Sorting Sequential Portfolios in Automated Planning

Submission #1549

Abstract

Recent work on portfolios of problem solvers have shown that they are able 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. Experimental results on Automated Planning show that sorting the component solvers using a greedy approach can significantly improve the performance of the portfolio over time.

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.

This work focuses on static sequential portfolios. A portfolio is termed *static* if its behavior can not change once it has been configured —i. e., neither the component solvers and their allotted time nor the order of the execution sequence. Also, if the portfolio invokes solvers in sequence, then it is termed *sequential*, as opposed to *parallel* portfolios, which run multiple solvers concurrently. Additionally, according to the rules of most international problem-solving competitions, 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 in overall quality or coverage; i.e., a measure of the solution cost wrt the best solution and the number of solved problems respectively) 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. The resulting portfolio will improve anytime behavior while preserving performance. Hence, 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.

The problem discussed in this work is applicable to real-world problems. For instance, during the Hurricane Sandy (2012) there were lots of blackouts in New York City. Thus, several electrical failures (generators, electrical lines, etc.) had to be repaired to restore the electricity as quickly as possible. This situation combines two problems, a logistic problem (parts that are needed to fix the electrical components) and a scheduling problem (Power Restoration Problem) (Hentenryck, Coffrin, and Bent 2011). The last one is very similar to the problem described in this work. The electrical components that must be repaired by workers are the problems to be solved by planners. The reward (power restored) for repairing each particular component is the number of problems solved (or the quality of the solutions found) by a planner 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 in the network as quickly as possible (coverage or quality over time).

The contributions of this work are summarized as follows:

1. We address an open problem in the automated design of portfolios that has not been theoretically analyzed yet: the problem of sorting the execution sequence of the component solvers in a sequential portfolio. We focus on ordering these solvers with the purpose of maximizing the probability of providing the largest overall quality or coverage at any point in time.
2. We introduce a formal definition for the problem of ordering the component solvers in a sequential portfolio as a function defined over time.
3. We propose two algorithms to solve the problem. The first algorithm solves the problem optimally for a given data set. The second one is based on a greedy approach to find a solution with good quality. We empirically show that

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

solving the problem greedily provides near-optimal solutions very quickly.

4. We define three ordering algorithms inspired on the sorting criteria used by state-of-the-art portfolios in Automated Planning. We apply these algorithms, a random ordering algorithm and both proposed approaches, to the optimal and satisficing tracks of the IPC 2011. Finally, we report the quality of the different orderings compared with the portfolio optimally ordered for these tracks.

The paper is organized as follows: first, Section 2 introduces Related Work. Next, Section 3 defines the problem addressed in this work. Section 4 describes the algorithm proposed to find the optimal sorting of a sequential portfolio. Section 5 presents the algorithm proposed to solve the aforementioned problem using a greedy search approach. Section 6 shows the experimental results in optimal and satisficing planning. Finally, Section 6 presents the conclusions of this work and introduces future work.

Background

In this section, we describe the state-of-the-art portfolio approaches in Automated Planning, focusing on the ordering strategies used to sort their component solvers.

Fast Downward Stone Soup (FDSS) (Helmert, Röger, and Karpas 2011) was one of the awarded planners in the IPC 2011. In particular, two portfolios (denoted as FDSS as well, FDSS-1 and FDSS-2) were generated using the FDSS technique to the sequential (optimal and satisficing) tracks of 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. Once the search algorithm has finished, the order of the component planners in the resulting portfolio is sorted. 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.

PBP (Gerevini, Saetti, and Vallati 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. The configuration relies on some knowledge about the performance of the planners in the portfolio for the specific domain and the observed utility of automatically generated sets of macro-actions. 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) described BUS, one of the first portfolio approaches for planning. BUS runs 6 planners in a round-robin scheme. The portfolio halts as soon as 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 by $\frac{P(A_i)}{T(A_i)}$, 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.

An alternative approach to the previous ones consists of configuring sequential portfolios using learnt performance models (Roberts and Howe 2006). This approach learns the success rates of the candidate planners on a training problems set. For each input problem, a sequential portfolio is configured and its component planners are sorted by decreasing probability of success. The resulting portfolio is executed using a round robin strategy until a solution is found.

Formal Description

In this work, we analyze the relationship between the order of the component solvers and the performance of the static sequential portfolio over time.

