Journal of Artificial Intelligence Research 50 (2014) 639–696 Submitted 02/14; published 07/14

Planning through Automatic Portfolio Configuration: The PbP Approach

Alfonso Emilio Gerevini ALFONSO.GEREVINI@UNIBS.IT
Alessandro Saetti ALESSANDRO.SAETTI@UNIBS.IT
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Brescia
Via Branze 38, I-25123 Brescia, Italy

Mauro Vallati M.VALLATI@HUD.AC.UK
School of Computing and Engineering
University of Huddersfield
Huddersfield, West Yorkshire, HD1 3DH, UK

Abstract

In the field of domain-independent planning, several powerful planners implementing different techniques have been developed. However, no one of these systems outperforms all others in every known benchmark domain. In this work, we propose a multi-planner approach that automatically configures a portfolio of planning techniques for each given domain. The configuration process for a given domain uses a set of training instances to: (i) compute and analyze some alternative sets of macro-actions for each planner in the portfolio identifying a (possibly empty) useful set, (ii) select a cluster of planners, each one with the identified useful set of macro-actions, that is expected to perform best, and (iii) derive some additional information for configuring the execution scheduling of the selected planners at planning time. The resulting planning system, called PbP (Portfolio-based Planner), has two variants focusing on speed and plan quality. Different versions of PbP entered and won the learning track of the sixth and seventh International Planning Competitions. In this paper, we experimentally analyze PbP considering planning speed and plan quality in depth. We provide a collection of results that help to understand PbP's behavior, and demonstrate the effectiveness of our approach to configuring a portfolio of planners with macro-actions.

1. Introduction

During the last fifteen years, the field of automated plan generation has achieved significant advancements, and several powerful domain-independent planners are today available, e.g., for propositional planning, FF (Hoffmann & Nebel, 2001), LPG (Gerevini, Saetti, & Serina, 2003), SGPlan (Chen, Hsu, & Wah, 2006), Fast Downward (Helmert, 2006), and LAMA (Richter & Westphal, 2010). Moreover, while each of such systems performs very well on a (more or less large) class of planning domains and problems, it is well-known that no one outperforms all the others in every available benchmark domain (see, e.g., Roberts & Howe, 2009). It would then be useful to have a multi-planner system that automatically selects and combines the most efficient planner(s) in a portfolio for each given domain.

The performance of the current planning systems is typically affected by the structure of the search space, which depends on the considered planning domain. For many domains, the planning performance can be improved by exploiting some knowledge about the domain structure that is not explicitly given as part of the input formalization, but that can be automatically derived from it. In particular, several approaches encoding additional knowledge in the form of macro-actions have been proposed (e.g., Botea, Enzenberger, Müller, & Schaeffer, 2005; Newton, Levine, Fox, & Long, 2007). A macro-action (*macro* for short) is a sequence of actions that can be planned at one time like a single action. When using macros there is an important tradeoff to consider. While their use can speedup the planning process, because it reduces the number of search steps required to reach a solution, it also increases the search space size, which could slow down the planning process. Moreover, it is known that the effectiveness of macros can depend on the planning algorithm: a set of macros can increase the performance of a planner, but decrease it, or be irrelevant, for another.

In this paper, we propose an approach to automatically configuring a portfolio of existing planners, possibly using a useful set of macros for each of them. The configuration relies on a statistical analysis of the performance of the planners in the portfolio and the usefulness of some automatically generated sets of macros, considering a set of training problem instances in a given domain. The configuration knowledge that is automatically generated by this analysis consists of a cluster of planners defined by: an ordered subset of the planners in the initial portfolio, which at planning time are combined using a round-robin strategy; a set of useful macros for each planner; and some sets of planning time slots. The planning time slots specify the amount of CPU time to be allocated to each planner in the cluster during planning. The resulting planning system is called PbP (*Portfolio-based Planner*).

The current implementation of PbP incorporates two systems for the generation of macros and nine efficient planners, but its architecture is open to consider any other (current or future) planner as an additional or alternative system. If PbP is used without configuration knowledge, all planners in the portfolio are scheduled (without macros) by a simple round-robin strategy where some predefined CPU-time slots are assigned to the (randomly ordered) planners. If PbP is used with the configuration knowledge for the domain under consideration, only the selected cluster of planners (possibly using the relative selected sets of macros) is scheduled, their ordering favors the planners that during configuration performed best, and the planning time slots are defined by the computed configuration knowledge. As for the selection and exploitation of macros in PbP, it is worth noting that the planners in the portfolio configured by PbP do not necessarily use the macros learned for them. In the configuration process, the system evaluates each planner in the portfolio with each set of macros computed for it, as well as with the empty macro set, as if they were independent planning systems.

PbP has two main variants: PbP.s, focusing on speed, and PbP.q, focusing on plan quality. A preliminary implementation of PbP.s (Gerevini, Saetti, & Vallati, 2009) entered the learning track of the sixth international planning competition (IPC6) and was the overall winner of this competition track (Fern, Khardon, & Tadepalli, 2011).[1] More recently, a

1. As observed by the IPC6 organizers, surprisingly, for the IPC6 problems the use of the configuration knowledge does not considerably speedup this version of PbP.s. The reasons are some implementation bugs concerning both the configuration phase and the planning phase, and the inefficient use of

revised and optimized version of PbP with both the speed and quality variants entered the learning track of the seventh competition (IPC7), and it was again the winner of this competition track (Coles, Coles, Olaya, Celorrio, López, Sanner, & Yoon, 2012).

A large experimental analysis presented in this paper provides a collection of results that help to understand the performance behavior of PbP and the effectiveness of its portfolio configuration methods. In particular, the analysis (i) confirms the very good performance of PbP in the context of the IPC6-7 benchmarks, (ii) compares PbP with other existing approaches to configure a planner portfolio, (iii) evaluates the accuracy of PbP's approach to identify an effective cluster of planners and the strength of using a (configured and unconfigured) multi-planner with respect to a single planner, (iv) investigates the usefulness of macros in the considered benchmarks, showing that PbP selects useful macro sets, and (v) examines the execution scheduling configuration of PbP for the selected planners in the configured portfolio, demonstrating that its default strategy works well compared to other possible strategies considered in the analysis.

Several ideas and techniques investigated in the context of PbP use or build on previous work. Besides presenting and evaluating an effective approach to configuring a planner portfolio, the research presented in this paper corroborates, validates or evaluates some "hunches" and empirical studies done by other researchers in planning. In particular, our experimental analysis confirms that certain sets of macros can be very useful to accelerate planning speed or improve plan quality (Botea et al., 2005; Coles & Smith, 2007; Newton et al., 2007) while others are harmful, that diversity of the planning techniques is important in the construction of an effective planner portfolio, as observed by, e.g., Roberts and Howe (2009), and that the round-robin scheduling of the planner execution times is a robust strategy for a planner portfolio (Howe, Dahlman, Hansen, vonMayrhauser, & Scheetz, 1999; Roberts & Howe, 2006).

The remainder of the paper is organized as follows. Section 2 discusses related work; Section 3 describes the PbP approach; Section 4 presents the results of our experimental study; finally, Section 5 gives the conclusions.

2. Related Work

In this section, after a brief presentation of the most prominent work on algorithm portfolio design in automated reasoning, we describe related work by others on planner portfolio design in the automated planning, pointing out some important differences between PbP and the most related work. Other specific differences and similarities will be indicated in the following sections presenting our technical results.

2.1 Algorithm Portfolio Design in Automated Reasoning

In the field of automated reasoning, the idea of using a portfolio of techniques has been investigated by several researchers. A prominent example is the work by Gomes and Selman (2001), who conducted a theoretical and experimental study on the parallel run of stochastic algorithms for solving computationally hard search problems. Their work shows under what

some Linux shell scripts (evident especially for small or easy problems), which were corrected after the competition obtaining much better results (Gerevini et al., 2009).

conditions running different stochastic algorithms in parallel can give a computational gain over running multiple copies of the same stochastic algorithm in parallel.

Many papers on algorithm portfolio design concern the definition of models to select the best algorithm(s) for an instance of a certain problem according to the values of some predetermined features of the instance (Rice, 1976). For example, algorithm portfolios have been designed with this aim to solve instances of SAT, MaxSAT, and QBF (Matos, Planes, Letombe, & Marques-Silva, 2008; Pulina & Tacchella, 2007; Xu, Hutter, Hoos, & Leyton-Brown, 2008). SATzilla is a prominent example of an algorithm portfolio designed for SAT (Xu et al., 2008). SATzilla uses machine learning techniques to build a predictor of the runtime of a class of SAT solvers. When SATzilla attempts to solve an instance of the SAT problem, it computes the values of some features of the instance, predicts the performance of the SAT solvers it incorporates, selects the most promising SAT solvers and order them accordingly to their predicted performance, and finally runs the selected SAT solvers using the established ordering and the predicted required CPU times.

Matos et al. (2008) propose an algorithm portfolio solving the MaxSAT problem. The portfolio computes the values of several features of a given instance of the MaxSAT problem, estimates the runtime for each solver in the portfolio, and then solves the instance with the estimated fastest solver. The estimation is done using a (linear) model configured by performing ridge regression (Marquardt & Snee, 1975). Similarly, Pulina and Tacchella (2007) study an algorithm portfolio solving the QBF problem. They identify some features of the QBF problem, and investigate the usage of four inductive models to select the best solver to use according to the values of the identified features.

2.2 Planner Portfolio Design in Automated Planning

Regarding automated planning, some prominent planners combining one or more algorithms have been proposed. Blackbox (Kautz & Selman, 1999) can use a variety of satisfiability engines (the initial version also included the Graphplan algorithm), and FF (Hoffmann & Nebel, 2001), LPG (Gerevini et al., 2003; Gerevini, Saetti, & Serina, 2006) and SGPlan5 (Chen et al., 2006) include a “backup strategy” using an alternative search technique that is run when the default method fails. The algorithm combination in these systems is straightforward and does not use an automatic portfolio configuration.

Previous work on planner portfolios includes the approach proposed by Howe and collaborators (Howe et al., 1999; Roberts & Howe, 2007, 2009; Roberts, Howe, Wilson, & desJardins, 2008). In the rest of the paper, we will refer to Howe and collaborators’ approach using the name of their first planner portfolio, BUS (Howe et al., 1999), although our analysis of this approach will consider their most recent techniques for the planner portfolio configuration. Their approach learns models of performance for a set of planners. At planning time, the round-robin policy is used to schedule the runs of the planners in such a set, and the learned models are exploited to determine the order of the runs. The configuration-knowledge derived by this approach is domain-independent: the performance models of the planners are built by using several predictive models of the WEKA data mining package (Witten & Frank, 2005), and the set of planners forming the portfolio is determined through a set covering algorithm over the solved training problems across several different planning domains.

The work on BUS originally inspired our approach. PbP has some similarities with it, but it computes and uses very different configuration knowledge, and the methods for selecting and ordering the portfolio planners are considerably different. The portfolio configuration of PbP generates domain-optimized clusters of planners, and the selection and ordering of PbP is based on a statistical analysis of the planners' performance over a set of training problems using the Wilcoxon sign-rank test, also known as the "Wilcoxon matched pairs test" (Wilcoxon & Wilcox, 1964).[2] Finally, their system does not compute, analyze or use macros, and does not consider plan quality.

Similarly to the work of Howe et al. (1999), and Roberts and Howe (2007), the techniques used by Cenamor, de la Rosa, and Fernàndez (2013), and Fawcett, Vallati, Hutter, Hoffmann, Hoos, and Leyton-Brown (2014) learn models of performance of a set of planners according to some predetermined features. In the work of Cenamor et al. (2013), such features are derived from the SAS+ representation of the planning problem. In this approach, the learned models are used to determine which planners should be run, in which order, and for how long. The selected planners run sequentially either using an amount of CPU time uniformly assigned or determined from the predicted execution time. The experimental results in the work of Cenamor et al. (2013) show that for problems in domains different from those used to learn the models, the configured portfolios perform worse than running an unconfigured portfolio consisting of all the incorporated planners with uniform CPU time assigned to each of them.

The work described by Fawcett et al. (2014) is focused on generating models for accurately predicting planners runtime. Such models exploit a large set of instance features, derived from the PDDL and SAS+ representations of the problem, a SAT encoding of the planning problem, and (short) runs of planners. The experimental results in the work of Fawcett et al. (2014) indicate that the generated performance models are able to produce very accurate runtime predictions.

Fast Downward Stone Soup (here abbreviated FDSS) is an approach to selecting and combining a set of forward-state planning techniques (Helmert, Röger, & Karpas, 2011). Using the IPC6 scoring function, FDSS evaluates a class of candidate techniques on the basis of their performance over a set of training problem instances from different domains, and builds a domain-independent sequential portfolio of forward planners by a hill-climbing algorithm searching a space of possible sequential combinations of the evaluated candidate techniques. The automatic portfolio configuration in FDSS and PbP aims at building different types of planning systems: a single efficient domain-independent planner portfolio in FDSS; an efficient domain-optimized portfolio planner for each given domain in PbP. The configuration processes and the resulting configured portfolios of FDSS and PbP are significantly different. In particular, PbP configures a portfolio of generic planners (using different styles of planning), each one with a (possibly empty) set of useful learned macros, which are not considered in FDSS because of its domain-independent purpose. Moreover, the execution scheduling strategy of PbP runs the selected planners in round-robin rather than sequentially.

2. In the context of planning, the Wilcoxon sign-rank test has been previously used also in the work of Long and Fox (2003), Gerevini, Haslum, Long, Saetti, and Dimopoulos (2009), Gerevini et al. (2009), and Roberts and Howe (2009).

ParLPG (Vallati, Fawcett, Gerevini, Hoos, & Saetti, 2013b) and Fast Downward-autotune (Fawcett, Helmert, Hoos, Karpas, Röger, & Seipp, 2011) configure the parameters of planners LPG and Fast Downward (Helmert, 2006), respectively, using a set of training problems of a given domain in order to obtain combinations of parameters for these two planners that perform especially well in the given domain. Both these frameworks uses the stochastic local search procedure ParamILS to search for high-performance configurations of parameters by evaluating promising configurations (Hutter, Hoos, & Stützle, 2007; Hutter, Hoos, Leyton-Brown, & Stützle, 2009). An extended version of FDSS (Seipp, Braun, Garimort, & Helmert, 2012) involves twenty one configurations of Fast Downward, obtained by configuring its parameters through Fast Downward-autotune for twenty one different domains (Fawcett et al., 2011), that are combined by several alternative sequential strategies allocating the CPU times to them.

ASAP (Vallati, Chrpa, & Kitchin, 2013a) is a recent system for selecting the most promising planner from a set of candidates planners that derives much of its power from the use of entanglements (Chrpa & Barták, 2009; Chrpa & McCluskey, 2012). Entanglements are relations between planning operators and predicates used to reformulate the domain model by removing unpromising operator instances or restricting the applicability of some actions to certain states. A problem over the resulting modified domain can become significantly easier to solve for a planner. On the other hand, since ASAP uses an approximate method to decide entanglements, which is PSPACE-complete (Chrpa, McCluskey, & Osborne, 2012), a problem that is solvable with the original domain can become unsolvable with the reformulated domain. Given a planning domain modified by entanglements and a set of planners, ASAP identifies the most promising of these planners as the one with the highest IPC score (Jiménez, Coles, & Coles, 2011) over a set of training problems.

3. Automated Planner Portfolio Design in PbP

In this section, after introducing some preliminaries defining the problem of configuring a planner portfolio and its execution to solve planning problems, we describe the architecture and techniques of our approach to configure and execute a planner portfolio.

3.1 Preliminaries on Configuring and Executing a Planner Portfolio

Differently from most of the existing work on algorithm portfolio design of which we are aware, PbP does not design the planner portfolio for solving a specific instance of the planning problem according to the values of some predetermined features of the instance. Instead, planning problems are gathered according to their planning domains, and the planner portfolio is designed for the whole domain. The basis of this choice is the empirical observation that often there exists a single planner or a combination of planners that performs generally better for all or most of the problems of a domain. This seems something peculiar to automated planning that does not hold for other types of reasoning problems, and it makes PbP somewhat atypical in the general literature on algorithm portfolio design.

Let $\mathcal{D}$ be a planning domain, T a CPU-time limit, and $\mathcal{P}$ a set of n planners (initial portfolio), each of which with its predefined parameter values. The *problem of configuring* $\mathcal{P}$ *for* $\mathcal{D}$ consists of computing a set of triples $\{\langle P_i, M_i, S_i\rangle \,|\, i = 1 \ldots m\}$, where: $1 \leq m \leq n$, $P_i \in \mathcal{P}$, M_i is a (possibly empty) set of macro operators learned for P_i in domain $\mathcal{D}$, and S_i

is a sequence of increasing CPU times. These CPU times (real numbers) are called *planning time slots*, and are such that each time is lower than or equal to T.

The output set of triples identified by a portfolio configuration algorithm is the *configured (planner) portfolio of $\mathcal{P}$ for $\mathcal{D}$*, which in the rest of the paper will also be called a selected *planner cluster* (or simply cluster). Depending on how planners, macros and planning time slots are chosen, there can be many candidate solutions to a portfolio configuration problem. A special case, that we call the *unconfigured (planner) portfolio*, is defined as $\{\langle P_i, \emptyset, S_{pre}\rangle \mid i = 1 \dots |\mathcal{P}|\}$, where S_{pre} is predefined as $\langle 0.1, 1, 10, 100, 1000\rangle$ (in seconds).

Like BUS, PbP uses the round-robin policy for scheduling the runs of the planners in the configured portfolio. Let $\Pi = \{\langle P_i, M_i, S_i\rangle \mid i = 1 \dots m\}$ be a planner portfolio configured for a domain $\mathcal{D}$. Portfolio Π is executed to solve a planning problem in $\mathcal{D}$ by a round-robin scheduling of m processes where: each process corresponds to running a planner P_i with macros M_i ($P_i + M_i$ for short), according to an order and time slices derived from sequences $S_{1\dots m}$. More precisely, the circular order of the m planners in Π is determined by considering the m values $t_{1\dots m}$ defined by the first planning time slot in each of the m sequences $S_1 \dots S_m$. If $t_i < t_j$, P_i is ordered before P_j; if $t_i = t_j$, the relative order of P_i and P_j is arbitrarily decided (i.e., in this case P_i runs before P_j iff $i < j$), for every $i, j \in 1 \dots m$ with $i \neq j$. Each planner $P_i + M_i$ is initially run until the total CPU time allocated to this process is t_i, or the planner terminates earlier. If a planner $P_i + M_i$ does not terminate within the assigned planning time slot t_i, then it is suspended, and it resumes the next time a time slot is assigned to it. No additional CPU time is assigned to those processes that have already terminated. When, according to the circularity of the order, a planner $P_i + M_i$ resumes its execution, the total CPU time assigned to it (from the start of its execution) is equal to the next unprocessed time slot in S_i (i.e., the j-th value of S_i for the j-th time $P_i + M_i$ runs).

Figure 1 shows an example of the round-robin scheduling for portfolio $\{\langle P_1, M_1, \langle 10, 40, 160, \dots\rangle\rangle, \langle P_2, M_2, \langle 20, 60, 180, \dots\rangle\rangle\}$, assuming that $P_1 + M_1$ terminates after using 80 CPU time units, and $P_2 + M_2$ after using 120 CPU time units. $P_1 + M_1$ runs before planner $P_2 + M_2$, because the first time slot of $P_1 + M_1$ (i.e., 10) is lower than the first time slot of $P_2 + M_2$ (i.e., 20). The round-robin scheduler suspends $P_1 + M_1$ after 10 time units, and gives $P_2 + M_2$ 20 time units of CPU time. This process is repeated suspending $P_1 + M_1$ when the total execution of $P_1 + M_1$ has consumed 40 time units, and suspending $P_2 + M_2$ when the total execution of $P_2 + M_2$ has consumed 60 time units. At the next iteration, $P_1 + M_1$ should be suspended when its total execution time reaches 80 time units, but, before the end of its third time slot, i.e., at time 140, $P_1 + M_1$ terminates and needs no more CPU time. Then, $P_2 + M_2$ resumes its run, and terminates at time 200. In this example, the planners of the portfolio use only their first three time slots.

Given a set of *training problems* in a domain $\mathcal{D}$, we propose an approach to configuring an initial planner portfolio for $\mathcal{D}$ through a statistical analysis about the performance of the planners in the initial portfolio with some alternative sets of computed macros. The effectiveness of the determined configured portfolios can then be evaluated over a set of *test problems* in $\mathcal{D}$, that in our experimental analysis are disjoint from the training problem set and that, if not specified otherwise, are always formed by known benchmark problems.

