m$ in order to obtain a globally better CP solver. When a new unseen instance i comes, the portfolio solver tries to predict which are the best constituent solvers $s_1, \dots, s_k$ $(k \leq m)$ for solving i and then runs such solver(s) on i. This solver selection process is clearly a fundamental part for the success of the approach and it is usually performed by exploiting 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 [11, 33] are often portfolio

[⋆] Work partially supported by Aeolus project, ANR-2010-SEGI-013-01.

solvers. Despite the proven effectiveness of the portfolio approach in the CSP case, and in particular in the SAT field, a few studies have tried to apply portfolio techniques to Constraint Optimization Problems (COPs). In these problems constraints are used to narrow the space of admissible solutions and then one has to find a solution that minimizes (maximizes) a specific objective function. This is done by using suitable constraint solvers integrated with techniques for comparing different solutions. Clearly a COP is 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 explain in part the lack of exhaustive studies on portfolios consisting of different COP solvers. Indeed, to the best of our knowledge, a few works deal with portfolios of COP solvers and some of them refer only to a specific problem like the *Traveling Salesman Problem* [17], while others use runtime prediction techniques for tuning the parameters of a single solver [39].

Nevertheless, this area is of particular interest since in many real-life applications we do not want to find 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 formalize a suitable model for adapting the “classical” satisfaction-based portfolios to address COPs, providing also a metric to measure portfolio performances. Then, by using an exhaustive benchmark of 2670 instances, we test the performances of different portfolio approaches using portfolios composed from 2 to 12 different solvers. In particular, we adapt two among the best effective SAT portfolios, namely SATzilla [38] and 3S [18], to the optimization field. We compare their performances w.r.t. some off-the-shelf approaches - built on top of the widely used ML classifiers - and w.r.t. SUNNY, a promising portfolio approach recently introduced in [2] that (unlike those mentioned above) does not require an offline training phase.

Empirical results indicate that these approaches always significantly outperform the Single Best Solver available. The performances of the SATzilla and 3S inspired approaches are better than the ones obtained using off-the shelf approaches, even though not as much as when used for solving CSPs [1]. Finally, we observe that the generalization of SUNNY to COPs appears to be particularly effective, since this algorithm has indeed reached the peak performances in our experiments.

Paper structure. In Section 2 we introduce the metrics adopted to evaluate the portfolio approaches for COPs. 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. We finally give concluding remarks and discuss future work in Section 6.

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 portfolio is then conceptually easy: the goal is to maximize the number of solved instances, solving them in the least time 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 lunched afterwards.

Although the ideal goal is to prove the optimality in the least time possible, in the real world there is often the need of compromises. 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 to find the optimal value (or proving its optimality). In order to study the effectiveness of the portfolio approaches we therefore need new and more sophisticated evaluation metrics. In this work we then propose to give 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 given if the optimality is proven, while a punishment is given if no solution are found without proving unsatisfiability.
In particular, given an 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 $[0.25, 0.75]$, weighting 0.25 and 0.75 respectively the worst and the best known solutions of the known COP solvers.

In order to formally define the scoring function and to evaluate the quality of a solver, we denote with U the universe of the available solvers and with T the solving timeout in seconds that we are willing to wait at most. We use the function `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 `unk`, if s does not find any solution for i; `sat`, if s finds at least a solution for i but does not prove the optimality; `opt` or `uns` if s proves optimality or unsatisfiability. Similarly, we use the function `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.

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. We define the *scoring value* of s (shortly, score) on the instance i at a given time t as a function `score` such that $\texttt{score}(s, i, t)$ can be either:

(i) 0 if $\texttt{sol}(s, i, t) = \texttt{unk}$

(ii) 1 if $\texttt{sol}(s, i, t) \in \{\texttt{opt}, \texttt{uns}\}$

(iii) 0.75 if $\texttt{sol}(s, i, t) = \texttt{sat}$ and $\texttt{MIN}(i) = \texttt{MAX}(i)$

