Learning and Exploiting Configuration Knowledge for a Portfolio-based Planner

Alfonso E. Gerevini and **Alessandro Saetti** and **Mauro Vallati**
Dipartimento di Elettronica per l'Automazione, Università degli Studi di Brescia,
Via Branze 38, 25123 Brescia, Italy
{gerevini,saetti,mauro.vallati}@ing.unibs.it

Abstract

In the recent years the field of automated plan generation has significantly advanced and several powerful domain-independent planners have been developed. However, no one of these systems clearly outperforms all the others in every known benchmark domain. It would then be useful to have a multi-planner system capable of automatically selecting and combining the most efficient planning technique(s) for each given domain. In this paper we propose a planner, called PbP (Portfolio-based Planner), which automatically configures a portfolio of existing planners, possibly using a useful set of macro-actions for each of them. The configuration relies on some knowledge about the performance of the planners in the portfolio and the observed usefulness of sets of macro-actions, which is automatically generated by a statistical analysis considering a set of training problems for the domain under consideration. The configuration knowledge for the given domain consists of a promising combination of planners in the portfolio, each one with a (possibly empty) set of macro-actions, and additional information specializing their round-robin scheduling at planning time. PbP has two variants, one focusing on speed (PbP.s) and one on plan quality (PbP.q). A preliminary version of PbP.s entered the learning track of the sixth IPC, and was the overall winner of this competition track. An experimental analysis presented in the paper confirms the effectiveness of PbP.s, indicates that PbP.q performs better than the IPC6 planners, shows that the learned configuration knowledge can be very useful for PbP.s/q, and demonstrates that PbP.s/q can perform much better than the basic planners forming the portfolio.

Introduction

The field of automated plan generation has recently significantly advanced. However, while several powerful domain-independent planners have been developed, no one of these clearly outperforms all the others in every known benchmark domain. It would then be useful to have a multi-planner system that automatically selects and combines the most efficient planner(s) 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. In many domains, the planning performance can be improved by deriving and exploiting knowledge about the domain structure that is not explicitly given in the input formalization. In particular, several approaches encoding additional knowledge in the form of macro-actions have been proposed, e.g., (Armano, Cherchi & Vargiu 2003; Botea *et al.* 2005; Coles & Smith 2007; Newton *et al.* 2007). A macro-action (or macro) is a sequence of actions that can be planned at one time like a single action. When using macro-actions there is an important tradeoff to consider. While their use can speedup the planning process, because it reduce the number of search steps required to reach a solution, it also increases the search space size and this could slow down the planning process. Moreover, the effectiveness of a set of macro actions can depend on the particular planner using it.

In this paper we propose a planner, called PbP (*Portfolio-based Planner*), which automatically configures a portfolio of domain-independent planners. The configuration relies on some knowledge about the performance of the planners in the portfolio and the observed usefulness of automatically generated sets of macro-actions. This configuration knowledge is "learned" by a statistical analysis and consists of: an ordered selected subset of the planners in the initial portfolio, which at planning time are combined through a round-robin strategy; a set of useful macro-actions for each selected planner; and some sets of planning time slots. A planning time slot is an amount of CPU-time to be allocated to a selected planner (possibly with a set of macro-actions) during planning.

When PbP is used without this additional knowledge, all planners in the portfolio are scheduled by a simple round-robin strategy where predefined and equal CPU-time slots are assigned to the (randomly ordered) planners. When PbP uses the knowledge computed for the domain under consideration, only the selected cluster of planners (and relative sets of macro actions) is scheduled, their ordering favors the fastest planners for the domain under consideration, and the planning time slots are defined by the learned knowledge.

It should be noted that in our framework the computed macro-actions are not always used by the planners that PbP selects. Assume that a planner P is in the cluster of planners selected by PbP for solving problems in a domain D. If in the learning phase PbP observes that a set of macro-actions computed for P does not improve the performance of the selected cluster for the training problems in D, then

the learned configuration does not include these macro actions for P.

PbP has two variants: PbP.s focusing on speed, and PbP.q focusing on plan quality. A preliminary implementation of PbP.s entered the learning track of the sixth international planning competition (IPC6) and was the overall winner of this competition track (Fern, Khardon & Tadepalli 2008). However, as observed by the IPC6 organizers, for the IPC6 problems the use of the learned knowledge does not speedup the competition version of PbP.s significantly. This behaviour depends on some implementation bugs contained in the preliminary version of PbP.s concerning both the learning phase and the planning phase, and on the inefficient use of some Linux shell scripts (evident especially for small or easy problems).

Recently, we have developed a new version of PbP.s that is completely implemented in C, does not contain these bugs and, furthermore, extends the set of the integrated basic planners with Lama (Richter & Westphal 2008), the planner which won the deterministic track of IPC6. In the rest of the paper, PbP.s denotes this newer version.