The proposed approach is implemented in a planning system called PbP (*Portfolio-based Planner*). In the following, depending on the context, PbP will be used to indicate either

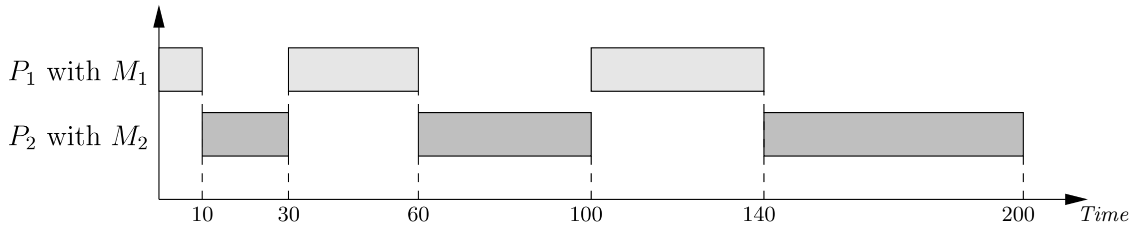

Figure 1: An example of the round-robin scheduling of PbP when running portfolio $\{\langle P_1, M_1, \langle 10, 40, 160, \ldots\rangle\rangle, \{\langle P_2, M_2, \langle 20, 60, 180, \ldots\rangle\rangle\}$ on a given planning problem, assuming that planner P_1 using macros M_1 takes a total of 80 CPU-time units to terminate and P_2 with M_2 takes a total of 120 CPU-time units.

its method for configuring the planner portfolio, or the generated configured portfolio. In the experimental evaluation of the configured portfolios generated by PbP, as a baseline planner portfolio, we will use the unconfigured planner portfolio, that will be also called the *unconfigured version of* PbP and denoted with PbP-nok (while PbP will indicate the generated configured planner portfolio).

3.2 Architecture and Components of PbP

The architecture of PbP consists of the following five main components, which are combined as described in Figure 2.

3.2.1 Macro-Actions Computation

For each integrated planner, PbP computes some sets of alternative macros using the following two approaches.

- **Wizard, PhD thesis version** (Newton et al., 2007). This system implements three learning techniques based on offline evolutionary methods, which use genetic operators to compute macros for a given planner from the plans solving a set of training problem instances of an input domain. The three learning techniques are called *chunking*, *bunching*, and *clumping*: chunking learns individual macros from the original domain operators; bunching learns bunches of macros from a given pool of macros (such as the macros learned by the chunking process); and clumping learns both individual macros and sets of macros simultaneously. The learned macros are filtered by a fitness value. The fitness value reflects some filtering criteria including the number of solved problems and the CPU time required to solve the training problems using the domain operators augmented with the learned macros. For each computed macros, if the fitness value of a macro is lower than a threshold, the macro is discarded. Therefore, for each planner incorporated into PbP (expect Macro-FF), PbP using Wizard can generate at most three sets of macros for the planner. In order to determine the sets of macros to be used in the configured portfolio, the performance of the planner will then be evaluated by PbP with/without using the sets of learned and filtered macros over the training problems. This evaluation is performed by the "Planner cluster

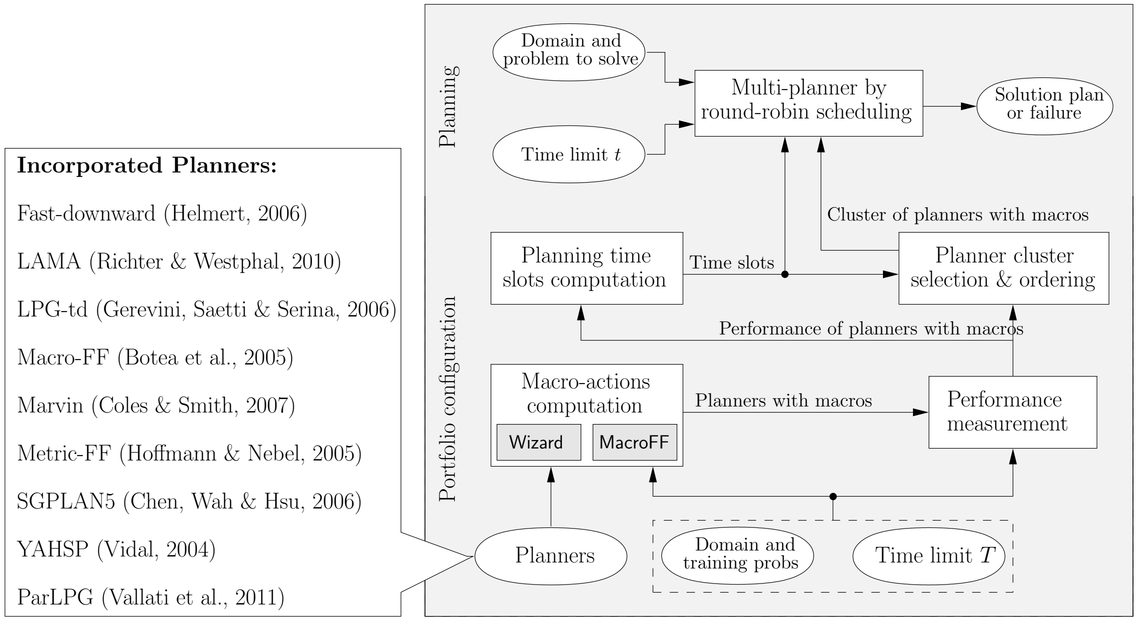

Figure 2: A sketch of PbP's architecture.

selection and ordering" component. For simplicity, the sets of learned macros will be identified by the names of the techniques used to derive them.

- **Macro-FF** (Botea et al., 2005; Botea, Müller, & Schaeffer, 2007b). The approach implemented in Macro-FF (Botea et al., 2005) computes macros by analyzing the solutions of a set of training problem instances, so that the macros that appear frequently and that significantly reduce the required search effort are preferred. In particular, first Macro-FF solves the training problems using an enhanced version of FF; then it generates macros by considering the frequency the sequences of actions forming macros appear in the computed solutions.[3] After the macro generation, Macro-FF solves the training problems using the computed macros, ranks the macros in terms of the obtained search effort gain, and using this ranking selects at most five sets of macros $M_{1..5}$, where M_i with $i = 1..5$ is the set of macros formed by the i best learned macros. The version of the approach integrated into PbP contains the enhancements described by Botea et al. (2007b). Since the macros learned by Macro-FF are coded using an ad-hoc language, in PbP the five learned sets of macros $M_{1..5}$ are used only by the Macro-FF planner.

3.2.2 Planner Performance Measurement

This is the most expensive computation step in the configuration of the portfolio. PbP runs each integrated planner expect Macro-FF with and without the three sets of macros

3. In our experiments presented in Section 4, we observed that Macro-FF computes no macros only if the enhanced version of FF solves no training problem.

learned for it by Wizard on the input training problem set, using the input CPU time limit T for each planner run. Similarly, Macro-FF runs with and without the five sets of macros learned by itself. The current implementation of PbP incorporates eight well-known successful planners, Fast Downward (Helmert, 2006), LAMA (Richter & Westphal, 2010), LPG-td (Gerevini et al., 2006), Macro-FF (Botea et al., 2005, 2007b), Marvin (Coles & Smith, 2007), Metric-FF (Hoffmann, 2003), SGPlan5 (Chen et al., 2006), YAHSP (Vidal, 2004) and a recent version of LPG (ParLPG) using a dedicated configuration phase to automatically optimize the setting of a collection of parameters governing the behavior of several parts of the system (Vallati et al., 2013b). Basically, running ParLPG consists in running LPG using a domain-specific parameter configuration. Every other incorporated planner runs using its default parameter configuration. For Marvin, this implies that during planning it can learn and memorize macros to escape from plateaus. For each run, PbP measures the planner performance in terms of: number of problems solved within T, CPU time required for solving the training problems, and quality of the computed solutions. For the incremental planners, i.e., LPG, ParLPG and LAMA, PbP measures the quality of all the solutions generated for each problem and the corresponding CPU times. Finally, note that for the macro-actions computation Macro-FF and Wizard already run the incorporated planners and hence, in principle, the performance of the planners with macros could be measured when Macro-FF and Wizard compute them. However, this has some technical difficulties and, for simplicity, PbP duplicates the runs of the (incorporated) planners.

3.2.3 Planning Time Slots Computation

The method for computing the planning time slots in PbP is a variant of the CPU-time allocation strategy proposed by Roberts and Howe (2007) for the round-robin planner scheduling. Let $\langle p_1, \ldots, p_n \rangle$ be a sequence of increasing percentages, and t_{p_i} $(i \in \{1, \ldots, n\})$ the minimum CPU time required by a planner P with a set of macros M learned for it ($P+M$ for short) in order to solve a percentage of training problems equal to p_i. During PbP's configuration of the planner portfolio, the planning time slots S of $P+M$ are defined as $S = \langle t_{p_1}, \ldots, t_{p_n} \rangle$.

The difference between the planning time slots in PbP and in the approach of Roberts and Howe can be explained by the following example. Assume that the computed planning time slots for planner A using macros M_A ($A+M_A$) are $\langle 0.20, 1.40, 4.80, 22.50, \ldots \rangle$ and that those for planner B using macros M_B ($B+M_B$) are $\langle 14.5, 150.8, \ldots \rangle$. Then, for this pair of planners, differently from the approach of Roberts and Howe, PbP extends the first time slot for $A+M_A$ (0.20) to 4.80, i.e., to the greatest time slot of $A+M_A$ which is smaller than the first time slot of $B+M_B$; similarly for the subsequent time slots. If the first time slot of $A+M_A$ were not extended, the slowest planner $B+M_B$ would initially run for a CPU time much greater than the CPU time initially assigned to the fastest planner $A+M_A$, and for many problems that planner $A+M_A$ quickly solves (e.g., using one CPU second), PbP would perform much slower. It is worth noting that using this time slot extension we observed a high performance gain only for small and easy problems.

In the rest of the paper, the sequence of increasing percentages $\langle p_1, ..., p_n \rangle$ used to define the planning time slots is called the *problem coverage percentage vector (PCPV)*. The default

PCPV in PbP is the sequence ⟨25, 50, 75, 80, 85, 90, 95, 97, 99⟩ ($n = 9$), which is the same used in the work of Roberts and Howe (2007).

3.2.4 Planner Cluster Selection and Ordering

This is the last step of the configuration process of PbP. PbP selects a cluster of planners in the initial portfolio (as described in Section 3.3), each one with a (possibly empty) set of *useful* macros, according to the measured performance and the computed planning time slots.

As for the macro selection, note that PbP has no explicit independent mechanism for selecting the macros to be used in the configured portfolio, and that macros are not shared between planners because the tools used to learn them (Wizard and Macro-FF) generate macro sets for a specific input planner. Planners and their macro sets are selected together, since *the planner cluster selection of* PbP *considers a candidate planner using two different sets of macros learned for it as two different candidate planners.*

The execution order of the planners in the selected cluster is implicitly defined by the increasing first planning time slots associated with the planners. Section 3.3 describes the planner cluster selection in detail.

3.2.5 Multi-Planner by Round-Robin Scheduling

After PbP has configured the planner portfolio for the domain under consideration, when a problem instance of this domain is encountered, PbP runs the selected ordered cluster of planners (each one using the relative selected set of macro-actions) by a round-robin scheduling algorithm using the computed planning time slots, that is similar to the one investigated in many portfolio algorithms (see, e.g., Howe et al., 1999; Roberts & Howe, 2006, 2007). Alternative planner scheduling strategies are possible, such as sequential execution or/and using configured planning time slots. However, according to the experimental results that will be presented in Section 4.8, the default round-robin strategy with the planning time slots derived from the default PCPV is robust and performs generally well. Concerning termination of the resulting multi-planner, PbP.s terminates if either a given (execution) CPU-time limit t is exceeded, returning failure, or *one* among the selected planners computes a solution (output of PbP.s); PbP.q terminates if either time t is exceeded, or *all* the selected planners terminate. If PbP.q generates no solution within t, it returns failure; otherwise, it returns the best computed solution.

3.3 Selecting a Planner Cluster

After the performance measurement and time slot computation phases, PbP analyzes the obtained results to identify the best cluster of planners and macros for the domain under consideration and the given CPU-time limit T. This is done by *simulating*, for every cluster C of at most k planners, each with a (possibly empty) set of macros, the round-robin execution of the planners in C for solving the same training problems used in the performance measurement phase.[4] The simulation is done using the data from the runs conducted for

4. In our experiments parameter k is set to 3. If k were greater than 3, we experimentally observed that for the considered benchmark domains and problems the cluster selected by PbP would be the same. The maximum number of possible combinations between the planners currently incorporated into PbP

the performance measurement phase (the planners are not re-run), ignoring the data of the planners that always perform worse than another incorporated planner (i.e., any planner that performs worse than another one across all the training problems of the domain is discarded). The CPU-time limit for each simulated execution of a cluster is T (the same time given to each run of a single planner during the performance measurement phase). The performances of the simulated cluster runs are compared by a statistical analysis based on the Wilcoxon sign-rank test (Wilcoxon & Wilcox, 1964). This test applies to a set of paired observations (a sample from a larger population), and tells us if it is plausible to assume that there is no correlation between the pairwise observed quantities. In our case, these paired observations are, e.g., the simulated runtimes of two clusters on the same training problem instance, and "no correlation" between them means it is equally likely that we will see one cluster solving a problem faster than the other as it is that we will see the opposite on a sample of problems.

For our purposes, the Wilcoxon sign-rank test is appropriate because it does not require us to know the sample distribution, and makes no assumption about this distribution. That is, we have no way to know a priori how hard a planning problem is, and hence we have no distribution of the simulated performance of the clusters. Consequently, as stated by Gibbons and Chakraborti (2003), it is critical that we use a non-parameterized test, such as the Wilcoxon sign-rank test. We have also investigated the usage of other methods to compare the performance of the simulated runs of planner clusters, including the IPC score function that was also used by Vallati et al. (2013a). However, we experimentally observed that, for the IPC7 domains, such a method is less effective than the usage of the Wilcoxon sign-rank test.

In PbP, the performance measure considers either the CPU time (PbP.s) or the plan quality (PbP.q). The data for carrying out the test in PbP.s are derived as follows. For each planning problem, the system computes the difference between the simulated execution times of the compared clusters. If a planner cluster does not solve a problem, the corresponding simulated time is twice the CPU-time limit;[5] if no cluster solves the problem, this problem is not considered. The difference between the simulated times is normalized by the value of the best simulated time under comparison (e.g., if cluster C_1 requires 200 seconds and cluster C_2 220, then the difference is 10% in favor of C_1). The absolute values of these differences are then ranked by increasing numbers, starting from the lowest value. (The lowest value is ranked 1, the next lowest value is ranked 2, and so on.) The ranks of the positive differences and the ranks of the negative differences are summed, yielding two values r_+ and r_-, respectively. If the performance of the two compared clusters is not significantly different, then the number of the positive differences r_+ is approximately equal to the number of the negative differences r_-, and the sum of the ranks in the set of the positive differences is approximately equal to the sum of the ranks in the other set. Intuitively, the test considers a weighted sum of the number of times a cluster performs better

and the considered sets of macros is 38; hence, with $k = 3$, the maximum number of clusters that can be evaluated by run simulation is $\sum_{i=1}^{i=k} \binom{38}{i} = 9177$. This is the number of clusters with at most 3 different combinations of planners and macros over the 38 in the current implementation.

5. This is the minimum value that ensures the performance gap for a problem solved by one cluster of planners and unsolved by the other compared cluster is bigger than the performance gap for any problem solved by both the compared clusters.

than the other compared one. The sum is weighted because the test uses the performance gap to assign a rank to each performance difference.

When the number of samples is sufficiently large, the T-distribution used by the Wilcoxon sign-rank test is approximately a normal distribution, which is characterized by two parameters called the *z-value* and the *p-value*. The higher the z-value, the more significant the difference of the performance is. The p-value represents the level of significance in the performance gap. If the p-value is greater than 0.05, then the null hypothesis that the performance of the compared pair of planners is statistically similar is refused, and the alternative hypothesis that their performance is statistically different is accepted. Otherwise, there is no statistically significant evidence that they perform differently, and PbP considers that they perform pretty much similarly.

The results of the Wilcoxon sign-rank test are used to form a directed graph where the nodes are the compared clusters, and an edge from a cluster C_1 to another cluster C_2 indicates that C_1 performs better than C_2. Such a graph has already been used by Long and Fox to present the results of the 3rd International Planning Competition (Long & Fox, 2003). Each strongly connected component of this graph is collapsed into a single node representing the elements in the clusters of the collapsed nodes. From the resulting DAG, PbP considers only the nodes without incoming edges (the graph root nodes). If there is only one root node, this is the selected cluster, otherwise PbP uses some secondary criteria to select the most promising cluster among the root nodes. These criteria are the number of solved problems, the sums of the ratios between the (simulated) CPU times of the planners in the compared clusters, and the first planning CPU-time slots of the involved planners. Specifically, PbP selects the cluster among the root nodes such that its simulation solves the highest number of training problems. To break the ties, for every pair of selected clusters x and y PbP computes the ratio $\frac{|s_x - s_y|}{\max\{s_x, s_y\}}$, where s_x and s_y are the sums of the (simulated) CPU times of clusters x and y, respectively; if such a ratio is greater than threshold 0.05, the compared cluster with the worst sum of CPU times is discarded. If the number of remaining clusters is still greater than one, PbP selects the cluster with the lowest first planning CPU-time slots of the involved planners. Finally, the remaining ties are broken by selecting the cluster randomly, but in our experiments no cluster has ever been randomly selected.

The method used to select a cluster of planners and macros in PbP.q is similar, but it applies to the plan qualities resulting from the cluster execution simulation, rather than to the CPU times as done by PbP.s. For this simulation, PbP.q considers also the intermediate solutions (i.e., those that are generated before the last one, which has the best quality) and the relative CPU times computed by the basic incremental planners in the considered clusters. If these solutions were ignored, the simulated plan quality for the clusters including incremental planners could be much worse than the actual quality. For example, assume that the CPU-time limit is 900 seconds, FF computes a solution with quality 50 using 100 seconds, LAMA computes two solutions with quality 20 and 19 using 120 and 880 seconds, respectively. If the intermediate solutions of LAMA were ignored, the estimated plan quality for the cluster formed by planners FF and LAMA would be equal to the quality of the plan generated by FF (the second solution generated by LAMA could be computed by the cluster using 980 seconds, but this is greater than the CPU time limit), although the intermediate (first) solution of LAMA is much better than the FF's solution.

Finally, note that if the performance of the incorporated planners is measured with CPU-time limit T, then the portfolio of PbP.s/q can be (re)configured for any time limit $t \leq T$ by simply ignoring the solutions computed after time t in the simulation of the planner cluster performance. If $t \geq T$, then $t - T$ is equally distributed among the planners in the selected cluster. If a planner terminates before its allocated time, the remaining time is also equally distributed to the other planners that are still running.

3.4 The Integrated Basic Planners

In this subsection, we give a very brief description of each of the nine basic planners that are currently incorporated in PbP. Much more detailed information is available from the corresponding referred papers.

Metric-FF (Version 2.1; Hoffmann, 2003). Metric-FF inherits the main ideas used in FF (Hoffmann & Nebel, 2001). FF's search strategy is a variation of hill-climbing over the space of the world states, and in FF the goal distance is estimated by solving a relaxed task for each successor world state. Compared to the first version of FF, Metric-FF is enhanced with goal orderings pruning techniques and with the ordering knowledge provided by the goal agenda. Moreover, it deals with level 2 of PDDL2.1 (Fox & Long, 2003), i.e., numerical state variables, numerical action preconditions and effects.

YAHSP (Version 1.1; Vidal, 2004). YAHSP extends the search procedure of FF with some information extracted from FF's relaxed plan. For each evaluated world state, YAHSP exploits a look-ahead strategy in a complete best-first search by employing actions from the relaxed plans in order to find the beginning of a valid plan that can lead to a reachable world state.

MacroFF (Botea et al., 2005, 2007b). Macro-FF extends FF with support for using macro-operators during the search, and with engineering enhancements. One of the main features of the planner version integrated into PbP is the use of *iterative macros* (Botea et al., 2007b), i.e., runtime combinations of macro operators, which are instantiated by attempting to use as many actions from FF's relaxed plan as possible. In the search procedure of FF, the iterative macros that can be successfully instantiated are considered for the generation of the next world states.

Marvin (Release 1; Coles & Smith, 2007). Marvin is another planner based on FF. The main improvement w.r.t. FF is memorizing the plateau-escaping action sequences discovered during the (local) search of FF. These action sequences form macros, which can be applied later when plateaus are once-again encountered by FF's search in order to escape from these plateaus quickly.

SGPlan (Version 5.22; Chen et al., 2006) with domain-modification script (Coles & Coles, 2011). SGPlan5 exploits a partition-and-resolve strategy to partition the mutual-exclusion constraints of a planning problem by its subgoals into subproblems, solves the subproblems individually using a modified version of the Metric-FF planner, and resolves those violated global constraints iteratively across the subproblems. It has been observed that the performance of SGPlan are affected by some rules detecting the domain name and the number of domain operators (Coles & Coles, 2011). In our work, we intend to consider the available implemented systems that have chances to perform well (possibly in combination with others) for at least one domain over a range of varied existing benchmark domains. SGPlan is definitely one of these systems. However, in order to prevent the usage of the domain-specific detection rules in SGPlan that, differently from the other planners incorporated in PbP, would make SGPlan domain-specific for some domains, we have induced SGPlan to behave domain-independently by a domain modification script, as proposed by Coles and Coles (2011). Such a script changes the domain name, adds a never-applicable action to the domain, and then runs SGPlan over the obtained domain. In addition, our domain-modification script also changes the names of domain operators.

