# Benchmarking Algorithm Portfolio Construction Methods

Mario Andrés Muñoz
munoz.m@unimelb.edu.au
School of Mathematics and Statistics,
The University of Melbourne
Parkville, VIC, Australia

Hamed Soleimani
hamed.soleimani@unimelb.edu.au
School of Mathematics and Statistics,
The University of Melbourne
Parkville, VIC, Australia

Sevvandi Kandanaarachchi
sevvandi.kandanaarachchi@rmit.edu.au
School of Sciences, RMIT University
Melbourne, VIC, Australia

## ABSTRACT

A portfolio is a set of algorithms, which run concurrently or interchangeably, whose aim is to improve performance by avoiding a bad selection of a single algorithm. Despite its high error tolerance, a carefully constructed portfolio, i.e., the smallest set of complementary algorithms, is expected to perform better than an arbitrarily constructed one. In this paper, we benchmark five algorithm portfolio construction methods, using as benchmark problems the ASLib scenarios, under a cross-validation regime. We examine the performance of each portfolio in terms of its riskiness, i.e., the existence of unsolved problems on the test set, and its robustness, i.e., the existence of an algorithm that solves most instances. The results demonstrate that two of these methods produce portfolios with the lowest risk, albeit with different levels of robustness.

## CCS CONCEPTS

• **General and reference → Empirical studies**; **Experimentation**; **Performance**; **Validation**.

## KEYWORDS

Algorithm Portfolios, Algorithm Selection, Benchmarking

**ACM Reference Format:**
Mario Andrés Muñoz, Hamed Soleimani, and Sevvandi Kandanaarachchi. 2022. Benchmarking Algorithm Portfolio Construction Methods. In *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3520304.3528880

## 1 INTRODUCTION

A key step in solving an optimization problem is to identify the best algorithm. Decades of research have taught us that some algorithms will out-perform in some problem instances and under-perform in others, with the differences in performance being attributable to: (a) the rules that govern the algorithm behavior, and (b) the characteristics of the instance's landscape. Consequently, to solve the so-called *Algorithm Selection Problem* (ASP) [6], current approaches use predictive models that take as inputs measurable features of an instance, and give as the output an algorithm recommendation [3]. At the core of this solution is the concept that a set of "complementary" algorithms, i.e., an *algorithm portfolio*, increases the probability of detecting better solutions using less resources [7].

Algorithm portfolios were initially introduced as a means to avoid poor performance due to equally poor selection, as each individual algorithm ran concurrently or interchangeably with the others [8]. Despite the inherent error tolerance of a portfolio, it is reasonable to expect that its careful construction would improve overall performance. Ideally, a portfolio should be small, as selecting from or running a larger number of algorithms leads to "over-choice" and diminishing returns from a more complex system. Nevertheless, compared to the ASP, there has been limited interest on *Portfolio Construction Problem* (PCP), i.e., *how to select the smallest subset of algorithms that also has an acceptable performance loss when compared with the complete algorithm set*, with a few approaches reported in the literature in later years.

The PCP has been explored in several problem domains. Almakhlafi and Knowles [1] evaluate seven methods for portfolio construction using 64 variations of a Genetic Algorithm as candidates. Muñoz and Kirley [5] propose ICARUS, a method that models the PCP as an election and uses the concept of 'uncovered sets' to determine the winners. Bischl et al. [2] used Sequential Forward Selection on ASLib, a library of algorithm selection scenarios. Kerschke et al. [3] proposes a method in which the benchmarks are divided into subsets by number of decision variables; then, the top $K$ algorithms per subset are selected; and finally, the portfolio is constructed by taking the intersection of these subsets. Wawrzyniak et al. [9] models the PCP as a mixed integer linear program, with the objective function being minimizing the loss compared with the VBS. With 72 heuristics as candidate algorithms and as benchmarks 12 sets of instances of the Berth Allocation problem.