Definition 1 *A static sequential portfolio ρ is a collection of n pairs $\langle s, t\rangle$, where s is a component solver and t is the time allotted to the execution of the component solver s.*

The static sequential portfolios considered in this work do not allow their component solvers to share any information among them. Thus, these solvers can not take advantage of their position in the portfolio by using information from the previous solvers, as for instance, a cost upper bound in satisficing planning. Since we are interested in portfolios ordering algorithms, we define next the notion of *ordering*.

Definition 2 *An ordering τ of a static sequential portfolio ρ defines the sequence in which the component solvers of ρ should be executed. Thus, a sorted static sequential portfolio ρ_τ is a sorted collection of n pairs $\langle s_i, t_i\rangle$, where s_i is the i th component solver and t_i is is the allotted time to s_i.*

A sequential portfolio is usually computed for a fixed time limit T_L, such that the portfolio achieves the best performance in time T_ρ, where $T_\rho \leq T_L$. The total allotted time T_ρ in the sequential portfolio is computed as $\sum_{\langle s,t\rangle \in \rho} t$. The fixed time limit T_L is usually taken from the settings of the international competitions or, in general, from the experimental set-up. Also, time is considered discrete in this work, in one second intervals.

The performance of a solver s is measured over a set of instances I. The solver s is executed with every instance $i \in I$ to obtain the set R_{si} of solutions. In case of solving problems in optimal planning, a solver s generates at most one solution $r \in R_{si}$ for each instance $i \in I$, since r should be optimal. Otherwise (satisficing planning), a solver can generate more than one solution. We have chosen two types of metrics to measure performance in this work:

1. Coverage, $C(s, I)$: the number of problems solved by solver s in the benchmark I within the time limit (each instance should be solved in a time less than or equal to T_L). It is the metric for the optimal track of IPC 2014.

2. Overall quality, $Q(s, I)$. Satisficing planners can generate more than one solution, each with a different quality. The quality of each solution $r \in R_{si}$ found by the planner s for an instance i, denoted as $q(s, i, r)$, is computed in the range $[0, 1]$ as $\frac{lowest\_solution\_cost_i}{solution\_cost_{sir}}$ if the instance i is solved by s. Otherwise, the solution quality is zero.

   The overall quality $Q(s, I)$ of a solver s with respect to a set of instances I is computed as the sum of the best quality solution $q(s,i,r)$ achieved for every instance $i \in I$. It is the metric for the satisficing track of the last three IPCs.

A component solver is a part of a sequential portfolio and it is always executed by that portfolio, as opposed to single solvers, which are independent. Thus, we define next the performance of a single solver.

Definition 3 *The performance of a solver s for a given problem set I, $P(s,I)$, is the coverage $C(s,I)$ (optimal planning) or the overall quality $Q(s,I)$ (satisficing planning) achieved by the solver s in I.*

This definition is valid for the performance of an ordered portfolio ρ_τ for a given benchmark I, $P(\rho_\tau, I)$, and the performance of a single solver s for a given I, $P(s,I)$. However, the performance $P(s,I)$ of a single solver s and the performance of a *component* solver s in a portfolio, $P(s,I,\rho_\tau)$, are not the same concept. While $P(s,I)$ is computed executing s in isolation, $P(s,I,\rho_\tau)$ is computed considering the solutions found by the previous solvers in ρ_τ as follows.

Definition 4 *Given the performance of the portfolio ρ_τ before using solver s, $P(\rho^b_{\tau_s}, I)$, and the performance after running s for its allotted time, $P(\rho^a_{\tau_s}, I)$, the performance of a solver s running in a portfolio ρ_τ is $P(s,I,\rho_\tau) = P(\rho^a_{\tau_s}, I) - P(\rho^b_{\tau_s}, I)$. Hence, it is the increase on the performance of portfolio ρ_τ after running s.*

Therefore, using these definitions, the performance of a solver s can be different when running it alone than when been executed in a portfolio. For instance, suppose that a given portfolio ρ for optimal planning consists of two solvers s_1 and s_2 which are allocated t_1 and t_2, respectively. Let us assume further than the performance of these two solvers over a set I of 20 planning instances is described as follows:

- s_1 solves instances 11 to 20. $P(s_1, I) = 10$ at $t_1 = 11$.
- s_2 solves tasks 1 to 18. Hence, $P(s_2, I) = 18$ at $t_2 = 5$.

