Portfolio-based Planning: State of the Art, Common Practice and Open Challenges

April 28, 2015

Mauro Vallati
Lukáš Chrpa
Diane Kitchin

Abstract

In recent years the field of automated planning has significantly advanced and several powerful domain-independent planners have been developed. However, none of these systems clearly outperforms all the others in every known benchmark domain. This observation motivated the idea of configuring and exploiting a portfolio of planners to perform better than any individual planner: some recent planning systems based on this idea achieved significantly good results in experimental analysis and International Planning Competitions. Such results let us suppose that future challenges of the Automated Planning community will converge on designing different approaches for combining existing planning algorithms.

This paper reviews existing techniques and provides an exhaustive guide to portfolio-based planning. In addition, the paper outlines open issues of existing approaches and highlights possible future evolution of these techniques.

Introduction

Automated Planning is one of the most prominent AI challenges; it has been studied extensively for several decades and led to many real-world applications (see, e.g., [20]). During the last decade, Automated Planning has achieved significant advancements. However, while several powerful domain-independent planners have been developed, none of them clearly outperforms all others in every known benchmark domain. It has also been observed that if a planner does not find a solution quickly, it is very likely it will not find it at all, even with very large CPU time horizons [25]. These observations motivate the idea of configuring and exploiting a portfolio of planners to achieve better overall performance than any individual planner. Moreover, portfolio-based approaches have been successfully applied to a number of combinatorial search domains, such as SAT [62], maxSAT [36], Answer Set Programming (ASP) [14], and QBF [43].

Recently, a number of planners based on the portfolio approach have been developed, and achieved impressive results in the last editions of the International Planning

Competition (IPC6-8) [11, 9, 34]: they won, or got very close to winning, in almost every track in which they took part. Achieved results, and the aforementioned observations, let us presume that the future of AI planning will not only be focused on developing new planning algorithms, as in the last decades, but also on designing promising techniques for combining and exploiting them.

This paper reviews existing techniques for configuring a portfolio of planning algorithms, in order to:

- give an overview of the state-of-the-art of portfolio-based planners;
- describe the decisions that have to be taken during the configuration process;
- stimulate the development of new high-performance planning frameworks based on this promising approach.

The overview of the state of the art is by no means to be considered complete, since the number of portfolio-based planning techniques is growing rapidly, but it has been designed to provide a comprehensive summary of approaches that have been exploited in planning.

The remainder of the paper is organised as follows. Firstly, we briefly introduce Automated Planning and algorithm portfolios; secondly, we present existing portfolio-based planners. We then describe the steps of portfolio configuration. Finally, we give conclusions and final remarks.

Background

This section introduces the definition of Automated Planning tasks and describes the idea behind portfolio-based approaches.

Automated Planning

Automated planning and specifically classical planning, deals with finding a (partially or totally ordered) sequence of actions transforming the static, deterministic and fully observable environment from some initial state to a desired goal state [20].

In the classical representation *atoms* are predicates. *States* are defined as sets of ground predicates. A *planning operator* $o = (name(o), pre(o), eff^{-}(o), eff^{+}(o))$ is specified such that $name(o) = op\_name(x_1, \ldots, x_k)$ (op_name is a unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator), $pre(o)$ is a set of predicates representing the operator's preconditions, $eff^{-}(o)$ and $eff^{+}(o)$ are sets of predicates representing the operator's negative and positive effects. *Actions* are ground instances of planning operators. An action $a = (pre(a), eff^{-}(a), eff^{+}(a))$ is *applicable* in a state s if and only if $pre(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus eff^{-}(a)) \cup eff^{+}(a)$.

A *planning domain* is specified via sets of predicates and planning operators. A *planning problem* is specified via a planning domain, initial state and set of goal atoms. A *solution plan* is a sequence of actions such that a consecutive application of the actions in the plan (starting in the initial state) results in a state that satisfies the goal.

The classical planning model can be extended, in order to handle a wider range of constraints and increase expressiveness. For instance, this is the case in Temporal planning, where actions have a duration that should be considered, or Uncertainty planning, that studies cases in which the environment is not fully observable and effects are non-deterministic. On this matter, the interested reader is referred to [20] and [15].

Algorithm portfolios

The first work in this area is from the 70s, by Rice [45]. In that paper, the author investigated and described the problem of selecting the best algorithm for a given instance of a problem, by considering the value of some characteristics of the instance. This problem is known as the *algorithm selection* problem.

The term *algorithm portfolio* was first introduced by Huberman et al. [27] to describe the strategy of running several algorithms in parallel. The idea was taken from economics, where portfolios are used to maximise a utility that has an associated risk [55].

The algorithm portfolio approach was also studied by Gomes and Selman [21]; who conducted a theoretical and experimental study on the parallel run of stochastic algorithms for solving computationally hard search problems. Several authors have since used the term for describing any strategy that combines multiple algorithms, considered as black-boxes, to solve a single problem instance. Examples of portfolio approaches in Artificial Intelligence can be found, for instance, in SAT, ASP, CSP and QBF [62, 14, 44].

Currently, a formal definition of *Algorithm portfolio* is missing. This is also due to the fact that it is not unusual, in AI, to have solvers that exploit more than one search technique. This is especially true in planning, where several state-of-the-art domain-independent planners have the so-called backup strategies, that are used when the main one fails [24, 16, 7]. Roberts and Siebers, the organisers of the IPC 2014 learning track, handle this issue from a different perspective: they attempted to define a “basic solver“, as follows.[1]

> A basic solver is any single (meta) algorithm that does not leverage more than one general purpose solver at its core. It can be a meta-algorithmic approach but it can only use one solver at its core. Parametrised variants of the same algorithm are still one solver. The core general purpose solver cannot itself be an ensemble of other solvers.

We do not completely agree with this definition. In particular, we disagree on considering parametrised variants of the same algorithm as one solver. If the parameter configuration can significantly change the behaviour of the algorithm, it is more appropriate to consider different configurations as different solvers. We believe that, in order to distinguish portfolios, it is fundamental to know the internal details of the considered system. Therefore, Definition 1 provides our practical definition of a portfolio-based planner.

[1] http://www.cs.colostate.edu/ ipc2014/

Definition 1. *We say that a planner is portfolio-based if it automatically selects and/or combines one or more techniques for solving one or more problem instances.*

With regards to Definition 1, it is worth noticing that:

- The term *techniques* is used in order to highlight that a portfolio-based planner can exploit a set of planning engines, i.e. algorithms that receive as input the problem description and provide plan(s) as output, or a set of heuristics that, given the description of the problem and the current state of the world, return the evaluated cost necessary to reach the goals.

- The selection and/or combination is considered as *automatic* if there is an automated process that provides the portfolio as output. Such a process can either be based on some training instances – as is common practice – or can only consider information collected while solving the given instance. It should be noted that a portfolio-based planner can possibly have no selection (all available techniques are used) or no combination (pure parallel execution), but at least one of them should be considered and automated. We emphasise the automatic process because, in our opinion, a system that runs a set of techniques without any automated selection or combination is not a portfolio-based planner. It can be seen as a generic bunch of techniques run together, which relaxes many of the constraints that otherwise should have been faced.

Approaches that exploit a backup strategy to use when the main one fails, are hereinafter not classified as portfolios. Systems like SATPlan [30, 31], which can use a variety of satisfiability engines are not considered portfolios since they do not automatically select the engine to use. In particular, we believe that SATPlan is a planning framework; it includes different modules –in this case, planning to SAT encoders and SAT solvers – and allows the user to select the preferred one.

