PbP2: Automatic Configuration of a Portfolio-based Multi-Planner

Alfonso E. Gerevini and **Alessandro Saetti** and **Mauro Vallati**

Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Via Branze 38, 25123 Brescia, Italy
{gerevini,saetti,mauro.vallati}@ing.unibs.it

Abstract

We present PbP2, an automated system that generates efficient domain-specific multi-planners from a portfolio of domain-independent planning techniques by (i) computing some sets of macro-actions for every planner in the portfolio, (ii) optimizing the parameter setting of the parameterized planners in the portfolio, (iii) selecting a promising combination of planners in the portfolio and relative useful macro-actions, and (iv) defining some running time slots for their round-robin scheduling during planning. The configuration of the portfolio yielding the multi-planner relies on some knowledge about the performance of the planners and relative macro-actions, which is automatically generated from a training problem set. PbP2 is a revision and extension of a preliminary version of this system (PbP) that was awarded at the learning track of IPC-2008.

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 can then be useful to have a multi-planner system that automatically selects and combines the most efficient planners 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 & 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.

Another aspect that can significantly affect the performance of a domain-independent planner is the setting of its parametric components, which can concern, e.g., the domain analysis or compilation performed during preprocessing, the heuristic functions used during search, and several other features of the search algorithm. For instance, LPG (Gerevini, Saetti & Serina 2003; Gerevini, Saetti, & Serina 2008) is a well-known efficient and versatile planning system with many components that can be configured very flexibly via 62 exposed configurable parameters, which jointly give rise to over 6.5×10^{17} possible configurations. ParLPG (Vallati *et al.* 2011) is a very recent system based on the idea of automatically configuring a generic, parameterized planner like LPG using a set of training planning problems in order to learn a configuration of the exposed parameters that perform especially well in the domains of these problems.

In this paper, we present PbP2, a system that generates efficient domain-specific multi-planners by automatically configuring a portfolio of domain-independent planning techniques. The configuration relies on some knowledge about the performance of the planners in the portfolio, the observed usefulness of automatically generated sets of macro-actions, and optimized parameter configurations for the parametric planners in the portfolio.

This configuration knowledge is automatically computed and consists of: an ordered selected subset of the planners in the initial portfolio, combined through a round-robin strategy; a set of useful macro-actions for each selected planner; an optimized parameter configuration for the selected parametrized planners; 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 PbP2 is used without this additional knowledge, in the generated multi-planner 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, using their default parameter configurations. When PbP2 uses the knowledge computed for the domain under consideration, only the selected cluster of planners (and relative sets of macro actions) is scheduled, and their ordering favors the fastest planners for the domain under consideration, and the planning time slots are defined by the learned knowledge; moreover, the selected parametrized planners are run using their optimized parameter configurations rather than the default configurations.

PbP2 has two variants: PbP2.s focusing on speed, and PbP2.q focusing on plan quality. A preliminary version of PbP2.s was awarded at the sixth international planning com-

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 PbP2.

petition (IPC6) (Fern, Khardon & Tadepalli 2008), while PbP2.q did not enter IPC6.

The most significant differences between PbP2 and PbP are: (1) the correction of some bugs and the optimization of some parts of the code, (2) the PbP2.q variant of the system configuration the portfolio for plan quality rather than planning speed, (3) the extension of the portfolio with the IPC6-awarded planner Lama (Richter & Westphal 2008), (4) the integration of a component for automatically configuring the parameters of highly parametrized planners, which in the current implementation is based on ParLPG (Vallati *et al.* 2011).

In recent work (Gerevini, Saetti, & Vallati 2009; 2009), we have presented an experimental analysis showing that, overall, the first three mentioned revisions of our system lead to significant improvements in the performance of the system. The evaluation of the impact of the last change, i.e., the integration of ParLPG, is ongoing.

The Portfolio-based Planner PbP2

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.

Architecture of PbP2

Table 1 shows the eight planners currently integrated into PbP2. The architecture of PbP2, sketched in Figure 1, consists of six main components, which are briefly described below.

Macro-actions computation. For each integrated planner, PbP2 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 PbP2. With this approach, PbP2 produces 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 PbP2 contains the enhancements described in (Botea, Müller & Schaeffer 2005; 2007). With this approach, PbP2 produces at most five sets of alternative macro-actions for Macro-FF.