Also, both solvers solve each aforementioned instances in one second. Figure 1 shows the performance over time of the two possible orderings for the given portfolio, $\rho_1 : (\langle s_1, 11\rangle, \langle s_2, 5\rangle)$ (red line) and $\rho_2 : (\langle s_2, 5\rangle, \langle s_1, 11\rangle)$ (blue line). $P(s_1, I)$ is equal to 10 since s_1 solved 10 instances within its time span $t_1 = 11$. However, the performance of s_1 in each portfolio is different. In ρ_1, the performance $P(s_1, I, \rho_1)$ is equal to 10, because s_1 is the first solver to be executed. However, the performance $P(s_1, I, \rho_2)$ of this solver in portfolio ρ_2 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 for a given benchmark I at time T_L, $P_\rho = P(\rho, I)$, is the same for every permutation τ of its component solvers. However, the performance of these orderings over time might differ. Thus, the order of the component solvers affects the performance of the portfolio over time. We aim to find the ordering τ^* of the given portfolio ρ such that this order maximizes the probability of providing the largest performance at any point in time given I; i.e. the ordering τ^* that shows the best performance curve $P(\rho_{\tau^*}, I)$ over the interval $(0, T_L]$.

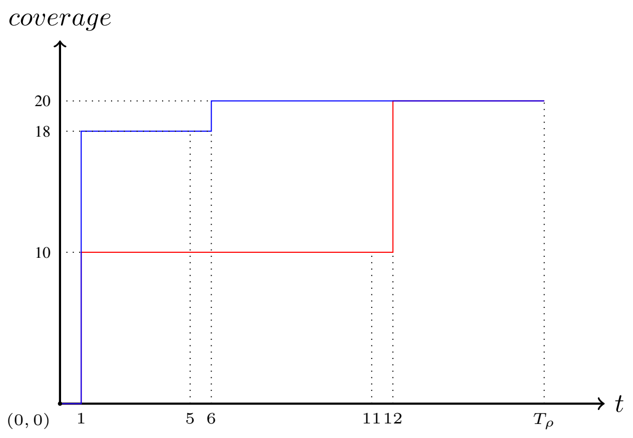

Figure 1: Performance of two different orderings of the same portfolio wrt. coverage.

The performance of solvers and portfolios are functions defined over time. Thus, these functions can be seen as *density functions*. This function describes the probability for the performance p to take a value x, where p is the performance $P(\rho_\tau, I)$ of a portfolio ρ_τ in a given benchmark I. Hence, the *density function* $f_{\rho_\tau}(p = x, t)$ is defined as the probability that the sorted portfolio ρ_τ reaches a performance p equal to x after t seconds. Accordingly, we define next the probability function of a sorted portfolio.

Definition 5 *For a particular sorted portfolio ρ_τ : $(\langle s_1, t_1\rangle, \langle s_2, t_2\rangle, \ldots, \langle s_n, t_n\rangle)$, the probability function $F_{\rho_\tau}(P \leq x, t)$, or the probability that the portfolio ρ_τ will reach a performance P equal to x or less in time t, is defined as $\sum_0^t f_{\rho_\tau}(p = x, t)$.*

This observation leads to propose the area inscribed by the 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 6 *Given a collection of n component solvers of a static sequential portfolio ρ find the permutation τ^* of solvers $s \in \rho$ for a given benchmark I such that it maximizes $\sum_{t=0}^{T_L} F_{\rho_\tau}(P \leq P_\rho, t)$.*

Since a static sequential portfolio ρ is composed of n solvers, there are $n!$ different sortings. All the different permutations of the same portfolio ρ achieve the same performance at time T_ρ.

Optimal Approach

The optimization problem stated in the previous section admits various formulations to solve it. In this section, we use heuristic search with an admissible heuristic function with the aim of finding the optimal ordering of the portfolio.

Specifically, we propose Depth First Branch and Bound (DFBnB), an admissible (thus complete) search algorithm that is typically used when the depth of the search tree is bounded. It explores a state space in depth-first order and finds many suboptimal solutions with increasingly better cost until the incumbent solution is proven to be optimal. DFBnB combines the space savings of depth-first search with heuristic information. An admissible heuristic value can be added to the g-value in order to increase the amount of pruning. Also, if the generated children of every given node are carefully sorted, a lower-cost solution can be found more quickly, further improving the pruning efficiency.