The space of algorithm portfolios, as we defined them, is large and includes approaches that use all the available algorithms as well as those that select a single algorithm. On the other hand, they all work for selecting (and possibly combining) algorithms in order to obtain improved performance.

Existing portfolio-based planners

In the field of automated planning, the idea of configuring and using a portfolio of techniques has been investigated by several researchers and has become a very interesting topic in the last few years.

In this section we present and discuss existing planning systems that explicitly configure, and either exploit or study, a portfolio of one or more planning techniques.

BUS

BUS [26] is the first work, in automated planning, based on the portfolio approach. It considers 6 different domain-independent planning engines. To solve a problem, BUS firstly determines the order in which the planners should be tried. It calculates an expected run time for each planner to solve the problem and an expected probability of success, and orders the incorporated planners by $P(A_i)/T(A_i)$ where $P(A_i)$ is the expected probability of success and $T(A_i)$ is the expected runtime of planner A on the problem i. The probability of success and the expected runtime of the incorporated planners are predicted by linear regression models that evaluate a set of features of the problem i. A different model is generated for each planner by analysing its performance on some training problems.

Given the planners' ordering, the planners are run in a round-robin like scheme, where each planner is run for the expected runtime needed for solving the current problem. If the planner solves the problem, the planning process halts. If the planner fails, then it is no longer considered by the round-robin. If the CPU time allocated to a planner finishes without solution or failure, the planner is suspended and the next one is run.

BUS achieved better results, in terms of the number of solved problems, than any of the incorporated planners. Due to the overhead required for extracting features and ordering the planners, it was slower then some of the included planners.

PbP

Inspired by a previous work of Roberts and Howe [50] (that will be described in the section *Studies on Planner Performance for Portfolios*), Gerevini and collaborators developed the Portfolio-based Planner PbP [17] (and lately, an enhanced version called PbP2 [18, 19], that includes a larger set of planners and a better engineered code), which automatically configures a domain-specific portfolio of domain-independent planners. PbP considered planners that showed good performance in past editions of the International Planning Competition, plus a domain-specific configuration of the well known system LPG [16], obtained by including the ParLPG system [59].

The configuration relies on some knowledge about the performance of the planners in the portfolio and the observed usefulness of automatically generated sets of macro-actions. This configuration knowledge is "learned" by a statistical analysis, that compares all the possible configured portfolios, and consists of: an ordered selected subset of the planners (at most three) in the initial portfolio, which at planning time are combined through a round-robin strategy; a set of useful macro-actions for each selected planner; and some sets of planning time slots [50]. A planning time slot is an amount of CPU time to be allocated to a selected planner (possibly with a set of macro-actions) during planning. For each integrated planner, PbP defines a sequence of increasing planning time slots, $\langle t_1, ..., t_n \rangle$. Each t_i is the CPU time that will be allotted to the planner during the testing phase. A t_i is defined as the CPU time required to solve a training problem during the performance measurement phase in a percentage p_i of cases. The sequence of increasing percentages $\langle p_1, ..., p_n \rangle$ from which the planning time slots are derived is defined by the vector $\langle 25, 50, 75, 80, 85, 90, 95, 97, 99 \rangle$. The execution order of selected planners is defined by the increasing CPU time slots associated with them, shortest first.

Both versions of PbP are able to configure two different portfolios: one focusing on speed and the other focusing on plan quality, in terms of the number of actions. Nevertheless, PbP can be used without this additional knowledge. Thus, the knowledge-free system schedules all planners by a round-robin strategy and it assigns the same time slot to the randomly ordered planners.

Fast Downward Stone Soup

Fast Downward Stone Soup (here abbreviated FDSS) [23] is a sequential portfolio planner that uses various heuristics and search algorithms that have been implemented in the Fast Downward [22] planning system. It is optimised for improving the quality of the solutions found. Several search algorithms are considered and used for solving some training problems. Given their results on such instances, they are combined through a hill-climbing search.

Portfolio generation starts from an initial portfolio which assigns a runtime of 0 to each search algorithm. FDSS then performs hill-climbing: in each step, a set of possible successors to the current portfolio are generated, which are like the current portfolio except that each successor increases the time limit of one particular algorithm by granularity. The best successor w.r.t. the IPC quality score among these candidates is selected and the configuration continues, for a total of timeout/granularity iterations. FDSS is able to configure two different portfolios: satisficing and optimal. The main difference is that the latter halts as soon as one of the solvers finds a solution, while the former continues until the maximum runtime of the portfolio is reached.

ASAP

An automatic Algorithm Selection Approach for Planning (ASAP) [58] is the only approach based on pure automatic algorithm selection. For a given domain, ASAP learns additional knowledge, in the form of macro-operators and entanglements [8], which is used for creating different encodings of the given planning domain and problems (i.e. planning domain/problem reformulation). Secondly, it explores the 2 dimensional space encodings (e)–planners (p), and finally, selects the best algorithm $\langle e, p \rangle$ for optimising the runtimes or the quality of the solution plans.

In ASAP, each *algorithm* has two dimensions: one dimension is represented by different encodings of a given domain, the other is represented by existing domain-independent planners. Planning engines have been selected accordingly to their good performance in International Planning Competitions, and the different planning techniques that they exploit. The algorithm which performed best, in terms of IPC score [9], on the training instances is selected. IPC score was selected since it is the commonly used metric for comparing the performance of planning systems. It considers, at the same time, coverage and runtime. On the other hand, it should be noted that the IPC score is "relative", in the sense that scores can vary while considering different sets of planners [9].

IBaCoP

The Instance Based Configured Portfolios (IBaCoP and IBaCoP2) [5] won the sequential satisficing track of the 2014 edition of IPC. They both consider the same set of planners, i.e. all the ones competing in the sequential satisficing track of IPC 2011 plus LPG-td, and configure the portfolio for maximising the quality of solution plans found. The systems differs in the way in which portfolios are configured. IBaCoP configures a domain-independent portfolio by using the Pareto efficiency technique [6] to select a sub-set of planners; selected planners are those that dominate all the others in at least one considered training domain in terms of either quality or CPU time. All the selected planners get the same running time in the testing phase. IBaCoP2 can be seen as an evolution of IBaCoP, since it considers only the planners selected by IBaCoP. Furthermore, IBaCoP2 performs an instance-based planner selection by using predictive models. Such models are used for predicting the performance of planners on testing problems, by taking into account planners' performance on some training instances, and correlating performance and instance structure through features extracted by problems' description. The planners are ordered following predictive model confidence. The running time is then uniformly divided among them.

Both IBaCoP and IBaCoP2 participated in various tracks of IPC 2014 achieving remarkable results in sequential satisficing and multi-core tracks.

Cedalion

Fast Downward Cedalion [54, 53] is a system that automatically configures a sequential portfolio of configurations of a parametrised planner, in this case Fast Downward. Given a set of training instances and a highly-parametrised planning engine, it iteratively selects the configuration – and the corresponding time to be allocated – that improves the current portfolio the most per time spent. At the end of each iteration, the training instances for which the current portfolio finds the best solution, in terms of plan quality, are removed from the training set. Configurations are generated by using the model-based algorithm configurator SMAC [28] on remaining training instances. Different SMAC are run in parallel in order to obtain different configuration of the planner. Cedalion took part in both the learning and deterministic tracks of IPC-14. It was awarded as the best learner of the learning track, and the second overall system in terms of quality of plans found. It was also the planner that solved the largest number of problems in the Agile subtrack of the deterministic IPC 2014.

