Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Sorting Sequential Portfolios in Automated Planning

Sergio Núñez and **Daniel Borrajo** and **Carlos Linares López**
Computer Science Department
Universidad Carlos III de Madrid (Spain)
{sergio.nunez,daniel.borrajo,carlos.linares}@uc3m.es

Abstract

Recent work in portfolios of problem solvers has shown their ability to outperform single-algorithm approaches in some tasks (e. g. SAT or Automated Planning). However, not much work has been devoted to a better understanding of the relationship between the order of the component solvers and the performance of the resulting portfolio over time. We propose to sort the component solvers in a sequential portfolio, such that the resulting ordered portfolio maximizes the probability of providing the largest performance at any point in time. We empirically show that our greedy approach efficiently obtains near-optimal performance over time. Also, it generalizes much better than an optimal approach which has been observed to suffer from overfitting.

1 Introduction

The notion of portfolio has been revived from the Modern Portfolio Theory literature [Markowitz, 1952] with the aim of improving the performance of modern solvers. This notion has been applied to some problem-solving tasks with remarkable results. Indeed, the results of the International Planning Competition 2014 (IPC 2014)[1] show that three awarded planners and twenty nine out of sixty seven participant planners in the deterministic tracks were portfolios or planners that consisted of a collection of solvers.

A portfolio is termed *static* if its behavior is not modified once it has been configured —i. e., neither the component solvers and their allotted time nor the order of the execution sequence can be altered. If the portfolio has the ability to define a new behavior for each input instance, then it is termed *dynamic*. Also, if the portfolio invokes solvers in sequence, it is termed *sequential*, as opposed to *parallel* portfolios, which run multiple solvers concurrently.

The component solvers of the sequential portfolios considered in this work are not allowed to share any information. Thus, they can not take advantage of their position by using information from previous executions (e.g., a cost upper bound in satisficing planning). Additionally, preemptive mode (i.e., the ability to stop execution and resume it later on if necessary) is not allowed.

The order in which the component solvers of a sequential portfolio are executed is a relevant issue that has not been analyzed in depth yet. Most successful portfolio approaches for Automated Planning only focus on maximizing performance (measured as overall quality or coverage) for a fixed time limit. We hypothesize that the order of the component solvers affects the performance of the portfolio over time. Thus, a sequential portfolio should be sorted if its performance over time is relevant. As a consequence, in optimal planning, the average time required to solve problems can be significantly reduced while in satisficing planning, lower-solution costs can be found more quickly, while preserving coverage or quality respectively.

An example of the interest in this particular problem can be found in the real world. The Power Restoration Problem [Hentenryck *et al.*, 2011] is very similar to the problem described in this work. The electrical components that must be repaired are problems to be solved by solvers. The reward (power restored) for repairing each particular component is the number of problems solved (or the quality of the solutions found) by a solver within its allotted time. This allotted time represents the time required (cost) to fix each component. Finally, the goal is to maximize the power flow (by repairing electrical components) in the network as quickly as possible (coverage or quality over time).

The contributions of this work are summarized as follows:

1. The problem of ordering component solvers in a sequential (static or dynamic) portfolio is formally defined.
2. Two different algorithms to solve the problem are proposed and compared. We empirically show that our greedy approach produces near-optimal solutions very quickly and that it generalizes much better than an optimal solution wrt to a specific training set which has been observed to suffer from overfitting.
3. An extensive evaluation is performed with the algorithms introduced here, a random ordering algorithm and others from the literature with data from the last three IPCs.

The paper is organized as follows: first, Section 2 introduces Related Work. Section 3 formally defines the problem targeted in this work. Sections 4 and 5 describe the optimal

[1] http://ipc.icaps-conference.org

and greedy approaches proposed. Section 6 reports the experimental results, and Section 7 ends with some conclusions and future work.

2 Background

Fast Downward Stone Soup (FDSS) [Helmert *et al.*, 2011] was one of the awarded planners in the IPC 2011. The FDSS technique explores the space of static portfolios that can be configured with a set of candidate planners using a hill-climbing search. The FDSS-1 portfolio for the satisficing track sorts planners by decreasing order of coverage. The component planners of FDSS-1 for optimal planning are sorted by decreasing memory usage. The remaining portfolios were sorted by other arbitrary orderings, which were not detailed by the authors.

PBP [Gerevini *et al.*, 2014] was the winner of the learning tracks of IPC 2008 and IPC 2011. It is a portfolio-based planner with macro-actions, which automatically configures a sequential portfolio of domain-independent planners for a specific domain. PBP uses the domain-specific knowledge with the purpose of selecting a cluster of planners for configuring the portfolio. This cluster of planners is sorted in ascending order of the allotted CPU time to run each solver.