DFBnB requires two parameters: the collection of pairs $\langle s, t\rangle$ that defines the portfolio ρ and the results set R_{si}. This set will be used to compute the area inscribed by the probability function of every combination of the component solvers of the given portfolio ρ.

The function used to guide DFBnB is $f(m) = g(m) + h(m)$, where $g(m)$ is the cost of the path from the initial state to the current node. In our problem, the g-value is the area inscribed by $F_{\rho_{\tau_m}}$ after $T_{\rho_{\tau_m}}$ seconds, where τ_m is the ordering considered in node m. If node m is not a leaf node, the sorting τ_m does not contain all the component solvers of the input portfolio ρ. The set A is composed of all solvers that have not been included yet in τ_m. Thus, $g(m)$ is computed using equation (1), where $T_{\rho_{\tau_m}}$ is equal to the sum of the time spans of every solver in ρ_{τ_m}.

$$g(m) = \sum_{t=0}^{T_{\rho_{\tau_m}}} F_{\rho_{\tau_m}}(P \leq P_\rho, t) \tag{1}$$

For instance, Figure 2 shows the probability function $F_{\rho_{\tau_m}}$ in the interval $[0, T_{\rho_{\tau_m}}]$ (blue area) and the estimated area inscribed by F_A in the interval $[T_{\rho_{\tau_m}}, T_{\rho_{\tau_m}} + T_A]$ (yellow area) in a particular state m of the DFBnB search. The search algorithm took $\rho = \{\langle s_1, t_1\rangle, \langle s_2, t_2\rangle, \langle s_3, t_3\rangle\}$ as the input portfolio. The aim of the blue points denoted with uppercase letters in Figure 2 is to allow us to define the areas inscribed by the probability functions $F_{\rho_{\tau_m}}$ and F_A. In this state, the algorithm has only selected the first solver, s_3, so that $\rho_{\tau_m} : \{\langle s_3, t_3\rangle\}$ and $T_{\rho_{\tau_m}} = t_3$. Thus. the area inscribed by $F_{\rho_{\tau_m}}$ is computed as the sum of triangular areas $area(OA_0T_0)$, $area(A_0A_1A_2)$, $area(A_1AA_3)$ and the rectangular areas $area(T_0A_0A_2T_1)$ and $area(T_1A_1A_3T_2)$.

On the other hand, $h(m)$ estimates the cost of reaching the goal from the current state. We have defined an admissible heuristic termed SQUARE that estimates optimistically 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 of the portfolio ρ at time T_L with a 100% probability, one second after the first solver in A starts its execution. It is computed as follows, where T_A is equal to the sum of the time spans of every solver in A:

$$\begin{aligned} h(m) &= F_{\rho_{\tau_m}}(P \leq P_\rho, T_{\rho_{\tau_m}}) + \textstyle\sum_{t=1}^{T_A-1} F_A(P \leq P_\rho, t) \\ &= P(\rho_{\tau_m}, I) + \textstyle\sum_{t=1}^{T_A-1} 1 \\ &= P(\rho_{\tau_m}, I) + T_A - 1 \end{aligned}$$

Following the example in Figure 2, the component solvers that have not been included in ρ_{τ_m} (s_1 and s_2) have been added to A, so that $T_A = t_1 + t_2$. Since these solvers

