Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling

An Automatically Configurable Portfolio-Based Planner with Macro-Actions: PbP

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

While several powerful domain-independent planners have recently been developed, no one of these clearly outperforms all the others in every known benchmark domain. We present PbP, a multi-planner which automatically configures a portfolio of planners by (i) computing some sets of macro-actions for every planner in the portfolio, (ii) selecting a promising combination of planners in the portfolio and relative useful macro-actions, and (iii) defining some running time slots for their round-robin scheduling during planning. The configuration relies on some knowledge about the performance of the planners in the portfolio and relative macro-actions which is automatically generated from a training problem set. PbP entered the learning track of IPC-2008 and was the overall winner of this competition track. An experimental study confirms the effectiveness of PbP, and shows that the learned configuration knowledge is useful for PbP.

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., (Botea, Müller and Schaeffer 2005; Newton et al. 2007). A macro-action is a sequence of actions that can be planned at one time like a single action. When using macro-actions there is a trade-off to consider. While their use can reduce the number of search steps required to reach a solution, it also increases the search space size. 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 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.

PbP has two variants: PbP.s focusing on speed, and PbP.q focusing on plan quality. PbP.s entered the learning track of the sixth international planning competition (IPC6), and was the overall winner of this competition track (Fern, Khardon and Tadepalli 2008). The paper includes some experimental results about an improved implementation of the competition version of PbP.s and about PbP.q. These results confirm the effectiveness of PbP.s, indicate that PbP.q performs better than the IPC6 planners, and show that, contrary to the preliminary version of PbP.s, the learned configuration knowledge is useful for PbP.

Our approach was inspired by and is related to (Howe et al. 1999; Roberts and Howe 2007), with some significant differences. Howe and collaborators' system configures the portfolio using all the training problems in a large set of input known domains, while PbP's configuration is specific for one given domain. Moreover, the planner selection and ordering methods 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 effective 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, and the ordering is simply based on the computed planning time slots. Finally, their system does not compute, analyze or use macro-actions, and does not consider plan quality.

Planner	Authors, date
Fast Downward	Helmert, 2006
Metric-FF	Hoffmann & Nebel, 2001
LPG-td	Gerevini, Saetti & Serina, 2005
MacroFF	Botea, Enzenberger, Müller & Schaeffer, 2005
Marvin	Coles & Smith, 2007
SGPlan5	Chen, Wah & Hsu, 2006
YAHSP	Vidal, 2004

Table 1: Domain-independent planners currently in PbP.

In the rest of the paper, first we describe PbP and then we present the experimental results.

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

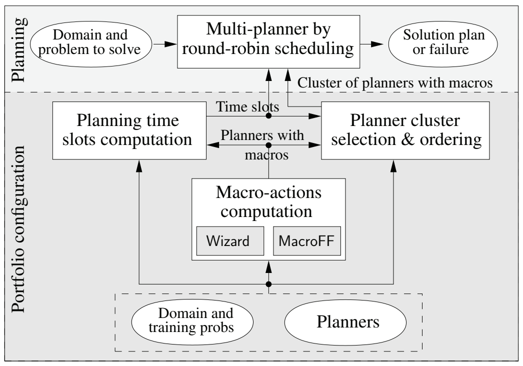
Table 1 shows the seven planners currently integrated into PbP. The architecture of PbP, sketched in Figure 1, consists of four main components, which are briefly described below.

Algorithms for computing macro-actions. For each integrated planner, PbP computes some sets of macro-actions using Wizard (Newton et al. 2007) and the techniques described in (Botea, Müller and Schaeffer 2005). With the first, PbP produces at most two alternative sets of planner-independent macro-actions; with the second, it produces at most five sets of alternative macro-actions for MacroFF.

Algorithms for computing the planning time slots of each considered planner. 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 (Roberts and Howe 2007), but with the technical difference explained in the following example. Assume that the computed planning time slots for planner A are $\{0.20, 1.40, 4.80, 22.50, \dots\}$ and that those for planner B are $\{14.5, 150.8, \dots\}$. 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.

Algorithms for selecting a cluster of planners using a (possibly empty) set of macro-actions and for ordering their runs. PbP runs each integrated planner with and without a set of computed macro-actions. Then, it selects a cluster of planners in the initial portfolio, each one with a (possibly empty) set of useful macro-actions. 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.

Algorithms for the planner scheduling and plan generation. 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. 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.