(iv) $\max\left\{0,\ 0.75 - 0.5 \cdot \dfrac{\texttt{val}(s, i, t) - \texttt{MIN}(i)}{\texttt{MAX}(i) - \texttt{MIN}(i)}\right\}$
if $\texttt{sol}(s, i, t) = \texttt{sat},\ \texttt{MIN}(i) \neq \texttt{MAX}(i)$ and i is a *minimization* problem

(v) $\max\left\{0,\ 0.25 + 0.5 \cdot \dfrac{\texttt{val}(s, i, t) - \texttt{MIN}(i)}{\texttt{MAX}(i) - \texttt{MIN}(i)}\right\}$
if $\texttt{sol}(s, i, t) = \texttt{sat},\ \texttt{MIN}(i) \neq \texttt{MAX}(i)$ and i is a *maximization* problem

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.[1]

As an example, 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, 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. If instead $T = 800$, the score assigned to s_1 becomes $0.75 - (40 - 10) * 0.5/(50 - 10) = 0.375$ 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.

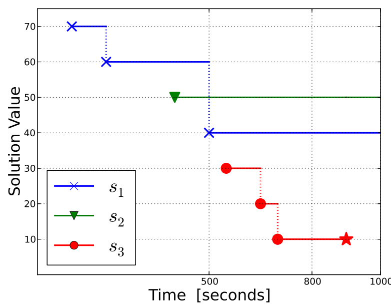


Fig. 1: Solver performances example.

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 founds 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 the logarithm of the objective function for scaling or considering the subtended area $\int_0^T \texttt{val}(s, i, t)\, dt$ may also be equally useful and justifiable in a real scenario. The exploration of the impact of such

[1] 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 solver executed by a portfolio solver for $t < T$ seconds could produce a solution that is worse than $\texttt{MIN}(i)$. This 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.

alternative choices is however outside the scope of this paper, and left as a future work. Moreover in this work we assume the independent execution of the solvers, leaving as a future research the study of portfolio approaches that exploit the collaboration between different solver in order to boost the search speed.

In order to compare different portfolio approaches, we then considered the following evaluation metrics:

– *Average Score* (AS): the average of the scores achieved by the selected solver(s) on all the instances of the dataset;
– *Percentage of Optimality Proven* (POP): the percentage of instances of the dataset for which optimality is proven;
– *Average Optimization Time* (AOT): the average time needed for proving optimality on every instance of the dataset, using a time penalty of T seconds when optimality is not proven.

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 have 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. 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 a timeout of $T = 1800$ 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 [29]. 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.

In order to reproduce the portfolio approaches, we have extracted for each instance a set of 155 features by exploiting the features extractor `mzn2feat` [3]. 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 $T/2 = 900$ 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 half of the timeout it is reasonable to assume that the recompilation of the MiniZinc model into FlatZinc format[2] would end up in wasting the time available to solve the instance. The final dataset Δ on which we conducted our experiments thus consisted of 2670 instances.

3.2 Portfolio composition

After running every solver 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 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:

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

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[3] [10] that in the following we refer also as *Single Best Solver* (SBS) of the portfolio. As a baseline for our experiments, 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 solver of the portfolio having highest score (by using the solving time for breaking ties).

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 selector. Moreover, we have also implemented a generalization of the recently introduced CSP portfolio solver SUNNY [2] 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

[2] FlatZinc [6] 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. arbitrary FlatZinc. We noticed that, especially for huge instances, the time needed for extracting features was strongly dominated by the FlatZinc conversion. However, for the instances of the final dataset this time was in average 10.36 seconds, with a maximum value of 504 seconds and a median value of 3.17 seconds.

[3] Following [1] methodology, CPX won all the elections we simulated using different criteria, viz.: *Borda*, *Approval*, and *Plurality*.

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 As in the case of satisfiability [1], off the shelf approaches were implemented by simulating the execution of a solver predicted by a ML classifier. We then built 5 different approaches using 5 well-known ML classifiers, viz.: IBk, J48, PART, Random Forest, and SMO, and exploiting their implementation in WEKA [15] 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 w.r.t. the score for such instance; 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.

3S (SAT Solver Selector) [18] 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, different 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 (with a peak AS of 0.78% more than the original one) 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. [4]

SATzilla [38] 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 [37] uses a weighted random forest approach provided with an explicit cost-sensitive loss function punishing misclassifications in direct

[4] The objective function of the best approach considered was obtained by replacing that of the IP problem defined in [18] (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 $\sum_t x_{S,t} \leq 1$, $\forall S$.