Howe et al. [Howe *et al.*, 2000] introduced BUS, which runs 6 planners in a round-robin scheme, until one component planner finds a solution. For each input instance, it extracts some features from the given planning task. The component planners are then sorted in descending order of the ratio $\frac{P(A_i)}{T(A_i)}$, where $P(A_i)$ is the expected probability of success of algorithm A_i, and $T(A_i)$ is the expected run time of algorithm A_i. Both estimations are provided by linear regression models based on instance features.

Cenamor et al. presented two strategies for configuring sequential portfolios [Cenamor *et al.*, 2015]: IBACOP distributes uniformly all the available time among all planners selected by a Pareto efficiency analysis; IBACOP2, uses learned models to select the five planners with the highest confidence for solving the input task and distributes time uniformly among them. Both portfolios won the satisficing track of the IPC 2014. On the other hand, CEDALION is a new successful portfolio generation technique from highly parameterized solvers [Seipp *et al.*, 2015]. Among these, only IBACOP2 sorts its component planners —using the confidence provided by the learned models.

3 Formal Description

In Automated Planning, the performance of a solver s is measured over a set of instances I (s can be either a solver or a portfolio). Every solver s is executed over every instance $i \in I$ to obtain the set R_{si} of solutions. This set contains every solution found by s within a given time bound t. We consider time as discrete in this work with a discretization of one second. Each element of R_{si} stores the cost of the corresponding solution and a timestamp with the time required to find it. In case of solving problems optimally, solvers generate at most one solution per instance. In satisficing planning, solvers can generate multiple solutions. Therefore, the performance of s over time is measured by evaluating a specific metric over time. The most usual metrics are:

1. Coverage, $C(s, I, t)$: number of problems in the benchmark I solved by the solver s in time less or equal than t per instance. It is the metric of the optimal track of the last three IPCs.
2. Overall quality, $Q(s, I, t)$. Quality is mapped to the interval $[0, 1]$ according to the expression $\frac{\text{cost}_i^*}{\text{cost}_{s,i}}$ where $\text{cost}_{s,i}$ is the best solution found by solver s for instance i in time less or equal than t and cost_i^* is the best solution found for the same instance by any solver within the same time bound. The *overall quality* is computed as the sum of the quality over all instances. It is the metric for the satisficing track of the last three IPCs.

In the following, $P(s, I, t)$ denotes generically either coverage, $C(s, I, t)$ or overall quality, $Q(s, I, t)$. Recall that every solver is executed t seconds on each instance to compute a specific metric. Next, we analyze the performance of portfolios of solvers over time.

Definition 1 *A sequential portfolio ρ is a collection of n pairs $\langle s_i, t_i \rangle_{i=1}^n$, where s_i is a component solver and t_i is the time allotted to its execution.*

The preceding definition does not define any *ordering* among solvers in a portfolio. Indeed, the performance of the portfolio in time $T \geq \sum_i t_i$ is the same for any ordering.

Definition 2 *An ordering τ of a sequential portfolio ρ is a full permutation over the solvers in ρ that defines the execution sequence, $\tau \doteq \{s_1, s_2, \ldots, s_n\}$.*

Let ρ_τ denote the sorted sequential portfolio of solvers in ρ whose execution ordering is given by τ.

In order to analyze the performance of a sorted sequential portfolio, we need to analyze separately the contribution of each solver to the overall performance.

Definition 3 *A partial ordering τ_k is a partial permutation over the first k solvers in the full permutation $\tau : \tau_k \doteq \{s_1, s_2, ..., s_k\}$.*

Let ρ_{τ_k} denote the sorted sequential portfolio of solvers in ρ that considers only the first k solvers according to the ordering in τ_k.

Therefore, the first solver, s_1, completes its execution after t_1 seconds, the second solver will finish after (t_1+t_2) seconds and, in general, the j-th solver s_j will complete its execution after $\sum_{k=1}^{j} t_k$ seconds. Now, we can define how to measure the performance of the resulting portfolio over time as shown below.

Definition 4 *The performance of a sorted sequence of solvers ρ_τ for a given problem set I over time t, $P(\rho_\tau, I, t)$ is defined as the sum of the best performance over all solvers in ρ executed in the order specified by τ for every instance in I in time less or equal than t.*

In case that $t < \sum_{i=1}^{n} t_i$ not all component solvers will be considered to compute $P(\rho_\tau, I, t)$. In this case, only the solutions found by those solvers in ρ_τ that could be run before t are considered. The timestamp of each considered solution is

equal to the time required to find it by solver s_k plus $\sum_{i=1}^{k-1} t_i$. Next, we define the performance of a component solver, s_i, that occupies the i-th position in a permutation τ. It is computed as the increase in performance of the ordered portfolio by adding s_i to the portfolio.