In this paper, we benchmark five portfolio construction methods, i.e., Random $K$ (RND), Top $K$ (TOPK), Sequential Forward (SF), Sequential Backward (SB), and Mixed Integer Programming (MIP), using as benchmark problems the ASLib scenarios [2]. As performance metrics, we consider the average *regret*, as a loss measure, which indicates the *riskiness* of a portfolio, and the entropy, as a measure of the uniformity in contribution from each algorithm, which indicates the *robustness* of a portfolio. The results indicate that MIP and SF produce the portfolios with the lowest risk. Although, SF does so with higher robustness. As expected, RND produced the riskiest portfolios on average; however, having better performance in some scenarios than SB and MIP. Moreover, as we accept a performance less-than-best from an algorithm, robustness of a portfolio increases.

## 2 PORTFOLIO CONSTRUCTION METHODS

Let $\mathcal{F}$ be the space of all possible instances, $F \subset \mathcal{F}$ be the training instances, and $F' \subset \mathcal{F}$ be the test instances, with $F \cap F' = \emptyset$. Let $\mathcal{A}$

be the candidate algorithms. Assume that, for all $f_i \in \{F \cup F'\}$, we have recorded a *cost* metric $\rho_{i,j}$ for all the algorithms $\alpha_j \in \mathcal{A}$. Let $\alpha_{i,b}$ represent the best algorithm for $f_i$, and $\rho_{i,b}$ be its cost. Then, the *regret* of an algorithm $\alpha_j$ for $f_i$ is defined as:

$$\tilde{\rho}_{i,j} = \frac{\rho_{i,j}}{\rho_{i,b}} \tag{1}$$

An algorithm $\alpha_j$ is better than $\alpha_{j'}$, i.e., $\alpha_j \succ \alpha_{j'}$, for $f_i$ if $\tilde{\rho}_{i,j} < \tilde{\rho}_{i,j'}$, and it is $\epsilon$-good if $\left(\tilde{\rho}_{i,j} - 1\right) \leq \epsilon$. We define an algorithm portfolio, $A \subset \mathcal{A}$, to be a subset of algorithms that solves the largest group of problems with a cost on the test instances, with $K = |A|$.

**Random** $K$ (RND) constructs a portfolio of a user-defined size $K$ by generating a random permutation in the range $[1, |\mathcal{A}|]$ [1].

**Top** $K$ (TOPK), which constructs a portfolio of a size $K$ by taking those algorithms that have the largest number of *wins* on the training set, i.e., the number of instances for which the algorithm is $\epsilon$-good. For TOPK, both $K$ and $\epsilon$ are user-defined parameters.

**Sequential Forward** (SF) starts with an empty portfolio and iteratively adds the algorithm which minimizes average maximum regret on the training instances [2].

**Sequential Backward** (SB) starts with the complete set of candidate algorithms, $\mathcal{A}$, and iteratively removes the algorithm that maximizes average maximum regret on the training instances. It is equivalent to the Racing method [1].

**Mixed Integer Programming** (MIP) is a type of optimization problem definition. In our case, we disregard the time-budget of the portfolio, as we assume that we will have at least the time established as a 'time-out' in the ASLib scenarios; and we introduce a cardinality constraint, limiting the number of algorithms to $K$. We define $x_j$ as a binary variable equal to '1' if $\alpha_j \in A$, and '0' otherwise; $u_{i,j}$ as a binary variable equal to '1' if $\alpha_j \in A$ is the best solver from the portfolio for an instance $f_i$, i.e., $\alpha_j = \alpha_{i,b}$, and '0' otherwise. Then, let the objective of the MIP is to minimize the average maximum regret of a portfolio on the training test, defined as:

$$G = \frac{1}{|F|}\sum_{i=1}^{|F|}\left[\max_{j=1,\ldots,|\mathcal{A}|} \tilde{\rho}_{i,j} u_{i,j}\right] \tag{2}$$

subject to having one algorithm in the portfolio capable of solving each instance, i.e., $\sum_{j=1}^{|\mathcal{A}|} u_{i,j} \geq 1$, $x_j \geq u_{i,j}$, $\forall f_i \in F$ and $\forall \alpha_j \in \mathcal{A}$.