This paper contains an experimental analysis about PbP.s/q, which

- confirms the good performance of PbP.s and indicates that PbP.q performs better than the IPC6 planners,
- shows that the learned configuration knowledge is useful for PbP.s/q,
- indicates that PbP.s/q performs better than the integrated planners,
- suggests that our approach effectively configures the portfolio w.r.t. other possible configurations, as well as the use of the computed macro-actions for the selected planners in the portfolio.

The paper is organized as follows. The 2nd section briefly presents the related work; the 3rd section describes our planning approach; the 4th section presents the results of our experimental study; finally, the 5th section gives the conclusions.

Related Work

The idea of configuring and using a portfolio of techniques has been investigated by several researchers in automated reasoning e.g., (Gomes & Selman 2001; Huberman, Lukose & Hogg 1997). Recently, Xu and collaborators (Xu *et al.* 2008) have proposed SATzilla, an automated approach for configuring a portfolio of SAT solvers, which is based on some empirical problem hardness models.

Regarding automated planning, Blackbox (Kautz & Selman 1999) can solve problems using a variety of satisfiability engines. Moreover, several state-of-the-art planners, e.g., (Hoffmann & Nebel 2001; Gerevini, Saetti & Serina 2003; Chen, Hsu & Wah 2006), include a "backup strategy" using an alternative search technique which is run when the main default method fails.

Vrakas and collaborators (Vrakas *et al.* 2003) propose some machine learning techniques for discovering additional knowledge that associates some features of the planning problems with specific values for the parameters of their planning system. The knowledge learned by PbP is different, and is used for configuring a portfolio of domain-independent planners.

Planner	Authors, date
FD	Helmert, 2006
Lama	Richter & Westphal, 2008
LPG-td	Gerevini, Saetti & Serina, 2005
Macro-FF	Botea, Enzenberger, Müller & Schaeffer, 2005
Marvin	Coles & Smith, 2007
Metric-FF	Hoffmann & Nebel, 2001
SGPlan5	Chen, Wah & Hsu, 2006
YAHSP	Vidal, 2004

Table 1: Domain-independent planners currently integrated into PbP.

The work that is most closely related to PbP is the system developed by Howe & collaborators (Howe *et al.* 1999; Roberts & Howe 2007). Like PbP, this system solves planning problems by a round-robin scheduling of domain-independent planners, but the learned knowledge is very different: the knowledge learned by PbP is specific for the problems of a given domain, while the knowledge learned by the Howe & collaborators' system is domain-independent and is used, at planning time, to configure the portfolio according to some features of the planning problem under consideration. Moreover, the methods for selecting and ordering the incorporated planners are different. Their system selects a set of planners through a set covering algorithm over the solved training problems, and orders them by exploiting some learned performance models. In PbP the selection is based on a statistical analysis comparing the CPU-times and plan qualities of the planners for the training problems in the given domain, and the ordering is based on the computed planning time slots. Finally, their system does not compute, analyze or use macro-actions, and does not consider plan quality.

The Portfolio-based Planner (PbP)

In this section, we give an overview of PbP's architecture and of the proposed implemented methods for selecting a cluster of planners and macro-actions for an input domain.

The PbP Architecture

Table 1 shows the eight planners currently integrated into PbP. The architecture of PbP, sketched in Figure 1, consists of five main components, which are briefly described below.

Macro-actions computation. For each integrated planner, PbP computes some sets of macro-actions using the following two approaches.

- Wizard (Newton *et al.* 2007). This system implements an offline evolutionary method, which computes macros by genetic operators from plans for a set of training problem instances of an input domain. The computed macro-actions are added to the domain formalization as additional actions, and hence they can be used by all the planners incorporated in PbP. With this approach, PbP pro-

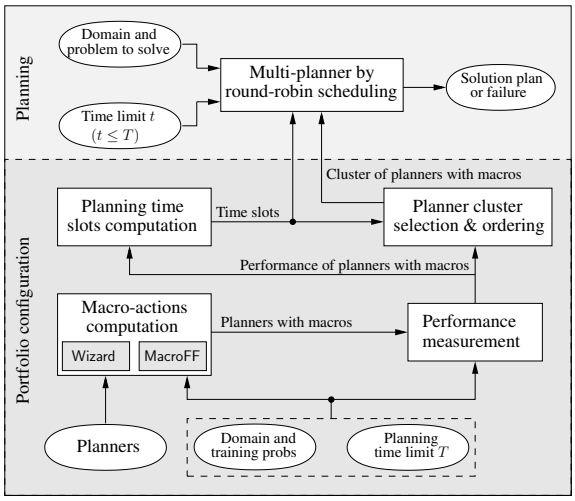

Figure 1: A sketch of PbP's architecture.

duces at most two alternative sets of macro-actions for each considered planner.