Studies on Planner Performance for Portfolios

It is worth discussing three works by Roberts and Howe [49, 51, 50]. The first work, which can be seen as an extension to BUS, is focused on using features for learning planner performance models, to be exploited for combining systems in round-robin portfolios. In the other two works, the first preliminary while the second extremely detailed and extensive, the authors focused on modelling and predicting the performance of planners, by considering systems and benchmarks up to the 2006 planning competition. In their work, Roberts and Howe also investigate how it is possible to combine planners, by predicting their performance, in order to maximise the coverage or minimise the required runtime for solving a problem. In the rest of this paper we will refer to this study as ModelPerformance.

In 2011, a portfolio approach system [41] was designed for evaluating the state-of-the-art domain-independent planners. They presented a general method based on linear programming to define the baseline sequential portfolio for a specific set of problems, against which the real performance of planners can be measured and evaluated. Given an objective function that balances the quality IPC score achieved by the portfolio, the time spent and/or memory used by the portfolio, Mixed-Integer Programming (MIP) is then used for combining the planners that will be used by the portfolio. In a subsequent work [42], the authors proved that it is possible to exploit MIP for deriving the optimal portfolio and they actually provided it for the optimal track of IPC 2011. Moreover, Núñez et al. tackled a very critical topic of portfolio configuration: how to understand the utility of training problems. They observed that a small set of highly informative problems allows the configuration of the best possible portfolio. In the rest of this paper we will refer to these studies as MIPstudy.

In 2012, Seipp et al. [52] studied the performance of several different automatically-obtained configurations of a single high-performance planner, combined in a portfolio. In particular, the authors exploited the FDSS framework for generating different sequential portfolios of Fast Downward configurations. Interestingly, they observe that the best performance in terms of quality, is achieved by using a "uniform" portfolio, that allocates the same amount of CPU time to all the considered planning systems.

More recently, in 2013, Cenamor et al. [4] built classification models for predicting whether a given planner will succeed or not on a specific problem, and regression models for predicting the runtime. For generating the predictive models they extracted a large set of features, derived from the SAS+ formulation [3]. They compared the performance of a number of strategies for configuring a portfolio, and evaluated them on planners and benchmarks of IPC 2011. In 2014, Fawcett et al. [10] provided an extensive analysis which includes a larger set of instance features, some more accurate predictive models, and an investigation of the relative importance of features. Hereinafter we will refer to these studies as IPCstudy

Portfolio-like Systems

In this Section we describe planning techniques that, according to Definition 1, can not be classified as portfolios but that, intuitively, are not just basic solvers. This is the case of ArvandHerd [56, 57]; it is a recent pure parallel planner. Following Definition 1, it is not a portfolio since it does not automatically select or combine planning techniques. It simultaneously runs, on different cores, different configurations of two significantly different planning approaches: four configurations of the random walk based domain-independent planner Arvand [37] and one configuration of the WA*-based domain-independent planner LAMA [47]. The Arvand configurations differ on the strategy used for generating random walks (biased, unbiased, considering helpful actions, etc.), on the initial walk length, the frequency with which walks are lengthened, and the factor by which they are lengthened. The communication between the processes is limited to those running Arvand. Specifically they share a walk pool and a single UCB [2] configuration selection system.

We noticed that in the multicore track of the 2011 and 2014 editions of IPC [9, 34], participants are either running simultaneously different search algorithms –without an

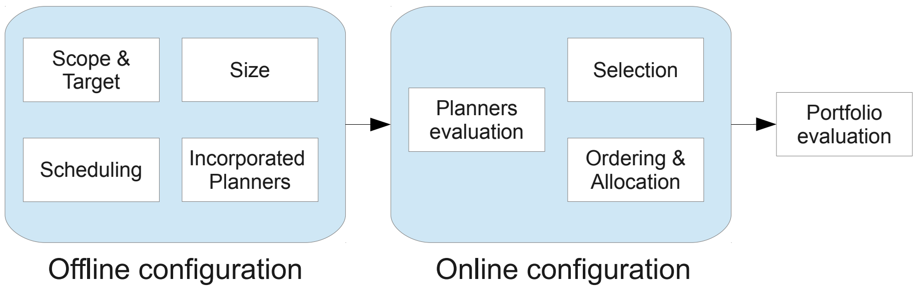

Figure 1: An overview of steps required for configuring a portfolio of planners. Terms *Online* and *Offline* are considered w.r.t. learning instances.

automated selection or combination process– or using some straightforwardly parallelised versions of sequential portfolios. An example of a system exploiting the former approach is Mp [48]. On the other hand, a few systems exploited communication and information sharing between techniques. This is the case of DAEyahsp [32] and ArvandHerd, that have been introduced since information sharing will be important for the future of portfolio-based planning (this will be discussed in subsequent sections).

Portfolio configuration

In this section we analyse every step of the portfolio configuration process for planning. Reference to existing systems (described in the previous section) are provided and discussed when relevant. For the sake of readability, hereinafter we will consider the configuration of a portfolio of planners. The discussion can be easily generalised to different planning-related techniques like heuristics.

Fig. 1 gives a high level description of the steps required for configuring a portfolio of planners. We can divide the decisions into two main sets: decisions to take *offline* and decisions to take *online*, w.r.t. the performance achieved by the incorporated planners on the learning problems used for the portfolio configuration. The former set concerns:

- *Scope*; the resulting configured portfolio can be domain-independent or domain-specific.
- *Target*; the function that the portfolio is configured for optimizing (e.g., runtime, quality of solutions).

- *Size*; minimum and maximum number of planners that can be selected during the configuration.
- *Scheduling strategy*; the strategy that will be used for running the selected planners (e.g., pure parallel, sequential, mixed, ...).
- *Incorporated planners*; the planners that are considered and that can be selected to be part of the portfolio.

The set of decisions to take online is:

- *Evaluation of the planners*; the performance metrics used for evaluating the planners on training instances.
- *Planner selection*; the techniques used for selecting the planners – out of the ones incorporated in the corresponding off-line step – to include in the portfolio (e.g., number of solved problems, statistical tests, ...).
- *Allocation strategies and planner ordering*; the strategy for deciding the CPU time allocated to the selected planners and the planners' execution order.

Finally, it is good practice to define the strategy for the evaluation of the performance of the configured portfolio on a subset of testing problems. It should be clear that most of the phases are strictly related, and they do not have a clear predefined ordering. In the remainder of this section we will describe each step of the two sets, in order to give the clearest representation of the whole configuration process.

Offline decisions

Target and scope

A portfolio of planners is configured for optimizing a predefined objective function. Typically these functions are very easy and concern three different performance areas, usually taken individually: runtimes, quality of solution plans (in terms of number of actions or actions cost) and number of solved problems. A classical target, that is often required in IPCs is to maximize the solution quality.

From the scope point of view, we can identify two different categories of portfolios:

1. domain-independent;
2. domain-specific;

A domain-specific portfolio is configured for solving problems from a given domain only, it should have great performance on the specific domain and – since it is focused on a single domain – it can exploit additional domain-related knowledge (e.g., macro-actions [39]). Domain-specific approaches usually have very good performance on the selected domain, and it might appear not worthwhile to look for further improvements. On the other hand, in domains in which problems' structure can significantly vary, performance can quickly decrease as testing problems differ from training ones.

A domain-independent portfolio can be used on every possible benchmark domain, therefore it is aimed at obtaining good performance on average. Intuitively, no domain-specific knowledge can be exploited. In this category there are either systems that exploit a static portfolio, i.e., it is always the same for every problem, or approaches that configure a different portfolio per instance. In order to select and combine planners for a given problem, they usually need to extract additional information, not given in the original description, about the problem to solve. Such information is extracted in the form of features related to the specific instance (e.g. number of objects), to the domain (e.g. number of operators), to the SAS+ representation of the problem (e.g., features of the causal graph) or to the performance of some planners (e.g., length of a relaxed plan). This is the case, for instance, for the techniques designed and compared in ModelPerformance and IPCstudy.

