Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Automatic Configuration of Sequential Planning Portfolios

Jendrik Seipp, Silvan Sievers, Malte Helmert
University of Basel
Basel, Switzerland
{jendrik.seipp,silvan.sievers,malte.helmert}@unibas.ch

Frank Hutter
University of Freiburg
Freiburg, Germany
fh@cs.uni-freiburg.de

Abstract

Sequential planning portfolios exploit the complementary strengths of different planners. Similarly, automated algorithm configuration tools can customize parameterized planning algorithms for a given type of tasks. Although some work has been done towards combining portfolios and algorithm configuration, the problem of automatically generating a sequential planning portfolio from a parameterized planner for a given type of tasks is still largely unsolved. Here, we present Cedalion, a conceptually simple approach for this problem that greedily searches for the ⟨parameter configuration, runtime⟩ pair which, when appended to the current portfolio, maximizes portfolio improvement per additional runtime spent. We show theoretically that Cedalion yields portfolios provably within a constant factor of optimal for the training set distribution. We evaluate Cedalion empirically by applying it to construct sequential planning portfolios based on component planners from the highly parameterized Fast Downward (FD) framework. Results for a broad range of planning settings demonstrate that – without any knowledge of planning or FD – Cedalion constructs sequential FD portfolios that rival, and in some cases substantially outperform, manually-built FD portfolios.

Introduction

Over the years the automated planning community has created a very large number of different planning algorithms, and it is well known that none of them dominates all others on all planning tasks. Since automatically choosing the best planner for a given task remains a mostly unsolved problem, many of today's most successful planners run a *sequential portfolio* of individual planners (Coles et al. 2012). For example, the winners of both the learning track and the deterministic optimization track of the 2011 International Planning Competition (IPC), PbP2 (Gerevini, Saetti, and Vallati 2011) and Fast Downward Stone Soup (FDSS) (Helmert, Röger, and Karpas 2011), as well as the winner of the IPC 2014 deterministic satisficing track, IBaCoP (Cenamor, de la Rosa, and Fernández 2014), are based on sequential portfolios. For a more general overview of work on sequential portfolios in the automated planning community we refer to Vallati (2012).

While planning portfolios combine the complementary strengths of existing planners on a heterogeneous set of benchmarks, performance on a particular homogeneous type of instances can often be improved by tuning a planner's parameters. Specifically, applied to highly parameterized planning systems, automated *algorithm configuration* has recently been shown to find novel planner instantiations that are customized to yield the highest available performance on particular types of benchmark instances (Fawcett et al. 2011; Vallati et al. 2013).

Due to these successes, several recent lines of work found in the Satisfiability Testing (SAT) literature use algorithm configuration to construct portfolios of a single parameterized algorithm based on algorithm selection (Rice 1976) and on parallel execution of solvers (Huberman, Lukose, and Hogg 1997); see the works by Xu, Hoos, and Leyton-Brown (2010), Kadioglu et al. (2010), and Hoos et al. (2012b). However, there does not yet exist a general procedure for the problem of constructing sequential portfolios from a highly parameterized algorithm.

In this paper, we introduce Cedalion, a conceptually simple yet effective procedure for this problem. Cedalion iteratively uses an algorithm configurator to add the ⟨parameter configuration, runtime⟩ pair to the portfolio that maximizes portfolio improvement per runtime spent. It does so by making the runtimes of the configurations in the portfolio part of the configuration space.

After discussing background, we give a detailed description of Cedalion and show that it yields theoretical approximation guarantees. In our empirical evaluation we create sequential portfolios from the highly parameterized Fast Downward (FD) framework for various types of planning tasks.

Background

In this section we discuss the existing work we base our new algorithm on.

Automated Algorithm Configuration & SMAC

Most planning systems have several free parameters that can be adjusted to optimize performance (e.g., solution cost or runtime). In recent years, the AI community has developed dedicated systems for this *algorithm configuration* problem (Hutter et al. 2009; Ansótegui, Sellmann, and Tierney 2009;

Birattari et al. 2010; Hutter, Hoos, and Leyton-Brown 2011). Formally, the problem can be stated as follows: given a parameterized algorithm with possible configurations $\boldsymbol{\Theta}$, a benchmark set Π, and a performance metric $m(\boldsymbol{\theta}, \pi)$ capturing the performance of configuration $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ on instance $\pi \in \Pi$, find a configuration $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ that maximizes m over Π , i.e., that maximizes

$$f(\boldsymbol{\theta}) = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} m(\boldsymbol{\theta}, \pi).$$

The most prominent applications of algorithm configuration in the planning literature configured the Fast Downward planning system (45 parameters, after discretization giving rise to 2.99×10^{13} combinations, see Fawcett et al. 2011) and the LPG planning system (62 parameters, after discretization giving rise to 6.5×10^{17} combinations, see Vallati et al. 2013), achieving average speedup factors in satisficing planning of up to 23 and 118, respectively. These applications used the local-search based algorithm configuration method ParamILS (Hutter et al. 2009), which requires a discretization of continuous and integer parameters.

