A Planner Based on an Automatically Configurable Portfolio of Domain-independent Planners with Macro-actions: PbP

Beniamino Galvani and **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}@ing.unibs.it

Introduction: the PbP Planner

In the last years, the field of fully-automated plan generation has significantly advanced. However, while several powerful domain-independent planners have been developed, no one of these clearly outperforms all the others in any known benchmark domain. It would then be desirable to have a multi-planner system capable of automatically selecting and using 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 encoded in the input domain formalization. Several approaches encoding additional knowledge about the domain in the form of macro-actions have been proposed (e.g., (Botea *et al.* 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 an important tradeoff to consider. On one hand, their usage can speedup the planning process, because it reduces the number of search steps required to reach a solution plan; on the other hand, it increases the size of the search space, and this could slow down the planning process.

In this paper, we propose a planner, called PbP (*Portfolio-based Planner*), which automatically configures a portfolio of domain-independent planners possibly using macro-actions. The configuration relies on some additional knowledge on the performance of the planners in the portfolio and on the observed usefulness of automatically generated sets of macro-actions. This knowledge is obtained by a statistical analysis and consists of: useful sets of macro-actions, an ordered (sub)set of the planners in the initial portfolio, and some sets of "planning time slots". A planning time slot is an amount of CPU-time to be dedicated to a particular planner using a specific (possibly empty) set of macro-actions selected during the learning phase. When PbP is used without this additional knowledge, PbP schedules all the planners in the portfolio by using a simple round-robin strategy where (predefined) equal CPU-time slots are assigned to the (randomly ordered) planners. On the contrary, if PbP uses the computed additional knowledge for the domain under consideration, only a (sub)set of the planners composing the portfolio is scheduled, their ordering favors the fastest planners for the domain under consideration, and the assigned planning time slots are not necessary equal. Moreover, PbP runs the selected planners with different sets of (possibly empty) macro-actions, which in the learning phase are selected w.r.t. a specific planning system.

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: The list of the domain-independent planners currently used in PbP.

It should be noted that in our framework the computed macro-actions are not always used by a planner. Assume that a planner P performs very well in a domain D, and thus it is in the set of the planners selected by PbP for solving the problems in D. If in the learning phase PbP observes that the set of macro-actions computed for planner P does not improve the performance of P in D, then PbP schedules the run of P without using macro-actions.

Our approach is closely related to the work of Howe and colleagues (Howe *et al.* 1999; Roberts & Howe 2007), but with some significant differences. Differently from our approach, Howe *et al.*'s system does not configure the portfolio for a specific domain (while PbP does so); instead the portfolio is configured using all the training problems in *every* input domains; moreover, the planners selected by this system are a set covering the whole training problem set, while in our approach the selection of the planners is based on a statistical analysis, which considers the CPU-times consumed by the planners to solve the problems in a specific given domain. Finally, Howe *et al.*'s system does not compute, analyze or use macro-actions.

The PbP Architecture

Table 1 shows the seven planners considered in the version of PbP that took part in the learning track of IPC-6. The architecture of PbP, sketched in Figure 1, consists of four

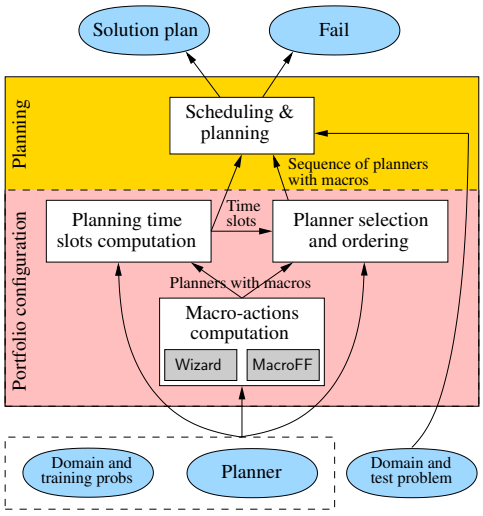

Figure 1: A sketch of the PbP architecture.

main components, which are briefly described below.

Algorithms for computing macro-actions. For each planner P in Table 1, PbP computes a set of macro-actions for the domain under consideration. Macro-actions are computed using Wizard (Newton *et al.* 2007) and the techniques described in (Botea, Müller & Schaeffer 2005).

Algorithms for computing the planning time slots of each considered planner. For each planner P in Table 1, 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 by (Roberts & Howe 2007), but with the technical difference explained in the following example. Assume that the computed planning CPU-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 CPU-time slot for A to 4.80, which is the greatest CPU-time slot of A which is smaller than the first time slot of B (and similarly for the following CPU-time slots). If we did not extended the first CPU-time slot of A, 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 the problems that planner A quickly solves, PbP would perform significantly slower.

Algorithms for selecting a subset of the planners (possibly using macro-actions) and for ordering their runs. PbP runs each planner P in Table 1 with and without using the macro-actions computed for P. Then, it selects a subset of the planners in the initial portfolio, each one with a (possibly empty) set of "useful" macro-actions.

For every subset S of the planners in Table 1, PbP simulates the run of S using the planner scheduling algorithm for solving the input problem set of the domain under consideration. Then, PbP selects the best subset(s) of the tested planners w.r.t. a statistical analysis based on the Wilcoxon sign-rank test (also known as the "Wilcoxon matched pairs test") (Wilcoxon & Wilcox 1964) applied to the CPU-times resulting from the simulation. In order to break the possible tie between a pair of planner subsets, PbP considers other parameters about the observed planning performance, such as the number of solved problems, the sums of the ratios between the (simulated) CPU-times of the involved planner subsets, and the first planning CPU-time slots of the involved planners.

Moreover, in the version of PbP that took part in the competition, the execution order of the selected subset of planners is trivially defined by the increasing CPU-time slots associated with the planners.

Algorithms for the planner scheduling and plan generation. PbP runs the selected ordered planners (using their selected subset of macro-actions) by a round-robin scheduling algorithm using the computed planning time slots.

Conclusions and Future Work

We have briefly described 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.

In a preliminary experimental study, we have observed that for several planning domains, the computed additional knowledge improves the performance of PbP without learned knowledge, and that often PbP performs better than any planner forming the initial portfolio considered in the current implementation.

Future work includes the development of algorithms for selecting the planners from the portfolio taking into account some problem dependent parameters. Moreover, we intend to study the integration of the techniques proposed in (Roberts & Howe 2007) for ordering the selected planner runs into our framework.

References

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

A. Botea, M. Müller and J. Schaeffer. 2005. Learning Partial-Order Macros from Solutions. In *Proc. of 15th Int. Conf. on Automated Planning and Scheduling (ICAPS-05)*.

M. A. H. Newton, J. Levine, M. Fox, and D. Long. 2007. Learning Macro-Actions for Arbitrary Planners and Domains. In *Proc. of the 17th Int. Conf. on Automated Planning and Scheduling (ICAPS-07)*.

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)*.

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

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