Portfolio size

In this section we use the terminology of Xu et al. in [62] who define:

> An (a, b)-of-n portfolio as a set of n incorporated planners and a technique for selecting among them at least a and no more than b algorithms to be executed. We also use the terms a-of-n portfolio to refer to an (a, a)-of-n portfolio, and n-portfolio for an n-of-n portfolio.

In order to exemplify, let us recall and classify some of the existing portfolio-based planners. ASAP has a 1-of-n structure, and PbP a (1, 3)-of-n. BUS exploits n-of-n structures. FDSS is a bit more complex to categorise since the number of selected planners is defined by a heuristic algorithm; the most correct way for describing its structure is (1, n)-of-n. FDSS2 exploits and compares different structures; as in FDSS it relies on heuristic algorithms for some of them, but it is also able to use all the included planners together; it goes from a (1, n)-of-n to n-of-n, depending on the selected approach for combining planners.

Scheduling strategy

Portfolios can be *parallel* (all algorithms are executed concurrently), *sequential* (the execution of one algorithm only begins when the execution of the previous algorithm has ended), or *mixed* (some combination of parallel and sequential). Formally, given a cutoff CPU time for solving a problem T, a set of planners $P = \{p_1, p_2, ..., p_n\}$, and the time allocated to each planner as $t(p_i)$:

- Sequential: planners are executed following a given order and $\sum_{i=1}^{n} t(p_i) = T$.
- Parallel: planners are executed concurrently on different CPUs, and $t(p_i) = T$.

In parallel portfolios, there are enough CPUs for running all the selected planners in pure parallel. The portfolio can finish as soon as some of the planners finds a solution if the criterion is to minimise runtime. Otherwise, all the planners have spent their maximum available time. This can be due to the fact that a planner does not find a solution, or that it found a solution, and it is incrementally improving its quality. While a parallel portfolio seems in principle easy to implement, it becomes complex to deal with planners that share information, such as the best solution found.

On the contrary, sequential portfolios run all the selected planners on a single CPU. This strategy executes the planners to their maximum allotted time and quits at the first success or after all planners have either spent their time or found a solution. While it is easy to implement, this strategy requires refined techniques for estimating the amount of CPU time to allot to each planner. Moreover, if the portfolio's target is minimising runtime, it is crucial to find the best order among the selected planners. In this case, it is important to include in the portfolio the fastest planner for solving the given problem, but it is even more important to schedule it as soon as possible. Intuitively it is better to have an approach which firstly runs quick planners, rather than an approach that includes the best possible planner, but schedules it very late. For this reason – and also for avoiding features extraction on problems that can be quickly solved – the idea of pre-solving has been introduced in SATZilla [62]: the pre-solver is a system that is able to quickly solve a large number of problems.

Finally, a mixed strategy tries to mix the two previous techniques. This is usually done by "simulating" parallelism on a single CPU; for instance this could be done by using round-robin scheduling as in PbP.

Obviously, there is not a clear limit to the combinations that it is possible to obtain. It is theoretically possible, for instance, to configure a set of several sequential portfolios and execute them in parallel on different CPUs.

Incorporated planners

One of the most important decisions to take while building portfolio-based planning systems, is choosing the planning algorithms to consider for the configuration of the portfolio.

The AI planning community constantly designs faster and more efficient heuristics and algorithms for solving Automated Planning problems. Currently, there is a large collection of domain-independent planners that can be considered while configuring a portfolio framework. The first temptation is, evidently, to consider *all* the available planning systems. This requires a dramatically high amount of CPU time for evaluating the planners on learning problems (step described in the following section), as well as a large amount of human-time for configuring and compiling all the required sources. Therefore selecting all the existing planners is suitable only for the configuration of domain-independent portfolios, for which the evaluation step is done once.

Another possibility is to consider different configuration of the same planning framework. This can only be done on planners, like Fast Downward, that are highly parametrised and include numerous algorithms and techniques for planning. In such planners it is presumable that, if correctly configured, their different configurations work well on several different search space structures.

A third option is to consider a number of systems that are believed to be competitive and, hopefully, efficient. This is commonly done by selecting winners –or top performers– of various editions of the IPC, by selecting planners that exploit very different planning strategies, or even by considering all the planners that took part in some editions of the IPC.

Summarising, it is important to include a large selection of uncorrelated planning techniques: including a very small set of algorithms will probably lead to poor performance on some domains. On the other hand, including a lot of planners will take a remarkable amount of CPU time for evaluating them on learning problems.

Online decisions

In the following we detail decisions that are taken online, with regards to the performance of the considered planners on the training instances. In other words, the following decisions must take into account the performance on training instances.

Evaluation of the planners incorporated

This is, generally, the computationally most expensive step in the configuration of a portfolio.

Firstly, the learning instances, on which the incorporated planners will be evaluated, must be selected. For configuring domain-independent portfolios, it is common practice to use a set of the IPC's benchmark domains and problems; that is helpful because they have been generated by human experts and, moreover, there exist official results for a preliminary evaluation of their hardness. On the contrary, random generators are typically used for configuring domain-specific portfolios. These generators have some parameters that can be used for tuning the problems' difficulty; by working on them, it is possible to finely set the hardness of problems. On this matter, Núñez et al. [42] showed that given a training set, the same results – in terms of portfolio configuration – achieved by using the whole training set can be obtained by considering a small subset of the training set. For selecting the subset, it is useful to consider the number of planners that solved each problem: those solved by a reduced number of solvers are informative. Even though it is not guaranteed that the resulting portfolio will be able to generalise on different instances, it will work well on problems similar to those used for training.

In order to evaluate the incorporated planners on the selection of learning problems, the performance metrics must be defined. It is usual to measure whether a plan is found or not (success or failure), the runtime needed for finding solutions and the quality of solutions. All of them are useful for configuring a portfolio for optimising any target function, as described in the section *Target and scope*.

When the learning instances have been selected and the metrics have been defined, all the incorporated planners have to be run on the learning instances. Since each planner has its own way of declaring success, they are not very standardised. It is important to develop a code to automatically extract these metrics from planners' output. Moreover, if incremental planners[2] are incorporated, it will be essential to define the way to measure their performance. In PbP, for instance, the authors handle this by measuring the quality of all the solutions generated for a problem, and the corresponding CPU times needed.

[2]Planners that are able to incrementally optimize solution plans after finding an initial satisficing one.

Planner selection

Selecting the planners to include in the portfolio is strictly related to the number of incorporated planners and the maximum allowed size of the configured portfolio. This step could be redundant in some portfolio structures: n-of-n design does not require any selection. The configured portfolio includes all the incorporated planners, and is based on the hypothesis that typically planners either solve a problem quickly or not at all [25]. This strategy is reasonable when all the following hold:

- the number of incorporated planners is limited;
- the incorporated planners have either really good mean performance or very good performance on some domains;
- the maximum amount of CPU time for solving a problem is large, with regards to the number of planners;

Specifically, with regards to the third condition, this refers to the fact that all the planners have enough CPU time to run. Reasonably, this means that each planner should run for at least a few tens of seconds: this is because, according to [25], if a planner does not find a solution quickly, it will not find it at all. Finally, we would emphasise that if the target of the portfolio is minimising the runtime, including all the incorporated planners will possibly make it hard to effectively order them.