Fast Downward (Version 1.0.1; Helmert, 2006). Fast Downward (abbreviated with FD) translates the input PDDL problem specification into its multi-valued state variable representation SAS+ (Bäckström & Nebel, 1995), and searches for a plan in the space of the world states using a heuristic derived from the *causal graph*, a particular graph representing the causal dependencies of SAS+ variables. PbP integrates the 2006 version of the planner. The main improvement compared to the earlier version of the planner that in 2004 won the propositional satisficing track of IPC4 is the addition of *safe abstraction*, a form of problem simplification that allows the planner to solve certain kinds of simple problems without search.

LAMA (Version 2008; Richter & Westphal, 2010). LAMA is built on Fast Downward, using SAS+ state variables and multi-heuristic search. Its core feature is the use of a pseudo-heuristic derived from landmarks, propositions that must be true in every solution of a planning task. Moreover, a weighted $A^\star$ search is used with iteratively decreasing weights, so that the planner continues to search for plans of better quality.

LPG-td (Gerevini et al., 2006). LPG-td inherits the main ideas used in LPG (Gerevini et al., 2003). LPG uses stochastic local search in a space of partial plans represented through *action graphs*. The search steps are certain graph modifications transforming an action graph into another one. LPG-td includes more accurate heuristics for selecting the graph modifications than those in LPG.

ParLPG (Version IPC7; Vallati et al., 2013b). ParLPG is a recent system based on the idea of automatically configuring a generic, parameterized planner using a set of training planning problems in order to obtain speed-optimized planners that perform especially well for the domains of these problems. ParLPG uses the FocusedILS variant of the off-the-shelf, state-of-the-art automatic algorithm configuration procedure ParamILS (Hutter et al., 2007, 2009), and the planning system LPG (ver. 1.0), which has several components that can be configured very flexibly via many exposed configurable parameters.

4. Experimental Analysis

In this section, we present the results of a large experimental study about PbP with the following main goals:

(G1) describing the configured portfolios and analyzing the configuration process of PbP (Section 4.2);

(G2) analyzing the efficiency of PbP.s/q in terms of speed and plan quality in the context of the planning competitions IPC6-7 (Section 4.3);

(G3) comparing the performance of the planner portfolio configured by PbP.s/q versus other planning systems based on planner portfolios (Section 4.4);

(G4) evaluating the effectiveness of using the (automatically computed) domain-specific configuration knowledge in PbP.s/q (Section 4.5);

(G5) comparing the performance of the planner portfolio configured by PbP.s/q versus the single basic planners in the portfolio, and evaluating the accuracy of the planner cluster selection in PbP.s/q (Section 4.6);

(G6) analyzing which kind of macros is selected by PbP for the planners in the configured portfolio, evaluating the effectiveness of using the selected macro set, and understanding PbP.s/q's accuracy for selecting the most useful set (Section 4.7);

(G7) investigating some possible alternative methods for scheduling the execution of the planners in the selected cluster, and understanding the effectiveness of the default round-robin strategy in PbP.s/q (Section 4.8).

This experimental study uses various versions of PbP, the most important of which are listed in Table 1. For G1, we show the CPU time of each configuration step, and we evaluate if the size of the training problem set can be important to derive effective configured portfolios. For G2, PbP is compared with the planners that entered the learning track of IPC6-7 and the winner of the deterministic track of IPC7. For G3, the performance of PbP is analyzed w.r.t. FDSS (Seipp et al., 2012) and BUS, the portfolio approach proposed by Roberts and Howe (2007). Although both BUS and FDSS propose to design domain-independent planner portfolios, in principle they can also be used, like PbP, to generate domain-optimized planning systems. We will experimentally investigate also such an use of these approaches, comparing them with PbP. For G4, we show the results of three different experimental comparisons: comparison between PbP configured using the learned domain-specific knowledge (DSK), the unconfigured version of PbP (PbP-nok) and the randomly configured version of PbP (PbP-rand); comparison of the performance gaps of PbP and PbP-nok w.r.t. the gaps of the IPC6-7 planners with/without their learned knowledge; and comparison of PbP using the DSK, PbP configured for a single domain but without using macros, and PbP configured across all IPC7 domains (PbP-allD). For G5, we have conducted three experiments in which: the performance of PbP and of each incorporated planners are compared; the performance of PbP is analyzed w.r.t. the best incorporated planner (without using macros) for every IPC7 domain; and, finally, PbP is compared with the best cluster of incorporated planners (possibly using macros) for every IPC7 domain. For G6, we compare the performance of the planners forming the clusters selected by PbP using (i) no macros, (ii) the set of macros selected by PbP, and (iii) the best performing set of macros; moreover, we show and comment some features of the sets of macros selected and used by PbP. Finally, for G7 we perform two experimental analysis: comparison of the clusters selected by PbP using some different scheduling strategies, and comparison of the performance of PbP using different PCPVs (PbP with R1-R2/S1-S2).

PbP versions	
PbP (default)	Last version of PbP configured by computing the domain-specific knowledge (DSK)
PbP-IPC6	Version of PbP that entered IPC6 configured using DSK
PbP-IPC7	Version of PbP that entered IPC7 configured using DSK
PbP-nok	Unconfigured portfolio
PbP-rand	Randomly configured portfolio
PbP-noM	Configuration without macros
PbP-allD	Configuration without macros and across all IPC7 domains
PbP with S1	Configuration using sequential scheduling of the planners with uniform time slots
PbP with S2	Configuration using sequential scheduling of the planners with *non*-uniform time slots
PbP with R1 $\equiv$ PbP	Configuration using round-robin scheduling of the planners with the default PCPV
PbP with R2	Configuration using round-robin scheduling of the planners with different PCPVs
PbP with 10/30/60	Configuration using 10/30/60 training problems

Table 1: Main variants of PbP generating different types of planner portfolio configurations used in the experimental analysis.

Before presenting and discussing the results of the experimental analysis, we describe the experimental settings.

4.1 Experimental Settings

The experiment evaluating PbP.s/q with respect to the other IPC6-7 planners considers all IPC6-7 benchmark domains (Fern et al., 2011; Jiménez et al., 2011), while the other experiments focus on the most recent IPC7 domains. Regarding the training problems used in the experiments, for the IPC6 domains they are the same as those of IPC6; for the IPC7 domains, they are a set of 540 problems of various sizes (60 problems for each IPC7 domain, unless otherwise specified for the particular experiment under consideration) that have been generated using the problem generator made available by the organizers of IPC7 (for IPC7, no explicit set of training problems was provided). The training problems are used for both learning macros and configuring the portfolio. Since the learning procedure of Wizard can run a planner over the training problems several times, in order to make the training not too much time consuming, half of the training problem set was designed to be formed by problems that took up to 30 seconds to solve by some planner; the other half is formed by problems that took up to about 450 seconds (half of the CPU time limit used in the testing phase) to solve.

Regarding the test problems, we used the same problems as those used in IPC6-7: the IPC6 test problems were used for evaluating the performance of PbP.s/q with respect to the planners that entered IPC6; the IPC7 test problems, that are generally larger and much more difficult than the IPC6 problems, were used for evaluating PbP.s/q with respect to the IPC7 planners, as well as for all other experiments in our analysis.

All our experiments have been conducted using the last version of PbP.s/q, which is not exactly the same as the one that entered and won IPC7 (PbP-IPC7 for short) for three reasons:[6] (a) PbP-IPC7 was not properly compiled because of the lack of some C-libraries on the competition machine, which was discovered only after competition; (b) PbP-IPC7 contains a very minor syntax bug about the format of the output plans that for few IPC7 domains made all generated plans invalid to the program validating them used in the competition (Howey,

6. The code of the last version of PbP is available from `http://chronus.ing.unibs.it/pbp/`.

Long, & Fox, 2004); and (c) PbP-IPC7.s uses SGPlan5 without the domain-modification script that induces SGPlan5 to behave domain-independently. Point (a) negatively affected the performance of PbP-IPC7.s/q, because one of the incorporated planners (Macro-FF) could not run when selected. For (b), many valid plans generated by PbP-IPC7.s/q were rejected by the plan validator of IPC7. Point (c) changed the composition of some clusters selected by PbP-IPC7.q that include SGPlan5, but it does not make the performance of PbP.q and PbP-IPC7.q substantially different. The only difference between the planner clusters selected by PbP-IPC7.s and those of PbP.s concerns domain `Blocksworld`, as the cluster of PbP-IPC7.s consists of ParLPG without macros, while the cluster selected by PbP.s is ParLPG using the Bunching set of macros computed by Wizard.

For the comparison with the IPC6 planners, the results of PbP.s/q were obtained by running its last version on a machine similar to (same CPU frequency and amount of RAM) the one used to obtain the official IPC6 data (an Intel Core(tm) Quad-processor Q6600 with 3 Gbytes of RAM). For the comparison of PbP.s/q and the IPC7 planners, all systems were run using the same machine of IPC7 (a Quad-core Intel Xeon 2.93 GHz with 4 Gbytes of RAM) that the IPC-organizers made available to us for this experiment. Unless otherwise specified, the other experiments were conducted using a Quad-core Intel Xeon(tm) 3.16 GHz with 4 Gbytes of RAM.

The experimental analysis required many days of CPU time. Unless otherwise indicated, as in IPC6-7, the CPU-time limit of each run of PbP.s/q was 15 minutes, PbP.s/q used the default configuration process (the CPU-time limit for each simulated execution of a planner cluster was 15 minutes), and the planners of the configured portfolio were run by the round-robin scheduling described in Section 3.2. The performance data of each planner in PbP.s/q incorporating a randomized algorithm (i.e., LPG, ParLPG and LAMA) were obtained by a single run for each considered problem instance.

The experimental comparisons with the test instances will generally use three alternative methods: the average of the performance data, the IPC7 score function (Jiménez et al., 2011), and the same Wilcoxon sign-rank test used for the planner cluster selection during configuration. Given two compared planners and a problem set, the average CPU time of each planner is computed over the problems in the set that are solved by *at least one* of the compared planners, and using the CPU-time limit (900 seconds) as the CPU time of the planner when it does not solve a problem; the average plan quality is computed over the problems solved by *both* the compared planners.

The IPC7 score function is defined as follows. Concerning planning speed, if a planner P solves a problem π using t CPU time, it gets a *time score* equal to $\frac{1}{1+\log_{10}(t/t^*)}$, where t^* is the best time over the times required by the planners under comparison for solving π. Concerning plan quality, if P generates a plan with l actions solving π, it gets a *quality score* equal to $\frac{l^*}{l}$, where l^* is the number of actions in the shortest plan over those computed by the compared planners for π. If P does not solve π, then it gets zero score (both for speed and quality). Given a domain D, the time (quality) score of planner P for D is the sum of the time (quality) scores assigned to P over all the considered test problems in D. The IPC7 score function for speed is a refinement of the IPC6 score function. Both the IPC6 and IPC7 time scores are defined according to how much slower a planner performs than the best performing one, but the IPC6 score penalizes slowdowns more heavily than the IPC7 score. For our experiments, we observed that using the IPC6 function, instead of the IPC7 function, gives similar general results that are slightly more favorable to PbP.s.

As for the Wilcoxon sign-rank test, the null hypothesis is that the performance of a compared pair of planning systems is statistically similar. The level of confidence we used is $p = 0.001$. If the analysis involves the comparison of more than two planning systems, then, in order to maintain the confidence level used when only one hypothesis is tested (i.e., only a pair of planners is compared), the confidence level has been modified by the Bonferroni's correction (Shaffer, 1995). For our analysis, the usage of the Bonferroni's correction implies that, if the experimental result we obtain by the Wilcoxon sign-rank test derives from the comparison of n planning systems, then the used confidence level is $\frac{0.001}{n-1}$. Moreover, for the plan quality comparison using the Wilcoxon sign-rank test, the quality of the plans computed by two compared planners is normalized by the length of the best plan for all the test problems solved by these planners. Since the Wilcoxon sign-rank test uses a ranking of the differences between values in each sample pair, if we compared the absolute plan length directly, without normalization, such differences in values between domains could result in an unintended bias, with small relative differences in a benchmark domain with large solution plans weighted as more important than larger relative differences in a domain with small plans.

4.2 Overview of the Configured Portfolios Generated by PbP

This section concerns experimental goal G1: we give some information about the configured portfolios (multi-planners) generated by the default version of PbP.s/q (see Table 1), the relative CPU times used for the automated portfolio configuration, and the size of the training problem set used for configuring PbP. Table 2 shows the planners in the clusters selected by PbP for every IPC6-7 domain. For each planner in the cluster, the table also indicates in brackets the sets of macros selected by PbP, which are available from `http://chronus.ing.unibs.it/pbp` (the computed planning time slots in the clusters are omitted for brevity and clarity). For example, for Depots, PbP.q selects the cluster formed by (i) Macro-FF with the two learned macros that most frequently appear in the Macro-FF's plans solving the training problems, (ii) ParLPG without any of the computed macros, and (iii) SGPlan5 with the set of macros obtained by the chunking macro generation method of Wizard. From the configured portfolios in Table 2 we can derive the following observation:

Experimental result 4.2.1 *The planner clusters selected by* PbP *often are formed by different sets of planners and macros: overall all nine basic planners are helpful (each of them is selected by* PbP.s/q *at least once), and different sets of macros are considered more helpful than others, including, in few cases, the empty set.*

Concerning planning speed, we observe that for most domains PbP.s relies on a single planner possibly using a set of macros. In particular, for 7 of the 15 considered domains ParLPG outperforms the other incorporated planners, and hence for these domains the selected cluster contains only ParLPG. The main reason of the better performance of ParLPG is that it uses LPG with a parameter configuration that is (automatically) optimized for every considered domain, and this can greatly speedup the planner (Vallati et al., 2013b).

Domains	PbP.s	PbP.q
IPC6 domains		
Gold-miner	YAHSP (Cl)	Macro-FF (M1), LAMA (B), LPG (0)
Matching-BW	ParLPG (–)	Marvin (0), LAMA (–), LPG (B)
N-puzzle	ParLPG (–)	Fast Downward (–), LAMA (–), LPG (0)
Parking	Macro-FF (M2)	FF (0), LAMA (–)
Sokoban	ParLPG (–)	Macro-FF (M2), LPG (B)
Thoughtful	FF (–), YAHSP (–)	Macro-FF (M5), Marvin (–), LAMA (–)
IPC7 domains		
Barman	SGPlan5 (B)	SGPlan5 (Cl), FF (–), LAMA (–)
Blocksworld	ParLPG (B)	ParLPG (–), LPG (B)
Depots	Macro-FF (M2), ParLPG (0)	Macro-FF (M2), ParLPG (0), SGPlan5 (Ch)
Gripper	ParLPG (–)	Marvin (–), ParLPG (–)
Parking	Macro-FF (M2)	FF (0), LAMA (–)
Rovers	ParLPG (–)	LAMA (–), ParLPG (–)
Satellite	ParLPG (–)	ParLPG (–), Marvin (0)
Spanner	ParLPG (–)	LPG (–)
TPP	Macro-FF (M1)	LAMA (–), SGPlan5 (Ch)

Table 2: Planners and sets of macros (in round brackets) in the cluster selected by PbP for the IPC6-7 domains. "–" and "0" indicate that no macros was generated and selected, respectively; "Ch", "B" and "Cl" abbreviate the three sets of macros *Ch*unking, *B*unching and *Cl*umping generated by Wizard, respectively; M1–M5 are the five sets of macros generated by Macro-FF. The order of the planners listed in the clusters corresponds to the order in which they run.

We observed that, in a previous version of PbP that entered IPC6 without ParLPG, the selected clusters were even more varied.

It is interesting to observe that when PbP selects Macro-FF for the configured portfolio this planner always uses a non-empty set of macros. The fact that in the selected cluster Macro-FF always uses one among the learned sets of macros indicates that the macro construction and exploitation methods incorporated into Macro-FF are effective for this planning system.

Table 3 gives the CPU times used by PbP.s for the different phases of the portfolio configuration applied to the IPC7 domains, for which a machine with a Quad-core Intel Xeon(tm) 3.16 GHz and 4 Gbytes of RAM was used.[7] The configuration times of PbP.q are similar to those of PbP.s for the macro extraction and cluster simulation phases, while they are higher for the performance measurement, because the incorporated incremental planners can use the whole CPU-time limit in order to find good quality plans. Although configuring PbP for a specific domain requires a considerable amount of CPU time, it should be considered that such a configuration needs to be done only once, since the generated configured portfolio (selected planner cluster) can be used for all the problems in the domain.

Finally, in order to understand if small sets of training problems can be sufficient to derive informative DSK for test problems that are larger than the training ones, we have

7. For every IPC7 domain, the parameter configuration of ParLPG required about 1400 hours.

IPC7 Domains	Macro Extraction Macro-FF	Macro Extraction Wizard	Performance Measure	Simulation & Selection	Total
Barman	37.5	28.5	121.6	0.02	187.6
Blocksworld	16.4	44.2	92.7	0.02	153.4
Depots	7.1	82.9	92.4	0.02	182.4
Gripper	45.3	12.9	96.8	0.02	155.0
Parking	23.8	86.5	163.0	0.02	273.3
Rovers	18.2	0.0	57.3	0.02	75.6
Satellite	14.1	41.3	60.0	0.02	115.4
Spanner	5.25	0.8	110.7	0.02	116.8
TPP	19.3	3.9	34.5	0.02	57.8

Table 3: CPU hours used by the configuration of PbP.s for the IPC7 domains: extraction of macros with Macro-FF and Wizard (2nd and 3rd columns), performance measurement phase (4th column), cluster run simulation and best cluster selection (5th column), total configuration time (6th column).

compared the performance of PbP configured using the default number of 60 training problems and using half and one-sixth of these training problems. (The range of the problem size is the same for each of the three sets of training problems.) The results of this analysis are in Table 4. Of course, the lower the number of training problems is, the cheaper the training of PbP is. On the other hand, the DSK computed using few training problems can sometime be much less effective and informative than the DSK obtaining using larger sets.

For Depots, PbP.s with the DSK derived from 60 training problems performs much better than with the DSKs derived from 30 and 10 training problems; for all the other domains, the performance of PbP.s with the three compared DSKs is similar or the same. It is interesting to observe that Depots is the only domain for which the cluster of PbP.s has two planners. For this domain, the cluster of PbP.s derived from 60 training problems consists of Macro-FF and ParLPG: for 16 training problems ParLPG hands a solution to PbP.s, while for the other 44 training problems the solution of PbP.s is obtained by Macro-FF. If the DSK is derived from 30 or 20 training problems, either Macro-FF or ParLPG is not part of the configured cluster of PbP.s and this makes PbP.s performing worse.

For Depots, Satellite and TPP, PbP.q with the DSK derived from 60 training problems performs much better than with the DSK derived from 30 or 10 training problems. For all the other domains, the performance of PbP.q is similar or the same.

4.3 Performance of PbP and the IPC6-7 Planners

This section concerns experimental goal G2: we experimentally evaluate the performance of PbP in the context of IPC6-7 with the aim of showing that it is competitive with other recent planning systems using domain specific learned knowledge. Since at the time of writing several IPC6-7 planners and the relative domain specific knowledge are not available, for this experiment we used the official competition data (CPU times, plan qualities and number of solved problems) and the results we obtained by running the last version of PbP.

IPC7 Domains	Time score 60	30	10	Mean CPU time 60	30	10	# solved problems 60	30	10
Depots	26.0	2.6	2.6	31.9	312.5	312.5	26	4	4
Parking	7.4	4.9	5.8	172.8	127.7	281.3	8	5	7
All domains	33.4	7.0	8.4	110.2	209.8	292.6	34	9	11

IPC7 Domains	Quality score 60	30	10	Mean plan length 60	30	10	# solved problems 60	30	10
Blocksworld	29.9	29.6	29.6	269.9	272.8	272.8	30	30	30
Depots	24.7	8.6	9.2	151.7	156.2	151.2	26	10	10
Parking	4.8	2.8	2.0	76.0	61.0	61.0	5	3	2
Satellite	29.6	0.0	27.8	–	–	–	30	0	28
TPP	14.8	7.7	0.0	–	–	–	15	8	0
All domains	133.8	78.7	98.6	325.3	326.6	326.0	136	81	100

Table 4: Time/quality score, average CPU time/plan length and number of solved problems of PbP.s/q configured with DSK computed by using a set of either 60 (default version of PbP), 30 or 10 training problems. The domains considered are the IPC7 domains for which the training phase of PbP.s/q derives different DSKs for training problem sets with different sizes.