- Macro-FF (Botea *et al.* 2005; Botea, Müller & Schaeffer 2007). The approach incorporated into the Macro-FF system (Botea *et al.* 2005) computes the macros by analyzing the solutions of a set of training problem instances, so that the macros that appear frequently and that reduce the required search effort significantly are preferred. This version of the approach integrated into PbP contains the enhancements described in (Botea, Müller & Schaeffer 2005; 2007). With this approach, PbP produces at most five sets of alternative macro-actions for Macro-FF.

Performance measurement. This is the most expensive computation step in the configuration of the portfolio. PbP runs each integrated planner with and without the sets of learned macro-actions for the input training problems and the planning CPU-time limit T, measuring their performance in terms of: number of problems solved within T, CPU-time required for the solved training problem, and quality of the computed solutions. For the incremental planners, i.e., LPG and Lama, PbP measures the quality of all the solutions generated for a problem and the corresponding CPU-times.[1]

Planning time slots computation. For each integrated planner, PbP defines the planning time slots as the CPU-times used to solve the following percentages of problems during the learning phase: $\{25,\ 50,\ 75,\ 80,\ 85,\ 90,\ 95,\ 97,\ 99\}$. A similar method is also used in the round-robin scheduling defined in (Roberts & Howe 2007), but with the technical difference explained in the following example. Assume that the computed planning time slots for planner A are $\{0.20, 1.40, 4.80, 22.50, \ldots\}$ and that those for planner B are $\{14.5, 150.8, \ldots\}$. Then, for this pair of planners, PbP extends the first time slot for A (0.20) to 4.80, i.e., to the greatest time slot of A which is smaller than the first time slot of B; similarly for the subsequent time slots. If the first time slot of A were not extended, the slowest planner B would initially run for a CPU-time much greater than the CPU-time initially assigned to the fastest planner A, and for many problems that planner A quickly solves (e.g., using one CPU-seconds), PbP would perform significantly slower.

Planner cluster selection & ordering. PbP selects a cluster of planners in the initial portfolio, each one with a (possibly empty) set of useful macro-actions, according with the measured performance and the computed planning time slots. Moreover, the execution order of the planners in the selected cluster is defined by the increasing CPU-time slots associated with the planners. More on this in the next section of the paper.

Multi-planner by round-robin scheduling. PbP runs the selected ordered planners (each one using the relative selected set of macro-actions) by a round-robin scheduling algorithm using the computed planning time slots for an input (test) problem. We developed two versions of PbP, called PbP.s and PbP.q. Concerning termination of the resulting multi-planner, PbP.s is interrupted if either a given CPU-time limit T is exceeded (returning failure), or *one* among the selected planners computes a solution (output of PbP.s). PbP.q's execution is interrupted 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.

Selecting a Cluster of Planners and Macro-actions

At configuration time, for each given test domain, PbP considers the execution of all the integrated planners, and selects a subset of them on the basis of their observed performance for a training problem set in the domain and a given CPU-time limit t.

After having run each planner with every computed set of macro-actions (one run for each set) for the training problem set of domain D and for CPU-time limit T, PbP analyzes the results (CPU-times and plan qualities) to identify the best cluster of planners and macro-actions for D and T. This is done by *simulating*, for each cluster C of at most k planners, each with a (possibly empty) set of macro-actions, the round-robin execution of the planners in C for solving the same training problems within T.[2] The simulation is conducted using the data from the previous runs (the planners are not re-run), possibly ignoring the data of the planners that always performs worse than another incorporated planner, and the simulated performances of the clusters are compared by a statistical analysis based on the Wilcoxon sign-rank test (also known as the "Wilcoxon matched pairs test") (Wilcoxon & Wilcox 1964). The Wilcoxon test has also been used in (Long & Fox 2003; Gerevini *et al.* 2009; Roberts & Howe 2009), but for different purposes. The first two papers contain details about the test and a discussion on its adequateness for comparing planner performances.

[1] An incremental planner produces a sequence of solutions with increasing plan quality which are generated with increasing CPU times.

[2] k is a parameter that in our experiments was set to 3.

Domain	PbP.s	PbP.q
D1	YAHSP (63), FF (33), SGP (33)	LPG (0), Lama (100), MFF (20)
D2	Lama (–), Marvin (62.5)	Marvin (0), Lama (–), LPG (100)
D3	FF (–), FD (–)	Lama (–), LPG (0), FD (–)
D4	FF (0)	Lama (–), FF (0)
D5	SGP (50), LPG (100)	LPG (50), MFF (40)
D6	FF (–), YAHSP (–)	Lama (–) + MFF (100), Marvin (–)

Table 2: Planner clusters and percentages of macros (in round brackets) selected by PbP for the IPC6 domains Gold-miner (D1), Matching-BW (D2), N-puzzle (D3), Parking (D4), Sokoban (D5) and Thoughful (D6). MFF and SGP abbreviate Macro-FF and SGPlan5, respectively. "–" indicates that no macro-action was generated.