Definition 5 *Let $P_{s_i}(\rho_\tau, I)$ denote the performance of a component solver s_i in a partial permutation τ_i (that considers only the first i solvers in τ) wrt the benchmark I:*

$$P_{s_i}(\rho_\tau, I) = P(\rho_{\tau_i}, I, t_{s_i}) - P(\rho_{\tau_{i-1}}, I, t_{s_{i-1}})$$

where τ_i denotes the partial permutation of all solvers in τ until s_i; t_{s_i} is the sum of the allotted times of all solvers in ρ_{τ_i}; and, s_{i-1} denotes the previous solver of s_i in the partial permutation.

The performance of a solver s_i is defined as a function of the ordered portfolio ρ_τ since different solvers in ρ or different orderings τ would yield different performances —i. e., the performance of a solver depends on the previous solvers. Consider a sequential portfolio ρ for optimal planning which consists of two solvers s_1 and s_2 which are allocated 4 and 7 seconds respectively. Let us assume that the performance of these solvers with respect to a set I of 20 planning tasks is:

- s_1 solves instances 11 to 20. Hence, $P(s_1, I, 4) = 10$.
- s_2 solves tasks 1 to 18, resulting in $P(s_2, I, 7) = 18$.

Assume further that both solvers solve the aforementioned instances in one second each. Therefore, $P(s_1, I, 1) = 10$ and $P(s_2, I, 1) = 18$. Figure 1 shows the performance over time of the two possible orderings for the given portfolio, $\tau_1 : (s_1, s_2)$ (red solid line) and $\tau_2 : (s_2, s_1)$ (blue dashed line). $P(s_1, I, 4)$ is equal to 10 since s_1 solved 10 instances within its time span. However, the performance of s_1 in each portfolio is different. $P_{s_1}(\rho_{\tau_1}, I)$ is equal to 10 because s_1 is the first solver to be executed. However, the performance $P_{s_1}(\rho_{\tau_2}, I)$ is equal to two since instances 11–18 have already been solved by the previous solver s_2. As shown in Figure 1, the performance of a sequential portfolio ρ_τ at time $T = \sum_i t_i$, $P(\rho, I, T)$, is the same for every permutation τ. However, the performance of these orderings over time differs. Figure 1 also exemplifies a case where the portfolio achieving its maximum performance sooner is not the one with the best overall performance and, indeed, the first portfolio (red solid line) inscribes a smaller area than the second one.

The key observation is that the performance of ordered portfolios ρ_τ over time can be seen as *bivariate density functions*, $f_{\rho_\tau}(x, t)$. They are defined as the probability that the sorted portfolio ρ_τ reaches a performance P equal to x in t seconds: $Prob(P = x, T = t)$. Accordingly, we define the probability function as follows.

Definition 6 *Let $Prob(P \leq x, T = t)$ denote the probability that a portfolio ρ_τ reaches a performance equal to x or less in time t:*

$$Prob(P \leq x, T = t) = \sum_x f_{\rho_\tau}(x, t)$$

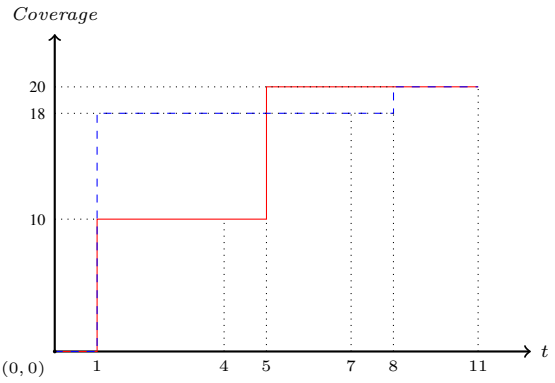

Figure 1: Performance of two different orderings of the same portfolio with respect to coverage.

This observation leads to propose the area inscribed by this probability function as the optimization criteria to compare different permutations τ of the same portfolio ρ. Thus, we define the optimization task as follows.

Definition 7 *Given a collection of n component solvers of a sequential portfolio ρ find the permutation τ^* of solvers $s \in \rho$ for a given benchmark I such that it maximizes $F_{\rho_\tau}(x, t)$:*

$$\begin{aligned} F_{\rho_\tau}(x, t) &= Prob(P \leq x, T \leq t) \\ &= \textstyle\sum_t Prob(P \leq x, T = t) \end{aligned}$$

As a result, this task will sort the component solvers of the input portfolio ρ with the aim of maximizing the area inscribed by the probability function shown in Definition 6 of the resulting ordering τ (see Definition 2).