In most of the cases, it is necessary to select only a subset of all the incorporated planners. In [42], the authors showed that it is possible to derive the optimal selection of solvers, with regard to the coverage, for the optimal track by using MIP techniques. When runtime has to be optimised, also the order of planners in the portfolio is critical. In that case, the number of possible configurations exponentially increases with the allowed maximum size of the portfolio, it is often computationally impossible to offer an exhaustive comparison: in those cases the most convenient approach is using heuristic techniques. For instance, a large selection of heuristics have been exploited and compared in the FDSS and FDSS2 papers [23, 52].

Other techniques can be adopted for pre-selecting a number of planners. This first step allows a reduction in the number of systems, and to perform a sophisticated analysis on the remaining ones. For instance, in ModelPerformance only planners that solved at least a predefined percentage of learning problems have been taken into account.

On the contrary, if the number of possible portfolios is limited, for instance because the maximum number of planners per portfolio is manually fixed, it is suggested to exhaustively compare all of them. The comparison can be done by a statistical analysis, as in PbP, or by evaluating the performance of planners using some metrics like the IPC scores [9, 34].

Allocation strategies and planner ordering

In this step of the portfolio configuration, the CPU time allocated to selected planners and planners' execution order are computed w.r.t. the selected scheduling (as described in section *Scheduling strategy*). It must be noted that the planners' ordering is fundamental for portfolios focusing on criteria defined upon time (such as speed), but irrelevant on portfolios with targets that are independent of time (such as coverage). Another common metric, namely the quality of solution plans, can be either time-dependent (its increment is measured over time) or time-independent (it is assessed at the end of the allotted time). In the latter case, which is commonly used in IPCs, for optimising quality the order of planners is irrelevant as well. On the other hand, the optimisation of quality over time depends strongly on the planners' scheduling.

If parallel portfolios do not need complex techniques for allocating CPU time to a selected planner, it is a critical step for portfolios with different scheduling (both serial and mixed): giving too much (not enough) CPU time to a planner, could significantly worsen performance.

For serial portfolios, the classical strategy is to equally divide the maximum amount of CPU time through all the selected planners [52]. This strategy, even though it is very easy, has been shown to achieve significant results in terms of quality of plans. Other approaches involve the estimation of time needed by a planner to solve a problem. Such estimation can be done heuristically or by exploiting empirical predictive models which relies on features evaluation.

Finally, in mixed portfolios the order of planners is not as critical as in serial ones. This is because planners are not run only once, but they are kept in the scheduling. On the other hand, a few works have investigated this strategy (ModelPerformance and PbP) by using the same approach. Time slots, to be used in a round-robin scheduling, corresponds to the CPU time needed by planners to solve increasing percentages of training instances.

Evaluating the portfolio

Typically, a portfolio is configured by evaluating the performance of incorporated planners on a set of learning instances, that are somehow related to the testing problems. Since the portfolio has been configured on problems different from the one on which it will be used, it is essential to evaluate its performance on (a subset of) testing instances.

Currently, no standard procedures have been defined and exploited in the existing portfolio-based planners for the evaluation step. Commonly, they are evaluated on a subset of the benchmark problems used in the IPCs.

In very large experimental studies, like IPCstudy and ModelPerformance ones, configured portfolio performance is usually estimated by N-fold cross-validation or leave-one-out techniques. The former randomly divides the training set in N equal size slices, regardless of the planning domains, the $N-1$ of the slices are used for training and the remaining one for testing. The process is repeated N times, with each of the slices used exactly once as a testing one. The estimated performance is the average of performance on testing slices. The leave-one-out technique is similar to cross-validation, but it is done considering planning domains. For each step a different domain is considered for testing, while all the others are used for training. It should be noted that both approaches are forms of cross-validation, but their aims are different. N-fold allows testing of how the performance generalises on different instances of known planning domains, while the leave-one-out approach provides information about the capability of the portfolio in generalising on a previously unseen domain.

Challenges

In this section we provide a list of what we consider to be open issues or future avenues in portfolio-based approaches for AI Planning. We are aware that this list is not complete, but we are highlighting the most important ones: we have not included those challenges and future avenues that are dependent on the selected ones. This is the case, for instance, for challenges related to determining the minimum and maximum portfolio size, which mainly depends on the planner selection strategy.

Definition

Although there are attempts to define either "portfolios" or "basic solver", none of the existing definitions can in all cases clearly distinguish whether the planner is a portfolio or basic solver. The need for such a distinction is driven by the necessity for fair comparison and evaluation of basic solvers. It might be unfair to compare basic solvers against portfolios and, moreover, this could have a detrimental effect on the development of new basic solvers. This is especially important in competitions such as IPC or SAT competition. In the SAT competition 2013, basic ("core") solvers were defined as follows[3]:

> Core solvers – this type of solvers are allowed to use at most two different SAT solving engines for all runs and at any time during one track. Type of solvers that fall into this category are single engine solvers like: minisat, glucose, lingeling, SAT4J, clasp, CCASat, Sparrow, sattime and hybrid solvers like: CCCeq. A preprocessor is not considered a solver so that a hybrid solver could additionally run a preprocessor. A portfolio approach consisting of only two different core solvers can also participate in this track.

Although such a definition seems to clearly distinguish between basic solvers and portfolios, there are some critical points. For instance, it is not clear why using "two" different SAT solving engines can be still considered as a basic solver. In the IPC 2014 Learning track, basic solvers have been awarded separately, however, earlier in this paper we made some critical points against the used definition of the basic solver. In the IPC 2014 Deterministic track, an "innovative planner award" has been introduced to support developing of novel (core) planning techniques. This award is based on the jury's judgement, so no clear criteria are given.

In this paper we provided a practical way for classifying planning approaches, which relies on the analysis of: (i) selection and combination process, and (ii) use of considered planning techniques. Nevertheless, some planning systems are still hard to classify, and stay in the "grey zone". Previous attempts to make a clear cut distinction between basic solvers and portfolios have shown themselves to be controversial on

[3]http://satcompetition.org/2013/description.shtml

some points. For a better classification, we have to raise and answer several questions. For example, how to classify planning systems that exploit concurrent runs of a search technique that share information and visit different areas of the search space?

Target

Every portfolio must have a target function to optimise. Typically these functions are very easy and concern three different performance, usually taken individually: runtimes, quality of solution plans (in terms of number of actions or action cost) and number of solved problems.

Most of the existing approaches are optimised for finding good quality plans (in terms of the number of actions or action costs) or for maximising the number of solved problems, and all of them exploit configured portfolios composed of several different planners. The existing systems that are able to configure a domain-specific portfolio for minimising runtime are PbP (and its latest version, PbP2) and ASAP. By observing the runtimes-configured portfolios that PbP generated for IPC-6 benchmark domains [17, 11], it is noticeable that a single planner (possibly with additional knowledge extracted from the domain in the form of macro-actions) is often selected. Such a trend is also confirmed by ASAP, that demonstrated extremely good performance in terms of runtime, while exploiting a single planner per domain. It would be interesting to offer an in-depth analysis for better understanding this behaviour. Is it related to the scheduling strategy, to the knowledge extracted from the domain, to the size/structure of the considered testing problems, or is it typical of domain-specific portfolios focusing on speed?

Probably, it is possible to find some planners that work particularly well on a subset of problems of a specific domain, while a single planner (possibly with some type of additional domain-related knowledge) is able to achieve good performance in the average case. This is especially true in domains where problems' structure vary a lot. Thus, in such domains, a more specialised portfolio (*class-specific* rather than domain-specific) will be appropriate.