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 (15 minutes, as in IPC6); 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 favour 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 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 test is approximately a normal distribution, which is characterised 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. PbP uses a default confidence level equal to 99.9%; hence, if the p-value is greater than 0.001, then the hypothesis that the performance of the compared sets 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 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. 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 include 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.

The method used by PbP.q is similar, but it applies to the plan qualities resulting from the cluster execution simulation. For this simulation, PbP.q also considers 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, consider the cluster $\{\mathsf{FF}, \mathsf{Lama}\}$, and assume that the CPU-time of the last solution computed by Lama for a problem p is close to the limit T. If the intermediate solutions of Lama were ignored, the estimated plan quality for $\{\mathsf{FF}, \mathsf{Lama}\}$ would be equal to the quality of the plan generated by FF,[3] although the quality of the intermediate solutions of Lama could be much better than the quality of the plan computed by FF.

Finally, note that if the performances of the incorporated planners are measured with CPU-time limit T, then the portfolio of PbP.s/q can be 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.

Table 2 shows the clusters of planners and the relative percentages of macros selected by PbP for each IPC6 domain (Fern, Khardon & Tadepalli 2008), considering all macros computed for that domain. Interestingly, for the IPC6-domains the portfolio configuration resulting from the learned knowledge is variegated and all the eight basic planners are helpful (since each of them is selected by PbP.s/q at least once). Moreover, sometimes PbP.s/q uses (a portion of) the computed macros and sometimes it does not use them at all.

Experimental Results

In this section, we present the results of an experimental study about PbP with four main goals:

- Evaluating the efficiency of PbP.s/q in terms of speed and plan quality by comparing them with the planning approaches that entered IPC6;
- Testing the effectiveness of the learned configuration

[3] If Lama is run together with FF, the total running time of Lama can be much less than the CPU-time limit T, and hence it does not have enough time to compute the last solution generated when T CPU-time is available.

Planner	Solved (%)	Time score (max = 180)	Quality score (max = 180)
CABALA	1.67	0.003	2.40
DAE1	18.3	0.005	23.7
DAE2	18.3	0.008	23.5
MacroAltAlt	28.9	4.771	37.3
ObtuseWedge	65.0	74.80	76.2
PbP.s	**95.6**	**110.1**	**110.9**
PbP.q	**95.0**	8.5	**167.0**
REPLICA	31.7	4.581	19.4
RFA1	47.2	14.87	52.4
RFA2	26.1	4.11	29.1
Roller	30.6	5.31	19.4
Sayphi-Rules	26.1	1.719	20.7
Wizard+FF	56.7	31.89	72.7
Wizard+SGPlan	51.1	32.69	65.8

Table 3: Percentages of solved problems within the IPC6 CPU-limit (15 min), scores for the time and plan quality of the planners that took part in the learning track of IPC6.

knowledge for PbP.s/q in terms of speed and plan quality;

- Showing that, overall, there is no single basic planner incorporated in PbP that performs better than PbP.s in terms or speed or better than PbP.q in terms of plan quality;
- Evaluating the accuracy of the strategy used by PbP.s/q for configuring the portfolio of the considered planners with respect to other possible configurations, and the usage of the computed macros for the planners in the selected cluster.

The set of problems we used for learning the domain knowledge and the set of problems used for testing PbP are disjoint. The experimental analysis uses the benchmark problems of the learning track of IPC6 and a collection of (hard) problems in the well-known `Depots` domain (Long & Fox 2003).

- The IPC6 benchmarks are 720 problems in six domains: `Gold-miner`, `Matching-bw`, `N-Puzzle`, `Parking`, `Sokoban` and `Thoughtful`. A description of these domains can be found in (Fern, Khardon & Tadepalli 2008). We used 540 of these problems for the learning phase and 180 problems for testing phase.
- `Depots` is a domain about moving crates between locations by trucks, and stacking crates onto pallets by hoists. For each n in $\{5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$, we randomly generated 105 instances of the `Depots` problem with 7 locations, 3 trucks, 7 hoists and n crates; five of these 105 instances have been used for learning the knowledge.

The experimental tests were conducted on an Intel Xeon(tm) 2.6 GHz machine, with 3 Gbytes of RAM. Unless otherwise specified, as in IPC6, the CPU-time limit was 15 minutes.

IPC6 Domains	#Probs	#Probs solved by			
		PbP.s +k	PbP.s nok	PbP.q +k	PbP.q nok
`Gold-miner`	30	30	30	30	30
`Matching-BW`	30	29	26	30	26
`N-puzzle`	30	29	26	29	26
`Parking`	30	25	19	24	19
`Sokoban`	30	30	30	30	30
`Thoughful`	30	29	29	28	29
Total	180	**172**	160	171	160

Table 4: Total number of test problems (2nd column), number of problems solved by PbP.s with/out the learned knowledge (3rd/4th column), and number of problems solved by PbP.q with/out the knowledge (5th/6th column) for the IPC6 domains.