Here, we employ the improved sequential model-based algorithm configuration method SMAC (Hutter, Hoos, and Leyton-Brown 2011) that no longer requires discretization. SMAC uses predictive models of algorithm performance (Hutter et al. 2014) to guide its search for good configurations. More precisely, it uses previously observed $\langle$configuration, performance$\rangle$ pairs $\langle \boldsymbol{\theta}, f(\boldsymbol{\theta}) \rangle$ and supervised machine learning (in particular, random forests, see Breiman 2001) to learn a function $\hat{f} : \boldsymbol{\Theta} \to \mathbb{R}$ that predicts the performance of arbitrary parameter configurations (including those not yet evaluated). In a nutshell, after an initialization phase, SMAC iterates the following three steps: (1) use the performance measurements observed so far to fit a random forest model $\hat{f}$; (2) use $\hat{f}$ to select a promising configuration $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ to evaluate next, trading off exploration in new parts of the configuration space and exploitation in parts of the space known to perform well; and (3) run $\boldsymbol{\theta}$ on one or more benchmark instances and compare its performance to the best configuration observed so far.

SMAC is an anytime algorithm that interleaves the exploration of new configurations with additional runs of the current best configuration to yield both better and more confident results over time. As all anytime algorithms, SMAC improves performance over time. While SMAC provably converges for finite configuration spaces, it often only finds close-to-optimal configurations for realistic time budgets and challenging configuration problems.

Automated Portfolio Construction & Hydra

Our method for constructing sequential portfolios is closely related to Hydra (Xu, Hoos, and Leyton-Brown 2010), which automatically constructs *selection-based* portfolios. Given a parameterized algorithm framework with a space of algorithms $\boldsymbol{\Theta}$, Hydra starts with an empty set of candidate algorithms $\mathcal{C} = \emptyset$ and iteratively calls an algorithm configuration method to add algorithms $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ that *complement* $\mathcal{C}$ best. In each iteration, it constructs a portfolio from the current set $\mathcal{C}$ using the portfolio-based algorithm selector SATzilla (Xu et al. 2008) and measures its performance $m(\mathcal{C}, \pi)$ on each benchmark instance $\pi \in \Pi$ (in the first iteration, $m(\emptyset, \pi) = -\infty$). The performance metric Hydra maximizes via its algorithm configuration method is then

$$m_{\mathcal{C}}(\boldsymbol{\theta}, \pi) := \max(m(\mathcal{C}, \pi), m(\boldsymbol{\theta}, \pi));$$

in words, algorithm configuration searches for a configuration $\boldsymbol{\theta}$ that most increases the average performance of the current portfolio under an oracle that for each instance selects the better of the current portfolio and $\boldsymbol{\theta}$. (Of course, in practice, the SATzilla algorithm selector performs worse than an oracle; the assumption is that the error is small enough to not hurt much.) The configuration $\boldsymbol{\theta}$ returned by algorithm configuration with this metric $m_{\mathcal{C}}$ is then added to the current portfolio and the procedure iterates. Hoos et al. (2012b) introduced parHydra, a very similar method as Hydra to determine a subset of parameter configurations to run in a *parallel* portfolio.

Sequential Portfolios

So far, work on constructing sequential portfolios has focused almost exclusively on the case where the space of algorithms or parameter configurations $\boldsymbol{\Theta}$ is small and finite, and where we already know the runtimes of each $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ for solving each $\pi \in \Pi$. Streeter, Golovin, and Smith (2007) showed that, even when the $|\boldsymbol{\Theta}| \cdot |\Pi|$ runtimes are known, it is NP-hard to find an optimal portfolio. They introduced an algorithm exponential only in $|\boldsymbol{\Theta}|$, and an efficient greedy algorithm with the best possible approximation ratio (unless P=NP). Núñez, Borrajo, and Linares López (2012) derived a mixed integer programming (MIP) representation of the problem and solved it using the GNU Linear Programming Kit. In a similar spirit, Hoos et al. (2012a) derived an answer set programming (ASP) representation of the problem and solved it with the ASP solver Clasp. Of course, these approaches are not practical for the combinatorial configuration spaces typical in highly parameterized algorithm frameworks since they require access to the runtimes of all $|\boldsymbol{\Theta}|$ parameter configurations (for reference, in our work $|\boldsymbol{\Theta}| = 2.99 \times 10^{13}$). Instead, similar to the Hydra approach, we will use algorithm configuration to explore this space.