Training instances

Two main decisions have to be taken with regard to the training instances. Firstly, a large set of training instances have to be generated or selected from existing collections; secondly, an informative subset of them is used for the actual training of the portfolio. The two decisions can be taken at the same time, but they address two significantly different problems. While the former focuses on identifying reasonable –neither trivial nor too complex– instances, the latter deals with the issue of investigating the informativeness of the considered instances. To this extent, it should be noted that the problem of evaluating the utility of training instances has been addressed by Núñez et al. in [42].

Traditionally, training problems are selected from IPC's benchmark or obtained by random generators that offer some parameters to tune the problem's difficulty. The former approach is limited to the number of already existing domains and instances. In terms of number of instances, hundreds of problems are available, but the actual number of domains is small – some domains are also analogous – and problems are usually generated by a single generator; therefore their structure is often similar. Moreover, many benchmark problems from earlier IPCs are trivial for the current state-of-the-art planners as observed in [51, 52].

On the other hand, the latter approach – which relies on exploiting random generators – has two main limitations: (i) although in some domains the solvability of problems can be guaranteed, it is generally not trivial to guarantee the problems' solvability; (ii) the generators' parameters are domain-specific and tuning them to generate good quality learning examples implies significant domain expertise. In order to avoid the requirement for domain expertise, Fern et al. [12] introduced an approach based on random walks. It generates a random initial state and takes *n* random actions to produce a state sequence. In domains where complexity depends on the number of objects, this approach is not enough. Fuentetaja and Borrajo [13] proposed an approach for generating problems using a given "sample" problem: the learning process has the bias imposed by this sample problem and could generate unsolvable problems, but it allows the generation of a random exploration of problems of increasing objects. A combination of these two approaches would possibly lead to the autonomous generation of learning problems.

Planner selection and ordering

A striking result [52] is that, in terms of the quality of solutions found, none of the more sophisticated strategies for configuring domain-independent static portfolios, performs better than the uniform portfolio (i.e., *all* the incorporated planners are selected, in a random order, and have the same amount of CPU time). Clearly, such an approach can be used when the number of incorporated planners is limited. A similar behaviour is observed in IPCstudy; although the best strategy consists of selecting the best 10 planners with regard to the confidence, and allocating them the same amount of CPU time, the approach that uses all the 27 planners in a random order is able to achieve comparable performance. Moreover, the latter is often better than strategies which select a small number of planners.

Additionally, Seipp et al. [52] indicate that portfolio performance can be improved much more by diversifying the set of incorporated planners than by adjusting selected planners' runtimes or ordering. From these results we can argue that configuring a portfolio for maximising plan quality is useless if enough (but not too many) different planning techniques are considered. Intuitively, we believe that each scheduled planners should have at least a few hundreds of allotted CPU time seconds. Moreover, a very specific portfolio configuration (selecting planners, allocating different runtimes and giving a specific order to them) could be wasteful because the selection techniques could make mistakes.

We are not totally convinced by this result, we believe that the aforementioned approaches work well on testing instances that are from similar distribution, or similar domains, of training problems. Moreover, allotting short CPU time to planners might prevent solving large instances, where most of the time is spent in generating the data structures. Even though some works showed that most planners either solve a problem fast or not at all, our intuition is that this is true on deterministic IPC benchmark instances, but is not always true on larger instances, such as those from the learning tracks of IPC. On this matter, Gerevini et al. [19] showed that on large instances, a single planner can require almost all the available (900 seconds) CPU time for finding a plan. On the other hand, while the selection of planners is critical for optimising metrics defined upon time, the ordering of planners could be irrelevant if portfolios are configured for optimising other criteria that are independent of time, as previously discussed.

Predictive model

A large number of approaches build and exploit predictive models for configuring domain-independent portfolios of planning systems. These approaches are based on feature extraction and evaluation. Usually, a very large set of features, characterising both the domain and the problem is used. Nowadays, a few hundred features are available to the planning community. On the basis of the observed performance of planners on training instances, performance and features are correlated for predicting the behaviour of planners on unseen instances, by generating *empirical predictive models* [29]. These models have shown to be useful for selecting and combining planning algorithms by predicting their performance –see for instance results described in IPCstudy and ModelPerformance, or the performance of IBaCoP –, but are generated by using techniques, like random forests, that can hardly be analysed. This significantly limits the understanding of how features affect performance, which is fundamental for exploiting the knowledge encoded under the form of predictive models for improving existing planners. In particular, being able to identify a small set of features that strongly affects the behaviour of a planning engine, would allow experts to better understand the planning process and, subsequently, to design solutions that enhance planners and planning techniques. Understanding what affects planners' performance, would possibly lead experts to understand also *why* planners are affected, and therefore shed some light on them and, generally, on automated planning.

Automated framework

Existing systems, since most of them has been developed for participating in IPCs, do not have an automated configuration process that allows the configuration of different types of portfolios. It would be useful, for a better understanding of portfolios' performance, to have a framework (hopefully, user-friendly) that is able to automatically generate several different classes of portfolios and to compare all of them through different techniques. Such a framework will provide an easy tool for studying the performance of portfolios and to evaluate the impact of new ideas in configuration steps. Moreover, that framework would also suggest a potential method for testing new planners, based on measuring the performance improvements obtained in several different portfolios by adding them as incorporated planners. This is similar to techniques applied in areas of AI different from planning [63].

In different fields of Artificial Intelligence such tools already exist (e.g. HAL [38]). Regarding AI Planning, a well known existing tool is itSIMPLE [61, 60]. This Knowledge Engineering tool currently focuses on disciplining the design cycle of planning problems. It includes a sort of testing phase, which consists of running a set of included planners on the generated domains and measuring their performance using different metrics. Improving the testing phase by integrating a portfolio-based evaluation would result in a very complete tool for evaluating domain encodings and planner combinations.

Share information

Existing portfolio approaches use planning systems as black-boxes. Usually selected planners do not share any kind of information, knowledge or evaluations about the search space of the current problem. On the other hand it is true that some existing systems share some sort of information: FDSS and PbP (while exploiting the portfolio for optimising plan quality) share between included planners the best solution found so far; most notably, ArvandHerd shares the walks explored and the control strategy for allocating the next ones. Even though, according to our definition, it would not be classified as a portfolio, it shows an interesting example of communication between planning techniques.

On the other hand, some existing planners already implement a form of information sharing. This is the case of Fast Downward, that by virtue of its multiple open lists it is able to to alternate and combine heuristics and preferred operators in the search procedure [46].

In order to push forward the performance of planners based on a portfolio of planning systems, they should share information, communicate and cooperate to reach the goal. The information to be shared is that which contemporary planning systems use to guide their search; e.g. heuristic estimates, preferred operators, landmarks, dead ends, subplans. To this end, we believe that treating each portfolio member as an agent will allow portfolio approaches to exploit techniques used in multi agent systems. We allow the agents to share information, however, a communication protocol is needed. It is also important to carefully balance communication and search; the overhead introduced by the communication should not significantly worsen the performance of agents (selected planners). Works in this area have been recently carried out by Nissim et al. [40], investigating the cooperation between agents that have private information they do not want to reveal, and specific capabilities.

Evaluation

Typically, a portfolio is configured by evaluating the performance of the incorporated planners on a set of learning instances, that are somehow related to the testing problems. Since the portfolio has been configured on problems different from the one on which it will be used, it is essential to evaluate its performance on (a subset of) testing instances. A configured portfolio must achieve, at least, better performance than every individual incorporated planner, so it is good practice to compare against all of them. After that, the main questions are:

1. given the selected structure of the portfolio, did we correctly configure it?

2. is the selected portfolio structure suitable for our target and scope?