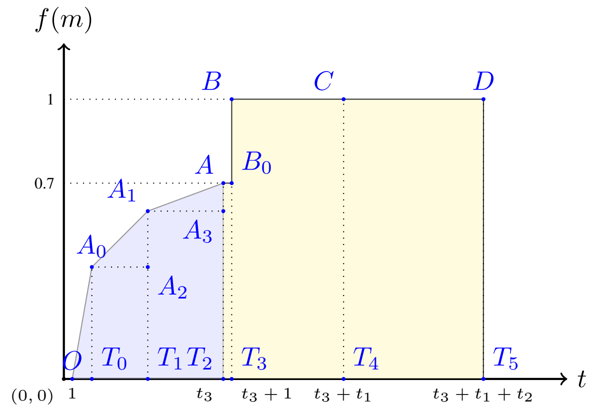

Figure 2: Example of the computation of $f(m)$

have not been selected by the search algorithm to be part in ρ_{τ_m}, the area inscribed by F_A is computed using the SQUARE heuristic. As it can be seen in Figure 2, the estimated yellow area is equal to the sum of the rectangular areas $area(T_2AB_0T_3)$ (performance of ρ_{τ_m}, since s_1 does not solve any problem during the first second, given that time is discrete), $area(T_3BCT_4)$ (solver s_1) and $area(T_4CDT_3)$ (solver s_2). $area(T_2AB_0T_3)$ is equal to $P(\rho_{\tau_m}, I)$, so that $area(T_2AB_0T_3) = 0.7$. The value of $f(m)$ in the current node is computed as:

$$\begin{aligned} f(m) = \quad & area(OA_0T_0) + area(A_0A_1A_2) \\ + \quad & area(A_1AA_3) + area(T_0A_0A_2T_1) \\ + \quad & area(T_1A_1A_3T_2) + 0.7 + t_1 + t_2 - 1 \end{aligned}$$

Heuristic Approach

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

The performance $P(s, I, \rho_\tau)$ of a component solver s can be approximated with a straight line in the interval $[\delta, \delta + t]$, where δ is the sum of the allotted time to the previous solvers in the portfolio ρ_τ and t is the allotted time to s. The slope of this line is computed as the performance $P(s, I, \rho_\tau)$ of solver s after t seconds divided by t. We have selected the slope as a heuristic because it is a conservative approximation of the growth in performance of a component solver in the portfolio. Also, it considers the performance provided by each component solver to the performance achieved by the previous solvers, since $P(s, I, \rho_\tau)$ is computed considering every solution found by the previous solvers in the portfolio, as described above.

We propose a hill-climbing search in the space of permutations of the input portfolio. The search will be guided by a new heuristic termed SLOPE. This heuristic consists of selecting at each step the component solver s_i which has the largest ratio $P(s_i, I, \rho_\tau)/t_i$, where $P(s_i, I, \rho_\tau)$ is the performance of the solver s_i in the portfolio ρ_τ over the training set I. This ratio will be re-computed at each step since the value $P(s_i, I, \rho_\tau)$ of each component solver is computed according to the performance achieved by the previous solvers, which are the same in every step except that a new component solver is added in every iteration. As a reminder $P(s_i, I, \rho_\tau)$ is the increase in performance by using s after the previous solvers in the portfolio. Thus, the value of $P(s_i, I, \rho_\tau)$ at each step can only be lower or equal than its value from the previous step.

The proposed approach takes the same parameters as DFBnB described in the previous section. Algorithm 1 starts from an initial empty portfolio. At each step, it computes the ratio $P(s_i, I, \rho_\tau)/t_i$ for every component solver in ρ. Then, the component solver with the largest ratio $P(s_i, I, \rho_\tau)/t_i$ is added to the tail of the sequential portfolio and removed from the set ρ. Finally, the algorithm returns the sorted sequential portfolio.

Algorithm 1 Hill-climbing algorithm with SLOPE heuristic

```
Input: A set of pairs $\langle solver, time \rangle$ $\rho$ and a results set $R_{si}$
Output: A sorted collection of pairs $\langle s_i, t_i \rangle$
  portfolio := {}
  repeat
    bestSuccessor := arg max$_{C \in \rho}$ slope(portfolio, $\rho$, $R_{si}$)
    portfolio := portfolio $\cup$ bestSuccessor
    $\rho$ := $\rho \setminus$ bestSuccessor
  until $\rho$ = {}
  return portfolio
```

Experimental Setup and Results

This section presents the evaluation of the proposed algorithms against other ordering strategies. To select those, we looked at the ordering algorithms used by state-of-the-art portfolios in Automated Planning, as FDSS and PBP. Inspired by those ordering algorithms, we have defined the following criteria to compare against our solutions:

- Shorter Time Spans (STS): sorts the component solvers of a given portfolio by the allotted time to run each solver (in increasing order).
- Memory Failures (MF): uses the number of times that each component planner exceeds the available memory limit to sort the given portfolio (in decreasing order).
- Decreasing Coverage (DC): uses the number of solved problems by each component solver to sort the input portfolio (in decreasing order).