PbP versus the IPC6 Planners

In order to show the efficiency of our approach, we compare the performance of PbP using the learned knowledge with the performance of the planners that entered IPC6.[4] Since several IPC6 planners are not available, for this analysis we consider the official competition data, even though the machine that we used for testing PbP is slightly slower than the machine used for IPC6.

Table 3 gives experimental results for the percentage of solved problems, time and plan quality. The performance scores in the table were obtained by the same simple evaluation function used for IPC6. If a planner P solves a problem π with time T (with plan quality Q), it gets a score equal to T^*/T (Q^*/Q), where T^* is the best time required by the planners under comparison to solve π (Q^* is the smallest number of actions in the plans computed for π); otherwise, it gets zero score. The time (quality) score for planner P is the sum of the time (quality) scores assigned to P over all the competition problems. The results in Table 3 indicate that PbP.s with the knowledge is much faster than the IPC6 planners (PbP.s solves many more problems, and also has much greater time and quality scores) and that PbP.q with the knowledge is clearly the best planner in terms of plan quality.

Remarkably, PbP.s/q solves almost all IPC6 benchmark problems within 15 CPU-minutes, and PbP.q almost always computes the best plan. In contrast, the time score of PbP.q is low, since PbP.q usually runs more than one planner and stops only when all the selected planners terminate or the CPU-time limit is exceeded.

On the Effectiveness of the Learned Knowledge

Table 4 shows the number of problems solved by PbP.s/q with and without the learned configuration knowledge for the IPC6 test problems.[5]

[4] A collection of short papers describing the IPC6 planners of the learning track can be found in (Fern, Khardon & Tadepalli 2008).

[5] When PbP is used without this additional knowledge, all planners in the portfolio are selected with no macro, equal CPU-time slots and random execution order.

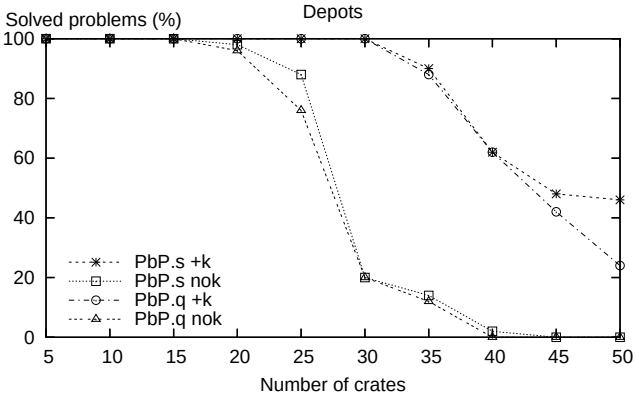

Figure 2: Percentage of solved problems of PbP.s with/out using the learned knowledge w.r.t. an increasing numbers of crates for Depots domains.

The performance gaps for the considered CPU-time limit are small. However, we note that with the 15 CPU-minutes limit, many IPC6 problems are inadequate to observe possible problem-solved improvements in PbP, because they can be quickly (relative to the limit) solved by most of the incorporated basic planners. We observed that, for CPU-time limits smaller than 15 minutes, PbP.s with the learned knowledge solves many more problems than without the knowledge. For instance, the percentage of problems solved by PbP.s in 10 seconds is 71, while without the knowledge it is only 39.

Moreover, we observed significant performance gap even for large CPU-time limits when testing PbP with a much harder set of problems. Figure 2 shows an example of such a gap for our collection of hard Depots problems. The results indicate that the learned knowledge can be very useful for PbP.s/q.

We now analyze the impact of using the learned knowledge for speed and plan quality. Figure 3 gives an overall picture of the performance of PbP.s/q with and without the learned knowledge using CPU-time limits ranging from 1 second to 1000 seconds. For every considered CPU-time limit, PbP.s with knowledge is much faster than without it. In contrast, for large CPU-time limits, there is no significant difference between PbP.s with and without knowledge in terms of plan quality. This result indicates that for PbP.s the learned knowledge is very helpful in terms of speed, while it does not help to improve plan quality considerably (but the second observation is not surprising, since this knowledge is generated ignoring plan quality).