The most prominent sequential portfolio approach for domain-independent planning is Fast Downward Stone Soup (FDSS) (Helmert, Röger, and Karpas 2011), for which two satisficing and two optimal versions exist. Each FDSS portfolio runs multiple instantiations of the Fast Downward system for prespecified time slices. The construction of these FDSS portfolios relied on substantial domain expertise by planning experts; we will evaluate our fully automated approach against this baseline. FDSS showed excellent performance in IPC 2011, winning the deterministic optimization track and coming in second in the satisficing track.

Another recent sequential portfolio construction approach we compare against is due to Seipp et al. (2012). Their approach assumes that the benchmark set Π is comprised of known homogeneous subsets $\Pi = \Pi_1 \cup \cdots \cup \Pi_n$ and uses algorithm configuration to find specialized configurations $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$ for each of these subsets. Then, in a second

step, it uses one of several different methods for combining $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$ in a sequential portfolio. In contrast, Cedalion does not require knowledge about homogeneous subsets but operates directly on Π.

Definitions

Before we present and analyze our portfolio construction algorithm we give some definitions concerning planning tasks, sequential portfolios and quality metrics.

Informally, a *classical planning task* π consists of an initial state, a goal description and a set of operators. Solving π entails finding an operator sequence that leads from the initial state to a goal state. In *satisficing planning* any such sequence is valid but there is a preference for cheap solutions. On the other hand, in the setting of *optimal planning* only solutions with minimal cost are accepted. For *agile planning* the task is to find solutions as fast as possible, regardless of the solution cost. The last setting we consider in this work is the *learning setting*, which corresponds to satisficing planning with the difference that planners are allowed to learn on a training set in a first phase, before being evaluated on a test set in a second phase.

We define $c(\langle \boldsymbol{\theta}, t \rangle, \pi)$ as the *cost* of the solution a planning algorithm with configuration $\boldsymbol{\theta}$ finds for planning task π within time t, or as ∞ if it does not find a solution in that time. Furthermore, we let $c^\star(\pi)$ denote the *minimum known solution cost* for task π (approximated by a set of baseline planners). Following IPC evaluation criteria, we define the *solution quality* $q_{\text{sol}}(\langle \boldsymbol{\theta}, t \rangle, \pi) = \frac{c^\star(\pi)}{c(\langle \boldsymbol{\theta}, t \rangle, \pi)}$ as the minimum known solution cost divided by the solution cost achieved by $\boldsymbol{\theta}$ in time t. Note that for optimal planning the quality is either 0 or 1.

A *sequential planning portfolio* P is a sequence of pairs $\langle \boldsymbol{\theta}, t \rangle$ where $\boldsymbol{\theta}$ is a configuration of a planning algorithm and $t \in \mathbb{N}_{>0}$ is the time $\boldsymbol{\theta}$ is allowed to run for. We denote the portfolio resulting from appending a component $\langle \boldsymbol{\theta}, t \rangle$ to a portfolio P by $P \oplus \langle \boldsymbol{\theta}, t \rangle$.

We now define two quality scores $q(P, \pi)$ that evaluate the performance of a portfolio P on task π. Satisficing and optimal portfolios are assigned the maximum *solution quality* any of their components achieves, i.e.,

$$q_{\text{sol}}(P, \pi) = \max_{\langle \boldsymbol{\theta}, t \rangle \in P} q_{\text{sol}}(\langle \boldsymbol{\theta}, t \rangle, \pi).$$

Following IPC evaluation criteria, for agile portfolios the *agile quality* is defined as

$$q_{\text{agile}}(P, \pi) = \frac{1}{1 + \log_{10}(t(P, \pi)/t^\star(\pi))},$$

where $t(P, \pi)$ is the time that portfolio P needs to solve task π (note that $t(P, \pi) = \infty$ if P fails to solve π) and $t^\star(\pi)$ is the minimum time any planner needs (approximated by a set of baseline planners). We set $q_{\text{agile}}(P, \pi) = 1$ if $t(P, \pi) < t^\star(\pi)$ or $t(P, \pi) < 1$.

A portfolio's score on multiple tasks is defined as the sum of the individual scores, i.e. $q(P, \Pi) = \sum_{\pi \in \Pi} q(P, \pi)$, and the score of the empty portfolio is always 0.

Algorithm 1 : Cedalion. Construct a sequential portfolio maximizing the quality score q using a configuration space $\boldsymbol{\Theta}$ for instances Π and total portfolio runtime T.

```
1:  function CEDALION(Θ, Π, q, T)
2:      P ← ⟨⟩
3:      t_used ← 0
4:      while t_max = T − t_used > 0 do
5:          ⟨θ, t⟩ ← CONFIGURATOR(P, Θ × [1, t_max], Π, m_P, q)
6:          if q(P ⊕ ⟨θ, t⟩, Π) = q(P, Π) then
7:              return P
8:          P ← P ⊕ ⟨θ, t⟩
9:          if quality metric q is q_sol then
10:             Π ← {π ∈ Π | q_sol(P, π) < 1}
11:         else
12:             Π ← {π ∈ Π | t(P, π) = ∞}
13:         t_used ← t_used + t
14:     return P
```

Cedalion: A Greedy Construction Algorithm

We now introduce Cedalion, our new approach for constructing sequential portfolios from highly parameterized algorithms. Given an algorithm with parameter configuration space $\boldsymbol{\Theta}$, a set of tasks Π, a quality score q and the total portfolio runtime T, Cedalion iteratively constructs a sequential portfolio.