BUS sorted its component solvers by decreasing order of the expected probability of success of each solver divided by the expected run time of that solver. Thus, BUS computes a portfolio (which contains its six candidate solvers) and sorts its component solvers for each input instance. This approach aims to minimize the expected cost (time) to solve every instance problem, as in our case for optimal planning. However, we always use the same time slot for each component solver and the same order in the resulting portfolio. Also, we do not use probability estimations. Instead, we use the performance (coverage or quality) provided by each component solver in the resulting portfolio[2].

We have used the area inscribed by the probability function of the resulting portfolios as metric to assess the different algorithms. Since DFBnB finds the optimal sorting of a given portfolio, we have compared the area inscribed by the probability function of each sorted portfolio with the optimal sorting using equation (2). This equation assigns a score from 0 to 1, where higher scores stand for better anytime behaviors:

$$score(\rho_\tau) = \frac{\sum_{t=0}^{T_L} F_{\rho_\tau}(P \leq P_\rho, t)}{\sum_{t=0}^{T_L} F_{\rho_{\tau^*}}(P \leq P_\rho, t)} \qquad (2)$$

In this section, we have used as input the portfolios computed with the technique described in (Núñez, Borrajo, and Linares López 2012), which has been shown to work well in Automated Planning. Given the range of tasks (satisficing and optimal planning), we do not believe we introduced a bias for the sorting algorithm by using a particular portfolio generation technique. Besides, our approach is independent of the algorithm used to generate the input portfolio.

We evaluate our approaches over the training set I because: first, we are not comparing the performance of different portfolio generation techniques; secondly, we are afraid that differences obtained over a different set might be due to differences in the performance of the portfolio generation technique. Thus, by fixing the training set we expect to highlight the differences due solely to the sorting criteria.

The hardware used to run the experiments is an Intel Xeon 2.93 GHZ quad core processor (64 bits) with 8 GB of RAM memory. Also, the time limit T_L used in every experiment for automated planning is equal to 1800 seconds.

Satisficing Planning

In our first experiment, we have generated the input portfolio using all the participant planners and all the planning tasks defined for the sequential satisficing track of the IPC 2011. The results set R_{si} has been generated by running each component planner with every planning instance from the benchmark of the aforementioned track.

First, we have defined the optimal ordering of the given portfolio by running DFBnB. It took seven days to compute it. Next, we have applied Algorithm 1, STS, MF, DC and a random ordering algorithm to the input portfolio. The time taken to compute the resulting orderings was less than one second. Next, we have computed the quality of these ordered portfolios using equation (2). Finally, we show the area inscribed by the resulting ordered portfolios in the satisficing track of the IPC 2011.

Table 1 shows the score of these portfolios. Obviously, the score of the portfolio sorted with DFBnB is equal to 1. The difference between the score of the portfolio sorted with STS and the optimal sorting is equal to 0.0244. However, the

[2]Anyway, we tried to compare BUS against our approach but unfortunately, the source code is not available.

difference between SLOPE and the best portfolio is 0.0024. These values show how far each solution is from the best anytime behavior for the input portfolio. Therefore, the performance over time of the portfolio sorted with our greedy approach is ten times closer to the optimal performance than the performance of STS.

Note that the fact that we obtain better results than using those three sorting algorithms does not imply that the complete portfolio generation algorithm works better than FDSS or PBP. We are only comparing approximations of their sorting algorithms, which are in turn only one of the steps needed to configure a portfolio.

Ordering algorithm	Score
DFBnB	1.0000
SLOPE PORTFOLIO	0.9976
STS PORTFOLIO	0.9756
DC PORTFOLIO	0.9438
MF PORTFOLIO	0.9383
RANDOM	0.9298

Table 1: Score of the sorted portfolios in satisficing planning.

Figure 3 shows the area inscribed by the probability function of the resulting ordered portfolios. As it can be seen, the area defined by SLOPE is very close to the optimal sorting. Also, the probability values of STS are always lower than the values of our approach.