4 Optimal Approach

We first use heuristic search with an admissible heuristic function to find the optimal ordering with respect to a given benchmark. Specifically, we propose Depth First Branch and Bound (DFBnB). It requires two parameters: a sequential portfolio ρ and the set R_{si}, which will be used to compute the area inscribed by the probability function of every combination of the component solvers.

To find the optimal ordering τ^*, DFBnB starts with the empty permutation τ_0. Each node m contains the current partial permutation τ_m and the set A of solvers that are not yet in ρ_{τ_m} (initially, $A = \{s_i \mid s_i \in \rho\}$ i. e., all solvers in ρ). The successors of each node are generated by adding a solver $s \in A$ to the current permutation (and thus removing it from A). Each node defines a partial permutation of the component solvers in ρ, while each leaf node defines a full permutation of all solvers in ρ.

DFBnB uses $f(m) = g(m) + h(m)$. The g-value is the area inscribed by $F_{\rho_{\tau_m}}$ after $T_{\rho_{\tau_m}}$ seconds, where $T_{\rho_{\tau_m}}$ is equal to the sum of the time spans of every solver in ρ_{τ_m}:

$$g(m) = F_{\rho_{\tau_m}}(P(\rho, I, T_{\rho_{\tau_m}}), T_{\rho_{\tau_m}})$$

Suppose that DFBnB is initially given the portfolio $\rho = \{\langle s_1, 540\rangle, \langle s_2, 630\rangle, \langle s_3, 630\rangle\}$. Assume also that the DFBnB search is in a state m, where only the solver s_3 has been

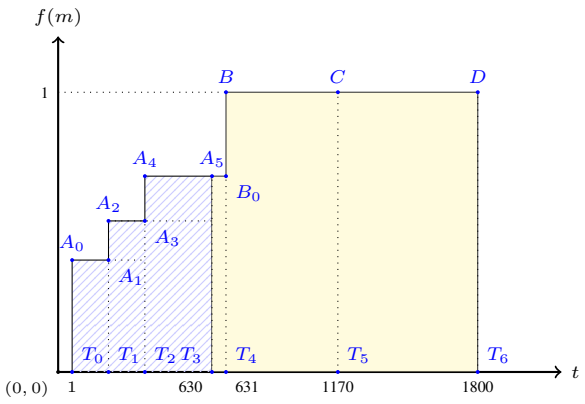

Figure 2: Example of the computation of $f(m)$

selected, so that $\rho_{\tau_m} : \{\langle s_3, 630\rangle\}$ and $T_{\rho_{\tau_m}} = 630$. Figure 2 shows the area inscribed by the probability function $F_{\rho_{\tau_m}}$ in the interval $[0, 630]$ (blue line pattern area), where points denoted with uppercase letters allow us to define the areas inscribed by the probability function. Thus, the area inscribed by $F_{\rho_{\tau_m}}$ is computed as the sum of the rectangular areas $area(T_0A_0A_1T_1)$, $area(T_1A_2A_3T_2)$ and $area(T_2A_4A_5T_3)$.

In relation to $h(m)$, we have defined an admissible heuristic termed SQUARE that optimistically estimates the area inscribed by the probability function F_A. It just assumes that the order of the solvers contained in A is not relevant so that the portfolio will reach the performance $P(\rho, I, T)$ of the portfolio ρ at time T with a 100% probability, one second after the first solver in A starts its execution. $h(m)$ is computed as follows:

$$\begin{aligned} h(m) =& Prob(P \leq P(\rho, I, T), T = T_{\rho_{\tau_m}}) + \\ & F_A(P(\rho, I, T), T_A) \\ =& Prob(P \leq P(\rho, I, T), T = T_{\rho_{\tau_m}}) + T_A - 1 \end{aligned}$$

where T_A is equal to the sum of the time spans of every solver in A. The area inscribed by F_A is composed of two rectangular areas. The first one is defined in the interval $[T_{\rho_{\tau_m}}, T_{\rho_{\tau_m}} + 1]$, the first second of the execution of the first solver in A. This area is represented by the first term in $h(m)$. It is computed as the probability value of reaching a performance equal to x or less in time $T_{\rho_{\tau_m}}$ multiplied by the time interval (one second). The second area is defined by the probability function F_A in the interval $[T_{\rho_{\tau_m}} + 1, T_{\rho_{\tau_m}} + T_A]$.