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 constructs its selector based on $m(m-1)/2$ pairwise cost-sensitive decision forests.[5] 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 $P_{i,j} = 1 - \texttt{score}(j, i, T)$.

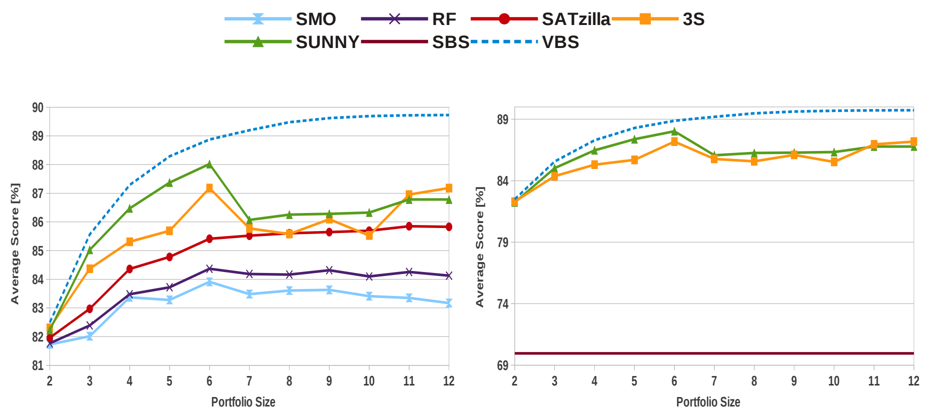
SUNNY [2] is a brand new lazy algorithm portfolio that, different from previously mentioned approaches, does not need an offline training phase. For a given instance i and a given portfolio P, SUNNY uses a k-NN algorithm to select from the training set a subset $N(i, k)$ of the k instances closer to 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 neighborhood $N(i, k)$ and by allocating to each solver of S a time proportional to the number of solved instances in $N(i, k)$.

Even in this case, 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 neighborhood and we allocated 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 have instead assigned to it a slot of time proportional to $k - h$ where h is the maximum score achieved by the sub-portfolio S.

3.4 Validation

In order to validate and test each of the above approaches we used a 5-repeated 5-fold cross validation [4]. The dataset Δ was randomly partitioned in 5 disjoint folds $\Delta_1, \ldots, \Delta_5$ treating in turn one fold Δ_i, for $i = 1, \ldots, 5$, 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 1800 seconds. For estimating the solving 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 metrics introduced in the previous section.

[5] For more details, we defer the interested reader to [37]



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

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

Fig. 2: AS performances.

4 Results

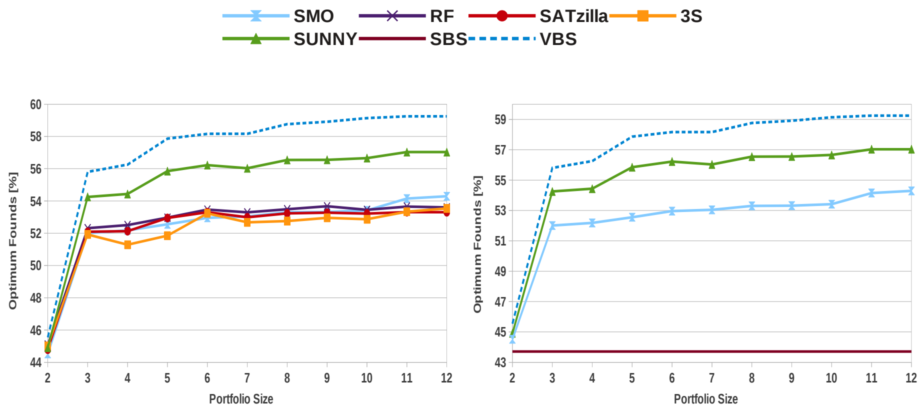
In this section we present the obtained experimental results.[6] In Fig. 2, 3, 4 we summarize the results obtained by the various techniques considering portfolios of different sizes and by using the Average Score (AS), the Percentage of Optimality Proven (POP), and the Average Optimization Time (AOT) metrics introduced 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. The source code developed to conduct and replicate the experiments is available at `http://www.cs.unibo.it/~amadini/lion_2014.zip`

4.1 Average Score

Fig. 2a shows the AS performances of the various approaches, setting as baseline the performances of the Virtual Best Solver (VBS). Fig. 2b for the sake of readability visualizes the same results considering the VBS baseline, the two best approaches only (SUNNY and 3S) and the Single Best Solver (SBS) performance as additional baseline.