Planning
Domain and problem to solve
Multi-planner by round-robin scheduling
Solution plan or failure
Cluster of planners with macros
Time slots
Planning time slots computation
Planners with macros
Planner cluster selection & ordering
Portfolio configuration
Macro-actions computation
Wizard
MacroFF
Domain and training probs
Planners

Figure 1: A sketch of PbP's architecture.

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. In PbP, the performance measure considers either the CPU-time (PbP.s) or the plan quality (PbP.q).

After having run each planner with every computed set of macro-actions (one run for each set) for the training problem set of a domain D, PbP analyzes the results (CPU-times and plan qualities) to identify the best cluster of planners and macro-actions for D. This is done by *simulating*, for each cluster C of at most k planners with a (possibly empty) set of macro-actions, the round-robin execution of the planners in C for solving the same training problems.[1] The simulation is conducted using the data from the previous runs, and the simulated performances of the clusters are compared by a statistical analysis based on the Wilcoxon sign-rank test.[2]

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

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

[2] The Wilcoxon test has also been used in (Long and Fox 2003; Gerevini et al. 2009; Roberts and 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.

Domain	PbP.s	PbP.q
D1	YAHSP (63)	SGPlan5 (33) + LPG (100) + FF (33)
D2	FD (–) + YAHSP (14.3)	FD (–) + YAHSP (86) + LPG (0)
D3	SGPlan5 (50)	FD (–) + MFF (20) + LPG (100)
D4	YAHSP (0)	Marvin (0) + FF (0) + YAHSP (0)
D5	FD (–) + YAHSP (–)	FD (–) + FF (–) + Marvin (–)
D6	MFF (20)	YAHSP (100) + MFF (40) + 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), `Sokoban` (D3), `Parking` (D4), `Thoughful` (D5) and `N-puzzle` (D6). FD and MFF abbreviate Fast Downward and MacroFF.

differences are then ranked by increasing numbers, starting from the lowest value. 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 r_+ and r_- are similar. Intuitively, the test considers a weighted sum of the number of times a cluster performs better than the other compared one. The method used by PbP.q is similar, but it applies to the plan qualities resulting from the cluster execution simulation.

The Wilcoxon test is characterised by a probability value p, which represents the level of significance of the performance gap. PbP uses a default confidence level equal to 99.9%; hence, if p 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.

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. Table 2 shows the clusters of planners and the relative percentages of macros selected by PbP for each IPC6 domain (Fern, Khardon and Tadepalli 2008).

Experimental Results

The preliminary implementation of PbP.s that entered the learning track of IPC6 contains some implementation bugs and uses some inefficient Linux shell scripts. We have fixed these bugs and implemented a new version of PbP.s completely in C. Unless otherwise specified, in the following PbP.s denotes the newer version.

In this section, we present the results of an experimental study about PbP with three main goals: (i) confirming the efficiency of PbP.s; (ii) evaluating the efficiency of PbP.q in terms of plan quality by comparing it with other IPC6 planners, (iii) testing the effectiveness of the learned knowledge

Planner	Solved (%)	Time	Quality
CABALA	1.67	0.004	2.50
DAE1	18.3	0.007	31.7
DAE2	18.3	0.012	31.4
MacroAltAlt	28.9	4.458	44.9
ObtuseWedge	65.0	76.54	89.7
PbP.s	**90.0**	**107.9**	**117**
PbP.q	**93.0**	8.759	**151**
REPLICA	31.7	6.910	24.1
RFA1	47.2	14.15	61.5
RFA2	26.1	3.864	34.4
Roller	30.6	7.667	24.2
Sayphi-Rules	26.1	3.312	28.7
Wizard+FF	56.7	40.16	88.4
Wizard+SGPlan	51.1	36.58	76.3

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.

Planner	Solved (%)	Time	Quality
Lama	75.6	13.6	126.2
PbP.s	90.0	159.7	108.5
PbP.q	92.8	28.3	141.8

Table 4: Percentages of solved problems, time and quality scores (competition evaluation functions) of Lama and PbP.

for improving PbP's performance. The experimental tests were run on a machine slightly slower than the one used for IPC6. The CPU-time limit was the same as in IPC6.

PbP versus the IPC6 Planners

Table 3 compares PbP with the learned configuration knowledge and the planners that entered the learning track of IPC6 in terms of percentage of solved problems, time and plan quality.[3] The performance scores in Table 3 were obtained by the same simple evaluation scheme used in 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 compared planners 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 the new version of PbP.s performs much better than the IPC6 planners (PbP.s solves many more problems, and also has much greater time and quality scores) and that PbP.q is clearly the best planner in terms of plan quality. The time score of PbP.q is low because PbP.q stops only when all the selected planners terminate or the CPU-time limit is exceeded. Moreover, we observed that PbP.s/q has much better time/quality scores than those of every integrated planner.