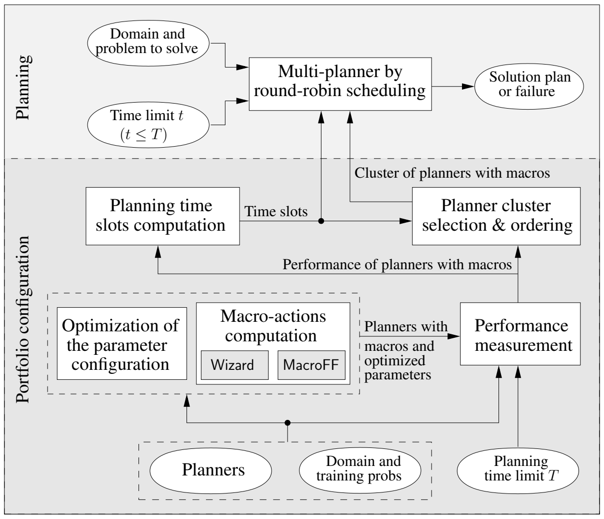
Planning
Domain and problem to solve
Time limit t ($t \leq T$)
Multi-planner by round-robin scheduling
Solution plan or failure
Cluster of planners with macros
Planning time slots computation
Time slots
Planner cluster selection & ordering
Performance of planners with macros
Portfolio configuration
Optimization of the parameter configuration
Macro-actions computation
Wizard
MacroFF
Planners with macros and optimized parameters
Performance measurement
Planners
Domain and training probs
Planning time limit T

Figure 1: A sketch of PbP2's architecture.

Optimization of the parameter configuration. PbP2 includes a module for automatically configuring highly parameterized planners in the portfolio. In the current implementation, the only such planner is LPG, which is configured using ParLPG (Vallati *et al.* 2011). ParLPG is based on the FocusedILS variant of the off-the-shelf, state-of-the-art automatic algorithm configuration procedure ParamILS (Hutter *et al.* 2009). FocusedILS uses a well-known stochastic local search procedure (Iterated Local Search) to search for high-performance configurations of a given parameterized algorithm. Given a planning domain and set of training problem instances in this domain, ParLPG generates an optimized configuration for the 62 parameters exposed by this planner, evaluating LPG's performance either in terms of planning speed (ParLPG.s) or of plan quality (ParLPG.q).

Performance measurement. This is the computationally most expensive task in the configuration of the portfolio. PbP2 runs each integrated planner with and without the sets of learned macro-actions and optimized planning parameters (in the current implementation, only of LPG) 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, PbP2 measures the quality of all the solutions generated for a problem and the corresponding CPU-times.[1]

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

Planning time slots computation. For each integrated planner, PbP2 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, PbP2 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), PbP2 would perform significantly slower.

Planner cluster selection & ordering. PbP2 selects a cluster of planners in the initial portfolio, each one with a (possibly empty) set of useful macro-actions and a (default or optimized) parameter configuration, 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. PbP2 runs the selected ordered planners (each one using the relative selected set of macro-actions and possibly optimized parameter configuration) by a round-robin scheduling algorithm using the computed planning time slots for an input (test) problem. Concerning termination of the resulting multi-planner, PbP2.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 PbP2.s). PbP2.q's execution is interrupted if either time T is exceeded, or *all* the selected planners terminate. If PbP2.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, PbP2 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 and without the optimized parameter configuration and 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, PbP2 analyzes the results (CPU-times and plan qualities) to identify the best cluster of planners, parameter configurations 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 and a (default or optimized) parameter configuration, 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.

In PbP2, the performance measure considers either the CPU-time (PbP2.s) or the plan quality (PbP2.q). The data for carrying out the test in PbP2.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. PbP2 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 PbP2 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 con-

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

nected component of this graph is collapsed into a single node representing the elements in the clusters of the collapsed nodes. From the resulting DAG, PbP2 considers only the nodes without incoming edges (the graph root nodes). If there is only one root node, this is the selected cluster, otherwise PbP2 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 PbP2.q is similar, but it applies to the plan qualities resulting from the cluster execution simulation. For this simulation, PbP2.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 PbP2.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.

Conclusions

PbP2 is a planning system 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. The system generates this configuration knowledge through an automated statistical analysis about the performance of the basic planners in the portfolio (possibly optimized by an automatic parameter configuration module) and the relative candidate sets of computed macro actions, using a set of 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, an optimized parameter configuration of the parameterized planners in the portfolio (in the current implementation only LPG) each one with a (possibly empty) set of macro-actions, and to define additional information specializing their round-robin scheduling at planning time.

[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.

References

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*.

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.

Gerevini, A.; Saetti, A.; and Serina, I. 2008. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence* 172(8-9):899–944.

Gerevini, A.; Saetti, A.; and Vallati, M. 2009. Learning and Exploiting Configuration Knowledge for a Portfolio-based Planner. In *Proc. of ICAPS-09 Workshop on Planning & Learning*.

Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An Automatically Configurable Portfolio-based Planner with Macro-actions: PbP. In *Proc. of ICAPS-09*.

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

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.

Vallati, M.; Fawcett, C.; Gerevini, A; Hoos, H.; Saetti, A. 2011. ParLPG: Generating domain-specific planners through automatic parameter configuration in LPG. In *Abstract booklet of IPC-7*.

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