All the considered approaches have good performances and they greatly outperform the SBS. As in the case of CSP [1, 3], it is possible to notice that

[6] 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 solving times we considered the CPU time by exploiting Unix `time` command.



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

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

Fig. 3: POP performances.

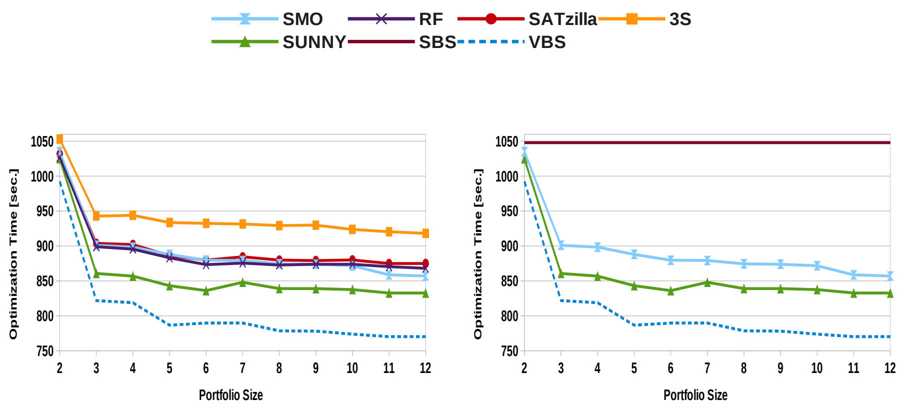
off-the-shelf approaches have usually lower performances even though the gap between the best approaches and them is not that pronounced.

The best portfolio approach is SUNNY that reaches a peak performance of 0.8802 using a portfolio of just 6 solvers and is able to close the 91.35% of the gap between the SBS and VBS. 3S however has performances close to those of SUNNY and in particular its best performance (0.8718 with 6 and 12 solvers) is very close to the peak performance of SUNNY. Strangely enough, we can notice that both SUNNY and 3S 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. We think that in this case it may be due to the noise of the addition of a solver that may disrupt the choices made by the k-NN algorithm on which SUNNY and 3S rely.

SATzilla often does not reach the performances of SUNNY or 3S, even though for big portfolio sizes its performances are rather close. Moreover its behavior is monotonic w.r.t. the increase of the size of the portfolio. Hence, it seems that SATzilla is more reliable and scalable and, as also noticed in [1], it is the only approach that does not present a degradation of performances for portfolios with more than 6 solvers.

4.2 Percentage of Optimality Proven

Looking at the number of optimality proven it is clear from Fig. 3a and 3b that there is a sharp demarcation of SUNNY w.r.t. other approaches. SUNNY appears to prove many more optimality w.r.t. the other techniques, reaching a



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

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

Fig. 4: AOT performances.

maximum POP of 57.03%. We think that the performances of SUNNY exploit the fact that it 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 behavior of the VBS. Thus, SUNNY seems able to properly use the addition of a solver to prove the optimality of instances exploiting the capability of the newly added solver.

Regarding other approaches, it can be observed by the overlapping of their curves in Fig. 3a that they are basically equivalent. What may seem surprising is that the best among them is SMO, which instead turned out to be the worst by considering the AS (Fig. 2a).

Even in this case, as shown in Fig. 3b all the portfolio approaches greatly outperform the SBS. SUNNY in particular is able to close the 85.73% of the gap between the SBS and VBS. Finally, note that there is a significant correlation between AS and POP (the Pearson coefficient computed taking into account every instance of all the test sets for all the portfolio sizes is about 0.78). Hence, maximizing the score is almost equivalent to maximizing the number of optimality proven.