Following the example in Figure 2, $A = \{s_1, s_2\}$ and $T_A = 540 + 630 = 1170$. Since these solvers have not yet been included in ρ_{τ_m}, the area inscribed by F_A (yellow solid area) is computed using the SQUARE heuristic. The estimated yellow solid area is equal to the sum of the rectangular areas $area(T_3A_5B_0T_4)$ (first term of the heuristic function), $area(T_4BCT_5)$ (solver s_1) and $area(T_5CDT_6)$ (solver s_2). The rectangular areas $area(T_4BCT_5)$ and $area(T_5CDT_6)$ are equal to $T_A - 1 = 1169$. Hence:

$$\begin{aligned} f(m) = \; & area(T_0A_0A_1T_1) + area(T_1A_2A_3T_2) \\ & + area(T_2A_4A_5T_3) + area(T_3A_5B_0T_4) + 1169 \end{aligned}$$

This technique yields optimal orderings for a specific set of planning tasks. However, it can suffer from overfitting when evaluating its performance over a different benchmark.

5 Greedy Approach

The time required to find the optimal solution increases dramatically with the number of component solvers, since there are $n!$ different orderings. Thus, we propose an alternative approach based on greedy search to quickly find a suboptimal solution with good quality.

We assume that the performance $P_{s_i}(\rho_\tau, I)$ can be approximated with a straight line in the interval $[\delta, \delta + t_i]$, where δ is the sum of the allotted time to $\rho_{\tau_{i-1}}$ and t_i is the execution time of s_i. The slope of this line is computed as the performance $P_{s_i}(\rho_\tau, I)$ divided by t_i. We have selected the slope as a heuristic (to be denoted as SLOPE) because it is a conservative approximation of the growth in performance of a component solver in the portfolio. Also, it considers the performance of each component solver *wrt* the performance achieved by the previous solvers in the term $P_{s_i}(\rho_\tau, I)$ —see Definition 5.

We propose hill-climbing in the space of partial permutations of the input portfolio ρ as the search technique. It takes the same parameters as DFBnB described previously. The search is initialized with the empty permutation τ_0, and the set $A = \{s_i \mid s_i \in \rho\}$. At each step, it selects the solver $s_i \in A$ which has the largest ratio $P_{s_i}(\rho_\tau, I)/t_i$. Then, the selected solver is added to the current permutation and removed from A. Finally, the algorithm returns the ordered portfolio ρ_τ.

6 Experimental Setup and Results

The proposed algorithms (SLOPE and DFBnB) are compared with other ordering strategies and their performance is reported. Inspired by the ordering criteria used by state-of-the-art portfolios in Automated Planning, we have defined the following algorithms to compare against our solutions.[2]

- Shorter Time Spans (STS): inspired by PBP, this algorithm sorts the component solvers of a given portfolio in increasing order of the allotted time to run each solver.
- Memory Failures (MF): inspired by FDSS, it uses the number of times that each component planner exceeds the available memory limit (and does not solve the task) to sort the given portfolio in decreasing order.
- Decreasing Coverage (DC): inspired by FDSS, it uses the number of problems solved by each component solver to sort the input portfolio in decreasing order.
- Random: sorts the solvers of the input portfolio randomly. This algorithm generates five random orderings and reports the average score of all generated orderings.
- Confidence: uses the confidence provided by the learned models to sort the input portfolio in decreasing order. It is defined by IBACOP2. Therefore, it only will be applied in the comparisons with dynamic input portfolios.

[2]We also tried to use the ordering criteria defined by BUS against our approach, but the source code is not available.

	Training Score				Test Score			
	IPC 2008		IPC 2011		IPC 2011		IPC 2014	
Ordering Algorithm	Optimal	Satisficing	Optimal	Satisficing	Optimal	Satisficing	Optimal	Satisficing
DFBnB 2008	1.0000	1.0000	-	-	0.9421	0.9381	-	-
DFBnB 2011	-	-	1.0000	1.0000	-	-	0.9830	0.9479
SLOPE PORTFOLIO	0.9944	0.9862	0.9993	0.9977	1.0000	0.9735	0.9764	0.9656
STS PORTFOLIO	0.9944	0.9642	0.9980	0.9756	1.0000	0.9576	0.9824	0.9786
RANDOM	0.9869	0.9604	0.8716	0.9545	0.9433	0.9591	0.8601	0.8450
DC PORTFOLIO	0.9985	0.9818	0.8579	0.9439	0.8435	0.9516	0.9031	0.7072
MF PORTFOLIO	0.9733	0.9334	0.6457	0.9472	0.9253	0.9437	0.6227	0.8267

Table 1: Training and test results of the sorted portfolios using MIP-configured portfolios.