In the learning track of IPC6 and IPC7, the competing teams were not aware of the domains used for the evaluation before submitting their systems. After code submission, the contest had two phases. In the first phase, the domains were released and the learning parts of the planners were run to automatically derive, for each domain, some additional knowledge using a set of training problems in the domain. In the second phase, after submitting the learned knowledge to the IPC organizers, the planners were run with the relative learned knowledge, and the resulting performance data were compared using the IPC score function. The interested reader can find more details about the IPC6-7 organization as well as a collection of short papers describing the IPC6-7 planners that entered the learning track in the work of Fern et al. (2011), and Jiménez et al. (2011).

For PbP, the knowledge derived in the first phase of the competition is the portfolio configuration knowledge described in the previous section of the paper. The knowledge learned by IPC6 planner ObtuseWedge consists of some special patterns, that extend the notion of an “n-gram” to include argument relations, and are used with the aim of speeding up the enforced hill-climbing search (Yoon, Fern, & Givan, 2008). The IPC6 planning systems Wizard+FF and Wizard+SGPlan learn a set of macro actions for planners FF and SGPlan5, respectively. As for the IPC7 planners, Bootstrap-Planner learns a domain-specific heuristic by combining a set of existing heuristics with weights obtained by evaluating the performance of the heuristics on the training problems (Arfaee, Zilles, & Holte, 2010). Finally, the knowledge learned by OALDAEYASHP (Brendel & Schoenauer, 2011), ParLPG, Fast Downward-autotune-speed and Fast Downward-autotune-quality (Fawcett et al., 2011) consists of domain-specific parameter configurations.

Table 5 gives an overall experimental evaluation of the best-performing planners in IPC6 (using the IPC6 domains) and of the best-performing planners in IPC7 (using the

Best IPC6 planners	Problem Solved (%)	Time score (max = 180)	Quality score (max = 180)
PbP.s	**97.0**	**105.5**	111.2
PbP.q	95.0	6.89	**169.1**
ObtuseWedge	65.0	73.5	75.6
PbP-IPC6.s	95.6	88.2	111.4
PbP-IPC6.q	92.8	6.43	132.3
RFA1	47.2	14.7	52.3
Wizard+FF	56.7	47.3	72.4
Wizard+SGPlan	51.1	56.1	65.2

Best IPC7 planners	Problem Solved (%)	Time score (max = 270)	Quality score (max = 270)
PbP.s	**87.4**	**232.7**	202.5
PbP.q	83.7	76.2	**221.7**
Bootstrap-Planner	4.07	3.28	10.93
Fast Downward-autotune-speed	77.0	110.0	170.8
Fast Downward-autotune-quality	32.2	35.3	64.3
OALDAEYASHP	7.41	5.70	3.76
ParLPG	57.04	104.0	146.0
PbP-IPC7.s	71.48	178.1	172.5
PbP-IPC7.q	70.37	71.1	192.7
LAMA-2011	37.67	37.9 (1st sol.)	82.4 (last sol.)

Table 5: Percentage of solved problems within 15 CPU minutes, and time and quality scores of PbP.s/q and the (best performing) planners that took part in the learning track of IPC6-7 for the domains and problems of IPC6-7. Larger scores indicate better performances. PbP-IPC6 and PbP-IPC7 indicate the versions of PbP that took part in IPC6 and IPC7, respectively; LAMA-2011 is the winner of the deterministic track of IPC7.

IPC7 domains), in terms of percentage of solved problems, planning speed and plan quality. All compared planners were run with the relative learned knowledge. From the data in Table 5, the following general experimental result can be derived.

Experimental result 4.3.1 *For the IPC6-7 domains and problems,* PbP.s *is generally faster than the compared IPC6-7 planners,* PbP.q *performs generally better in terms of plan quality, and* PbP.s/q *solves many more problems.*[8]

Remarkably, PbP.s/q solves a very high percentage of the IPC6-7 benchmark problems within 15 CPU minutes, and PbP.q almost always computes a plan that is better than the plan computed by any other competitor. In contrast, the time score of PbP.q is low, since

8. The version of PbP used for the comparison does not suffer the technical problems indicated in Section 4.1 that affected the performance of PbP at IPC7. At IPC7 other planners may have suffered similar problems, and their implementation might also have improved versions which we have not considered. However, we note that even the version of PbP.s/q that entered IPC7 performs generally better than the other competing planners.

PbP.q usually runs more than one planner and stops only when all the selected planners terminate or the CPU-time limit is exceeded.

An analysis of the competition results (planner CPU times and plan qualities) using the Wilcoxon sign-rank test instead of the IPC score functions for the performance comparison confirms that PbP.q generates significantly better quality plans ($z = -3.920$, $p < \frac{0.001}{7}$). The p-value obtained by this analysis is 0.004 (with z-value equal to -2.846). Since the p-value is not below the adjusted critical value of $\frac{0.001}{7}$, the null hypothesis (the performance of PbP is similar to the performance of the other IPC6-7 planners in terms of speed) is accepted, and thus the research hypothesis (the performance of PbP is statistically different) is rejected. However, it is worth pointing out that the critical value of 0.001 is quite hard to reach, especially given that we also apply an experiment-wise error adjustment. If we had set a less stringent critical value, say 0.05, then the adjusted critical value would be $\frac{0.05}{7} = 0.0071$, and p-value of 0.004 would be significant.

Table 6 gives details about the performance comparison for each IPC7 domain. In terms of speed, PbP.s has the best performance in eight out of the nine domains considered in the analysis; the only domain where it does not perform best is `Parking`, where Fast Downward-Autotune-speed performs better. Similarly, in terms of quality, PbP.q has the best performance in seven out of nine domains, it performs as well as ParLPG and PbP.q in one domain (`Spanner`), and it performs worse than Fast Downward-Autotune-speed in two domains (`Parking` and `TPP`). It is worth noting that in principle a portfolio approach should incorporate the planners most promising for attempting the problems of a domain. The current version of PbP integrates the planners that have established the state-of-the-art when PbP was developed, and at that time Fast Downward-Autotune-speed was not available. The results in Table 6 indicate that our portfolio-based approach would reach better performance, if it also incorporated such a planner. For instance, it is likely that PbP would select this planner for domain `Parking`, greatly improving its performance for this domain.

Finally, we comment on the relative performance of PbP and the winner of the deterministic *satisficing* track of IPC7, the 2011 version of LAMA. Of course, we cannot expect that a domain-independent planner, such LAMA, performs better than a planner exploiting (learned) specific domain knowledge. On the other hand, it is definitely a desired property that the other way around holds: a planning system that uses some form of (automatically acquired) domain specific knowledge is effective only if it performs better than a state-of-the-art domain-independent planner that does not use such additional knowledge.

The last lines of Tables 5 and 6 indicate the global and domain-by-domain performance of LAMA-2011 with respect to the planners of the learning track of IPC7, considering the score functions of this competition track.[9] For this comparison, the CPU time limit used to run LAMA is 15 minutes, the same time limit as the one used to run PbP.s/q and the other planners that took part in the learning track of IPC7. It is worth noting that the IPC7 domains of the learning track are propositional, and the IPC7 problems do not require the optimization of an explicit specified plan metric; for these problems, both LAMA and PbP minimize the number of actions. It can be seen that PbP.s/q performs substantially

9. Although the experimental comparison considers both planning time scores and plan quality scores, it should be noted that the deterministic track of IPC7 focused on plan quality, and hence LAMA-2011 has presumably been developed focusing on quality rather than speed. In this sense, the results about plan quality in our comparison with LAMA-2011 are more meaningful than those about planning speed.

IPC7 Planners	Solved problems								
	Barman	BW	Depots	Gripper	Parking	Rovers	Sat	Spanner	TPP
Bootstrap-Planner	0	11	0	0	0	0	0	0	0
FDA-speed	**30**	29	20	**30**	**20**	**30**	19	0	**30**
FDA-quality	0	27	0	1	9	**30**	7	0	14
OALDAEYASHP	0	20	0	0	0	0	0	0	0
ParLPG	0	**30**	17	**30**	0	27	**30**	**30**	15
PbP-IPC7.s	29	29	26	**30**	0	27	**30**	**30**	0
PbP-IPC7.q	**30**	**30**	10	**30**	5	**30**	**30**	**30**	0
PbP.s	**30**	**30**	**26**	**30**	8	27	**30**	**30**	25
PbP.q	**30**	**30**	**26**	**30**	5	**30**	**30**	**30**	15
LAMA-11	2	29	0	0	5	**30**	13	0	20

IPC7 Planners	Time score								
	Barman	BW	Depots	Gripper	Parking	Rovers	Sat	Spanner	TPP
Bootstrap-Planner	0.0	3.28	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FDA-speed	14.3	12.1	10.9	9.84	**18.7**	13.5	6.39	0.81	23.5
FDA-quality	0.0	9.31	0.0	0.55	5.67	11.4	1.92	0.51	5.96
OALDAEYASHP	0.0	5.70	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ParLPG	0.0	16.3	9.18	15.3	0.0	17.5	17.7	16.2	11.9
PbP-IPC7.s	**28.8**	17.3	**25.1**	**30.0**	0.0	**27.0**	**30.0**	**30.0**	0.0
PbP-IPC7.q	22.9	8.27	3.05	8.47	2.62	8.21	10.2	7.34	0.0
PbP.s	**28.8**	**30.0**	**25.1**	**30.0**	7.14	**27.0**	**30.0**	**30.0**	**24.8**
PbP.q	22.9	8.27	7.95	8.47	2.62	8.21	10.2	7.34	7.37
LAMA-11 (1st sol.)	0.52	10.6	0.0	0.0	2.8	10.0	3.5	0.0	10.5

IPC7 Planners	Quality score								
	Barman	BW	Depots	Gripper	Parking	Rovers	Sat	Spanner	TPP
Bootstrap-Planner	0.0	3.76	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FDA-speed	26.9	13.2	20.0	28.8	**17.2**	24.2	15.7	0.0	**24.8**
FDA-quality	0.0	13.3	0.0	0.0	9.00	22.8	6.67	0.0	12.6
OALDAEYASHP	0.0	10.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ParLPG	0.0	21.7	8.31	28.6	0.0	21.4	28.5	**30.0**	7.54
PbP-IPC7.s	28.4	24.1	16.8	27.5	0.0	19.3	26.5	**30.0**	0.0
PbP-IPC7.q	**30.0**	**29.8**	9.01	**29.9**	4.09	**30.0**	**30.0**	**30.0**	0.0
PbP.s	28.4	21.1	16.8	27.5	9.08	19.3	26.5	**30.0**	23.9
PbP.q	**30.0**	**29.8**	**23.0**	**29.9**	4.09	**30.0**	**30.0**	**30.0**	14.9
LAMA-11 (last sol.)	1.86	21.8	0.0	0.0	3.8	24.7	11.0	0.0	19.3

Table 6: Number of solved problems, and time/quality scores of the (best performing) IPC7 planners for each IPC7 domain. FDA, LAMA-11, BW and Sat abbreviate Fast Downward-Autotune, LAMA-2011, Blocksworld and Satellite, respectively.

better than LAMA-2011. The results in Table 5 show that PbP.s/q solves many more IPC7 problems, and it achieves considerably better overall time and quality scores with respect to LAMA-2011's first and best quality solutions, respectively. The results in Table 6 show that: PbP.s has a much higher speed performance for every domain, and a much higher quality performance for most of the domains; PbP.q has a much higher quality performance for seven domains, while it performs similarly for the other two domains, and it has a much higher speed performance for most of the domains.

Moreover, since in the deterministic track of IPC7 the CPU-time limit was 30 minutes, we compared LAMA-2011 and PbP.s/q over the problems of the learning track using this limit for the first planner, but keeping 15 CPU minutes for the second. The extra CPU time for LAMA-2011 does not considerably change the results of the comparison: overall, the total time scores of LAMA-2011 and PbP.s/q are 61.9 and 231.9/116.5, respectively; the total quality scores of LAMA-2011 and PbP.s/q are 80.7 and 227.6/206.2, respectively; LAMA-2011 solves 101 problems while PbP.s/q solve 238/230 problems.

The previous experimental analysis of PbP.s/q and LAMA-2011 is summarized in the following claim, suggesting that if a portfolio-based planner is (automatically) configured for a given domain, it can perform much better than a state-of-the-art fully domain-independent planner.

Experimental result 4.3.2 *For the benchmark domains of the learning track of IPC7, the configured versions of* PbP.s/q *perform better than the IPC7 winner of the deterministic track.*

Since PbP *without* configuration knowledge (PbP-nok) is a fully domain-independent planner, it is also interesting to see how well PbP-nok performs w.r.t. LAMA-2011. For this experimental comparison, we also used the benchmark domains and problems of the deterministic track of IPC7, with the same CPU-time limit of IPC7 for each run (30 minutes). Moreover, since the deterministic track of IPC7 focused on plan quality, measured as total action cost, we considered only the quality version of PbP-nok. While LAMA-2011 optimizes total action cost, PbP-nok.q and the incorporated planners consider number of actions for plan quality. Although our analysis relies on the total action cost, and hence is somewhat in favor of LAMA-2011, we observed that PbP-nok.q is competitive with LAMA-2011. For the problems of the IPC7 deterministic track, the total quality score and number of solved problems are slightly lower for PbP-nok.q than for LAMA-2011 (214.8 against 232.2, and 239 against 250, respectively). The lower quality score of PbP.q is mainly because of two of the fourteen IPC7 domains (`Elevator` and `Parcprinter`), where PbP-nok.q obtains much lower scores (1.0 against 19.0 and 3.0 against 19.6, respectively). For the test problems of the learning track of IPC7, PbP-nok.q performs even better than LAMA-2011 (IPC quality score: 168.8 versus 97.5; solved problems: 181 versus 105).

Experimental result 4.3.3 *For the benchmark domains of the deterministic and learning tracks of IPC7,* PbP.q *without configuration knowledge (*PbP-nok.q*) is competitive with the winner of the IPC7 deterministic track.*

Given that PbP.q without configuration performs already well, a performance improvement obtained by exploiting the computed configuration knowledge is even more notable. Section 4.5 shows that the portfolio configuration of PbP.s/q is very useful to improve performance.

4.4 Performance of PbP and Other Planner Portfolios

This section concerns experimental goal G3: we compare PbP with two planner portfolio approaches: FDSS (Helmert et al., 2011) and BUS (Roberts & Howe, 2007).

4.4.1 PbP versus FDSS

Table 7 shows the performance of PbP.s/q w.r.t. FDSS with and without using macros.[10] The results of this comparison can be summarized as follows.

Experimental result 4.4.1 *For the benchmark domains of the learning track of IPC7, in terms of number of solved problems* PbP.s/q *performs always better than* FDSS*, except for domains* Rovers *and* TPP*, where* FDSS *solves few problems more than* PbP.s *and* PbP.q*, respectively. In terms of time score,* PbP.s *always performs better than* FDSS*. In terms of quality score,* PbP.q *performs always better except for* TPP*.*

We think there are at least four reasons why in our experiments PbP performed generally better than FDSS. The main reason is that, while PbP is separately configured for every considered domain, FDSS always uses the same configuration determined from the problem instances of IPC1–6, that were designed using problem distributions quite different from those of the learning track of IPC7 (Seipp et al., 2012). Other reasons are (a) the diversity of the planning methods implemented in the planners incorporated into PbP and FDSS, (b) the usage of macros in PbP.s/q, and (c) the different portfolio configuration techniques of the two compared systems. Concerning (a), if we consider for instance domain Spanner, PbP.s/q outperforms FDSS because PbP's configured portfolios use ParLPG/LPG (see Table 2). While every planner incorporated into FDSS uses heuristic forward search techniques, ParLPG/LPG uses heuristic techniques searching a space of partial plans, which seems more effective for this domain. As for (b), we tried to learn macros for FDSS using Wizard, but unfortunately no useful macro was learned for this planning system. Therefore, we tested the performance of FDSS using the same macros learned by Wizard and selected by PbP.s/q for the planners in the configured portfolios (see Table 2). The results in Table 7 indicate that, while using these macros sometimes greatly improves the performance of PbP, they are not really effective for FDSS.

Finally, in order to better understand the importance of (c), we also developed and compared with PbP a new variant of FDSS, called FDSS^d, restricting the differences between FDSS and PbP to their configuration techniques. Specifically, FDSS^d has the following similarities and differences w.r.t. the original FDSS. While FDSS^d uses the same configuration techniques of FDSS, it configures the planner portfolio separately for each input domain (instead of for a set of domains altogether), uses macros, and integrates the same planners as PbP (instead of a set of forward-state planners). Then, the most important differences between PbP and FDSS^d are the method for the planner cluster selection and the scheduling strategy used for running the planners forming the clusters that, as described in Section 2, are substantially different.

Like for PbP, we computed two sets of domain-optimized portfolio configurations of FDSS^d: FDSS.s^d focusing on speed, and FDSS.q^d focusing on plan quality. For all the IPC7 domains except Depots, the planner clusters selected by FDSS.s are the same as those of PbP.s. For Depots, the cluster of FDSS.s consists of Macro-FF using macro set M1 and Macro-FF using macro set M2, while the cluster of PbP.s consists of ParLPG using no macro and Macro-FF using macro set M2. For all domain in which FDSS.s and PbP.s have the

10. The version of FDSS that was run in this experiment uses 15 of the 38 variants of Fast Downward analyzed by Helmert et al. (2011).

Planners	# solved problems									
	Barman	BW	Depots	Gripper	Parking	Rovers	Sat	Spanner	TPP	Total
PbP.s	**30**	**30**	**26**	**30**	**8**	27	**30**	**30**	**25**	**236**
PbP.q	**30**	**30**	**26**	**30**	5	**30**	**30**	**30**	15	226
FDSS	0	21	0	0	0	28	14	0	17	80
FDSS+M	0	10	0	0	0	28	14	0	17	69
PbP-nok.s	23	17	8	24	0	20	11	13	7	123
PbP-nok.q	23	18	8	25	0	17	11	13	10	125

Planners	Time score									
	Barman	BW	Depots	Gripper	Parking	Rovers	Sat	Spanner	TPP	Total
PbP.s	**30.0**	**30.0**	**26.0**	**30.0**	**7.8**	**27.0**	**30.0**	**30.0**	**23.7**	**234.5**
PbP.q	23.0	8.27	8.03	8.47	2.54	8.21	10.2	7.3	7.1	83.2
FDSS	0.0	9.7	0.0	0.0	0.0	17.1	8.8	0.0	8.5	44.1
FDSS+M	0.0	3.6	0.0	0.0	0.0	17.1	8.8	0.0	9.0	38.5
PbP-nok.s	6.7	7.1	5.1	10.3	0.0	9.8	4.7	5.3	3.2	52.2
PbP-nok.q	6.2	5.4	3.0	7.9	0.0	5.8	4.1	5.3	3.8	41.5

Planners	Quality score									
	Barman	BW	Depots	Gripper	Parking	Rovers	Sat	Spanner	TPP	Total
PbP.s	28.4	21.3	17.2	27.6	**5.4**	18.2	26.3	**30.0**	**20.3**	194.7
PbP.q	**30.0**	**29.9**	**26.0**	**30.0**	5.0	**27.9**	**29.5**	**30.0**	14.2	**222.5**
FDSS	0.0	13.4	0.0	0.0	0.0	24.9	12.2	0.0	16.1	66.6
FDSS+M	0.0	12.9	0.0	0.0	0.0	24.9	12.2	0.0	15.4	65.4
PbP-nok.s	22.5	10.3	7.6	18.1	0.0	18.7	8.9	13.0	6.3	105.4
PbP-nok.q	22.5	12.3	7.4	19.9	0.0	15.8	9.8	13.0	9.3	110.0

Table 7: Number of solved problems, and time/quality scores of PbP, PbP-nok, FDSS with/without using macros for each IPC7 domain. FDSS+M, BW and Sat abbreviate FDSS using macros, Blocksworld and Satellite, respectively.

same cluster, the cluster is formed by a single planner. Hence, running it by the sequential scheduling and by the round-robin scheduling is the same thing, and the compared planner portfolios have the same performance.

The planner clusters selected by FDSS.q are in Table 8. For domains Gripper, Satellite and TPP, they are the same as those of PbP.q but, in these cases they are formed by more than one planner. For domains Satellite and Gripper we observed that FDSS.q performs differently from PbP.q because of the different scheduling strategy. Table 9 shows the results of the experimental comparison between PbP and FDSS^d (results are omitted when the compared clusters are the same and they are formed by a single planner). Overall, we can derive the following observation.