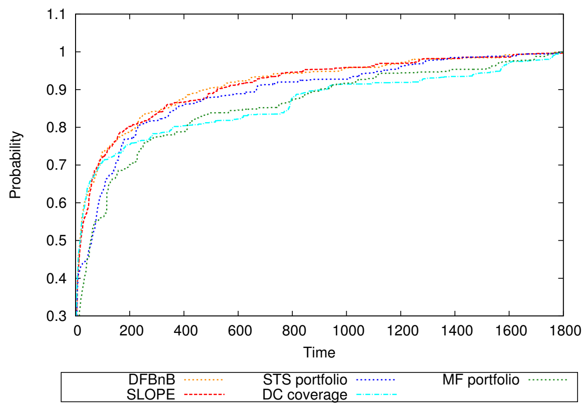

Figure 3: Area inscribed by the probability function of the ordered portfolios in satisficing planning.

Optimal Planning

The input portfolio has been computed using all the participant planners and the whole collection of planning tasks defined in the IPC 2011. Also, the input set R_{si} has been generated considering all the planning task defined for this track and the component planners of the input portfolio.

We have run DFBnB and Algorithm 1 with the input portfolio. The time required by each algorithm to compute the ordered portfolio was less than one second. Also, we have sorted the input portfolio with the three sorting criteria defined in this section: STS, MF, DC and a random algorithm. Finally, we have assessed the ordered portfolios using equation (2). The results of this comparison are shown in Table 2, which shows that the anytime behavior of the portfolio sorted with our greedy approach (denoted as SLOPE) is extremely close to the anytime behavior of the optimal order of the input portfolio.

Ordering algorithm	Score
DFBnB	1.0000
SLOPE PORTFOLIO	0.9993
STS PORTFOLIO	0.9980
DC PORTFOLIO	0.9717
MF PORTFOLIO	0.9505
RANDOM	0.6995

Table 2: Score of the ordered portfolios in optimal planning.

As mentioned above, we measured the performance of our ordering technique over the training instance. As a result, the performance of all sorting techniques (shown in Tables 1 and 2) is very close to 1. We consider that improving the overall behaviour of the resulting portfolio in this range is remarkable.

Conclusions and Future Work

Most state-of-the-art approaches in the automated design of portfolios did not focus on sorting their component solvers. Only some approaches have experimentally defined a specific order to execute their solvers. However, this order was defined using empirical criteria. We have addressed this open problem with the aim of improving the performance of the sequential portfolios over time. We have presented a formal definition of the problem of sorting the component solvers in static sequential portfolios. We deal with a new problem in the automated design of portfolios, where the anytime behavior of the resulting portfolios shall be optimized. Hence, the order of component solvers in the automated process of configuring portfolios becomes relevant.

In addition, we have introduced two algorithms to solve the problem defined in this work. The first algorithm aims to solve the problem optimally for a given data set using DFBnB and an admissible heuristic. Optimality is only guaranteed for the given data set. The second one is a greedy approach, which proposes a new heuristic based on the slope of the performance of each solver in the portfolio. Our results show that the performance of the portfolio over time can be significantly improved by sorting the component solvers, since the score of the portfolios randomly sorted is the lowest. Also, the greedy sorting algorithm outperforms others under the same conditions. Besides, the portfolios sorted with this greedy approach show a similar performance to the optimal sortings of the given portfolios. Moreover, the greedy approach takes less than one second to sort a sequential portfolio while the time required by DFBnB increases exponentially with the number of component solvers.

In the future, we will apply the sorting algorithms to dynamic portfolios approaches, where the configuration of each component planner is computed at runtime.

References

Gerevini, A.; Saetti, A.; and Vallati, M. 2014. Planning through Automatic Portfolio Configuration: The PbP Approach. *J. Artif. Intell. Res. (JAIR)* 50:639–696.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast Downward Stone Soup: A Baseline for Building Planner Portfolios. *In ICAPS 2011 Workshop on Planning and Learning* 28–35.

Hentenryck, P. V.; Coffrin, C.; and Bent, R. 2011. Vehicle Routing for the Last Mile of Power System Restoration. In *Proceedings of the 17th Power Systems Computation Conference (PSCC11), Stockholm, Sweden*.

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

Markowitz, H. 1952. Portfolio Selection. *The Journal of Finance* 7(1):77–91.

Núñez, S.; Borrajo, D.; and Linares López, C. 2012. 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.

Roberts, M., and Howe, A. E. 2006. Directing a Portfolio with Learning. *In Ruml, W., and Hutter, F., eds., AAAI 2006 Workshop on Learning for Search, 129-135*.