For finding an answer to the former question, the best strategy is to compare the configured portfolio with an oracle: a portfolio with same structure but configured exactly on the testing problems. This is the strategy adopted by Núñez et al. in MIPstudy for generating a baseline of the performance, to compare with other planners.

For the latter question, there is still no answer. This problem has not been considered yet. Ideally, it would be good to have some "guidelines" that provide indications about the portfolio structure to exploit with regards to some specific needs. Generally, it would be enough to compare against differently structured portfolios, that share the same training instances and incorporated planners. It should be noted that selecting the most appropriate portfolio structure for this comparison is -at least- as difficult as selecting the preferred portfolio configuration. Currently, the most convenient strategy is to compare with state-of-the-art portfolio-based planners, e.g. by selecting them from recent IPCs.

Reformulation

Techniques for reformulating domain or problem models have been studied in planning since the 1970s, and include online and offline approaches for identifying macro-actions [33, 39], entanglements [8], or for decomposing complex operators into simpler ones [1]. State-of-the-art portfolios that exploit reformulation techniques, consider –for the purpose of portfolio configuration– a reformulated model and a planning engine as a single algorithm. This means that the same planner, provided with different input, is treated as if it were two completely unrelated algorithms. ASAP, as well as PbP, exploit this approach. While this approach reduces the complexity, at the cost of a potentially significant increase in the number of considered algorithms, it does not allow to directly relate the effectiveness of reformulation with domains' structure and planners' strategy. An alternative which should be investigated could be based on a hierarchical selection approach that, given a testing problem, selects a promising reformulation technique –on the basis of previously observed performance on similar structures– and then selects a planning engine that has shown good performance on reformulated problems that share some similarities with the considered one.

Hierarchical selection processes have been exploited in some areas of AI like SAT [62], and it might be fruitful to test them also in planning. It can both boost portfolio performance and provide insights into the actual reformulation impact on different kind of problems.

Updating Knowledge

In state-of-the-art portfolio-based planners, as well as in most of the portfolio approaches exploited in AI, the knowledge used for configuring a portfolio is extracted once and hardly updated. This has been called *one-shot learning approach* [35], since no evolution of the knowledge is considered. The drawback of this approach is that in a dynamic environment, like real-world planning applications, the type of problems that the system is expected to solve changes. Currently, no investigation in planning has been performed for providing any mechanism to evaluate if the performance of the configured portfolio is decreasing and it should be trained again. Intuitively, a possible solution is to periodically re-train the portfolio, regardless of actual performance. This can be computationally expensive and, moreover, raises the question of when is the right time to train. On the other hand, since a large number of problem-related features are nowadays available [10], it is worth considering to exploit them also for measuring if the structure of current problems is significantly different than the structure of training ones.

Evolution of knowledge exploited by portfolio approaches is a novel topic also for AI in general, and a first work in this direction has been done in 2013 by Malitsky et al. [35].

Planner Portfolios for different paradigms

A number of planner portfolios have been recently developed. We notice that they are mostly focused on classical planning. In particular, the majority deal with satisficing planning, while a few approaches – in many cases straightforward implementations of satisficing systems – are able to deal with optimal planning. We are not aware of portfolio-based planners for temporal planning or probabilistic planning. On the one hand, this is probably due to the fact that a large number of planners are available for classical planning. On the other hand, having portfolios of systems that are able to handle different planning paradigms would be undoubtedly helpful. Firstly, it can lead to an improvement of the state-of-the-art by stimulating development of portfolios as well as new planning techniques. Secondly, it might shed some light on aspects that affect planners performance in order to predict planners' performance more accurately.

Conclusions

The existing AI Planning technology offers a large, growing set of powerful techniques and efficient domain-independent planners, but none of them outperforms all the others in every known planning domain. This observation, and the results achieved by portfolio approaches in different fields of AI, motivated the idea of configuring and exploiting a portfolio of planners to achieve better performance than any individual planner: recently, several different high-performance portfolio-based planners have been developed.

In this paper we described the idea of algorithm portfolio, and outlined the existing planning systems based on this approach. Then, we listed the different decisions that have to be taken for configuring a portfolio of planners, and divided them in two subsets: decisions to take *offline* and decisions to take *online*, w.r.t. the learning problems used for the portfolio configuration. We exhaustively described every configuration step, and analysed how existing portfolio approaches in planning deal with them. Finally, we focused on the challenges of existing techniques for configuring a portfolio of planners, in order to

- give an overview of the state-of-the-art of portfolio-based planners;

- describe the decisions that have to be taken during the configuration process;
- stimulate the development of new high-performance planning frameworks based on this approach.

This analysis is motivated by the excellent results achieved by portfolio-based planning systems in recent IPCs: they won, or got very close to winning, in almost every track in which they took part. These impressive results let us suppose that future of AI Planning will be related to algorithms and techniques for effectively combining planners, in order to obtain results that cannot be achieved by a single domain-independent planner. We recognise that further studies are needed to analyse the highlighted open issues and to increase the performance that can be achieved by exploiting a portfolio approach in AI Planning, since we are confident that portfolio techniques, having only recently been extensively applied in AI Planning, will lead to further significant improvements in the near future. Such improvements will be twofold: on the one hand, a further enhancement of portfolio-based approaches; on the other hand, the amelioration of basic solvers led by a better comprehension of planners' behaviour.

References

[1] C. Areces, F. Bustos, M. Dominguez, and J. Hoffmann. Optimizing planning domains by automatic action schema splitting. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS*, 2014.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

[3] C. Bäckström and B. Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11:625–655, 1993.

[4] I. Cenamor, T. de la Rosa, and F. Fernández. Learning predictive models to configure planning portfolios. In *Proceedings of the 4th workshop on Planning and Learning (ICAPS-PAL 2013)*, pages 14–22, 2013.

[5] I. Cenamor, T. de la Rosa, and F. Fernández. Ibacop and ibacop2 planner. In *The Eighth International Planning Competition. Description of Participant Planners of the Deterministic Track*, 2014.

[6] Y. Censor. Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization*, 4(1):41–59, 1977.

[7] Y. Chen, B. W. Wah, and C.-W. Hsu. Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research (JAIR)*, 26:323–369, 2006.

[8] L. Chrpa and T. L. McCluskey. On exploiting structures of classical planning problems: Generalizing entanglements. In *20th European Conference on Artificial Intelligence (ECAI-2012)*, pages 240–245, 2012.

[9] A. Coles, A. Coles, A. G. Olaya, S. Jiménez, C. L. Lòpez, S. Sanner, and S. Yoon. A survey of the seventh international planning competition. *AI Magazine*, 33:83–88, 2012.

[10] C. Fawcett, M. Vallati, F. Hutter, J. Hoffmann, H. H. Hoos, and K. Leyton-Brown. Improved features for runtime prediction of domain-independent planners. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS*, pages 355–359, 2014.

[11] A. Fern, R. Khardon, and P. Tadepalli. The first learning track of the international planning competition. *Machine Learning*, 84:81 – 107, 2011.

[12] A. Fern, S. W. Yoon, and R. Givan. Learning domain-specific control knowledge from random walks. In *Proceedings of the 14th International Conference on Automated Planning & Scheduling (ICAPS)*, pages 191–199, 2004.

[13] R. Fuentetaja and D. Borrajo. Improving control-knowledge acquisition for planning by active learning. In *Proceedings of the 17th European Conference on Machine Learning (ECML)*, pages 138–149. Springer, 2006.

[14] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, M. T. Schneider, and S. Ziller. A portfolio solver for answer set programming: Preliminary report. In *Logic Programming and Nonmonotonic Reasoning*, pages 352–357. Springer, 2011.