Concerning the results for PbP.q using the IPC6 problems in Figure 3, we observe that in terms of plan quality PbP.q with the learned knowledge always perform better than without it. While the performance improvements are generally clear, they are much less strong than those observed for PbP.s in terms of speed. The main reason is that many of these problems can be quickly solved by most of the incorporated basic planners (and incrementally optimized by LPG and Lama), and PbP.q without knowledge runs all them. However, there are domains and problems for

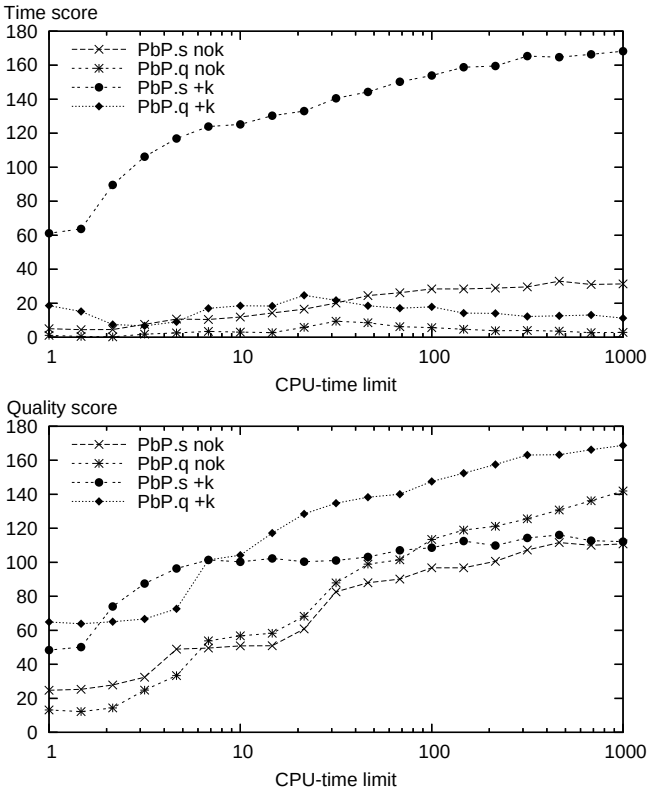

Figure 3: Time scores (upper plot) and plan quality scores (bottom plot) of PbP.s/q with/out the learned knowledge w.r.t. an increasing CPU-time limit (ranging from 1 to 1000 seconds) for the IPC6 domains.

PbP.q	5	10	15	20	25	30	35
+k	21.6	40.5	62.3	88.5	118	174	194
nok	21.7	40.9	63.2	101	140	191	213

Table 5: Average number of actions in the plans computed by PbP.q using the learned knowledge ("+k" line) and PbP.q without the knowledge ("nok" line) for Depots.

Planner	ΔSolved (%)	ΔTime	ΔQuality
DAE1	+17.2	±0.0	+21.8
DAE2	+8.9	±0.0	+10.9
MacroAltAlt	+1.1	+0.12	±0.0
ObtuseWedge	+17.3	+31.5	+24.0
PbP.s	**+3.3**	**+93.4**	−0.7
PbP.q	**+6.7**	+7.32	**+25.8**
Sayphi-Rules	+2.2	−6.2	−5.6
Wizard+FF	−6.6	+11.7	−15.0
Wizard+SGPlan	−1.7	+8.1	−3.1

Table 6: Performance gaps between the IPC6 planners with/out learned knowledge in terms of % of solved problems, time and quality scores. The planners that, according to the IPC6 results, work only with knowledge are omitted.

which very significant improvements can be obtained also in terms of plan quality.

Table 5 shows the quality of the plans computed by PbP.q

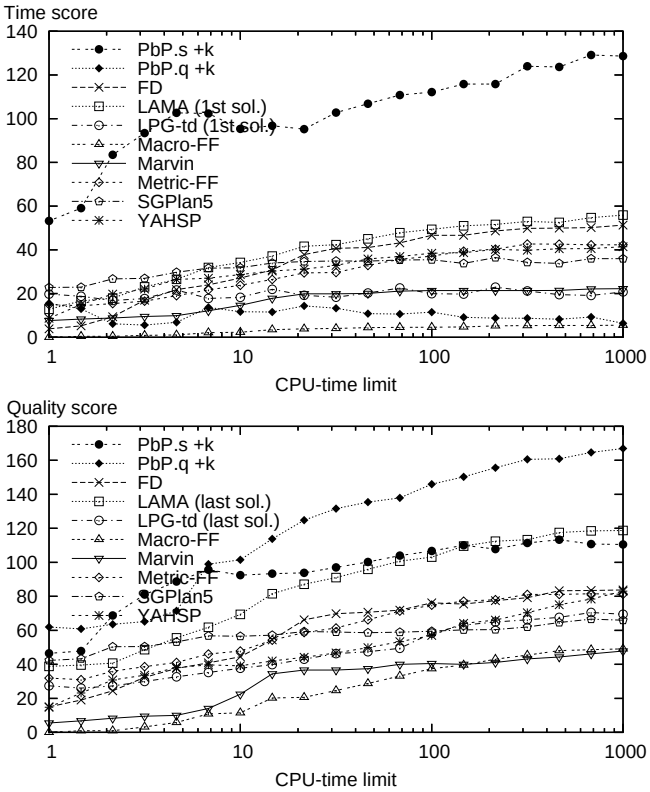

Figure 4: Time scores (upper plot) and quality scores (bottom plot) of PbP.s/q using the learned knowledge and of the incorporated planners w.r.t. an increasing CPU-time limit (ranging from 1 to 1000 seconds) for the IPC6 domains.