## 3 EXPERIMENTAL METHODOLOGY

We use 10-fold cross-validation to validate these methods. Moreover, we assume that a perfect oracle exists, i.e., an algorithm selector that takes the best solver from the portfolio on each instance for our performance calculations. The TOPK and MIP methods were implemented in Python 3.7 with Gurobi 9.5.0, while the remaining methods were implemented in MATLAB r2020a. Postprocessing was carried out in R.

### 3.1 Benchmark instances: The Algorithm Selection Library

ASLib [2] is a library widely used on Algorithm Selection competitions [4]. It provides a repository of algorithm selection scenarios collected from the literature, a standardized format able to express a wide variety of situations, and a set of tools in R and Python for reading the data. The main objective of ASLib is to provide a way for a researcher to systematically and fairly compare their methods.

At the time of writing, ASLib contained 43 scenarios broadly divided into "problems", e.g., satisfiability (SAT), quantified Boolean formula (QBF), maximum satisfiability (MAXSAT). Each scenario provides for each instance a vector of features, i.e., a group of measurements that describe those characteristics known to make an instance difficult, and a vector of algorithm performances, i.e., a measure of the resources required by each algorithm to successfully solve an instance. For our purpose, we assume that any algorithm that has "timed-out" is unable to solve the instance, i.e., it was unsuccessful, replacing the time-out value for `NaN`. We ignore any instance for which all algorithms have been unsuccessful, as it makes not difference which algorithm is used in these cases. We ignored those scenarios which included algorithm "features" that describe the complexity of each algorithm, because not all scenarios have this information available, and the "TTP-2016" scenario as the performance information did not match the other scenarios. This left us with 32 scenarios for analysis.

### 3.2 Metrics of portfolio performance

We calculate two metrics to determine the performance of the portfolio. The first one corresponds to the base-10 logarithm of the average regret, as given by (2), although calculated on the test set, and giving a regret value of $10^9$ to any `NaN`. The second one is the Entropy of the portfolio [1], defined as:

$$H(A) = -\sum_{j=1}^{|A|} p_j \ln p_j \tag{3}$$

where $p_j$ is the probability that $\alpha_j$ is $\epsilon$-good. We define an *ideal* portfolio, $\mathcal{I} \subset \mathcal{A}$, as the one such that (a) it solves all instances, and (b) each component algorithm solves an equal number of unique instances. In other words, assuming a uniform distribution of problems, the algorithms in this portfolio are complementary. In such portfolio, the probability of $\alpha_j$ of being good on the training and tests sets is $1/|\mathcal{I}|$ and its entropy is '1'. On the other hand, we define a *dominated* portfolio as the one such that a single algorithm solves all instances, and all others solve none. In such portfolio, the entropy is '0'. With these definitions, we can define a normalised entropy as:

$$\tilde{H}(A) = \frac{H(A)}{H(\mathcal{I})} \tag{4}$$

where $H(\mathcal{I})$ is the entropy of the ideal portfolio, of the same cardinality as $A$. For portfolios with cardinality higher than 1 and for which a single algorithm is $\epsilon$-good for each instance, the entropy is expected to be bounded between $[0, 1]$.

These measures capture two aspects of performance. Regret is a loss measure, giving the worst case cost of a given portfolio. Therefore, a large regret indicate that one or more problems remained without solution. A smaller regret indicate that a less efficient algorithm was selected. A value close to one indicates that a competitive algorithm was selected. On the other hand, entropy is a measure of robustness of the portfolio, as it cares about the diversity of both the algorithms and the instances. An ideal portfolio has both regret and entropy equal to '1'. A portfolio with high regret and entropy close to '1' is risky as it is likely to fail in some problems, but is probably more robust. Similarly, a portfolio with regret close to '1' and entropy close to '0' is also risky, because while the portfolio worked well on the test problems, it is likely to be less robust if the test instances were biased.