[15] H. Geffner and B. Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013.

[16] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research (JAIR)*, 20:239 – 290, 2003.

[17] A. Gerevini, A. Saetti, and M. Vallati. An automatically configurable portfolio-based planner with macro-actions: PbP. In *Proceedings of the 19th International Conference on Automated Planning & Scheduling (ICAPS-09)*, pages 350 – 353. AAAI Press, 2009.

[18] A. Gerevini, A. Saetti, and M. Vallati. PbP2: Automatic configuration of a portfolio-based multiplanner. In *Working notes of 21st International Conference on Automated Planning & Scheduling (ICAPS-11) – 7th International Planning Competition*, 2011.

[19] A. Gerevini, A. Saetti, and M. Vallati. Planning through automatic portfolio configuration: The PbP approach. *Journal of Artificial Intelligence Research (JAIR)*, 50:639–696, 2014.

[20] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers, 2004.

[21] C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.

[22] M. Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research (JAIR)*, 26:191 – 246, 2006.

[23] M. Helmert, G. Röger, and E. Karpas. Fast Downward Stone Soup: A baseline for building planner portfolios. In *Proceedings of the ICAPS-11 Workshop of AI Planning and Learning (PAL)*, 2011.

[24] J. Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research (JAIR)*, 20:291–341, 2003.

[25] A. Howe and E. Dahlman. A critical assessment of benchmark comparison in planning. *Journal of Artificial Intelligence Research (JAIR)*, 17:1 – 33, 2002.

[26] A. Howe, E. Dahlman, C. Hansen, A. vonMayrhauser, and M. Scheetz. Exploiting competitive planner performance. In *Proceedings of the 5th European Conference on Planning (ECP-99)*, pages 62 – 72. Springer, 1999.

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

[28] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[29] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artif. Intell.*, 206:79–111, 2014.

[30] H. Kautz and B. Selman. Unifying sat-based and graph-based planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 318–325. IJCAI Organization, 1999.

[31] H. Kautz, B. Selman, and J. Hoffmann. Satplan: Planning as satisfiability. In *Abstract Booklet of the 5th International Planning Competition*, 2006.

[32] M. R. Khouadjia, M. Schoenauer, V. Vidal, J. Dréo, and P. Savéant. Multi-objective AI planning: Evaluating dae YAHSP on a tunable benchmark. In *Proceedings of the 7th International Conference on Evolutionary Multi-Criterion Optimization (EMO)*, pages 36–50, 2013.

[33] R. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77, 1985.

[34] C. L. Lpez, S. J. Celorrio, and ngel Garca Olaya. The deterministic part of the seventh international planning competition. *Artificial Intelligence*, 2015.

[35] Y. Malitsky, D. Mehta, and B. O'Sullivan. Evolving instance specific algorithm configuration. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search SOCS*, pages 132–140, 2013.

[36] P. J. Matos, J. Planes, F. Letombe, and J. Marques-Silva. A MAX-SAT algorithm portfolio. In *the 18th European Conference on Artificial Intelligence ECAI*, pages 911–912, 2008.

[37] H. Nakhost and M. Müller. Monte-carlo exploration for deterministic planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1766–1771, 2009.

[38] C. Nell, C. Fawcett, H. H. Hoos, and K. Leyton-Brown. HAL: A framework for the automated analysis and design of high-performance algorithms. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION)*, pages 600–615. Springer, 2011.

[39] M. H. Newton, J. Levine, M. Fox, and D. Long. Learning macro-actions for arbitrary planners and domains. In *Proceedings of the 17th International Conference on Automated Planning & Scheduling (ICAPS)*, pages 256–263, 2007.

[40] R. Nissim and R. I. Brafman. Distributed heuristic forward search for multi-agent planning. *J. Artif. Intell. Res. (JAIR)*, 51:293–332, 2014.

[41] S. Núnez, D. Borrajo, and C. L. Lòpez. How good is the performance of the best portfolio in ipc-2011? In *Proceedings of ICAPS-12 Workshop on International Planning Competition*, 2012.

[42] S. Núñez, D. Borrajo, and C. L. López. Performance analysis of planning portfolios. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search SOCS*, pages 65–71, 2012.

[43] L. Pulina and A. Tacchella. A multi-engine solver for quantified boolean formulas. In *Principles and Practice of Constraint Programming - CP*, pages 574–589, 2007.

[44] L. Pulina and A. Tacchella. A multi-engine solver for quantified boolean formulas. In *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP)*, pages 574–589. Springer, 2007.

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

[46] S. Richter. *Landmark-Based Heuristics and Search Control for Automated Planning*. PhD thesis, Griffith University, 2010.

[47] S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research (JAIR)*, 39:127 – 177, 2010.

[48] J. Rintanen. Engineering efficient planners with SAT. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI-12)*, pages 684–689. IOS Press, 2012.

[49] M. Roberts and A. Howe. Directing a portfolio with learning. In *AAAI 2006 Workshop on Learning for Search*, pages 129–135, 2006.

[50] M. Roberts and A. Howe. Learned models of performance for many planners. In *Proceedings of the ICAPS-07 Workshop of AI Planning and Learning (PAL)*, 2007.

[51] M. Roberts and A. Howe. Learning from planner performance. *Artificial Intelligence*, 173(5-6):536–561, 2009.

[52] J. Seipp, M. Braun, J. Garimort, and M. Helmert. Learning portfolios of automatically tuned planners. In *Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS)*, pages 369 – 372. AAAI Press, 2012.

[53] J. Seipp, S. Sievers, M. Helmert, and F. Hutter. Automatic configuration of sequential planning portfolios. In *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI)*, 2015.

[54] J. Seipp, S. Sievers, and F. Hutter. Fast downward cedalion. In *The Eighth International Planning Competition. Description of Participant Planners of the Deterministic Track*, 2014.

[55] W. F. Sharpe and W. Sharpe. *Portfolio theory and capital markets*, volume 217. McGraw-Hill New York, 1970.

[56] R. Valenzano, H. Nakhost, M. Müller, J. Schaeffer, and N. Sturtevant. Arvandherd: Parallel planning with a portfolio. In *Proceedings of the 20th European Conference on AI (ECAI-12)*, pages 786 – 791. IOS Press, 2012.

[57] R. Valenzano, H. Nakhost, M. Müller, J. Schaeffer, and N. Sturtevant. Arvandherd 2014. In *The Eighth International Planning Competition. Description of Participant Planners of the Deterministic Track*, 2014.

[58] M. Vallati, L. Chrpa, and D. E. Kitchin. An automatic algorithm selection approach for planning. In *IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1–8, 2013.

[59] M. Vallati, C. Fawcett, A. Gerevini, H. Hoos, and A. Saetti. Automatic generation of efficient domain-specific planners from generic parametrized planners. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search (SOCS)*, 2013.

[60] T. Vaquero, J. R. Silva, and J. C. Beck. Analyzing plans and planners in itSIMPLE3.1. In *Proceedings of the Knowledge Engineering for Planning and Scheduling workshop. The 20th International Conference on Automated Planning & Scheduling (ICAPS)*, 2010.

[61] T. Vaquero, R. Tonaco, G. Costa, F. Tonidandel, J. R. Silva, and J. C. Beck. itSIMPLE4.0: Enhancing the modeling experience of planning problems. In *Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS-12) – System Demonstration*, pages 11–14, 2012.

[62] L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research (JAIR)*, 32:565–606, 2008.

[63] L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown. Evaluating component solver contributions to portfolio-based algorithm selectors. In *Theory and Applications of Satisfiability Testing - SAT*, pages 228–241, 2012.