As shown in Algorithm 1, Cedalion starts with an empty portfolio P (line 2) and then iteratively uses an algorithm configurator (e.g., SMAC) to find configurations $\boldsymbol{\theta}$ and their runtime t that complement the current portfolio P best (line 5; see next paragraph for details). If appending the pair $\langle \boldsymbol{\theta}, t \rangle$ to P does not change the portfolio quality anymore, we converged and Cedalion terminates (line 6). Otherwise, the pair is appended to P (line 8) and all tasks that are solved optimally (line 10) or, in agile planning, solved at all (line 12) by P, are removed from Π in order to focus on other instances in the next iteration. This process iterates until the sum of the runtimes in the portfolio components exceeds the threshold T (line 4).

Cedalion's subsidiary algorithm configuration method CONFIGURATOR (line 5) returns a ⟨configuration, runtime⟩ pair that maximizes the portfolio improvement per additional runtime spent. More formally, given a quality score q it returns a pair $\langle \boldsymbol{\theta}, t \rangle \in \boldsymbol{\Theta} \times [1, t_{\text{max}}]$ that approximately maximizes the following portfolio improvement metric m_P for the current portfolio P across all instances π in the remaining set of instances Π:

$$m_P(\langle \boldsymbol{\theta}, t \rangle, \pi) = \frac{q(P \oplus \langle \boldsymbol{\theta}, t \rangle, \pi) - q(P, \pi)}{t}. \quad (1)$$

We let the configurator jointly optimize $\boldsymbol{\theta}$ and its runtime t by adding t to the configuration space. Its domain is bounded from above by the remaining portfolio time $t_{\text{max}} = T - t_{\text{used}}$. We further discretize t to integer values in $[1, t_{\text{max}}]$ to account for the fact that CPU limits can usually be set in units of seconds. As a result the parameter space for the configurator is $\boldsymbol{\Theta} \times [1, t_{\text{max}}]$ (line 5).

Note that by plugging in different quality measures q, the performance metric m_P applies to all of satisficing, optimal,

and agile planning. Per additional time unit spent, for optimal planning it maximizes the new number of tasks solved optimally; for satisficing planning the increase in solution quality; and for agile planning the increase in agile quality.

Comparison to Hydra

Cedalion resembles the Hydra approach for constructing selection-based portfolios in that it starts with an empty portfolio P and then iteratively calls a subsidiary algorithm configuration procedure to find complementary configurations $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ to add to P. However, the resulting portfolios are suited for different use-cases: while Hydra creates portfolios for *algorithm selection*, Cedalion constructs a *sequential* portfolio. This means that Cedalion solves a more complex task than Hydra since it not only needs to select the configurations to include in the portfolio but also their order and respective runtimes.

Another important difference is that in the Hydra framework the algorithm configuration procedure evaluates each candidate configuration $\boldsymbol{\theta}$ by running it with the full runtime budget T. In contrast, in our approach, the evaluation of a candidate combination $\langle \boldsymbol{\theta}, t \rangle$ only requires time t. Since t is typically only a small fraction of T, our subsidiary algorithm configuration procedure can make much more progress in the same amount of time than when being used by Hydra.[1]

Cedalion is also more broadly applicable than Hydra since it does not require any instance features. Finally, in contrast to Hydra, Cedalion comes with theoretical performance guarantees, which we will discuss next.

Theoretical Analysis

We now analyze our algorithm theoretically, using results by Streeter et al., who studied the theory of a special case of Algorithm 1 that takes the time required by every planner $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ on every planning task $\pi \in \Pi$ as an input and uses a $\Theta(|\boldsymbol{\Theta}| \cdot |\Pi|)$ computation in place of our use of a configurator (Streeter, Golovin, and Smith 2007; Streeter and Golovin 2007; Streeter and Smith 2008). Of course, this approach would not scale to our configuration space with $|\boldsymbol{\Theta}| = 2.99 \times 10^{13}$, but the theoretical results transfer to Cedalion with quality score q_{sol} (albeit not with q_{agile}).