Tie-breaking is resolved by using the order in which solvers were initially specified. We have used an Intel Xeon 2.93 GHZ quad core processor with 8 GB of RAM. The time limit T used is 1800 seconds.

6.1 Static Input Portfolios

Equation (1) is used to measure the quality of the resulting orderings. This equation compares the area inscribed by the probability function of each sorted portfolio with the optimal sorting computed with DFBnB over the evaluation set. Thus, higher scores stand for better anytime behaviors.

$$score(\rho_\tau) = \frac{F_{\rho_\tau}(P(\rho, I, T), T)}{F_{\rho_{\tau^*}}(P(\rho, I, T), T)} \quad (1)$$

The performance of all the considered sortings has been evaluated with different evaluation test sets: either the training data set or an entirely new one (test set). The second type of experiments examines the generalization capabilities of all ordering approaches. Also, to avoid introducing any bias in the experimentation, different techniques for generating static sequential portfolios are considered:

- Mixed-integer programming technique (MIP). This approach focused on deriving the optimal static sequential portfolio for a particular metric and a given training set [Núñez *et al.*, 2012].
- Random portfolios. They consist of a random selection of planners from a pool of candidate planners (at most half the number of candidate planners) so that at least one problem is solved in each training domain. The time allotted to each component planner is also randomly chosen, such that the total time does not exceed T.
- Random-uniform portfolios. They are random portfolios where the total available time is uniformly distributed among all the component planners.

We have performed two sets of experiments. The first one considers all the planning tasks from the IPC 2008 to configure and sort the input portfolio (training set). The new domains defined in the IPC 2011 are then used to assess the performance of the resulting portfolio (test set). In this experiment, we have used the set of candidate planners considered in the design of FDSS (FDSS-1 or FDSS-2, depending on the experiment) to configure the input portfolios. The second set of experiments takes the whole collection of planning tasks from the IPC 2011 as training set and the domains of the IPC 2014 (which were not included in IPC 2011) as test set. The input portfolios have been configured considering all the participants of the IPC 2011 but LPRPGP.[3]

The size of the candidate and component planners sets are defined in the ranges $[8, 38]$ and $[3, 14]$ respectively. The smaller planner sets were used in optimal planning, since there were few participants in the last IPCs. As a reference, IBACOP2, the state-of-the-art portfolio, considers 12 candidate planners and five component planners.

The ordering generated by DFBnB for the training sets (IPC 2008 and IPC 2011) will be denoted as DFBnB 2008 and DFBnB 2011 respectively. Also, the ordering computed with DFBnB over the test sets only will be used to compute the test score of the orderings generated using the training set.

Table 1 shows the score of the resulting ordered portfolios using MIP portfolios. The training results show that the anytime behavior of the portfolio sorted with our greedy approach (SLOPE) is usually extremely close to the optimal performance. As it can be seen, the test results show that the orderings obtained by SLOPE and STS (using data from the IPC 2008) are the optimal orderings for the test set. Also, the permutation computed with the technique that generates the best ordering for the training data (DFBnB 2008) shows overfitting as expected; it is worse than the random ordering, and it does not generalize well to unknown domains. Moreover, the orderings generated with MF and DC usually perform worse than the RANDOM ordering. Finally, our greedy approach outperforms the other approaches with a remarkable score (IPC 2011) and generalizes well on the IPC 2014.

Table 2 presents the training and test results for the random and random-uniform portfolios. Since we are using random portfolios, we have executed 50 times the training and test phases, each one with a different random portfolio. As it can be seen, the SLOPE random portfolio achieves again a training score extremely close to the score of the best permutation (DFBnB solution) of the input portfolio. The test results for random portfolios show that the SLOPE portfolio outperforms others under the same conditions. Also, all the generated orderings usually perform better than RANDOM in the test set. The differences in test scores between SLOPE and STS are larger in satisficing planning than in optimal planning, mostly because in satisficing planning the ordering task is harder and there is much more variability in the areas of the portfolios.

Strikingly, the results for the random-uniform portfolio show that the SLOPE portfolio achieves a training score ex-

[3] We experienced problems with the CPlex license.