with/out knowledge for our collection of hard Depots problems, which only few incorporated planners solve within the limit. For more than 35 crates, PbP.q without knowledge solves no problem, while PbP.q with knowledge solves many problems (e.g., 62% with 40 crates); for the solved problems with more than 15 crates, on average, PbP.q with knowledge computes much better plans.

Finally, we compared the impact of using the learned knowledge in PbP and in the other planners that entered the learning track of IPC6. The results in Table 6 show that, for the IPC6 problems, the knowledge learned by PbP has the most effective impact in terms of speed (considering PbP.s) and plan quality (considering PbP.q).

On the Accuracy of the Cluster Selection

Figure 4 gives an overall picture of the performance of PbP.s/q with the learned knowledge w.r.t. the incorporated planners in terms of speed and plan quality using a CPU-time limit ranging from 1 second to 1000 seconds. The marked points on the PbP curves correspond to the different clusters of planners resulting from the configuration knowledge learned according to the corresponding CPU-time limit. Overall, for every considered CPU-time limit, PbP.s with the knowledge is much faster than the incorporated planners, and PbP.q generates better quality plans.

IPC6 Domains	Max score	Time score of PbP.s	Best	Worst
Gold-miner	30	29.2	30.0	0.41
Matching-BW	30	24.4	24.0	0.00
N-puzzle	30	16.0	26.0	0.29
Parking	30	20.9	20.9	0.00
Sokoban	30	26.9	27.1	0.00
Thoughful	30	18.1	28.7	0.00
Total	180	135.5	156.7	0.70

Table 7: Maximum time score (2nd column), time score of PbP.s (3rd column) and of the best/worst clusters (4th/5th columns) for the IPC6 domains.

These results indicate that, for the IPC6 benchmarks, there is no basic planner in the considered portfolio that could be used to obtain an overall performance better than or similar to the performance achieved by PbP.s for speed and PbP.q for plan quality.

Table 7 shows, for each IPC6 domain, the speed performance of PbP.s with the learned knowledge w.r.t. the best performing and the worst performing planner clusters. The data related to "Best" ("Worst") column is the maximum (minimum) sum of the time scores for all possible clusters (with at most three planners) over the set of testing problems of each IPC6 domain. The time score for PbP.s is often similar to the score of the best cluster and much greater (and hence much better) than the score of the worst cluster. It is also worth noting that the time score of PbP.s without the knowledge is much worse than the scores of both PbP.s with the knowledge and the best cluster. This result indicates that the method used by PbP.s for configuring the portfolio identifies a good cluster of planners in terms of speed. Finally, we observed a similar behaviour for PbP.q in terms of plan quality.

On the Effectiveness of Using Macros

In this section, we give some experimental results aimed at understanding the usefulness of the computed macro-actions and the accuracy of the choices made by PbP about using them for the planners in the clusters identified by the learned configuration knowledge. For each IPC6 domain with at least one non-empty set of computed macros and each planner in the selected cluster (see Table 2), we compared the performance (in terms of total time scores) of the planner in the cluster using (a) no macros, (b) the set of macros identified by PbP.s as useful, and (c) the set of macros among those computed that makes the planner performs best.

The results of this analysis are in Table 8 and show that, for the considered benchmark domains and problems, very often there is a set of macros that is useful (increase speed performance). Moreover, these experimental results indicate that the performance obtained by using the set of macros selected by PbP.s is usually similar to performance that could be achieved when using the best computed set of macros. The only exception is YAHSP with domain Gold-miner, for which, however, PbP.s selects a cluster containing two other planners with very accurate choices in terms of se-

Planner	No macros	PbP.s macros	Best macros
Gold-miner			
YAHSP	4.9 (30)	20.6 (30)	29.0 (30)
FF	2.5 (30)	30.0 (30)	30.0 (30)
SGPlan5	15.0 (17)	28.3 (30)	28.3 (30)
Matching-BW			
Marvin	10.9 (16)	9.9 (16)	10.2 (16)
Parking			
FF	25.0 (25)	25.0 (25)	6.3 (25)
Sokoban			
SGPlan5	11.1 (30)	26.4 (30)	26.8 (30)
LPG	3.4 (29)	30.0 (30)	30.0 (30)

Table 8: Time scores and number of solved problems (in round brackets) 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 IPC6 domains with at least one non-empty set of computed macros.

lected macros. Interestingly, the sets of macros computed for FF for Parking is *not* helpful (making its speed performance significantly worse), and PbP.s correctly detects this, choosing to run FF without macros.