We define the expected runtime $R(P, \Pi)$ of a portfolio P on benchmark set Π as its average time to success (i.e., to achieving the maximal possible solution quality for an instance). Defining the prefix portfolio P_T of P as consisting of P's first $\langle$planner, runtime$\rangle$ combinations $\langle \boldsymbol{\theta}_1, t_1 \rangle, \ldots, \langle \boldsymbol{\theta}_{|P_T|}, t_{|P_T|} \rangle$ such that $\sum_{j=1}^{|P_T|} t_j \leq T$, we define the expected quality that portfolio P achieves in time T on benchmark set Π as

$$Q(P, T, \Pi) = \mathbb{E}_{\pi \in \Pi}[\max_{\langle \boldsymbol{\theta}, t \rangle \in P_T} q_{sol}(\langle \boldsymbol{\theta}, t \rangle, \pi)].$$

[1]Computational savings by using shorter timeouts in algorithm configuration have also been exploited in the context of configuring on easy instances with the goal of scaling to hard instances (Mascia, Birattari, and Stützle 2013; Styles and Hoos 2013; Lindawati et al. 2013).

Finally, we let $R^* = \min_P R(P, \Pi)$ denote the lowest expected runtime achievable by any sequential portfolio based on the space of algorithms $\boldsymbol{\Theta}$ and let $Q^*(T) = \max_P Q(P, T, \Pi)$ denote the maximal quality any such portfolio can achieve in time T. We then use Theorems 6 and 7 of Streeter and Golovin (2007) to prove:

Theorem 1. *Using a configurator that, in iteration j of Cedalion, returns a solution $\langle \boldsymbol{\theta}, t \rangle$ within ϵ_j of the maximum of $m_P(\langle \boldsymbol{\theta}, t \rangle, \Pi)$ (see Equation 1), for any time $t \leq T$, the quality $Q(P, t, \Pi)$ of the portfolio P constructed by Cedalion is bounded by*

$$Q(P, t, \Pi) \geq (1 - (1/\exp(1))) \cdot Q^*(T) - \sum_{j=1}^{|P_T|} \epsilon_j \cdot t_j.$$

Simultaneously, the expected runtime of P is bounded by

$$R(P, \Pi) \leq 4R^* + \sum_{i=1}^{|P_T|} \left(\sum_{j=1}^{i} \epsilon_j \cdot t_j \right).$$

Thus, given a perfect configurator, Cedalion simultaneously achieves a $(1 - (1/\exp(1)))$ approximation for the quality achievable at any given time $t \leq T$, and a 4-approximation to the optimal runtime. Suboptimal configurators yield worse results but small errors ϵ_j do not escalate. This is important because with configuration spaces including 40+ categorical parameters it cannot be expected that blackbox configuration procedures can find the truly optimal configuration in realistic time. However, at least for some scenarios with few and discretized parameters, the SMAC configurator has empirically been shown to yield close to optimal performance in reasonable time (Hutter, Hoos, and Leyton-Brown 2011).

We also note that Theorem 1 only provides a performance guarantee for the set (or distribution) of benchmarks Π used for training the portfolio. We can use the same techniques as Hutter et al. (2009) to show that, given large enough training sets Π, we recover guarantees for an independent test set sampled from the same distribution as Π.

Experiments

We now study our automated portfolio construction algorithm empirically by building portfolios for the satisficing, optimal, agile and learning settings. For our experiments, we set the total portfolio limit T to 30 minutes for the satisficing, optimal and learning settings and to 1 minute for the agile setting and abort each component planner if it uses more than 2 GB of memory. Due to space reasons we list the found portfolios in a separate technical report (Seipp et al. 2014).

Configuration Space and Baselines In our experiments we consider the configuration space of Fast Downward (Helmert 2006) which allows us to compare Cedalion to the manually constructed Fast Downward Stone Soup (FDSS) (Helmert, Röger, and Karpas 2011) and to the most closely related works by Seipp et al. (2012) and Fawcett et

al. (2011). For comparability of results we use exactly the same reduced configuration space (see Fawcett et al. 2011) as these works. Since a fair evaluation requires that all compared approaches are able to choose from the same set of planners, we do not compare to any planners outside this space.

Benchmarks In order to determine training and test sets for the satisficing, optimal and agile settings, we first selected all instances of the corresponding IPC 2011 track (we used the tasks from the satisficing track in the agile setting) for which a set of baseline planners could find a solution and grouped them by domain. For each group we performed *stratified sampling* by dividing the instances evenly into two parts, one of which became part of our training set. The other half and additionally half of the instances which the baseline planners could not solve were added to the test set. We only trained on tasks solved by at least one baseline planner in order to avoid spending time on tasks that are probably too hard to solve in reasonable time anyway.

Algorithm Configuration We used 10 parallel SMAC runs in each iteration of Cedalion and chose the ⟨configuration, runtime⟩ pair that maximized the performance metric on the current instance set. Each of the SMAC runs was given a wall-clock time budget of 10h in all settings except optimal planning, where we used 30h to account for the fact that optimal planning is generally harder than satisficing planning.