We performed a parameter sweep, with $K = \{1, 2, \ldots, |\mathcal{A}|\}$, $\epsilon = \{0.00, 0.05, 0.10, 0.20, 0.50, 1.00, 2.00\}$. We expect that $\epsilon$ has minimal to no effect on regret, but a significant effect on entropy, as it enlarges what is accepted as "good". The results averaged over $\epsilon$ for each value of $K$. In order to compare the performance across scenarios, we scale the cardinality such that the maximum value of $K$ is '1' for each scenario. As the log-10 regret can be large, we scale its values in a scenario by dividing over the mean. Using these scaled values, we compute the area under the curve (AUC) of both the log-10 regret and the entropy curves. We use trapezoidal integration available in R package `pracma` to compute the AUC.

## 4 RESULTS

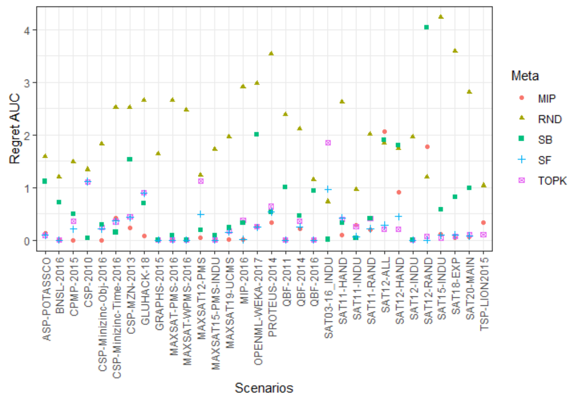
We examine the values of the AUC for each method. From Figure 1a we see that, for most scenarios, the highest regret AUC is achieved by RND, followed by SB, while MIP, SF and TOPK have very low regret AUC values for most scenarios. The entropy AUC in Figure 1b shows that for most scenarios the highest entropy AUC is achieved by SF while the lowest is achieved by SB. In some scenarios, such as SAT12-ALL, SAT12-HAND and SAT12-RAND, both MIP and SB have worse regrets than RND, indicating that the algorithms in these scenarios are more specialized. Recalling that good selectors maximize entropy and minimize regret, we see that SF and MIP are the better performing portfolio constructors.

To examine whether one method is better for a problem class, we separated the scenarios into categories. Figure 2 illustrates these results. By inspecting the regret in Figure 2a, we observe that MIP and SF perform better than the remainder methods, with RND having the poorest performance, and in most cases being unable to achieve a log-10 regret close to '0'. Furthermore, we see that for SAT scenarios, SF performs better than MIP. For CSP and MAXSAT scenarios, MIP performs better than SF. In terms of entropy AUC in Figure 2b, SF gives higher entropy for most scenario categories compared to other selectors. We see that entropy AUC for SF is higher than 0.6 for many scenarios, whereas the other selectors have only a few scenarios achieving an entropy higher than 0.6.