Random Portfolios								
	Training Score and Std. Deviation (average)				Test Score and Std. Deviation (average)			
	IPC 2008		IPC 2011		IPC 2011		IPC 2014	
Ordering Algorithm	Optimal	Satisficing	Optimal	Satisficing	Optimal	Satisficing	Optimal	Satisficing
DFBnB 2008	1.0000 - 0.0000	1.0000 - 0.0000	-	-	0.9559 - 0.0448	0.9445 - 0.0530	-	-
DFBnB 2011	-	-	1.0000 - 0.0000	1.0000 - 0.0000	-	-	0.9806 - 0.0170	0.9613 - 0.0450
SLOPE PORTFOLIO	0.9923 - 0.0095	0.9965 - 0.0038	0.9932 - 0.0109	0.9984 - 0.0030	0.9800 - 0.0245	0.9617 - 0.0478	0.9829 - 0.0115	0.9607 - 0.0409
STS PORTFOLIO	0.9850 - 0.0181	0.9900 - 0.0095	0.9794 - 0.0203	0.9752 - 0.0264	0.9797 - 0.0293	0.9595 - 0.0381	0.9774 - 0.0193	0.9394 - 0.0541
RANDOM	0.9661 - 0.0209	0.9571 - 0.0160	0.9143 - 0.0654	0.8394 - 0.0900	0.9255 - 0.0467	0.8792 - 0.0599	0.9485 - 0.0490	0.7895 - 0.1107
DC PORTFOLIO	0.9897 - 0.0113	0.9623 - 0.0284	0.9866 - 0.0154	0.9720 - 0.0246	0.9437 - 0.0606	0.8713 - 0.1082	0.9707 - 0.0180	0.8996 - 0.0858
MF PORTFOLIO	0.9426 - 0.0375	0.9332 - 0.0434	0.9393 - 0.0443	0.7977 - 0.1587	0.9438 - 0.0486	0.8671 - 0.1095	0.9773 - 0.0253	0.6877 - 0.2051
Random Uniform Portfolios								
DFBnB 2008	1.0000 - 0.0000	1.0000 - 0.0000	-	-	0.9370 - 0.0302	0.9086 - 0.0550	-	-
DFBnB 2011	-	-	1.0000 - 0.0000	1.0000 - 0.0000	-	-	0.9776 - 0.0139	0.8878 - 0.0486
SLOPE PORTFOLIO	0.9996 - 0.0010	0.9996 - 0.0008	0.9992 - 0.0013	0.9995 - 0.0015	0.9353 - 0.0306	0.9123 - 0.0567	0.9791 - 0.0143	0.8919 - 0.0453
STS PORTFOLIO	0.9560 - 0.0228	0.9494 - 0.0180	0.8941 - 0.0535	0.8429 - 0.0644	0.8588 - 0.0497	0.8254 - 0.0704	0.9261 - 0.0331	0.7897 - 0.0823
RANDOM	0.9648 - 0.0216	0.9664 - 0.0098	0.9411 - 0.0255	0.8745 - 0.0431	0.9311 - 0.0271	0.8525 - 0.0433	0.9487 - 0.0149	0.7982 - 0.0529
DC PORTFOLIO	0.9913 - 0.0104	0.9623 - 0.0142	0.9636 - 0.0131	0.9871 - 0.0112	0.9413 - 0.0329	0.8183 - 0.0782	0.9530 - 0.0144	0.8694 - 0.0437
MF PORTFOLIO	0.9442 - 0.0304	0.9442 - 0.0208	0.9412 - 0.0314	0.9086 - 0.0558	0.9566 - 0.0292	0.8371 - 0.0800	0.9605 - 0.0339	0.7711 - 0.0981

Table 2: Training and test results of the sorted portfolios using random and random-uniform portfolios.

tremely close to the score obtained by the optimal sorting despite the fact that the uniform method penalizes the SLOPE heuristic. This method also penalizes the STS algorithm. However, the STS portfolio performs worse than the RANDOM portfolio. As it can be seen, the difference in test score between SLOPE and STS (the two algorithms that are penalized by using the uniform time assignment) is quite large. The scores of SLOPE, DFBnB 2008 and 2011 are very close. However, the time required by our greedy approach is exponentially shorter than the time required by DFBnB. Overall, all ordering algorithms (but DFBnB) sort a given portfolio in less than one second, while DFBnB can take several days to sort a given portfolio (depending on the number of component planners).

6.2 Dynamic Input Portfolios

We now apply all the ordering algorithms defined above to IBACOP2, the winner of the IPC-2014 (satisficing planning). Instead of configuring the same portfolio for all test instances (as the approaches used in the previous experiments), IBACOP2 generates a different sequential portfolio for each input instance. Therefore, the score of each sorted ordering is computed as $\sum_{i \in I} \text{score}(\rho_{i\tau})$, where $\rho_{i\tau}$ is the ordered portfolio computed for each input instance i. The generation of each portfolio is based on learned data using a training set.