SMAC-uniform In order to establish a fair comparison to the most closely related approach by Seipp et al. (2012) we relearned their uniform portfolio for the new training set with SMAC instead of ParamILS. We used a time budget of 30h for the configurator. Although the authors only created portfolios for satisficing planning we also learned a uniform portfolio of optimal planners. Following their methodology, we grouped the tasks from our training set by domain and configured Fast Downward with SMAC for the 14 sets of tasks individually. The resulting satisficing and optimal portfolios, dubbed "SMAC-uniform", then run the respective 14 configurations sequentially with equal time shares.

Quality Evaluation We note that, as done by Helmert, Röger, and Karpas (2011) and Seipp et al. (2012), we evaluate all portfolios as anytime-search algorithms, where each iteration except the first uses the cost of the solution from the previous iteration as a maximum bound on g-values during search, thus possibly improving the overall quality. We do not, however, exploit this extension during the portfolio construction process in order to keep planning runs independent. All reported qualities take into account the solution costs of a set of baseline planners in addition to the planners in the respective tables.

Satisficing Planning

Table 1 compares the training and test performance of our satisficing Cedalion portfolio to that of LAMA 2011 (Richter, Westphal, and Helmert 2011), satisficing FDSS and the SMAC-uniform portfolio. We included LAMA 2011 since it won the IPC 2011 satisficing track and because its iterations can be considered a portfolio, even though they are not assigned maximum runtimes.

	LAMA	FDSS		SMAC	Cedalion
Quality		**1**	**2**	**uniform**	**10h**
Sum training (130)	111.86	104.46	99.25	119.32	**123.90**
barman (10)	8.57	7.94	7.92	**9.95**	9.58
elevators (10)	5.69	6.48	6.59	7.00	**9.60**
floortile (10)	2.08	3.13	2.80	4.68	**5.98**
nomystery (10)	5.84	6.61	6.63	9.00	**9.98**
openstacks (10)	8.46	7.78	7.66	**8.69**	8.45
parcprinter (10)	9.73	9.90	8.77	9.83	**9.95**
parking (10)	**8.78**	7.63	7.87	7.91	7.51
pegsol (10)	**10.00**	9.49	7.38	9.95	**10.00**
scanalyzer (10)	8.50	9.67	8.79	**9.72**	9.42
sokoban (10)	8.18	8.54	7.20	**9.81**	8.97
tidybot (10)	**8.84**	7.08	7.27	8.35	7.68
transport (10)	8.75	5.85	6.90	**9.78**	7.57
visitall (10)	8.04	1.84	1.37	9.98	**10.00**
woodworking (10)	7.39	**9.99**	9.84	8.33	9.20
Sum test (140)	108.84	101.93	96.99	122.98	**123.89**

Table 1: Quality scores on the training and test set for LAMA 2011, satisficing FDSS, the uniform portfolio trained with SMAC and Cedalion trained with budget 10h.

Not only did Cedalion achieve the highest overall quality on the training set (123.90), but it also had the highest score on the test set (123.89). While SMAC-uniform achieved an only slightly lower total quality (119.32 and 122.98), the quality gap to the other planners is substantial on both the training and test set.

Optimal Planning

Next, we evaluate Cedalion for optimal planning. We compare the generated portfolio to the optimal variants of FDSS and SMAC-uniform. The baseline planners that we used for creating the training set in this setting were the components of the FDSS 1 portfolio. Table 2 shows that the FDSS 1 portfolio solved all 94 tasks that were solved by any of its configurations. Unfortunately, this means that we cannot improve upon FDSS's coverage on the training set. Nonetheless, our Cedalion portfolio managed to solve 93 tasks which is only one task less than FDSS 1's coverage and on par with FDSS 2. The SMAC-uniform portfolio solved the lowest number of tasks (90) on the training set. The results on the test set were similar. The FDSS portfolios solved 93 and 92 tasks whereas Cedalion and SMAC-uniform solved 90 and 89 tasks, respectively.

We believe that the automatic methods have a hard time surpassing the hand-picked FDSS portfolios in the optimal planning setting since there are relatively few qualitatively different configurations for optimal planning in Fast Downward.

	FDSS		SMAC	Cedalion
Coverage	**1**	**2**	**uniform**	**30h**
Sum training (94)	**94**	93	90	93
floortile (10)	3	3	3	**4**
parking (10)	**4**	**4**	2	2
tidybot (10)	**7**	**7**	**7**	6
woodworking (10)	**6**	5	4	5
Remaining domains (100)	**73**	**73**	**73**	**73**
Sum test (140)	**93**	92	89	90

Table 2: Number of solved tasks on the training and test set for optimal FDSS, the uniform portfolio trained with SMAC and Cedalion trained with budget 30h. We group the domains where all portfolios had the same coverage.