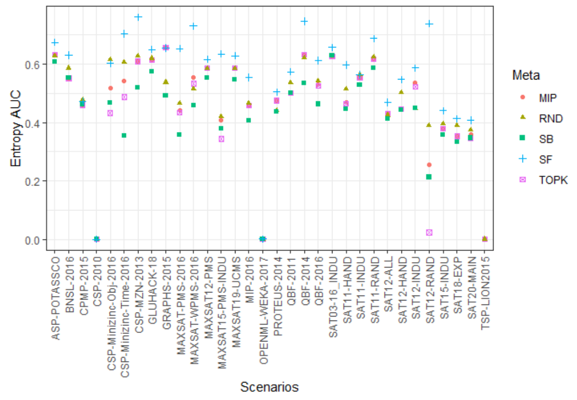
We investigate the interplay between regret and entropy for different selectors. Figure 3 shows the regret and entropy quadrants for MIP, RND, SB, SF and TOPK and their smoothed regret curves. We use regret and entropy AUC to draw the quadrants because each scenario has multiple values of regret and entropy for different values of $K$ for this set of selectors. The bottom right quadrant corresponds to low risk, robust portfolios, with entropy being high and regret being low. The top right quadrant correspond to high risk, robust portfolios. The top left quadrant correspond to high risk, concentrated portfolios. Finally, the bottom left quadrant corresponds to low risk, concentrated portfolios. From the regret-entropy quadrants in Figure 3 we see that SF produces low risk, robust portfolios, giving the best performances as most scenarios fall in the bottom right quadrant, and a few scenarios falling into the bottom left quadrant. As expected, RND produces risky but robust portfolios. Both MIP and TOPK produce similar portfolios, often being low risk and highly robust.

The graphs on the right in Figure 3 show log-10 regret for each scenario for each $K$. Thus, a scenario with 30 algorithms would produce 30 points for this graph. The cardinality is normalized to $[0, 1]$ for comparison purposes. Using the data points for all scenarios, we construct the "average" regret curve using local polynomial regression. These average-regret curves show how the regret decreases with increasing cardinality for each selector. For RND and SB, the average-regret curves start with $\log_{10}(\text{regret}) \approx 15$ and exhibit a slow descent, until $K < 0.75$. Towards the final part of the curve, RND and SB show a steeper descent. This shows that both RND and SB need higher cardinality to produce a better portfolio. In contrast, MIP, SF and TOPK start with $\log_{10}(\text{regret}) \approx 13$ and the descent at the beginning is steeper compared to RND and SB. That shows that when the cardinality increases, regret decreases faster.

In summary, these results indicate that while MIP produces the portfolios with the lowest regret, it does so by selecting a more concentrated set of algorithms compared with SF. Although RND performed the worst on average as expected, it performed better

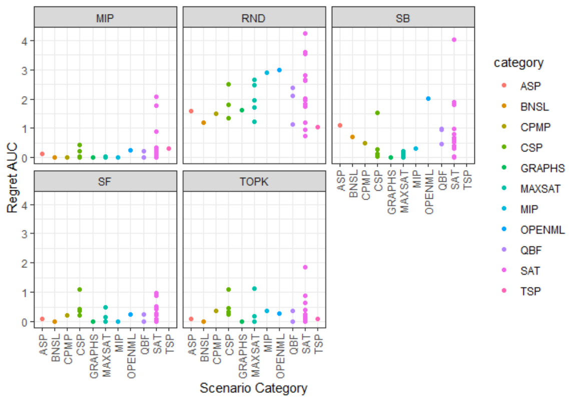


(a)

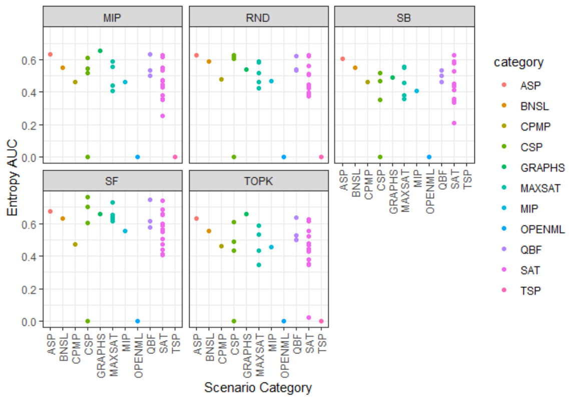


(b)