4.3 Average Optimization Time

When the AOT metric is considered we can notice that the 3S approach does not perform very well compared to the other approaches. We think that this is due to the fact that 3S is a portfolio that uses 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, 3S is still able to always outperform the SBS for each portfolio size.

While the performance of SATzilla and the off-the-shelf approaches appear to be very similar, even in this case we can observe the good performances of SUNNY that is able to close the 77.51% of the gap SBS/VBS reaching a peak performance of 832.62 seconds with a portfolio of 12 solvers.

The (anti-)correlation between AOT and AS is lower than the one between POP and AS (the Pearson coefficient is -0.72) but still considerable. On the other hand, the anti-correlation between AOT and POP is very strong (the Pearson coefficient is -0.99). This means that trying to maximize the average percentage score is like trying to minimizing the average solving time and to maximize the number of proven optimality.

Finally, we would like to mention that the AOT metric could be too strict and not so significant. In fact, if a solver finds the best value after few seconds and stops its execution without proving optimality it is somewhat over-penalized with the timeout value T. In future it may therefore be interesting to study other ways to weight and evaluate the solving time (e.g., a rationale metric could be to consider a properly normalized area under the curve time/value defined by each solver behavior).

5 Related work

As far as the evaluation of optimization solvers and portfolio approaches is concerned, there exist a variety of metrics used to rank them. Among those used in practice by well known solving competitions worth mentioning are those that rank the solvers by using the number of the solved instances first, considering solving time later in case of ties [27,33]. In [30] instead the ranking is performed by using a *Borda* count, i.e., a single-winner election method in which voters rank candidates in order of preference. Differently from the metrics defined in Section 2, these metrics address the quality of the solutions in a less direct way (i.e., by making pairwise comparisons between the score of the different solvers).

In the previous section we have already mentioned SATZilla [38] and 3S [18] as two of the most effectives 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 [21] and to [1] for CSPs.

As far as optimization problems are concerned, in the 2008 survey on algorithm selection procedures [34] 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 and we are not aware of more recent works addressing explicitly the construction of portfolio solvers for COPs. 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 [14, 17].

Other related work target the analysis of the search space of optimization problems by using techniques like landscape analysis [20], Kolmogorov complexity [8], and basins of attractions [28]. Some approaches like [23, 35] 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 built portfolios as done, for instance, by SATzilla [38] in the SAT domain or by [24] for optimization problems solved by using branch and bound algorithms.

Another related work is [36] where ML algorithms are used to solve the Knapsack and the Set Partitioning problems by a run-time selection of different heuristics during the search. In [19, 39] automated algorithm configurators based on AI techniques are used to boost the solving process of MIP and optimization problems. In [9] 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 solution quality of the solver answers. We then proposed three different metrics to evaluate and compare COP solvers. These criteria were used to compare different portfolio techniques adapted from the satisfiability world with others based on classifiers and with a recently proposed lazy portfolio approach.

The results obtained clearly indicate that exploiting portfolio approaches leads to better performances w.r.t. using a single solver. We conjecture that, especially when trying to prove optimality, the number of times a solver cannot give an answer is not negligible and that the solving times have a heavy-tail distribution typical of complete search methods [13]. Hence, a COP setting can be considered an ideal scenario to apply a portfolio approach 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 which (possibly) schedules more than one solver. In these cases the risk of choosing the wrong solver is reduced and, apparently, this is more important than performing part of the computation again, as could happen when two (or more) solvers are lunched on the same instance. We also noticed that the adaptation of methods deriving from SAT 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 should be done in order to obtain the same advantages of the SAT field . This is however left as a future work, as well as adapting and testing other promising portfolio approaches like [19, 26, 31] and using filtering [22] or benchmark generation techniques [16].

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 said, we would also like to study the impact of using other metrics to evaluate the solution quality of the solvers. On the basis of the empirical correlation among the metrics so far considered we are confident that the performance of portfolio approaches should be robust, i.e., the rank of good portfolios approaches does not depend on the specific metric used, provided that the metric is enough "realistic".