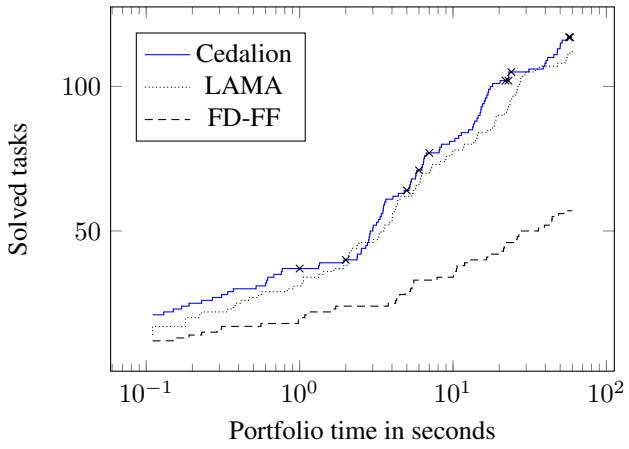

Figure 1: Coverage over time for Cedalion, LAMA 2011 and FD-FF on the test set. The crosses represent the times at which Cedalion switches to the next configuration.

Agile Planning

The next experiment studies our performance in the agile setting, comparing Cedalion with two Fast Downward configurations: greedy best-first search with deferred evaluation using the FF heuristic (Hoffmann and Nebel 2001) and preferred operators (FD-FF); and LAMA 2011 (Richter, Westphal, and Helmert 2011), winner of the IPC 2011 satisficing track. We chose to compare to these two planners since they are the fastest Fast Downward configurations we know.

On the training set Cedalion achieved an agile quality score of 117.62, outperforming both LAMA (105.03) and FD-FF (65.00). Similarly, on the test set Cedalion reached a score of 113.65 compared to 108.02 and 57.00 for LAMA and FD-FF, respectively. Cedalion was also the single best performer in 4 out of 14 domains and among the best in 6 additional domains.

Figure 1 shows that the number of tasks that Cedalion could solve within a given time is almost always higher than the corresponding numbers for LAMA 2011 and FD-FF.

	FD-Autotune		FD-Autotune-SMAC		Cedalion
Quality	**q**	**s**	**q**	**s**	**10h**
Sum training (540)	460.58	431.93	506.03	438.50	**533.68**
barman (60)	59.33	50.41	**59.70**	49.76	58.61
blocksworld (60)	28.10	27.56	34.18	27.56	**60.00**
depots (60)	39.26	56.65	57.17	57.34	**59.57**
gripper (60)	59.75	50.12	**59.86**	50.41	59.33
parking (60)	53.27	34.80	**59.09**	39.31	58.69
rover (60)	52.79	57.43	55.98	58.36	**59.63**
satellite (60)	57.71	48.65	57.44	49.62	**59.38**
spanner (60)	**60.00**	**60.00**	**60.00**	**60.00**	**60.00**
tpp (60)	52.79	43.44	58.03	43.77	**58.21**
Sum test (540)	463.00	429.04	501.46	436.14	**533.41**

Table 3: Quality scores for FD-Autotune.{q,s}, FD-Autotune-SMAC.{q,s} and Cedalion (with budget 10h) on the training set from Fawcett et al. (2011) and the test set using the same task distribution.

Learning Setting

To evaluate Cedalion in the learning setting we learned portfolios for the nine domains of the IPC 2011 learning track optimizing for solution quality. We used the tasks that Fawcett et al. (2011) generated to train FD-Autotune for that competition as training instances: 60 tasks per domain that take between seconds and minutes to solve. As a test set we generated 60 new instances using the same task distribution for each domain and made sure that the training and test set did not overlap. For each domain Fawcett et al. (2011) performed two configuration experiments with ParamILS (each of them using 10 ParamILS runs of 5 CPU days each), configuring for quality (FD-Autotune.q) and speed (FD-Autotune.s), respectively. In order to allow for a fair comparison, we re-ran their experiment with 5 SMAC runs of 5 CPU days each (FD-Autotune-SMAC.{q,s}).

Table 3 shows aggregated training performance of the five planners in the first row and detailed test performance in the bottom part. The total quality scores of both the training and test set show that in this scenario configuring with SMAC is more effective than with ParamILS and configuring for quality in fact achieves higher quality scores compared to when optimizing for speed.

Despite the fact that each domain is quite homogeneous and might thus intuitively not "need" a portfolio, Cedalion performed substantially better than even the best of the other planners (FD-Autotune-SMAC.q) on both the training and the test set with a total quality score of 533.68 vs. 506.03 and 533.41 vs. 501.46. Our portfolio was the single best performer in 5 out of 9 test domains and tied with the other planners in another.

Conclusion

We presented Cedalion, a novel algorithm that combines the strengths of sequential planning portfolios and algorithm configuration to automatically find portfolios of highly parameterized planners. At the core of our method lies the idea to make the time slices of sequential portfolios part

of the configuration space. Cedalion requires no domain-knowledge and automatically finds portfolios that improve upon state-of-the-art portfolios on training benchmark sets and similarly-distributed sets in several planning scenarios.