**Figure 1: Area under the curve (AUC) of the (a) log-10 regret and (b) entropy, across each scenario.**



(a) Regret by category



(b) Entropy by category

**Figure 2: Area under the curve (AUC) of the (a) log-10 regret and (b) entropy, across each scenario.**

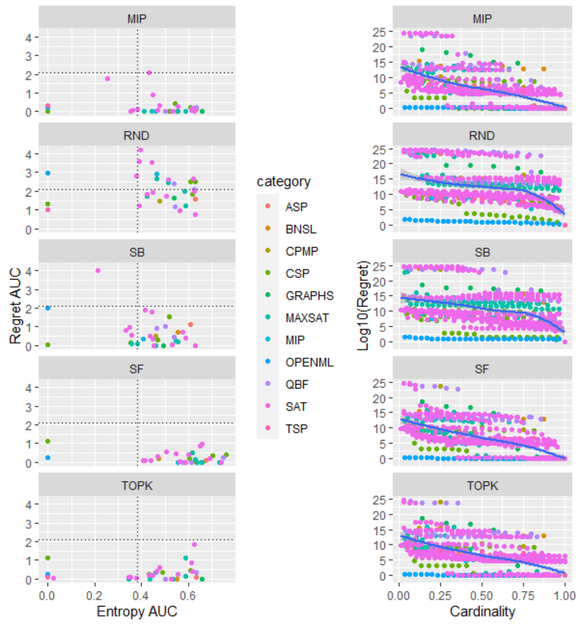


**Figure 3: Regret and entropy quadrants**

than SB depending on the instances. As we relaxed the threshold of "good" performance, by increasing $\epsilon$, we also allowed the probability of more than one algorithm being good. This transforms the algorithm selection problem from single class classification to multi-output classification, where more than one label can be valid. This is also an indicator of the diversity of the test instances, as more than one algorithm being good implies that perhaps they come from the same distribution of problems.

## 5 CONCLUSION

We propose a methodology for testing portfolio construction methods. Using the ASLib scenarios as benchmark problems [2], we test five methods: RND, TOPK, SF, SB and MIP. As performance metrics, we consider the average *regret* as a measure of the *riskiness* of a portfolio, and the entropy as a measure of the *robustness* of a portfolio. The results indicate that, the greedy SF was able to produce a portfolio with low risk and higher robustness, while MIP had the lowest risk portfolio. For some scenarios, RND produced better portfolios than SB and MIP, indicating that *No-Free Lunch* may also be in play in the Portfolio Construction Problem. As we accept less-than-best performance, portfolios increase their robustness, as more algorithms are deemed suitable. However, our results also indicate that the scenarios may lack some diversity on their instances.

## ACKNOWLEDGMENTS

Funding was provided by the Australian Research Council through grants No.: FL140100012 and CE140100049.

## REFERENCES

[1] A. Almakhlafi and J. Knowles. 2013. Systematic construction of algorithm portfolios for a Maintenance Scheduling Problem. In *2013 IEEE Congress on Evolutionary Computation.*
[2] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, et al. 2016. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence* 237 (2016), 41–58.
[3] P. Kerschke, H.H. Hoos, F. Neumann, and H. Trautmann. 2019. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation* 27, 1 (2019), 3–45.
[4] M. Lindauer, J.N. van Rijn, and L. Kotthoff. 2019. The algorithm selection competitions 2015 and 2017. *Artificial Intelligence* 272 (2019), 86–100.
[5] M.A. Muñoz and M Kirley. 2016. ICARUS: Identification of complementary algorithms by uncovered sets. In *2016 IEEE Congress on Evolutionary Computation (CEC).* IEEE, 2427–2432.
[6] J.R. Rice. 1976. The algorithm selection problem. In *Advances in computers.* Vol. 15. Elsevier, 65–118.
[7] H. Samulowitz and R. Memisevic. 2007. Learning to Solve QBF. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 1.* 255–260.
[8] D. Souravlias, K.E. Parsopoulos, I.S. Kotsireas, and P.M. Pardalos. 2021. *Algorithm Portfolios.* Springer International Publishing.
[9] J. Wawrzyniak, M. Drozdowski, and É Sanlaville. 2020. Selecting algorithms for large berth allocation problems. *European Journal of Operational Research* 283, 3 (2020), 844–862.