Table 4 compares the performances of the multiplanner in PbP.s and PbP.q (with the learned knowledge) w.r.t. Lama, the winner of the IPC6 classical track, for the benchmark problems of the IPC6 learning track. PbP.s solves many more problems and is generally much faster than Lama.

[3]The IPC6 test problem set and training problem set are different. These problems and a collection of short papers on the evaluated planners can be found in (Fern, Khardon and Tadepalli 2008).

Planner	ΔSolved (%)	ΔTime	ΔQuality
DAE1	+0.2	+0.0	+29.6
DAE2	+0.1	−0.0	+15.2
MacroAltAlt	+1.1	+0.12	−0.1
ObtuseWedge	+17.3	+33.6	+27.7
PbP.s	+**1.7**	+**84.2**	−10.8
PbP.q	+**2.8**	+8.99	+**2.0**
Sayphi-Rules	+0.0	−11.8	−8.8
Wizard+FF	−6.6	+18.4	−16.3
Wizard+SGPlan	−1.7	+15.0	−2.9

Table 5: 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.

PbP.s performs worse in terms of plan quality, but this is not surprising because Lama searches for good quality plans, while PbP.s focuses on speed. Interestingly, PbP.q solves more problems than Lama and performs better in terms of plan quality.

On the Effectiveness of the Learned Knowledge

Table 5 shows the performance gap of PbP and other IPC6 planners with and without the learned knowledge for the IPC6 test problems (positive values are improvements obtained by using learned knowledge). For PbP.s, the learned knowledge is very helpful in terms of speed, while it does not help to improve plan quality (but this is not surprising since this knowledge is generated ignoring plan quality).

It is important to note that, contrary to these results, the learned knowledge does not speedup the *competition version* of PbP.s for the IPC6 problems. This behaviour, observed by the IPC6 organizers, depends on some implementation bugs 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).

The results in Table 5 indicate that the use of the learned knowledge in PbP gives only small improvements in terms of solved problems and plan quality (using PbP.q) within the considered CPU-time limit. However, with this limit, many IPC6 problems are inadequate to observe possible plan-quality and problem-solved improvements in PbP, because they can be quickly (relative to the limit) solved by most of the incorporated basic planners, and incrementally optimized by LPG.

In order to better understand the effectiveness of using the learned configuration knowledge in terms of solved problems and plan quality, we conducted an additional experiment using a set of large problems in the well-known Depots domain. Every marked point on the curves of Figure 2 is the average value (time or % of solved problems) over 100 randomly generated Depots problems with 7 locations, 3 trucks, 7 hoists and the number of crates indicated over the x-axis. For this class of problems, PbP.s with knowledge performs significantly better w.r.t. the number of solved problems and planning speed. Finally, Table 6 shows the quality of the plans computed by PbP.q with/out knowledge for the set of large 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.

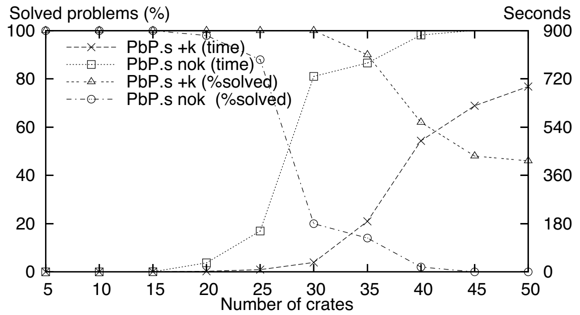


Figure 2: Percentage of solved problems (Depots domain) and average CPU-time (log. scale) of PbP.s with/out (+k/nok) the learned knowledge.

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 6: 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.

Conclusions

We have presented PbP, a planner based on an automatically configurable portfolio of domain-independent planners, which can compute and exploit additional knowledge about a given planning domain specified with PDDL. An experimental study shows that PbP.s performs better than the planners that entered the learning track of IPC6, that the knowledge learned by PbP.s can improve the efficiency of our approach significantly, and finally that for a collection of large problems the knowledge learned by PbP.q is very effective. Future work includes additional experiments and the extension of PbP to integrate further recent planners.

References

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

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

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and 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.

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

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

Newton, M.; Levine, J.; Fox, M.; and Long, D. 2007. Learning Macro-Actions for Arbitrary Planners and Domains. In *ICAPS07*.

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

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