In future work we would like to use Cedalion in other settings such as SAT.

Acknowledgments

This work was supported by the German Research Foundation (DFG) as part of Emmy Noether grant HU 1900/2-1.

References

Ansótegui, C.; Sellmann, M.; and Tierney, K. 2009. A gender-based genetic algorithm for the automatic configuration of solvers. In *Proc. CP 2009*, 142–157.

Birattari, M.; Yuan, Z.; Balaprakash, P.; and Stützle, T. 2010. F-Race and Iterated F-Race: An overview. In Bartz-Beielstein, T.; Chiarandini, M.; Paquete, L.; and Preuss, M., eds., *Experimental Methods for the Analysis of Optimization Algorithms*. Springer. 311–336.

Breiman, L. 2001. Random forests. *Machine Learning* 45(1):5–32.

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2014. IBaCoP and IBaCoPB planner. In *IPC-8 planner abstracts*, 35–38.

Coles, A.; Coles, A.; García Olaya, A.; Jiménez, S.; Linares López, C.; Sanner, S.; and Yoon, S. 2012. A survey of the Seventh International Planning Competition. *AI Magazine* 33(1):83–88.

Fawcett, C.; Helmert, M.; Hoos, H.; Karpas, E.; Röger, G.; and Seipp, J. 2011. FD-Autotune: Domain-specific configuration using Fast Downward. In *ICAPS 2011 Workshop on Planning and Learning*, 13–17.

Gerevini, A.; Saetti, A.; and Vallati, M. 2011. PbP2: Automatic configuration of a portfolio-based multi-planner. In *IPC 2011 planner abstracts, Planning and Learning Part*.

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.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Hoos, H.; Kaminski, R.; Schaub, T.; and Schneider, M. T. 2012a. aspeed: ASP-based solver scheduling. In *Technical Communications of ICLP 2012*, 176–187.

Hoos, H.; Leyton-Brown, K.; Schaub, T.; and Schneider, M. 2012b. Algorithm configuration for portfolio-based parallel SAT-solving. In *Proc. CoCoMile 2012*, 7–12.

Huberman, B. A.; Lukose, R. M.; and Hogg, T. 1997. An economics approach to hard computational problems. *Science* 265:51–54.

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: an automatic algorithm configuration framework. *JAIR* 36:267–306.

Hutter, F.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2014. Algorithm runtime prediction: Methods & evaluation. *AIJ* 206:79–111.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *Proc. LION 2011*, 507–523.

Kadioglu, S.; Malitsky, Y.; Sellmann, M.; and Tierney, K. 2010. ISAC – instance-specific algorithm configuration. In *Proc. ECAI 2010*, 751–756.

Lindawati; Yuan, Z.; Lau, H. C.; and Zhu, F. 2013. Automated parameter tuning framework for heterogeneous and large instances: Case study in quadratic assignment problem. In *Proc. LION 2013*, 423–437.

Mascia, F.; Birattari, M.; and Stützle, T. 2013. Tuning algorithms for tackling large instances: An experimental protocol. In *Proc. LION 2013*, 410–422.

Núñez, S.; Borrajo, D.; and Linares López, C. 2012. Performance analysis of planning portfolios. In *Proc. SoCS 2012*, 65–71.

Rice, J. R. 1976. The algorithm selection problem. *Advances in Computers* 15:65–118.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011 (planner abstract). In *IPC 2011 planner abstracts*, 50–54.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning portfolios of automatically tuned planners. In *Proc. ICAPS 2012*, 368–372.

Seipp, J.; Sievers, S.; Helmert, M.; and Hutter, F. 2014. Automatic configuration of sequential planning portfolios: Generated portfolios. Technical Report CS-2014-004, University of Basel, Department of Mathematics and Computer Science.

Streeter, M., and Golovin, D. 2007. An online algorithm for maximizing submodular functions. Technical Report CMU-CS-07-171, School of Computer Science, Carnegie Mellon University.

Streeter, M. J., and Smith, S. F. 2008. New techniques for algorithm portfolio design. In *Proc. UAI 2008*, 519–527.

Streeter, M. J.; Golovin, D.; and Smith, S. F. 2007. Combining multiple heuristics online. In *Proc. AAAI 2007*, 1197–1203.

Styles, J., and Hoos, H. 2013. Ordered racing protocols for automatically configuring algorithms for scaling performance. In *Proc. GECCO 2013*, 551–558.

Vallati, M.; Fawcett, C.; Gerevini, A.; Hoos, H. H.; and Saetti, A. 2013. Automatic generation of efficient domain-optimized planners from generic parametrized planners. In *Proc. SoCS 2013*, 184–192.

Vallati, M. 2012. A guide to portfolio-based planning. In *Proc. MIWAI 2012*, 57–68.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: portfolio-based algorithm selection for SAT. *JAIR* 32:565–606.

Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proc. AAAI 2010*, 210–216.