We have considered all the instances defined in the IPC 2011 to sort the input portfolios and the domains of the IPC 2014 (which were not included in IPC 2011) to evaluate the resulting orderings.

Ordering algorithm	Training score	Test score
DFBnB 2011	160.00	105.93
SLOPE	159.94	108.95
STS	144.64	107.32
RANDOM	149.30	107.89
DC	158.31	105.92
MF	152.87	107.10
CONFIDENCE	-	108.37

Table 3: Score of the resulting orderings using IBACOP2.

As it can be seen in Table 3, the SLOPE portfolio achieves again near-optimal solutions in the training set. The training score obtained by the DC portfolio is remarkable while the training score of the STS portfolio is worse than the score achieved by the random ordering (similarly to the random-uniform portfolios). It is mainly due to the uniform method, which is applied by IBACOP2 to distribute the available time among the component solvers. The CONFIDENCE portfolio does not show training score because it was ordered by its learned models. On the other hand, the test score of the resulting orderings show that SLOPE again outperforms others. However, the test score of all the permutations are close among them.

7 Conclusions and Future Work

In this work, we have presented a formal definition of the problem of sorting the component solvers in sequential portfolios. This open problem is addressed with the aim of improving the performance of the portfolios over time. In addition, we have introduced two algorithms to solve the aforementioned problem. The first one solves the problem optimally for a given data set using DFBnB and an admissible heuristic. Optimality is only guaranteed for the given training set. The second one is a greedy approach that uses the ratio between performance of each solver and execution time. Our results show that:

- The performance of the portfolio over time can significantly vary by using different ordering algorithms. Also, it can be improved by using a good ordering strategy, as shown by comparing against random orderings.
- DFBnB obtains an optimal ordering for the training set, but it does not generalize well to unknown domains. This optimal ordering shows overfitting when evaluating its performance over a different test set.
- Our greedy technique, SLOPE, computes orderings very fast, and obtains near-optimal solutions when compared against the optimal technique. Also, it generalizes much better than the state-of-the-art ordering techniques. The

good behavior of SLOPE does not depend on the algorithm used for generating the input portfolio, as shown by using randomly generated portfolios (with uniform and non-uniform times). We conjecture, in view of these results, that it is going to be difficult to find a better algorithm in terms of the balance between computation time, generalization power and quality of results.

In the future, we want to apply the ordering algorithms to portfolios generation techniques that generate several portfolios while solving the given input instance.

Acknowledgments

This work has been partially supported by MICINN project TIN2011-27652-C03-02.

References

[Cenamor *et al.*, 2015] Isabel Cenamor, Tomás de la Rosa, and Fernando Fernández. IBaCoP and IBaCoP2 Planner. *Planner description, ICP 2014*, 2015.

[Gerevini *et al.*, 2014] Alfonso Gerevini, Alessandro Saetti, and Mauro Vallati. Planning through Automatic Portfolio Configuration: The PbP Approach. *J. Artif. Intell. Res. (JAIR)*, 50:639–696, 2014.

[Helmert *et al.*, 2011] Malte Helmert, Gabriele Röger, and Erez Karpas. Fast Downward Stone Soup: A Baseline for Building Planner Portfolios. *In ICAPS 2011 Workshop on Planning and Learning*, pages 28–35, 2011.

[Hentenryck *et al.*, 2011] Pascal Van Hentenryck, Carleton Coffrin, and Russell Bent. Vehicle Routing for the Last Mile of Power System Restoration. In *Proceedings of the 17th Power Systems Computation Conference (PSCC11), Stockholm, Sweden*, 2011.

[Howe *et al.*, 2000] A. Howe, E. Dahlman, C. Hansen, M. Scheetz, and A. Von Mayrhauser. Exploiting competitive planner performance. *Recent Advances in AI Planning*, pages 62–72, 2000.

[Markowitz, 1952] Harry Markowitz. Portfolio Selection. *The Journal of Finance*, 7(1):77–91, March 1952.

[Núñez *et al.*, 2012] Sergio Núñez, Daniel Borrajo, and Carlos Linares López. Performance Analysis of Planning Portfolios. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS, Niagara Falls, Ontario, Canada, July 19-21, 2012*. AAAI Press, 2012.

[Seipp *et al.*, 2015] Jendrik Seipp, Silvan Sievers, Malte Helmert, and Frank Hutter. Automatic Configuration of Sequential Planning Portfolios. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI 2015, Austin, Texas, USA, January 2530, 2015*, 2015.