Experimental result 4.4.2 *For almost all the benchmark problems and domains of the learning track of IPC7,* PbP.s *is as fast as* FDSS.s^d*, and for* Depots *it is slightly faster;* PbP.q *computes plans that are always as good as or better than those computed by* FDSS.q^d*, and solves more problems.*

The performance gap between PbP and FDSS^d is lower than the gap between PbP and FDSS, but for Depots PbP.s performs slightly better in terms of speed and number of solved problems, and over all the IPC7 domains PbP.q performs considerably better in terms of plan quality. A rationale for this behavior is that, as we will show in Section 4.6,

IPC7 Domain	FDSS.q^d
Barman	S (Cl), FF, L, M
Blocksworld	M (B), FF (Ch), MFF (M1), FF (B),P, L, LPG (Ch), LPG (B)
Depots	LPG, M, FF (Ch), MFF (M1), FF (Cl), S (Ch), S (Cl), P, MFF (M2), L
Gripper	M, P
Parking	FF (Ch), FF, L
Rovers	FF, MFF (M1), L, LPG
Satellite	P, M
Spanner	P
TPP	L (–), S (Ch)

Table 8: Planners and sets of macros (in round brackets) in the cluster selected by FDSS^d for the IPC7 domains. S, L, M, MFF and P abbreviate SGPlan5, LAMA, Marvin, Macro-FF and ParLPG, respectively; “Ch”, “B” and “Cl” are the three sets of macros *Ch*unking, *B*unching and *Cl*umping generated by Wizard; M1–M5 are the five sets of macros generated by Macro-FF.

running planner clusters by a round-robin scheduling can be more robust than running them sequentially using possibly inadequate values of planning time slots. Another explanation, especially for the high performance difference in terms of plan quality, is the different way in which PbP and FDSS^d explore their portfolio configuration spaces. FDSS^d searches the planner cluster to use by a hill-climbing algorithm over the space of possible clusters, while PbP explores the whole space of possible clusters (with a bound on the number of planners in clusters). The selected clusters of PbP.s and FDSS.s^d are almost always the same because for the IPC7 domains and the considered training problems configuring the planner portfolio focusing on speed is quite easy, as in most cases a single planner (possibly using macros) outperforms every other planner. On the contrary, for these domains and the training problems, configuring the planner portfolio focusing on plan quality is more difficult for FDSS^d, because its search space contains local minima that prevent FDSS^d from finding the best-performing configuration (planner cluster), while a complete exploration of the search space allows PbP to identify it.

It is worth noting that the space of the planner clusters of PbP is much smaller than the spaces of FDSS and FDSS^d, since in the space of PbP there cannot be two different clusters formed by the same planners and the same relative macros, but different relative sequences of planning time slots (the sequence of planning time slots for a planner with the relative set of macros is derived according to the default PCPV). If this were not the case, the space of the clusters of PbP would be orders of magnitude greater, and the time required by PbP for simulating the cluster execution would not be negligible w.r.t. the time for the other configuration phases (see Table 3).

The performance comparison of PbP.s and FDSS^d using the Wilcoxon sign-rank test gives a statistical result that is compatible with the performance data in Table 9 and Experimental result 4.4.2: over all the IPC7 domains, there is no statistical difference between the planning CPU times of PbP.s and FDSS.s^d ($z = -1.257$, $p = 0.209$), while,

IPC7 Domains	Max score	Time score		Mean CPU time		# solved problems	
		PbP.s	FDSS.s^d	PbP.s	FDSS.s^d	PbP.s	FDSS.s^d
Depots	30	22.2	20.1	53.1	126.3	26	23

IPC7 Domains	Max score	Quality score		Mean plan length		# solved problems	
		PbP.q	FDSS.q^d	PbP.q	FDSS.q^d	PbP.q	FDSS.q^d
Barman	30	30.0	30.0	448.3	448.3	30	30
Blocksworld	30	30.0	14.4	233.5	340.2	30	20
Depots	30	24.2	19.2	159.9	169.1	26	21
Gripper	30	30.0	23.7	574.0	581.7	30	24
Parking	30	4.8	2.8	70.6	71.7	5	3
Rovers	30	29.8	17.3	580.3	600.2	30	18
Satellite	30	30.0	25.0	775.2	775.2	30	25
Spanner	30	30.0	30.0	326.0	326.0	30	30
TPP	30	15.0	15.0	370.1	370.1	15	15
All domains	270	223.8	177.4	433.1	448.7	226	186

Table 9: Maximum score, time/quality score, average CPU time/plan length and number of solved problems of PbP and FDSS^d on benchmark problems from Depots for planning speed, and from all the IPC7 domains for plan quality.

in terms of plan quality, PbP.q performs significantly better than FDSS.q^d ($z = -5.767$, $p < 0.001$).

4.4.2 PbP versus BUS

Although BUS was originally designed to generate a domain-independent configured planner portfolio, like FDSS, in principle it can also be used to build domain-specific configured portfolios. Domain specificity can be obtained simply by having all the training problems over the same domain. A fully-automated executable of BUS is not available, as the experimental results presented by Roberts and Howe (2007) were derived by simulation (Roberts & Howe, 2012). Thereby, in order to compare PbP and BUS, we implemented the BUS approach using the same planners and macros integrated into PbP, and we generated domain-specific configured portfolios using this implementation of BUS.

BUS selects the planners for the configured portfolio through a greedy set covering approximation algorithm over the sets of problems solved by the incorporated planners, and then the planners forming the clusters are ordered according to the ranking algorithm by Simon and Kadane (1975). The greedy set covering approximation algorithm iteratively selects a planner and reduces the set covering problem into a smaller one, until the original input set is fully covered (Cormen, Stein, Rivest, & Leiserson, 2001). Let D be the planning domain, P the set of selected planners, and S the set of test problems to cover. Initially, P is empty and S contains all 60 training problems. At each iteration, the algorithm chooses the planner with the largest set of solved problems in S, removes these problems from S, and adds the selected planner to P. If the number of planners with the largest set of solved problems in S is greater than one, the algorithm selects the first evaluated planner (the planner evaluation order is random). The process terminates when S is empty. The resulting set P contains the planners of the configured portfolio.

We experimentally observed that for almost every considered domain, since more than one incorporated planner solves all training problems in the domain, the set of planners forming the cluster selected by BUS for the domain consists of only one planner (except for domain Parking that has two selected planners, LAMA and FF using macro set Clumping). Moreover, the choice of this planner among those that solve all training problems is drastically affected by the random order in which the greedy set covering approximation algorithm evaluates the coverage of the planners. Hence, to derive an indication about the performance that can be reached by our implementation of BUS, we ran the portfolio configuration of BUS nine times, tested the obtained nine configured portfolios, and analyzed three sets of experimental results for the CPU time and three sets of experimental results for the plan quality. These three sets were derived using: the median performing configured portfolio over the nine generated for each considered domain, and the best/worst performing configured portfolio over *all* the possible portfolios that can be generated by the greedy set covering approximation algorithm of BUS. The results of this experimental comparison are given in Table 10 and summarized in the following observation.

Experimental result 4.4.3 *For the benchmark domains of the learning track of IPC7, in terms of time score and average CPU time,* PbP.s/q *performs much better than the worst and the median configured portfolios derived by* BUS*;* PbP.s *performs slightly better than the best configured portfolio that an oracle would select among those that can be derived by* BUS*, while* PbP.q *performs slightly worse. In terms of problem coverage (the criterion used by* BUS *to select the planners in the cluster),* PbP.s *solves the same number of problems as the best configured portfolio that can be derived by* BUS*.*

The results of Table 10 show that the performance obtained by the configured portfolios generated by BUS varies greatly, indicating that the planner-selection method of BUS is not very accurate to derive efficient domain-specific configured portfolios. We think that the main reason of this is that for the planner selection BUS only considers the problem coverage and ignores the CPU time and plan quality of the incorporated planners. However, it is important to note that the planner-selection method of BUS was originally proposed for different kinds of data sets (problem instances over a set of domains considered altogether and different from those used in our experiment) and with a different purpose (generating a domain-*in*dependent planner portfolio), for which BUS is a prominent approach that showed good performance (Roberts & Howe, 2009).

4.5 Effectiveness of the Computed Configuration Knowledge

This section concerns experimental goal G4. In order to understand the effectiveness of the automated portfolio configuration in PbP, we compare the performance of PbP with the computed configuration knowledge (PbP.s/q), with no configuration (PbP-nok.s/q), and with a random configuration (PbP-rand.s/q). In PbP-nok.s/q, all planners in the initial portfolio are selected, macros are not used, the planning time slots are the same for all planners, and their execution order is random. PbP-rand.s/q, is the same as PbP-nok.s/q except that a subset of at most three randomly chosen planners with a (possibly empty)

IPC7	Time score				Mean CPU time				# solved problems			
Domains	PbP.s	BUS			PbP.s	BUS			PbP.s	BUS		
		W.s	M.s	B.s		W.s	M.s	B.s		W.s	M.s	B.s
Barman	30.0	0.0	27.3	30.0	2.0	900.0	2.7	2.0	30	0	30	30
Blocksworld	**29.8**	7.4	16.1	23.5	**9.9**	603.2	135.1	19.7	30	10	30	30
Depots	20.6	7.6	14.3	**21.1**	191.4	590.9	205.0	**132.5**	26	11	22	26
Gripper	**29.5**	17.3	17.5	28.9	**18.2**	183.8	87.3	18.3	30	25	30	30
Parking	**7.9**	7.2	7.2	7.2	**364.1**	428.1	428.1	428.1	8	8	8	8
Rovers	26.9	1.7	8.7	26.9	50.5	840.4	411.5	50.5	27	5	18	27
Satellite	**30.0**	0.0	0.0	23.0	**28.3**	900.0	900.0	70.2	30	0	0	30
Spanner	30.0	15.0	22.4	30.0	16.9	208.1	151.1	16.9	30	30	30	30
TPP	**25.0**	0.0	8.2	21.3	**121.2**	900.0	508.6	175.3	25	0	15	25
All domains	**229.7**	56.2	121.7	211.9	**63.2**	627.0	299.6	70.3	236	89	183	236

IPC7	Quality score				Mean plan length				# solved problems			
Domains	PbP.q	BUS			PbP.q	BUS			PbP.q	BUS		
		W.q	M.q	B.q		W.q	M.q	B.q		W.q	M.q	B.q
Barman	30.0	0.0	29.5	30.0	–	–	–	–	30	0	30	30
Blocksworld	29.9	9.4	29.7	29.9	198.4	210.6	198.4	198.4	30	10	30	30
Depots	**24.4**	6.0	13.5	22.8	**143.7**	175.8	231.8	190.5	26	11	22	26
Gripper	29.0	15.2	15.2	**29.9**	577.0	876.6	876.6	**554.7**	30	30	30	30
Parking	4.3	7.8	7.8	7.8	87.8	77.4	77.4	77.4	5	8	8	8
Rovers	25.1	4.4	16.9	**29.0**	498.8	482.8	522.4	**420.0**	29	5	18	29
Satellite	**30.0**	0.0	0.0	28.5	–	–	–	–	30	0	0	30
Spanner	30.0	30.0	30.0	30.0	326.0	326.0	326.0	326.0	30	30	30	30
TPP	14.5	0.0	14.7	**21.5**	–	–	–	–	15	0	15	25
All domains	217.2	72.8	157.9	**229.4**	367.6	459.8	464.9	**358.7**	226	94	183	238

Table 10: Time/quality score, mean CPU time/plan length and number of solved problems of PbP.s/q, and the worst, median and best portfolios that can be derived by using BUS for each IPC7 domains. W.s, M.s and B.s denote the worst, median and best portfolios among those that BUS can derive with the lowest, median, and highest time score for each considered IPC7 test problems, respectively; similarly W.q, M.q and B.q denote the worst, median and best portfolios with the lowest, median, and highest quality score, respectively.

randomly chosen set of learned macros is used, instead of all planners, and a different random configuration of PbP.s/q has been generated for every IPC7 problem.

Figure 3 gives an overall picture of the results for all problems in the IPC7 domains considering different amounts of CPU times for the portfolio configuration; specifically, the time on the x-axis is the CPU-time limit given to each run of a planner (with a set of macros) during the performance measurement simulation phase, to each (simulated) run of the candidate clusters of planners during the planning cluster selection and ordering phase, and to each run of the configured portfolio during the test phase. The marked points on the curves for PbP.s/q correspond to performance scores of the different configured portfolios obtained for the different considered CPU-time limits. These results indicate that, for every considered CPU-time limit in the configuration phase, PbP.s/q clearly performs better than PbP-nok and PbP-rand. Moreover, a refined analysis considering each domain separately shows that PbP.s/q has the best performance also for every single considered domain, and that in terms of problem coverage for every considered CPU-time limit the gaps between

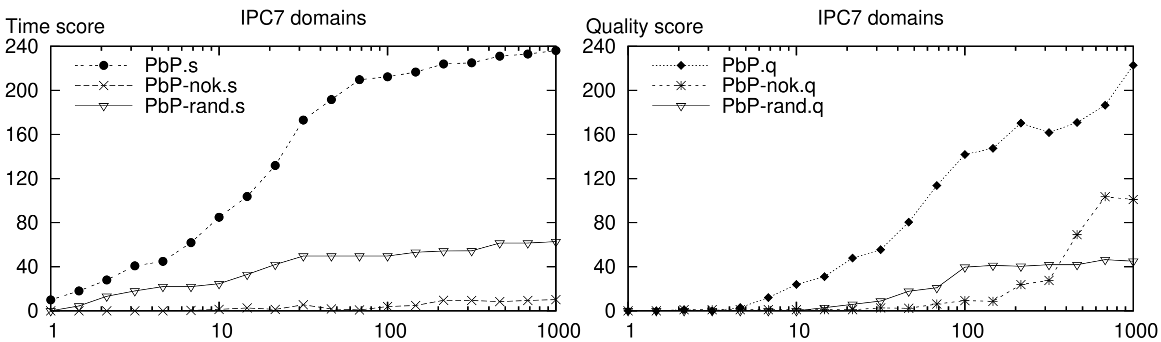

Figure 3: Time score (left plot) and quality score (right plot) of PbP, PbP-nok and PbP-rand with respect to an increasing CPU-time limit T (ranging from 1 to 1000 seconds) for the IPC7 domains.

PbP.s/q and the other two compared version of PbP are very similar to the gaps in the plots of Figure 3.

Experimental result 4.5.1 *The computed configuration knowledge can considerably improve the performance of* PbP.s/q *w.r.t. the unconfigured and randomly configured versions of* PbP *(*PbP-nok *and* PbP-rand*, respectively).*

In terms of planning speed, the performance comparison of the three considered versions of PbP.s/q, using the Wilcoxon sign-rank test gives a similar general result: PbP.s is statistically faster than the other versions ($z = -12.578$, $p < \frac{0.001}{2}$). In terms of plan quality, PbP.q performs statistically better than the unconfigured version ($z = -13.205$, $p < \frac{0.001}{2}$). For the comparison between PbP.q and PbP-rand.q, we analyzed 47 out of the 230 problems solved by PbP.q, because PbP-rand.q solves few problems and for the plan quality comparisons we consider only problems that are solved by both the compared planners. The results of the Wilcoxon test indicates that PbP.q performs similarly to PbP-rand.q ($z = -1.662$, $p = 0.096$). However, it should be noted that the low number of considered problems makes the statistical comparison through the Wilcoxon sign-rank test not very accurate and informative for deriving general conclusions about the relative performance in this case.

We have also tested a version of PbP-nok in which the incorporated planners are run using the predetermined time slot sequence S_{pre} and the planner runs are ordered using the same method used by PbP, which considers the relative performance of the planners on the set of the training problems instead of the random order. Overall the performance of PbP-nok remains much worse than the performance of (the planner cluster selected by) the configured version of PbP.

Table 11 analyzes the impact on performance of using DSK (i.e., for PbP, the computed configuration knowledge) in the best-performing planners that entered the learning track of IPC6-7. The results of this comparison confirm the strong positive impact of PbP's DSK.

Planner	ΔSolved (%)	ΔTime	ΔQuality
Best IPC6 planners			
ObtuseWedge	**+17.3**	+34.1	+23.7
PbP-IPC6.s	+3.6	+ 65.2	−3.0
PbP-IPC6.q	+0.9	+5.8	+0.7
Wizard+FF	−6.6	+21.0	−15.2
Wizard+SGPlan	−1.7	+17.6	−3.1
PbP.s	+5.0	**+82.5**	−3.2
PbP.q	+3.3	+6.3	**+37.5**
Best IPC7 planners			
BootstrapPlanner	+4.1	+3.3	+10.9
Fast Downward-autotune-speed	**+43.3**	+65.3	**+99.1**
Fast Downward-autotune-quality	+14.1	+16.0	+26.4
OALDAEYASHP	−18.9	−17.3	−40.4
ParLPG-speed	+9.3	+42.1	+15.6
PbP-IPC7.s	+5.6	+116.5	+16.8
PbP-IPC7.q	+7.4	+24.3	+40.9
PbP.s	+21.48	**+171.1**	+46.8
PbP.q	+20.74	+29.4	+69.8

Table 11: Performance gaps of the best-performing IPC6-7 planners with/without DSK in terms of percentage of solved problems, time and quality scores for the IPC6-7 benchmark domains and problems. Planner RFA1 is omitted because it works only with DSK.

Experimental result 4.5.2 *For the IPC6 domains and problems, the DSK computed for* PbP.s *and* PbP.q *has the strongest impact among the DSK of the IPC6 planners in terms of improved speed (Δtime) and plan quality (Δquality), respectively. The DSK computed for* ObtuseWedge *has the strongest impact in terms of percentage of additional solved IPC6 problems.*

The reason why the impact of the DSK computed by PbP is quite low in terms of additional solved IPC6 problems is that PbP.s/q solves almost all these problems even without DSK.

Experimental result 4.5.3 *For the IPC7 domains and problems, the DSK computed for* PbP.s *has the strongest impact in terms of improved speed (Δtime) among the DSK of the IPC7 planners. The use of the computed DSK in* Fast Downward-Autotune-speed *has the strongest impact in terms of percentage of additional solved problems and improved plan quality.*

Although in terms of percentage of additional solved problems and improved plan quality the use of DSK in PbP.s/q has not the highest impact, it leads to high improvements also in PbP.s/q, allowing it to achieve performance that is generally better than Fast Downward-Autotune-speed (see the "Quality Score" column of Table 5).

Finally, we conducted an experiment to understand if configuring PbP for a specific domain generates DSK that leads to better performance w.r.t. configuring the planner portfolio over a set of domains altogether. Table 12 compares the performance of PbP.s/q with the DSK, the DSK obtained without using macros (PbP-noM.s/q), and the configuration knowledge computed across all IPC7 domains (PbP-allD.s/q). The planner cluster of PbP-allD.s is formed by LPG and SGPlan5, while the planner cluster of PbP-allD.q is formed by LAMA, Marvin and SGPlan5. The results in Table 12 indicate that, even without considering the usage of macros, the portfolio configuration over all the considered domains together greatly decreases the performance of PbP.

Experimental result 4.5.4 *For the IPC7 domains, in terms of time score, average CPU time and number of solved problems,* PbP.s *performs much better than both* PbP-noM.s *and* PbP-allD.s*. In terms of quality score and number of solved problems,* PbP.q *performs much better than both* PbP-noM.q *and* PbP-allD.q*. In terms of average plan length, both* PbP.q *and* PbP-noM.q *perform usually better than* PbP-allD.q*.*

The results of the Wilcoxon sign-rank test applied to the comparison between PbP and PbP-noM confirm that, over all the IPC7 domains, PbP.s is significantly faster than PbP-noM.s ($z = -7.699$, $p < 0.001$) and, in terms of plan quality, PbP.q performs significantly better than PbP-noM.q ($z = -5.465$, $p < 0.001$). The high performance gap between PbP and PbP-noM, that is in favor of PbP, clearly indicates the usefulness of using macros, showing that a portfolio of planners and macros can be much more efficient than a portfolio of only planners.

4.6 Accuracy of the Planner Cluster Selection

This section concerns experimental goal G5. In order to test the accuracy of the planner cluster selection in PbP, we carried out three related experiments in which the performance of PbP using the computed configuration knowledge was compared with the performance of (a) every basic planner incorporated in the initial portfolio, (b) the best performing incorporated planner (without using macros) in each considered domain, and (c) the best performing planner cluster (possibly using macros) in each considered domain. In the following, Section 4.6.1 presents experiments (a) and (b), Section 4.6.2 experiment (c).

4.6.1 PbP and the Basic Portfolio Planners

Figure 4 gives an overall picture of the performance of PbP.s/q w.r.t. the performance of the basic planners (without macros) in terms of speed and plan quality, using a CPU-time limit for each run ranging from 1 to 1000 seconds. The time/quality scores of each compared system was derived by summing up the corresponding scores obtained by the system in each IPC7 domain. This analysis indicates that, for every considered CPU-time limit, PbP.s with DSK is generally much faster than the incorporated basic planners, and PbP.q generates better quality plans.