References

1. Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. An Empirical Evaluation of Portfolios Approaches for Solving CSPs. In *CPAIOR*, 2013.
2. Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. SUNNY: a Simple and Dynamic Algorithm Portfolio for Solving CSPs. *CoRR*, abs/1311.3353, 2013.
3. Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. An Enhanced Features Extractor for a Portfolio of Constraint Solvers. In *SAC*, 2014.
4. S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010.
5. Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
6. Ralph Becket. Specification of FlatZinc - Version 1.6. `http://www.minizinc.org/downloads/doc-1.6/flatzinc-spec.pdf`.
7. Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*. IOS Press, 2009.
8. Yossi Borenstein and Riccardo Poli. Kolmogorov complexity, Optimization and Hardness. In *Evolutionary Computation*, pages 112–119, 2006.
9. Tom Carchrae and J. Christopher Beck. Applying Machine Learning to Low-Knowledge Control of Optimization Algorithms. *Computational Intelligence*, 21(4):372–387, 2005.
10. CPX Discrete Optimiser. `http://www.opturion.com/cpx.html`.
11. Third International CSP Solver Competition 2008. `http://www.cril.univ-artois.fr/CPAI09/`.
12. Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artif. Intell.*, 126(1-2):43–62, 2001.
13. Carla P. Gomes, Bart Selman, and Nuno Crato. Heavy-Tailed Distributions in Combinatorial Search. In *CP*, 1997.
14. Haipeng Guo and William H. Hsu. 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.
15. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1), November 2009.

16. Holger H. Hoos, Benjamin Kaufmann, Torsten Schaub, and Marius Schneider. Robust Benchmark Set Selection for Boolean Constraint Solvers. In *LION*, 2013.
17. Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm Runtime Prediction: The State of the Art. *CoRR*, abs/1211.0906, 2012.
18. Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm Selection and Scheduling. In *CP*, 2011.
19. Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC - Instance-Specific Algorithm Configuration. In *ECAI*, 2010.
20. Joshua D. Knowles and David Corne. Towards Landscape Analyses to Inform the Design of Hybrid Local Search for the Multiobjective Quadratic Assignment Problem. In *HIS*, 2002.
21. Lars Kotthoff. Algorithm Selection for Combinatorial Search Problems: A Survey. *CoRR*, abs/1210.7959, 2012.
22. Christian Kroer and Yuri Malitsky. Feature filtering for instance-specific algorithm configuration. In *ICTAI*, 2011.
23. Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Learning the Empirical Hardness of Optimization Problems: The Case of Combinatorial Auctions. In *CP*, 2002.
24. Lionel Lobjois and Michel Lemaître. Branch and Bound Algorithm Selection by Performance Prediction. In *AAAI/IAAI*, 1998.
25. Alan K. Mackworth. Consistency in Networks of Relations. *Artif. Intell.*, 8(1):99–118, 1977.
26. Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm Portfolios Based on Cost-Sensitive Hierarchical Clustering. In *IJCAI*, 2013.
27. Max-SAT 2013. `http://maxsat.ia.udl.cat/introduction/`.
28. Peter Merz. Advanced Fitness Landscape Analysis and the Performance of Memetic Algorithms. *Evolutionary Computation*, 12(3):303–325, 2004.
29. Minizinc version 1.6. `http://www.minizinc.org/download.html`.
30. MiniZinc Challenge. `http://www.minizinc.org/challenge2012/results2012.html`.
31. Eoin OMahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry OSullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. *AICS 08*, 2009.
32. John R. Rice. The Algorithm Selection Problem. *Advances in Computers*, 15:65–118, 1976.
33. SAT Challenge 2012. `http://baldur.iti.kit.edu/SAT-Challenge-2012/`.
34. Kate Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41(1), 2008.
35. Kate Amanda Smith-Miles. Towards insightful algorithm selection for optimisation using meta-learning concepts. In *IJCNN*, 2008.
36. Orestis Telelis and Panagiotis Stamatopoulos. Combinatorial Optimization through Statistical Instance-Based Learning. In *ICTAI*, 2001.
37. 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.
38. Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla-07: The Design and Analysis of an Algorithm Portfolio for SAT. In *CP*, 2007.
39. Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-brown. 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.