Conclusions

We have presented PbP, a planner based on an automatically configurable portfolio of domain-independent planners, which can compute and exploit some additional knowledge about a given planning domain specified with PDDL. PbP generates this configuration knowledge through an automated statistical analysis about the performance of the basic planners in the portfolio and the relative candidate sets of computed macro actions, using a collection on training problems in the given domain. This analysis can be seen as a method by which PbP generalizes the observed performance of the incorporated planners and learned macros for the training problems to new problems in the same domain. The learned knowledge is exploited to select, for the given domain, a promising combination of planners in the portfolio, each one with a (possibly empty) set of macro-actions, and to define additional information specializing their round-robin scheduling at planning time.

An experimental analysis presented in the paper shows that the learned knowledge is very useful for obtaining a good configuration of the portfolio in PbP, especially in terms speed (using PbP.s), but also in terms of plan quality (using PbP.q). Moreover, PbP performs better than the IPC6 planners that entered the learning track of IPC6. Finally, overall PbP with knowledge performs better than every basic incorporated planner, and it effectively chooses a (possibly empty) set of macros to use for the planners in the selected cluster.

References

G. Armano, G. Cherchi and E. Vargiu. 2003. A Parametric Hierarchical Planner for Experimenting Abstraction Techniques. In *Proc. of 18th Int. Joint Conference on Artificial Intelligence*.

A. Botea, M. Enzenberger, M. Müller and J. Schaeffer. 2005. Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *JAIR*, v. 24:581–621.

A. Botea, M. Müller and J. Schaeffer. 2005. Learning Partial-Order Macros from Solutions. In *Proc. of ICAPS-05*.

A. Botea, M. Müller and J. Schaeffer. 2005. Learning Partial-Order Macros from Solutions. In *Proc. of IJCAI-07*.

Y. Chen, C. Hsu and B. Wah. 2006. Temporal Planning using Subgoal Partitioning and Resolution in SGPlan. *JAIR*, v. 26:323–369.

A. Coles and A. Smith. 2007. Marvin: A Heuristic Search planner with Online Macro-Action Learning. *Journal of Artificial Intelligence Research (JAIR)*, v. 28:119–156.

A. Fern, R. Khardon, P. Tadepalli, 2008. 6th IPC – Learning Track http://eecs.oregonstate.edu/ipc-learn/

A. Gerevini, P. Haslum, D. Long, A. Saetti and Y. Dimopoulos. 2009. Deterministic Planning in the Fifth International Planning Competition: PDDL3 and Experimental Evaluation of the Planners. *Artificial Intelligence*, 173(5-6):619–668.

A. Gerevini, A. Saetti and I. Serina. 2003. Planning through Stochastic Local Search and Temporal Action Graphs. *JAIR*, 20:239–290.

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

M. Helmert, M. Do, I. Refanidis. 2008. 6 IPC – Deterministic Part http://ipc.informatik.uni-freiburg.de/

J. Hoffmann and S. Edelkamp. 2005. The Deterministic Part of IPC-4: An Overview, *JAIR*, 24:519–579.

J. Hoffmann and B. Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *JAIR*, 14:253–302.

A. Howe, E. Dahlman, C. Hansen, A. vonMayrhauser and M. Scheetz. 1999. Exploiting Competitive Planner Performance. In *Proc. of 5th European Conf. on Planning (ECP-99)*.

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

H. A. Kautz and B. Selman. 1999. Unifying SAT-based and Graph-based Planning. In *Proc. IJCAI-99*.

D. Long and M. Fox. 2003. The 3rd International Planning Competition: Results and Analysis, *JAIR*, 20:1–59

M. Newton, J. Levine, M. Fox, and D. Long. 2007. Learning Macro-Actions for Arbitrary Planners and Domains. In *Proc. of ICAPS-07*.

S. Richter, M. Westphal 2008. The LAMA Planner Using Landmark Counting in Heuristic Search. In *Abstract booklet of the 6th Int. Planning competition*

M. Roberts and A. Howe. 2007. Learned Models of Performance for Many Planners. In *Proc. of ICAPS'07 Workshop of AI Planning and Learning*.

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

B. Selman, H. A. Kautz, B. Cohen. 1994. Noise Strategies for Improving Local Search. In *Proc. of AAAI94*.

D. Vrakas, G. Tsoumakas, N. Bassiliades, and I. Vlahavas. 2003. Learning Rules for Adaptive Planning. In *Proc. of ICAPS-03*.

L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown 2008. SATzilla: Portfolio-based Algorithm Selection for SAT. *JAIR* , 32:565–606.

F. Wilcoxon and R. A. Wilcox. 1964. Some Rapid Approximate Statistical Procedures, Lederle Laboratories.