Experimental result 4.6.1 *For the IPC7 domains, there is no basic planner in the considered input portfolio of* PbP *that achieves an overall performance better than or similar to the performance of* PbP.s *for speed, and of* PbP.q *for plan quality (except for very low CPU-time limits, where all compared planners perform similarly in terms of plan quality).*

IPC7	Time score			Mean CPU time			# solved problems		
Domains	PbP.s	noM	allD	PbP.s	noM	allD	PbP.s	noM	allD
Barman	30.0	12.0	10.9	2.0	72.9	138.6	30	30	30
Blocksworld	30.0	17.3	8.3	5.9	48.1	311.4	30	30	21
Depots	22.3	16.5	8.7	28.3	49.3	134.8	26	21	13
Gripper	30.0	30.0	15.0	18.2	18.2	175.1	30	30	30
Parking	7.8	6.2	0.0	–	–	–	8	7	0
Rovers	27.0	27.0	13.6	16.9	16.9	192.7	27	27	26
Satellite	30.0	30.0	18.3	28.3	28.3	151.0	30	30	30
Spanner	30.0	30.0	11.3	13.5	13.5	272.0	30	30	25
TPP	23.7	11.9	0.0	–	–	–	25	13	0
All domains	232.3	181.3	86.1	25.8	44.7	194.4	236	219	175

IPC7	Quality score			Mean plan length			# solved problems		
Domains	PbP.q	noM	allD	PbP.q	noM	allD	PbP.q	noM	allD
Barman	29.9	29.8	0.0	–	–	–	30	30	0
Blocksworld	29.9	18.5	13.4	222.6	307.0	340.8	30	26	16
Depots	25.8	7.2	2.8	153.0	163.5	171.0	26	9	3
Gripper	30.0	30.0	21.4	578.9	578.9	675.9	30	30	25
Parking	4.8	4.8	2.0	76.0	76.0	61.0	5	5	2
Rovers	29.6	29.6	22.1	634.9	634.9	650.6	30	30	23
Satellite	29.9	29.9	8.8	715.1	715.1	731.9	30	30	9
Spanner	30.0	30.0	8.0	284.8	284.8	284.8	30	30	8
TPP	14.8	14.8	12.3	370.1	370.1	374.5	15	15	13
All domains	224.7	194.5	91.2	484.5	498.8	535.4	226	205	99

Table 12: Time/quality score, average CPU time/plan length and number of solved problems for the speed and quality versions of PbP, PbP-noM (abbreviated with noM) and PbP-allD (abbreviated with allD) for the IPC7 domains.

The results of the Wilcoxon sign-rank test applied to this experiment confirm that PbP.s is significantly faster than every incorporated planner ($z = -5.773$, $p < \frac{0.001}{9}$), and that in terms of plan quality PbP.q performs significantly better than all them ($z = -3.920$, $p < \frac{0.001}{9}$) except ParLPG. According to the Wilcoxon sign-rank test, there is no statistical difference between the quality performances of PbP.q and ParLPG. The discrepancy between the results of this analysis and those in Figure 4 is generated by the different ways in which unsolved problems are handled by the quality score function and the Wilcoxon sign-rank test for comparing plan quality performance: the first considers all problems attempted by the compared planners (explicitly penalizing a planner with zero score for each unsolved problem), while the second considers only the subset of the test problems solved by both the compared planners; PbP.q solves many more problems than ParLPG (230 against 179), and this is reflected in the relative curves of Figure 4 for plan quality.

We observed that for domains Rovers, Satellite and Gripper all solutions of PbP.q are computed by ParLPG; for domains Blocksworld and Depots, PbP.q using ParLPG solves 5 and 3 problems, respectively; for the other considered domains, ParLPG is not part of the selected cluster of running planners. To better understand the importance of ParLPG in PbP, we have analyzed the performance of a version of PbP that does not incorporate ParLPG.

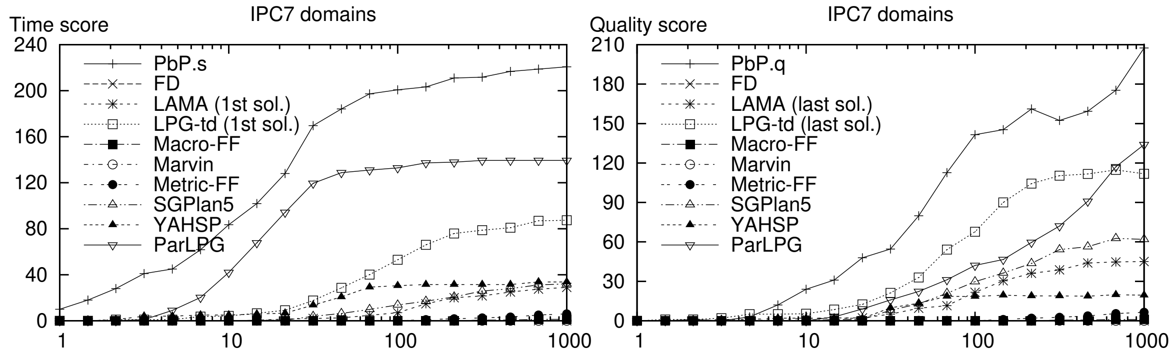

Figure 4: Time (left plot) and quality (right plot) scores of PbP.s/q with the relative computed configuration knowledge compared to the time and quality scores of the basic incorporated planners for the IPC7 domains, using an increasing CPU-time limit. FD abbreviates Fast Downward.

For the IPC7 domains, if PbP.s/q does not incorporate ParLPG, the problems solved by PbP.s/q decrease by about 10/12%, and, in terms of time score, PbP.s without ParLPG performs worse than ParLPG (156.1 vs. 176.5). However, in terms of quality score, PbP.q without ParLPG performs still much better than ParLPG (183.1 vs. 144.5). The results of this analysis show that the performance of PbP in terms of speed is drastically affected by ParLPG. On the other hand, the importance of having ParLPG in PbP.q is limited because the parameter configuration of ParLPG focused on speed.

The two main reasons explaining the observation derived from Experimental result 4.6.1 about the globally best performance of PbP.s/q are that no basic incorporated planner (even ParLPG) outperforms all the others in *every* considered benchmark domain, and that PbP effectively selects and combines the most efficient planners for each domain under consideration (possibly using a useful set of macro-actions).

One may wonder if the picture can be different when PbP.s/q is compared with the basic incorporated planners using a (possibly empty) set of macros. Figure 5 shows the results of this comparison, using a CPU-time limit for each run ranging from 1 to 1000 seconds. For the sake of readability, the names of the 38 combinations of basic incorporated planners and sets of macros (learned by Wizard and Macro-FF) have been omitted. The time/quality scores of each compared system was derived by summing up the corresponding scores obtained by any compared system in each IPC7 domain. If for a domain the combination of a planner P and a macro set M has M empty, then for that domain the combination is restricted to P.

The results in Figure 5 show that, in terms of CPU time, for the IPC7 domains there is no basic planner in PbP that, by using a learned macro set, achieves an overall performance better than or similar to the performance of PbP.s (except for very low CPU-time limits, where some compared planners with macros perform similarly). In terms of plan quality, for CPU-time limits lower than 20 seconds, there exist some basic incorporated planners using macros that perform better than PbP.q; for high CPU-time limits, PbP.q performs much better than every compared planner with macros. The combinations of basic incorporated

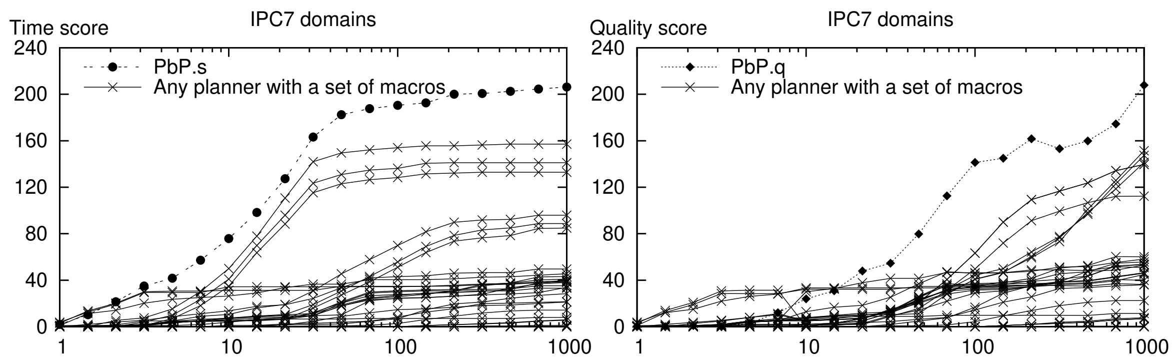

Figure 5: Time (left plot) and quality (right plot) scores of PbP.s/q with the relative computed configuration knowledge compared to the time and quality scores of the 38 combinations of incorporated planners and sets of macros for the IPC7 domains.

planners and sets of macros that for low CPU-time limits perform better than PbP.q are SGPlan5 using any set of learned macros, ParLPG using macro set "bunching", and YAHSP using macro set "clumping". For low CPU-time limits, these combinations of planners and macros have an overall performance better than PbP.q, essentially because they dominate over a single domain: `Barman` for SGPlan5, `Blocksworld` for ParLPG and YAHSP.

Since the analysis of Figure 4 considered the test domains altogether, in order to verify the supposition that also for a given single domain PbP performs better or not worse than every basic incorporated planner, we have compared PbP.s/q with the best-performing basic planner (according to the *test* problems and the relative IPC scores) for each considered domain. Such a planner, indicated with "BestP.s/q", is the single planner (without macros) that we would use if we had an oracle specifying the best basic incorporated planner for the test problems of a specific domain. The results of this experiment are shown in Table 13.

For domains `Gripper`, `Rovers`, `Satellite` and `Spanner` the planner cluster of PbP.s is the same as BestP.s. For the other considered domains, the time score and the average CPU time of PbP.s are much better than BestP.s. In terms of problem coverage, for three domains PbP.s solves a much higher number of problems; while for the other domains problem coverage is the same as BestP.s. These results show that, in order to achieve higher planning speed, using a cluster of planners or a useful set of macro-actions selected by PbP.s can be much better than using a single planner without macros. Sections 4.6.2 and 4.7 will study the usefulness of using a properly selected cluster of planners and a non-empty set of macros, respectively.

Experimental result 4.6.2 *There is no IPC7 domain for which* any *basic planner in the considered input portfolio of* PbP.s *is faster, achieves better time score, or solves more problems than* PbP.s*.*

Concerning plan quality, BestP.q contributes a great deal to the success of PbP.q, since for all domains except `Barman` and `Spanner` it is included in the cluster selected by PbP.q (see Table 2). For `Barman`, `Gripper`, `Parking`, `Rovers`, `Satellite`, `Spanner`, and `TPP`, in most cases BestP.q provides the solution to PbP.q.

IPC7 Domains	BestP.s	Max score	Time score		Mean CPU time		# solved problems	
			PbP.s	BestP.s	PbP.s	BestP.s	PbP.s	BestP.s
Barman	SGPlan5	30	30.0	12.0	2.0	72.9	30	30
Blocksworld	ParLPG	30	30.0	17.3	9.9	95.3	30	30
Depots	ParLPG	30	23.6	18.3	109.5	229.7	26	21
Gripper	ParLPG	30	30.0	30.0	18.2	18.2	30	30
Parking	FF	30	6.8	3.0	364.1	460.9	8	7
Rovers	ParLPG	30	27.0	27.0	25.1	25.1	27	27
Satellite	ParLPG	30	30.0	30.0	28.3	28.3	30	30
Spanner	ParLPG	30	30.0	30.0	16.9	16.9	30	30
TPP	ParLPG	30	24.4	11.2	121.2	531.4	25	14
All domains	–	270	232.8	178.8	39.9	79.3	236	219

IPC7 Domains	BestP.q	Max score	Quality score		Mean plan length		# solved problems	
			PbP.q	BestP.q	PbP.q	BestP.q	PbP.q	BestP.q
Barman	SGPlan5	30	30.0	29.8	449.3	452.9	30	30
Blocksworld	ParLPG	30	29.9	29.9	269.9	272.8	30	30
Depots	ParLPG	30	24.5	10.4	160.1	163.1	26	11
Gripper	ParLPG	30	29.0	29.9	577.3	570.1	30	30
Parking	FF	30	4.8	6.8	79.0	80.6	5	7
Rovers	ParLPG	30	30.0	30.0	694.7	694.7	30	30
Satellite	ParLPG	30	29.6	29.8	785.2	782.8	30	30
Spanner	ParLPG	30	30.0	30.0	326.0	326.0	30	30
TPP	LAMA	30	14.6	14.7	370.1	366.3	15	15
All domains	–	270	222.4	210.8	472.6	481.8	226	213

Table 13: Maximum score, time/quality score, average CPU time/plan length, and number of problems solved by PbP.s/q and the best planner (BestP.s/q) for the IPC7 domains.

Experimental result 4.6.3 *For the IPC7 domains, in terms of plan quality, the relative performance of* PbP.q *and the best-performing basic planner (BestP.q) that an oracle would choose is generally slightly in favor of* PbP.q*: for* Blocksworld *and* Depots PbP.q *performs better, for* Parking *BestP.q performs slightly better, and in the rest of the IPC7 domains they perform similarly.*

Concerning Parking, Table 13 shows that, for the used benchmark problems, the BestP.q-planner is FF, which is correctly contained in the cluster selected by PbP.q for this domain (see Table 2). However, this cluster also includes an additional planner (LAMA) that, for the tested problems and the considered CPU-time limit, does not give a useful contribution to PbP.q (no solution is found by LAMA), introducing some "noise" in the cluster selection. This and the fact that for Parking no useful set of macros is computed by PbP.q are the main reasons why PbP.q performs slightly worse than the BestP.q-planners for the considered test problems in domain Parking.

The Wilcoxon sign-rank test applied to this experiment confirms that, overall, PbP.s is significantly faster than the BestP.s-planner of each domain ($z = -3.134$, $p \approx 0.001$); while in terms of plan quality, the test results indicate that the performances of PbP.q and the

BestP.q-planner are not significantly different ($z = -1.157$, $p = 0.247$); in other words, the test cannot derive that one system performs statistically better than the other.

Finally, we have compared PbP.s/q and the best-performing combination $P + M$ of a basic planner P with a non-empty set M of macros learned for P in each IPC7 domain, except Spanner for which no macro are computed. In this experiment, the best macro set M for P in a domain D was chosen considering the performance of $P + M$ over the training problems of D. Overall, in terms of speed score and problem coverage, PbP.s performs similarly to $P + M$ in five domains, while it performs much better in three domains; in terms of quality score, PbP.q performs similarly in four domains and much better in other four domains. One of the reasons why $P + M$ can perform worse than PbP.s/q is that in some domains macros are harmful, and PbP.s/q correctly decides not to use them. This will be discussed also in the context of an experiment presented in Section 4.7, where we analyze the usefulness of macros and the accuracy of their selection in PbP.s/q.

4.6.2 PbP and the Best-Performing Portfolio Configuration

In order to test the accuracy of the planner cluster selection in PbP.s/q, we have also compared PbP with the computed configuration knowledge and the best-performing cluster of planners (with the useful macros) for each considered test domain. (The worst-performing cluster solves no problem.) Table 14 shows the results of this experiment considering two best-performing clusters with at most three planners: for each considered IPC7 domain, BestC.s is the planner cluster with the highest time score among those that can be obtained by PbP.s using the default PCPV; similarly, BestC.q is the planner cluster with the highest quality score. Therefore the data in the time/quality score columns "BestC.s/q" are the maximum values over the time/quality score sums of all planner clusters for the set of test problems of each IPC7 domain.

For every domain except Depots, the time score of PbP.s is the same as the one of the best cluster and is much greater than zero (and thus much better than the score of the worst cluster). Also in terms of average CPU time and problem coverage the performance of PbP.s and the best cluster are almost always the same. Only for domain Depots PbP.s and BestC.s perform slightly differently; in this case, the planners and relative macros in the cluster of PbP.s are different from those in BestC.s. In particular, Macro-FF is selected with a different set of macros, and this makes PbP.s slightly slower.

Concerning PbP.q, overall, in terms of plan quality there is no high performance gap with respect to the best cluster, although PbP.q performs worse for domain TPP. For this domain, the training problems used by PbP.q are not informative enough. This observation is supported by the fact that the best cluster computed using the *training* problems, instead of the test problems, is different from the one derived for the test problems. On the other hand, we observed that, if the size of the training problems is similar to the size of the test problems, the configured portfolios of PbP.q and BestC.q are the same.

The Wilcoxon sign-rank test confirms that, overall, the performance of PbP.s/q and the best cluster is not statistically significantly different ($z = -0.422$, $p = 0.673$, for the speed analysis; $z = -2.432$, $p = 0.015$, for the quality analysis). Moreover, we have also observed that PbP.s/q without configuration (PbP-nok.s/q) performs generally much worse than the

IPC7 Domains	BestC.s	Max score	Time score		Mean CPU time		# solved probs	
			PbP.s	BestC.s	PbP.s	BestC.s	PbP.s	BestC.s
Barman	SGPlan5 (B)	30	30.0	30.0	2.0	2.0	30	30
Blocksworld	ParLPG (B)	30	30.0	30.0	9.9	9.9	30	30
Depots	Macro-FF (M1)	30	21.2	24.5	165.9	82.5	26	28
Gripper	ParLPG (–)	30	30.0	30.0	18.2	18.2	30	30
Parking	Macro-FF (M2)	30	8.0	8.0	364.1	364.1	8	8
Rovers	ParLPG (–)	30	27.0	27.0	25.1	25.1	27	27
Satellite	ParLPG (–)	30	30.0	30.0	28.3	28.3	30	30
Spanner	ParLPG (–)	30	30.0	30.0	16.9	16.9	30	30
TPP	Macro-FF (M1)	30	25.0	25.0	121.2	121.2	25	25
All domains	–	270	231.2	234.5	47.4	36.7	236	238

IPC7 Domains	BestC.q	Max score	Quality score		Mean plan length		# solved probs	
			PbP.q	BestC.q	PbP.q	BestC.q	PbP.q	BestC.q
Barman	SGPlan5 (Cl)	30	29.9	30.0	449.3	448.3	30	30
Blocksworld	ParLPG (–)	30	29.9	29.9	269.9	272.8	30	30
Depots	MFF(M1),MFF(M2)	30	23.9	26.7	160.1	165.1	26	28
Gripper	ParLPG (–)	30	29.0	29.9	577.3	570.1	30	30
Parking	FF (0)	30	4.8	6.8	79.0	80.6	5	7
Rovers	ParLPG (–)	30	30.0	30.0	694.7	694.7	30	30
Satellite	ParLPG (–)	30	29.5	29.8	785.2	782.8	30	30
Spanner	LPG (–)	30	30.0	30.0	326.0	326.0	30	30
TPP	Macro-FF (M1)	30	14.8	24.4	370.1	379.5	15	25
All domains	-	270	221.8	237.5	461.2	487.6	226	240

Table 14: Maximum score, time/quality score, average CPU time/plan length, and number of problems solved by PbP.s/q and the best cluster (BestC.s/q) for the IPC7 domains. MFF abbreviates Macro-FF. The order of the planners listed in the cluster for Depots corresponds to the order in which they are run.

best cluster for speed and quality. Overall, from our experimental results we can derive the following observation.

Experimental result 4.6.4 *For the IPC7 benchmarks, in terms of time score, average CPU time and problem coverage,* PbP.s *performs as well as or, for* Depots*, similarly to BestC.s. In terms of quality score, average plan length and problem coverage,* PbP.q *performs as well as or similarly to BestC.q, except for* TPP*, in which the plan quality score and problem coverage of* PbP.q *are worse.*

Table 14 also shows that very often an oracle would use a single planner to either quickly solve the IPC7 problems or compute high-quality plans for them. Hence, one may argue that using clusters formed by more than one planner (possibly with a set of useful macros) is not useful. The rationale why the best clusters in Table 14 are formed by a single planner is that often any incorporated planner (even using macros) requires almost all the CPU time to solve each IPC7 test problem (except for domain Depots); thus the remaining time is usually not enough to improve the coverage or the quality of the (first) computed plan by running more than one planner. For the purpose of computing high-quality plans, if we use a set of test problems smaller than the IPC7 problems, then the picture is different.

Table 15 compares the performance of PbP and the best performing cluster of planners for some sets of randomly generated medium-size problems of the IPC7 domains (i.e., with size ranging between the largest training problems and the smallest testing problems). In this table, BestC.s/q indicates the clusters that an oracle would use to solve these sets of medium-size problems.

Experimental result 4.6.5 *For test problems over the IPC7 domains with sizes ranging between the training problem sizes and the IPC7 test problem sizes, for most of the IPC7 domains the best planner clusters for deriving high quality plans are formed by more than one planner.*

In general, a cluster of planners containing a certain planner performs worse than this planner alone when most of the planning problems of the domain for which the planner portfolio is configured are efficiently solved by the planner alone, and thus running also the other planners of the cluster is a waste of CPU time. A cluster formed by more than one planner performs better than any single portfolio planner only if for the considered domain there is no planner dominating all others in terms of either problem coverage and CPU time, or problem coverage and plan quality.

Interestingly, we observed that sometimes the cluster selected by PbP.q and the best cluster for the intermediate-size test problems are formed by a planner that solves all the problems, but produces low-quality plans, and other planners that produce higher-quality plans, but solve few problems. This is the case for Barman and TPP. For these domains, although the quality of the plans of SGPlan5 is low, having SGPlan5 in the cluster is very useful because it contributes to greatly improve the problem coverage of the cluster.

Finally, the results in Table 15 also indicate that sometimes the effectiveness of the configured portfolio can be greatly affected by the difference between the size/hardness of the training problems and the size/hardness of the test problems. In particular, the performance gap between PbP.q and the best cluster for the considered randomly generated intermediate-size problems of domain TPP is lower than between PbP.q and the best cluster for the IPC7 test problems of TPP. This indicates that, in terms of plan quality, the effectiveness of the planner portfolio configuration in PbP.q computed using relatively small training problems can gradually decrease when the size/hardness of the test problems is increased.

4.7 Macro Usefulness and Selection Accuracy

This section concerns experimental goal G6: we analyze the effectiveness of using the set of macros selected by PbP for each planner, and the accuracy of PbP for selecting the most useful set of macros among those computed by Wizard or Macro-FF for each planner in the configured portfolio. While it has been shown that Wizard and Macro-FF can often generate useful sets of macros that speed up planners (Botea et al., 2007b; Newton et al., 2007), it is also known that there is no guarantee that using macros always leads to improving the speed of a planner, and a "bad" set of macros could even make a planner *slower*. Moreover, usually the degree of usefulness of a set of macros depends on the specific planner that uses them.

Concerning macros in PbP.s, for each IPC7 domain with at least one non-empty set of computed macros and each planner in the selected cluster (see Table 2), we compared the

IPC7 domains (medium probs)	BestC.s	Time score		Mean CPU time		# solved probs	
		PbP.s	BestC.s	PbP.s	BestC.s	PbP.s	BestC.s
Barman	S (B)	30.0	30.0	1.5	1.5	30	30
Blocksworld	ParLPG (B)	30.0	30.0	7.3	7.3	30	30
Depots	MFF (M1), ParLPG (0)	28.9	29.7	57.4	52.5	30	30
Gripper	ParLPG (–)	30.0	30.0	13.2	13.2	30	30
Parking	MFF (M2)	22.0	22.0	308.7	308.7	22	22
Rovers	ParLPG (–)	30.0	30.0	17.6	17.6	30	30
Satellite	ParLPG (–)	30.0	30.0	14.3	14.3	30	30
Spanner	ParLPG (–)	30.0	30.0	13.6	13.6	30	30
TPP	Macro-FF (M1)	30.0	30.0	93.6	93.6	30	30
All domains	–	260.9	261.7	50.9	50.4	262	262

IPC7 domains (medium probs)	BestC.q	Quality score		Mean plan length		# solved probs	
		PbP.q	BestC.q	PbP.q	BestC.q	PbP.q	BestC.q
Barman	S (Cl), FF (–), M (–)	29.7	29.8	327.2	327.1	30	30
Blocksworld	P (–), MFF (M1), LPG (B)	29.5	29.6	174.7	173.2	30	30
Depots	MFF (M1), P (–), LPG (–)	26.4	28.5	143.1	145.4	28	30
Gripper	ParLPG (–)	30.0	30.0	472.3	472.3	30	30
Parking	FF (–), LAMA (–)	20.0	20.0	63.1	63.1	20	20
Rovers	ParLPG (–)	30.0	30.0	694.7	694.7	30	30
Satellite	ParLPG (–), Marvin (–)	30.0	30.0	524.6	524.6	30	30
Spanner	LPG (–)	30.0	30.0	257.2	257.2	30	30
TPP	MFF (M1), L (–), S (CH)	24.9	29.3	219.4	220.3	25	30
All domains	–	221.8	237.5	462.6	463.2	226	240

Table 15: Time/quality score, average CPU time/plan length, and number of problems solved by PbP.s/q and the best cluster (BestC.s/q) for sets of medium-size problems of the IPC7 domains. S, M, MFF, P, L abbreviate SGPlan5, Marvin, Macro-FF, ParLPG, and LAMA, respectively. The order of the planners listed in the clusters corresponds to the order in which they run.

number of solved problems, number of visited search nodes, average CPU time and time score using: (a) no macros, (b) the set of macros identified by PbP.s as useful for the planner, and (c) the set of macros among those computed for the planner that in terms of time score makes it perform best over the test problems. From the results of this experiment, which are given in Table 16, the following general observation can be derived.

Experimental result 4.7.1 *For the IPC7 domains, very often there is a candidate set of macros for a planner (computed by* Wizard *or* Macro-FF*) that greatly increases the speed performance of the configured portfolio, and* PbP.s *correctly selects it.*

Table 16 also indicates that, for most of the considered domains, the performance of the selected planners obtained using their sets of macros identified as useful by PbP.s is usually the same as the performance they can achieve when using their best sets of macros. This gives strong positive evidence about the effectiveness of PbP.s's approach to selecting a useful set of macros for each planner in the configured portfolio. In particular, the best set of macros is the same as the set of macros selected by PbP.s (see Table 2). The only exception where the sets of macros identified by PbP.s is different from the best set is the

Domain & Planner	with no macros				with PbP.s macros				with best macros			
	#S	#N	T	TS	#S	#N	T	TS	#S	#N	T	TS
Barman SGPlan5	30	–	72.9	12.0	30	–	1.8	30.0	30	–	1.8	30.0
Blocksworld ParLPG	30	3361	95.3	17.3	30	218.0	9.9	30.0	30	218.0	9.9	30.0
Depots Macro-FF	0	242678	–	0.0	26	33654	203.3	22.2	28	21231	105.1	26.2
Parking Macro-FF	2	1739	406.9	0.6	8	880.9	92.3	8.0	8	880.9	92.3	8.0
TPP Macro-FF	0	71594	600.0	0.0	25	2990	121.2	25.0	25	2990	121.2	25.0

Table 16: Number of solved problems (#S), number of visited search nodes (#N), average CPU time (T) and time score (TS) of the planners forming the cluster selected by PbP.s using no macro, the set of macros selected by PbP.s, and the best performing set of computed macros. The domains considered are the IPC7 domains with at least one non-empty set of computed macros. We indicate with "–" that the number of nodes visited by SGPlan5 could not be measured.

case of Macro-FF in domain Depots. However, as shown in Table 2, for Depots PbP.s selects a cluster that contains both Macro-FF with macro set M2 and ParLPG, obtaining an overall performance that we experimentally observed to be very similar to the performance of Macro-FF with the best set of macros, M1. It is worth noting that the candidate sets of macros computed for ParLPG and Depots are *harmful* (i.e., they make its speed performance much worse) and PbP.s correctly detects this, choosing to run ParLPG with zero macros (denoted with "ParLPG (0)" in Table 2).

The study of computing and using macros has usually been pursued with the main goal of speeding up planning, possibly making the quality of the computed plan lower than when macros are not used. Interestingly, in the context of PbP.q, in several cases macros are useful also for improving plan quality. Specifically, for nine over the fifteen IPC6-7 domains, the configuration phase of PbP.q selects clusters of planners with at least one planner using a non-empty set of macros (see Table 2). We experimentally observed, with both the training problems and the test problems, that there are two reasons why macros are useful to PbP.q:

- For some domains there are individual planners for which using macros leads to better quality plans. This is the case, e.g., for domains Barman and Blocksworld using planners SGPlan5 and LPG (first solution), respectively. This behavior has been observed also by Botea et al. (2005), Coles and Smith (2007), and Newton et al. (2007).

- If the selected cluster includes a planner configured to use a set of macros, usually such a planner quickly computes a solution. This can be somewhat helpful also for the test problems that *another* planner in the cluster can solve with better solutions, if it has enough CPU time, because a quick termination of the planner with macros leaves more CPU time to run the other cluster planner(s). Having more CPU time, can be important especially for the incremental planner(s) included in the selected cluster, like LAMA and ParLPG. There are many problem instances of domains Depots, Satellite and TPP for which we observed this behavior.

Experimental result 4.7.2 *For the IPC7 domains, the use of the macros selected by* PbP.q *can lead to better quality solutions.*

In general, the use of macros can make the plan search more effective because, e.g., by planning multiple actions in one search step the size of the possible plateaus and the depth of the local minima can be reduced. On the other hand, if a large number of macros is added to the domain, the size of the search space can drastically increase, making the problem harder to solve. In the rest of this section, we analyze the kind and number of macros selected and used by PbP. We consider both macro operators, i.e., parameterized macros defined as sequences of (primitive) domain operators, and macro actions, i.e., macros derived by instantiating the parameters of the macro operators.

Table 17 describes the macro operators in the sets selected by PbP.s for a planner in the configured portfolio (see Table 2) in terms of: number of aggregated operators, number of involved parameters, average numbers of macro-actions and primitive actions in the augmented domain, average plan lengths obtained by the considered planners without using macros, and using them but counting each planned macro actions as a single action.

From the data in Table 17, we can derive some interesting observations about the macros used by PbP for the considered domains. First, the macro operators used by PbP for a planner are no more than three, and often they aggregate few primitive operators. Secondly, for the planners that handle macros by simply adding instantiated macro operators to the domain definition (SGPlan5 and ParLPG), the average number of macro actions in the augmented domains is much lower or comparable to the number of primitive domain actions, even for domain Barman where SGPlan5 uses a large macro operators involving seven primitive operators and six parameters. Hence, for these planners and domains, macro actions do not drastically increase the search space. The picture is quite different for Macro-FF, for which the macro operators selected by PbP.s in domains Depots, Parking and TPP, if instantiated, generate a number of macro actions that on average is one or more orders of magnitude greater than the number of primitive domain actions. The reason why Macro-FF can successfully use macro operators even if the number of domain macro actions is huge is that this planner instantiates macro operators and filters macro actions at search time, according to a relaxed-plan heuristic applied to the current search state, rather than simply adding all macro actions to the original domain before planning.

The fact that in our experiment PbP never generates configured portfolios with large sets of macro actions added to the domain description seems to indicate that, if the number of macro actions is very high w.r.t. the number of primitive actions, this macro exploitation method usually makes the performance of a planner using them much worse. This observation was confirmed by an additional experiment in which we added the PDDL description of the macro operators learned by Macro-FF for domain Depots to the original description of Depots, and run Macro-FF using the resulting augmented domain. As shown in Table 17, for Depots the number of learned macro actions is about one order of magnitude greater than the number of primitive actions. We experimentally observed that with the augmented domain Macro-FF (without its own method of using macros) solves no Depots problem.

Domain & Planner	#operators for every m.	#parameters for every m.	#grounded macros	#actions	Plan length without m.	Plan length with m.
Barman SGPlan5+B	7,4	6,4	1645 (397)	15610 (3767)	452 (57)	374 (45)
Blocksworld ParLPG+B	2,3,2	2,2,2	17757 (5812)	11983 (5270)	415 (107)	153 (42)
Depots Macro-FF+M2	2,2	5,6	224053 (114600)	16005 (8269)	–	119 (26)
Parking Macro-FF+M2	5,2	8,5	billions (billions)	243223 (151979)	143 (23)	64 (11)
TPP Macro-FF+M1	6	9	billions (billions)	133145 (78545)	–	238 (51)

Table 17: Number of (primitive) operators forming each of the selected macro operators, number of parameters in each macro operator, average number of instantiated macro actions, average number of domain (primitive) actions, average plan length without using macros, and average plan length using macros and counting each planned macro action as a single action. Each number in the 2nd and 3rd columns refers to a different macro operator. Numbers in brackets are standard deviations. The domains considered are the IPC7 domains with at least one non-empty set of learned macros selected by PbP.s. "B" abbreviates the *B*unching macro set learned by Wizard; M1–M2 are two of the five sets of macros generated by Macro-FF. We indicate with "–" that no solution was found within the given CPU-time limit.

Moreover, the results about the average plan length in Table 17, show that plans with macro actions are much shorter than those computed from the original domain, if we count each macro as a single action. Given that during planning the application to the current search state of a macro (or possibly a combination of macros in Macro-FF) generates a single successor state, for the considered planners and domains, on average the distance between the initial search state and a goal state is much shorter when the search space includes macros, and hence searching a solution plan in this space can be much faster.

To conclude, we note that the usefulness of macros can also depend on factors different from those considered in our analysis, such as, e.g., the ratio between the number of useful instantiations of a macro operator (providing shortcuts towards a goal state) and the number of instantiations that guides the search towards a wrong direction (Botea, Müller, & Schaeffer, 2007a). Further factors that might affect the usefulness of macro-operators in planning are conjectured in the work by McCluskey and Porteous (1997).

4.8 Planner Cluster Scheduling

This section concerns experimental goal G7: we experimentally analyze some possible alternative strategies for scheduling the execution of the planners during the portfolio configuration of PbP and at planning time. In the first experiment, we investigate the use in PbP of four sequential and round-robin strategies with predefined and configured planning time slots. In the second experiment, we study the importance of choosing a specific PCPV defining the planning time slots (as described in Section 3.1) and in particular of PbP's default PCPV.

Let T be the input CPU-time limit, k the maximum number of planners in the cluster, and n the number of single planners, combined with a set of macros, in the portfolio (in our experiment, $T = 900$ seconds, $k = 3$, and $9 \leq n \leq 38$ depending on the number of computed macro sets). We experimentally compare the performance of PbP using the following strategies for the planner cluster execution during the portfolio configuration:[11]

S1. Sequential execution of each tuple of at most k planners with $\frac{T}{k}$ seconds for the run of every planner; the number of candidate configured portfolios is $\sum_{i=1}^{k} i!\binom{n}{i}$. For this and the next (S2) strategies, when a planner terminates before the end of its time slot, the remaining time of this slot is used to (uniformly) increase the slots of the subsequently running planners.

S2. For every combination of time slots $t_{1...k}$ such that $t_i \in \{0, 90, 180, 270, 360, 450, 540, 630, 720, 810, 900\}$, $i \in \{1 \cdots k\}$ and $t_1 + \cdots + t_k = T$, sequential execution of each tuple of k planners with t_i seconds for the run of the i-th planner in the sequence; the number of candidate configured portfolios is $\sum_{i=1}^{k} \binom{n}{i} \cdot O(u^{i-1})$, where u is the number of non-zero planning time slots lower than 900 (in our experiment $u = 9$).

R1. Round-robin execution of each set of at most k planners with the planning time slots derived from the default PCPV defined in Section 3.1 (this is PbP's default scheduling strategy); the number of candidate configured portfolios is $\sum_{i=1}^{k} \binom{n}{i}$.

R2. For every PCPV $p = \langle p_1, ..., p_9 \rangle$ in set P (defined below), round-robin execution of each set of at most k planners with the planning time slots derived from p; the number of candidate configured portfolios is $\sum_{i=1}^{k} \binom{n}{i} \cdot O(s^u)$, where s is the number of increments considered for each p_i (in our experiment $s = 4$).

Set P in R2 is formed by more than 100,000 PCPVs obtained by setting each percentage in the PCPV to a value ranging from l_i to u_i, with: l_1, ..., l_9 equal to 10, 15, 20, 25, 30, 35, 40, 45, 50; u_1, ..., u_9 equal to 70, 75, 80, 85, 90, 95, 98, 99, 100; and increment step of p_i equal to $\frac{u_i - l_i}{4}$. For instance, if $i = 1$, we have that the increment step of p_1 is $\frac{70-10}{4} = 15$. Consequently, the values used for the first percentage p_1 of the considered PCPVs are 10, 25, 40, 55, 70.

Concerning the execution order of the planners in a cluster, for each considered sequence in strategies S1 and S2, the order is defined by the planner order in the sequence (two sequences formed by the same planners are considered different clusters if the planners are differently ordered or they use different time slots); for each cluster of planners in strategies R1 and R2, the execution order is determined according to the increasing planning time slots associated with the planners in the cluster (this is the default execution order strategy).

The configuration phase of PbP using each of the four scheduling strategies generates four alternative clusters of planners, with relative planning time slots, which, at planning time, are run with the same corresponding scheduling strategies that were used at configuration time. It should be noted that the portfolio configuration using strategies S2 and R2 is

11. The planners of each candidate cluster are executed by *simulation*, as described in Section 3.2.

computationally much heavier than the configuration using S1 and R1, respectively, since many more candidate configured portfolios are considered. On the other hand, since PbP with S2 and R2 examines larger portfolio configuration spaces, in principle, it could obtain more accurate configured portfolios.

Tables 18 and 19 compare the performance of PbP configured using S1-S2 and R1-R2 for solving the IPC7 domains and problems. We observed that, in terms of speed, for all IPC7 benchmark domains except Depots, the considered scheduling strategies do not affect the selection of the best cluster, since PbP.s always selects the cluster formed by a single planner (possibly using macros). For Depots, as shown in Tables 18 and 19, PbP.s with the round-robin scheduling strategies solves more problems and is faster than with the sequential scheduling strategies.

Concerning plan quality, the best cluster selected by PbP.q contains more than one planner for every IPC7 domain. Overall, the following observation can be derived:

Experimental result 4.8.1 *For the IPC7 benchmark domains and problems,* PbP.q *with R1-R2 solves more problems than* PbP.q *with S1-S2 and, in terms of plan quality, overall it performs similarly to* PbP.q *with S1-S2.*

We think that the explanation why PbP.q with R1-R2 performs better in terms of number of solved problems is that using a round-robin strategy makes PbP.q more "robust" than using a sequential strategy with respect to possible incorrect ordering of the planner runs and inadequate values of the planning time slots decided at configuration time. When the training problems are not as difficult as those used at testing time (usually they are easier), some inaccurate estimation about the effectiveness of the learned configuration knowledge can arise. An under estimation of the time slot values or an incorrect planner execution order can damage more severely the sequential execution of the planners in the selected cluster, since each of these planners is run only once, using at most the estimated time slot, while in the round-robin execution each of them is iteratively run with its (multiple) time slots, until the total CPU-time limit is reached or all planners terminate.

In terms of plan quality evaluated through the IPC quality scores, PbP.q with R1-R2 tends to perform better than PbP.q with S1-S2. The main reason is that PbP.q with R1-R2 solves more problems than PbP.q with S1-S2, and the quality score for an unsolved problem is zero. If we consider the average plan quality (last four columns of Tables 18 and 19), we observe mixed results: in two domains PbP.q with R1-R2 performs best, in two worse, and in the other ones about the same. The discrepancy in the evaluation results using quality scores and average plan qualities is only apparent, since the quality score and the average quality evaluations have different assumptions about the way they consider unsolved problems. For the average plan quality, only the subset of the test problems solved by PbP using all the compared strategies are considered; while for the quality score, all test problems are considered.

Seipp et al. (2012) show that a sequential portfolio of 21 domain-independent state-based forward planners can solve more problems when the planning time slots are uniform, rather than configured over a set of training problems, because, for the considered planners and test problems, a planner either quickly solves a problem or does not solve it at all. In our context, we observed that if we sequentially run all n planners of PbP.q (i.e., up to 38 combinations of the 9 basic planners with/without the computed sets of macros)

IPC7	Time Score of PbP.s				Problems Solved by PbP.s			
Domains	S1	S2	R1	R2	S1	S2	R1	R2
Depots	20.8	17.8	22.2	21.0	26	20	27	26

IPC7	Quality Score of PbP.q				Problems Solved by PbP.q			
Domains	S1	S2	R1	R2	S1	S2	R1	R2
Barman	30.0	30.0	30.0	30.0	30	30	30	30
Blocksworld	16.7	16.7	30.0	30.0	21	21	30	30
Depots	6.1	6.1	24.4	25.2	8	8	25	27
Gripper	29.9	29.9	28.9	28.9	30	30	30	30
Parking	3.7	4.6	4.3	4.3	4	5	5	5
Rovers	29.0	29.0	25.2	25.2	29	29	30	30
Satellite	29.8	21.0	29.5	29.7	30	21	30	30
Spanner	30.0	30.0	30.0	30.0	30	30	30	30
TPP	13.7	13.7	14.7	12.8	14	14	15	13
All domains	188.9	181.0	217.0	216.1	196	188	225	225

Table 18: Time/quality score and number of solved problems of PbP.s/q using scheduling strategies S1-S2 and R1-R2 for the IPC7 benchmark domains and problems.

using uniform time slots, then only 137 test problems are solved (against the 225 solved by PbP); the n-planners uniform strategy performs as well as PbP.q only if the CPU-time limit is increased by several times (keeping 900 seconds for PbP.q). Differently from what observed in the work of Seipp et al. (2012), our experimental evaluation includes many problems that the n planners of PbP.q solve using considerable CPU time (e.g., the number of problems that can be solved by any planner incorporated in PbP, even using macros, within 10 seconds is only 80). Probably a reason of this different behavior is that the test problems of the IPC7 learning track are on average more difficult than the problems of the IPC7 deterministic track, which are the test problems used in the work of Seipp et al. (2012).

On the other hand, if PbP sequentially runs at most 3 planners, as in strategies S1-S2, instead of all the 38 possible combinations between the incorporated planners and the learned macros, we obtain a behavior similar to that observed by Seipp et al. (2012). In particular, the results in Tables 18 and 19 show that in terms of number of solved problems and speed, configuring the planning time slots for the sequential scheduling in some cases can even degrade the performance of PbP w.r.t. using the uniform distribution of CPU time (see the results in Tables 18 and 19 of PbP.s using S1 and S2 for Depots and of PbP.q using S1 and S2 for Satellite). However, in our context the uniform distribution of CPU time over the planners in the cluster selected by PbP is not the best one, since we experimentally observed that PbP with S2 clearly outperforms PbP with S1 if the configuration is done using the *test* problems rather than the training problems. We believe that the main reason for this behavior is that in our experiment the training problems are much smaller and easier than the test problems, which in several cases makes PbP with S2 (configured with the *training* problems) underestimate the CPU times required to solve the test problems.

IPC7	Average CPU Time of PbP.s				Std. Dev. of CPU Time of PbP.s			
Domains	S1	S2	R1	R2	S1	S2	R1	R2
Depots	256.2	360.8	185.2	185.0	122.3	250.4	206.1	78.2

IPC7	Average Plan Quality of PbP.q				Std. Dev. of plan quality of PbP.q			
Domains	S1	S2	R1	R2	S1	S2	R1	R2
Barman	449.3	449.3	449.3	449.3	55.3	55.3	55.3	55.3
Blocksworld	310.3	310.3	236.7	236.7	89.1	89.1	68.6	68.6
Depots	220.0	220.0	154.3	159.5	118.2	118.2	32.5	32.8
Gripper	570.1	570.1	588.7	588.7	45.2	45.2	38.0	38.0
Parking	84.0	83.3	82.0	82.0	11.8	27.4	24.8	24.8
Rovers	583.9	583.9	703.4	703.4	158.2	158.2	194.9	194.9
Satellite	747.8	747.8	751.6	751.6	161.3	128.5	183.8	183.8
Spanner	326.0	326.0	326.0	326.0	52.1	52.1	52.1	52.1
TPP	364.0	364.0	362.6	362.6	101.4	101.4	99.1	99.1
All domains	466.8	466.8	477.8	477.6	217.9	188.9	238.4	238.8

Table 19: Average and standard deviation of the CPU time/plan quality of PbP.s/q using scheduling strategies S1-S2 and R1-R2 for the IPC7 benchmark domains and problems.

Contrary to PbP with S1-S2, PbP with R1-R2 performs similarly according to all three evaluation criteria (solved problems, speed and plan quality). This result indicates that configuring the planning time slots by considering many alternative PCPVs does not lead to high improvements with respect to using the default predefined planning time slots, that in PbP configuring the values of the planning time slots is less crucial when using a round-robin strategy than when using a sequential strategy, and that PbP with R1-R2 is less sensitive to the different size of the problems used for configuration and testing.

Experimental result 4.8.2 *For the IPC7 benchmark domains and problems,* PbP.s/q *with R1-R2 is less sensitive to the definition of the planning time slots than* PbP.s/q *with S1-S2.*

In the rest of this section, we study the problem of configuring the PCPV used to define the planning time slots in the round-robin planner scheduling of PbP. In particular, we address the following questions focusing on the IPC7 benchmarks: *how important is setting the PCPV to a particular value for a given domain? If we had an oracle specifying the best PCPV for the test problems of a specific domain, how good would the default PCPV be with respect to it?*

The data used in this analysis were obtained as follows. For each PCPV p in the set P defined as well as for the scheduling strategy R2 of the previous experiment, PbP.s/q was run using the cluster selected by simulating the round-robin scheduling with the planning time slots derived from p as described in Section 3.1. Thereby PbP.s/q was configured more than 100,000 times with different predefined PCPVs and, consequently, different predefined planning time slots. The resulting configured portfolios were then run (by simulation) over the test problems of the learning track of IPC7.

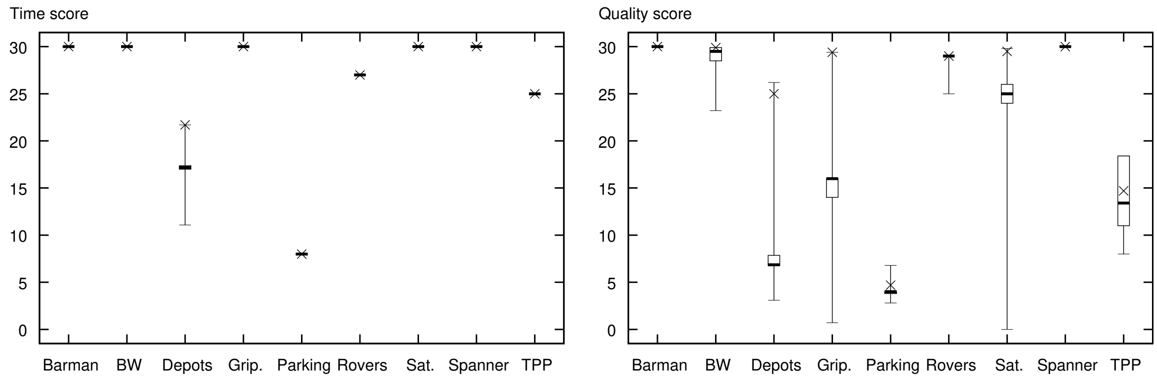

Figure 6: Distribution of the time (left plot) and quality (right plot) scores of PbP.s/q using more than 100,000 PCPVs for the IPC7 problems. `BW`, `Grip`. and `Sat`. abbreviate `Blocksworld`, `Gripper` and `Satellite`, respectively.

Figure 6 analyzes the time and quality scores of these configured portfolios through box and whisker plots. In each plot, the bottom of a whisker is the worst score; the bottom of a box is the lower quartile score; the band in a box is the median score; the top of a box is the upper quartile score; the top of a whisker is the best score; finally, each cross is the score of PbP.s/q for a domain using the default predefined PCPV. In the following, the PCPV corresponding to the configured portfolio obtaining the best time or quality score for a domain is called the *best-performing PCPV* for that domain. Since the best performing PCPV is derived from the observed performance on the *test* problems, it can be considered the best PCPV over P that an oracle would give us. From the experimental data used for Figure 6, we derive the following observation.

Experimental result 4.8.3 *Different IPC7 domains have different best-performing PCPVs for* PbP.

For each IPC7 domain where the length of the whisker in Figure 6 is zero, the cluster selected by PbP.s/q with *any* PCPV is formed by only a single planner, and hence for these cases the definition of the PCPV used to derive the planning time slots do not affect the performance of PbP (all available CPU time is assigned to the single selected planner). In the plot about speed this happens for all domains except `Depots`, while in the plot about plan quality, it happens only in domain `Spanner`. For domain `Barman`, all clusters selected by PbP.q using the configured PCPVs include SGPlan5 with a learned set of macros, and this is the only planner in the cluster finding solutions for the test problems of this domain.

For the domains in which PbP.s/q does not always select the same singleton planner cluster for all the PCPVs considered, the specific used PCPV can have a high impact on PbP's performance, as shown especially for domains `Depots`, `Gripper` and `Satellite` in the quality-score plot of Figure 6. Interestingly, we can observe that the default predefined PCPV used in PbP.s/q is generally a good choice, since very often the crosses in the plots appear at (or near to) the top position of the corresponding whiskers.

Experimental result 4.8.4 *For every IPC7 domain, the cluster selected by* PbP.s/q *using the default PCPV* ⟨25, 50, 75, 80, 85, 90, 95, 97, 99⟩ *performs similarly to* PbP.s/q *using the best-performing PCPV, except for* PbP.q *in domains* `Parking` *and* `TPP`*.*

For `Parking`, the best performance is obtained by running planners FF and LAMA with PCPV equal to $\langle 10, 15, 60, 65, 70, 75, 80, 95.5, 96.5\rangle$; for `TPP`, it is obtained by running planners LAMA, Macro-FF and SGPlan5 with PCPV equal to $\langle 10, 15, 20, 25, 30, 35, 40, 45, 50\rangle$. For these two domains, PbP.q with the default PCPV does not perform as well as with the best-performing PCPV (but still better than the median-performing PCPV). The main reason is that for these domains the IPC7 test problems are much larger (and harder) than those used for the training, which, as also observed in Section 4.6, can affect the accuracy of the portfolio configuration for the test problems in terms of the selected planner cluster and configured PCPVs.

Overall, the results of the experiment about configured and default PCPVs for PbP indicate that, if the round-robin planner scheduling is used, tuning the PCPV (and consequently the planning time slots) for a specific IPC7 domain does not greatly improve the performance of the resulting configured portfolio, since very often the default PCPV performs as well as the best PCPV specified by an oracle. Consequently, given that without the PCPV tuning the portfolio configuration is much simpler and faster, PbP uses the default version.

5. Conclusions

The existing automated-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 planning domain. From a practical perspective, it is then useful to consider a portfolio-based approach to planning that involves several techniques and planners. In this paper, we have proposed an approach to automatically configuring a portfolio of planners and learned macros for any given domain, that is implemented in the portfolio-based planner PbP. The computed configuration knowledge consists of a promising combination of basic planners in the portfolio, each one with a (possibly empty) set of useful macros, and some scheduling information for specializing their execution at planning time. The configured portfolio is obtained through an automated statistical analysis about the performance of a set of candidate clusters of planners and relative candidate sets of macros, using a collection of training problems in the given domain. The planner cluster performance is computed by simulating the cluster execution using the performance data from the runs of the individual basic planners (and relative sets of macros) in the portfolio.

The proposed approach to the portfolio planner configuration has been evaluated through a large experimental analysis, focusing on the IPC6-7 domains, with the aim of demonstrating its high efficiency, understanding the effectiveness of the automatic configuration, and investigating the importance of the main design choices. Several results have been derived from the various experiments of this analysis. The most important experimental results indicate that:

- the configured planner portfolios generated by PbP.s/q perform very well compared to other state-of-the-art planning systems using learning techniques, and much better than PbP-nok, i.e., the unconfigured planner portfolio of PbP (which is competitive with LAMA, a state-of-the-art domain independent planner);

- PbP.s/q performs much better than the other existing domain-independent portfolio-based planners, and often better than other domain-optimized planner portfolio approaches;
- the computed configuration knowledge is very useful and the selection of the planner cluster forming the configured portfolio is generally accurate for the given planning domain;
- while macros in a planning domain are not always helpful to a planner for improving its planning speed or plan quality, PbP.s/q generally selects helpful sets of macros;
- in the context of the proposed approach, the round-robin scheduling strategy of the planner cluster execution is a robust strategy with respect to the execution order of the cluster planners and their planning time slots; moreover, configuring the planning time slots is not crucial given the good default technique for deriving them currently implemented in PbP.s/q.

Besides evaluating the approach of PbP to configuring a planner portfolio with macros, the experimental analysis corroborates and validates some results, observations or empirical studies in previous work on other researchers in planning. These include the usefulness or harmfulness of macros for a set of prominent existing planners, the importance of diversity of the planning techniques in the construction of an effective planner portfolio, and the robustness of the round-robin scheduling of the execution times in a multi-planner system.

While the current version of PbP uses a portfolio formed by a specific set of selected techniques for plan synthesis, computation of macros and planner-parameter tuning, the architecture of PbP is open in the sense that additional or alternative (current or future) techniques can be integrated. Moreover, although we have chosen the Wilcoxon sign-rank test for comparing the candidate planner clusters and macro sets, demonstrating its effectiveness, other methods could be considered.

A limit of the current approach, which affects also other systems relying on knowledge learned from examples, is that when the training problem set is not representative of the test problems (e.g., most problems are much smaller or easier than the test problems), the computed portfolio configuration might not be very accurate for these problems. Knowing at configuration time “enough” information characterizing the test problems can obviously be very useful for generating representative training problem sets. For planning with PbP, we experimentally observed that, when the minimum/maximum number of objects involved in test problems is known, randomly generated training problem sets under these object bounds are sufficiently representative for an effective configuration of PbP.

We think that in future work it will be important to study and incorporate into PbP additional methods supporting the problem-based configuration of the portfolio planner. Such methods could refine the current domain-based configuration so that problems with different size or heuristically estimated hardness can have different, specialized configured portfolios. Moreover, it will also be important to extend PbP.q so that plan quality is measured in terms of plan action costs rather than number of plan actions.

Other directions for further research are investigating the use of PbP.s/q for optimal planning and for metric-temporal domains (Fox & Long, 2003), and extending the portfolios with additional automatically extracted domain-specific knowledge, such as entanglements (Vallati et al., 2013a). Finally, we intend to investigate the idea of making PbP fully domain-independent by computing many portfolio configurations (planner clusters) for different known domains, and using a classifier to match a new domain with the most promising stored configuration in terms of expected performance for the new domain. A similar idea was successfully developed for SAT (e.g., Xu et al., 2008).

Acknowledgments

Many ideas, techniques, and systems investigated in the paper use or build on important previous work in planning and portfolio design, without which our research would have not been possible. We thank all authors of such work, and in particular the authors of the planning systems and macro generators incorporated in PbP. A special thank to Mark Roberts and Adele Howe for clarifications on the configuration of their planner portfolio, to Beniamino Galvani for his help with the implementation of part of a preliminary version of PbP.s, and to the IPC7 organizers for letting us use the competition machine in one of the experiments conducted after the the competition. We would also like to thank the organizers of IPC6 and IPC7 for having developed and made available a large collection of useful benchmark domains, problems and some software tools that we used in our analysis. Finally, we thank the anonymous Reviewers and the Associate Editor for their very helpful and detailed comments.

References

Arfaee, S., J., Zilles, S., & Holte, R., C. (2010). Bootstrap learning of heuristic functions. In *Proceedings of the Third Annual Symposium on Combinatorial Search (SOCS-10)*, pp. 52–60. AAAI Press.

Bäckström, C., & Nebel, B. (1995). Complexity results for SAS+ planning. *Computational Intelligence*, *11*(4), 1–34.

Botea, A., Enzenberger, M., Müller, M., & Schaeffer, J. (2005). Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, *24*, 581–621.

Botea, A., Müller, M., & Schaeffer, J. (2007a). Fast planning with iterative macros. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence IJCAI-07*, pp. 1828–1833. AAAI Press.

Botea, A., Müller, M., & Schaeffer, J. (2007b). Learning partial-order macros from solutions. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS-05)*, pp. 231–240. AAAI Press.

Brendel, M., & Schoenauer, M. (2011). Instance-based parameter tuning for evolutionary AI planning. In *Proceedings of the Thirteenth Annual Genetic and Evolutionary Computation Conference (GECCO-11)*, pp. 259–260. ACM.

Cenamor, I., de la Rosa, T., & Fernàndez, F. (2013). Learning predictive models to configure planning portfolios. In *Proceedings of the ICAPS-13 Workshop on Planning and Learning.*

Chen, Y., Hsu, C., & Wah, B. (2006). Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research*, *26*, 323–369.

Chrpa, L., & Barták, R. (2009). Reformulating planning problems by eliminating unpromising actions. In *Proceedings of the Eighth Symposium on Abstraction, Reformulation, and Approximation, (SARA-09)*, pp. 50–57. AAAI press.

Chrpa, L., & McCluskey, T., L. (2012). On exploiting structures of classical planning problems: Generalizing entanglements. In *Proceedings of the Twentieth European Conference on Artificial Intelligence (ECAI-12)*, pp. 240–245. IOS Press.

Chrpa, L., McCluskey, T., & Osborne, H. (2012). Reformulating planning problems: A theoretical point of view. In *Proceedings of the Twenty-Fifth International Florida Artificial Intelligence Research Society Conference (FLAIRS-12)*, pp. 14–19. AAAI Press.

Coles, A., & Coles, A. (2011). LPRPG-P: Relaxed plan heuristics for planning with preferences. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS-11)*, pp. 26–33. AAAI Press.

Coles, A., Coles, A., Olaya, A., Celorrio, S., López, C., Sanner, S., & Yoon, S. (2012). A survey of the seventh international planning competition. *AI Magazine*, *33*(1).

Coles, A., & Smith, K., A. (2007). Marvin: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research*, *28*, 119–156.

Cormen, T. H., Stein, C., Rivest, R. L., & Leiserson, C. E. (2001). *Introduction to Algorithms* (3rd edition). McGraw-Hill.

Fawcett, C., Helmert, M., Hoos, H., Karpas, E., Röger, G., & Seipp, J. (2011). FD-Autotune: Domain-specific configuration using Fast Downward. In *Proceedings of the ICAPS-11 Workshop on Planning and Learning*.

Fawcett, C., Vallati, M., Hutter, F., Hoffmann, J., Hoos, H., H., & Leyton-Brown, K. (2014). Improved features for runtime prediction of domain-independent planners. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 355–359. AAAI Press.

Fern, A., Khardon, R., & Tadepalli, P. (2011). The first learning track of the international planning competition. *Machine Learning*, *84*(1), 81–107.

Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, *20*, 61–124.

Gerevini, A., Haslum, P., Long, D., Saetti, A., & Dimopoulos, Y. (2009). Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, *173*(5-6), 619–668.

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

Gerevini, A., Saetti, A., & Serina, I. (2006). An approach to temporal planning and scheduling in domains with predictable exogenous events. *Journal of Artificial Intelligence Research*, *25*, 187–231.

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

Gibbons, J., & Chakraborti, S. (2003). *Nonparametric Statistical Inference, Fourth Edition: Revised and Expanded*. Statistics: A Series of Textbooks and Monographs. CRC Press.

Gomes, C., P., & Selman, B. (2001). Algorithm portfolios. *Artificial Intelligence*, *126*(1-2), 43–62.

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

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

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

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Howe, A., Dahlman, E., Hansen, C., vonMayrhauser, A., & Scheetz, M. (1999). Exploiting competitive planner performance. In *Proceedings of the Fifth European Conference on Planning (ECP-99)*, pp. 62–72. Springer.

Howey, R., Long, D., & Fox, M. (2004). VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *Proceedings of the Sixteenth IEEE International Conference on Tools with Artificial Intelligence (ICTAI-04)*, pp. 294–301. IEEE.

Hutter, F., Hoos, H., H., & Stützle, T. (2007). Automatic algorithm configuration based on local search. In *Proceedings of the Twenty-second National Conference on Artificial Intelligence (AAAI-07)*, pp. 1152–1157. AAAI Press.

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2009). ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, *36*, 267–306.

Jiménez, S., C., Coles, A., & Coles, A. (2011). Seventh International Planning Competition IPC7 – learning part. In *http://www.plg.inf.uc3m.es/ipc2011-learning*.

Kautz, H., A., & Selman, B. (1999). Unifying SAT-based and graph-based planning. In *Proceedings of the Sixteenth International Joint Conferences on Artificial Intelligence (IJCAI-99)*, pp. 318–325. AAAI Press.

Long, D., & Fox, M. (2003). The third International Planning Competition: Results and analysis. *Journal of Artificial Intelligence Research*, *20*, 1–59.

Marquardt, D., W., & Snee, D. (1975). Ridge regression in practice. *The American Statistician*, *29(1)*, 3–20.

Matos, P., Planes, J., Letombe, F., & Marques-Silva, J. (2008). A MAX-SAT algorithm portfolio. In *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI-08)*, pp. 911–912. IOS Press.

McCluskey, T., L., & Porteous, J., M. (1997). Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence*, *95*(1), 1–65.

Newton, M., Levine, J., Fox, M., & Long, D. (2007). Learning macro-actions for arbitrary planners and domains. In *Proceedings of the Seventeenth International Conference on Automated Planning & Scheduling (ICAPS-07)*, pp. 256–263. AAAI Press.

Pulina, L., & Tacchella, A. (2007). A multi-engine solver for quantified boolean formulas. In *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP-07)*, pp. 574–589. Springer.

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

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

Roberts, M., & Howe, A. (2006). Directing a portfolio with learning. In *Proceedings of the AAAI 2006 Workshop on Learning for Search*, pp. 129–135.

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

Roberts, M., & Howe, A. (2009). Learning from planner performance. *Artificial Intelligence*, *173*(5-6), 536–561.

Roberts, M., & Howe, A. (2012). Personal communication. December 14.

Roberts, M., Howe, A., E., Wilson, B., & desJardins, M. (2008). What makes planners predictable?. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-08)*, pp. 288–295. AAAI Press.

Seipp, J., Braun, M., Garimort, J., & Helmert, M. (2012). Learning portfolios of automatically tuned planners. In *Proceedings of the Twenty-second International Conference on Automated Planning & Scheduling (ICAPS-12)*, pp. 368–372. AAAI Press.

Shaffer, J., P. (1995). Multiple hypothesis testing. *Annual Review of Psych*, *46*, 561–584.

Simon, H., & Kadane, J. (1975). Optimal problem-solving search: All-or-none solutions. *Artificial Intelligence*, *6*, 235–247.

Vallati, M., Chrpa, L., & Kitchin, D. (2013a). An automatic algorithm selection approach for planning. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 1–8. IEEE.

Vallati, M., Fawcett, C., Gerevini, A., Hoos, H., & Saetti, A. (2013b). Automatic generation of efficient domain-optimized planners from generic parametrized planners. In *Proceedings of the 6th Annual Symposium on Combinatorial Search (SOCS-13)*, pp. 184–192. AAAI Press.

Vidal, V. (2004). A lookahead strategy for heuristic search planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS-04)*, pp. 150–159. AAAI Press.

Wilcoxon, F., & Wilcox, R., A. (1964). *Some Rapid Approximate Statistical Procedures*. American Cyanamid Co., Pearl River, N.Y.

Witten, I., H., & Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco.

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

Yoon, S., Fern, A., & Givan, R. (2008). Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, *9*, 683–718.