Configuration and Learning Techniques for Efficient Automated Planning Systems – Abstract–

Mauro Vallati

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

Introduction

In this abstract we briefly present our thesis work, which is focused on three main directions: (i) speeding-up optimal SAT-based planners by exploiting learned domain knowledge, (ii) configuring a portfolio of planners for an input domain, and (iii) the automated configuration of planning algorithms. The main results of such a work are three planning systems: MacroSatPlan (Gerevini, Saetti & Vallati 2010), PbP (Gerevini, Saetti & Vallati 2009) and ParLPG (Vallati, Fawcett, Gerevini, Hoos & Saetti 2011).

The paper is organized as follows. The second section briefly describes MacroSatPlan and the third section briefly describes PbP. Finally, in fourth section, we present ParLPG: the system that we are currently focusing on.

MacroSatPlan

Planning as propositional satisfiability (SAT) is a powerful approach for computing optimal plans in terms of Graphplan plan length. SatPlan (Kautz and Selman 1992) is one of the most popular and efficient planning system adopting this approach. First, it computes a lower bound k of the optimal plan length. Then, using k as the planning *horizon*, i.e., a fixed time step after which actions cannot be executed, it translates the planning problem into a SAT problem Π, which is then solved by an existing SAT solver. If Π is satisfiable, then the assignment to propositional fluents satisfying the SAT problem is translated into a plan of actions that is a solution of the original planning problem. Otherwise (Π is unsatisfiable), the process is repeated using an increased value of k.

A critical weakness of the approach is that often the initial value of k is much less than the optimal plan length, and hence many unsolvable SAT problems can be generated and processed.

Moreover, in many domains the planning performance can be improved by deriving and exploiting knowledge. Well known examples of such knowledge are macro actions. A macro-action is a sequence of domain actions that can be planned at one time like a single action. Using macro-actions the planning process is often faster, but the length of the computed solution plan can be worse than optimal.

MacroSatPlan is a SAT-based optimal planner which exploits two types of knowledge learned for a given domain to speedup the SAT solving: (i) a predictive model based on some problem features estimating the optimal plan length, and (ii) useful sets of learned macro-actions.

The predictive model is learned using WEKA (Witten and Frank 2005), a well-known machine learning tool. Given a set of training problems for domain D, WEKA is used to identify a predictive model of the optimal plan length for a given problem in domain D from (i) the length of the optimal plan computed by SatPlan, (ii) some pre-identified features of the planning problem, and (iii) the length of the relaxed plan computed by FF (Hoffmann and Nebel 2001).

Macro-FF (Botea *et al.* 2005) is the system selected for computing macro-actions. The computed macros are subsequently used by a modified version of the SAT-solver MiniSAT (Eèn and Sörensson 2003), during planning phase.

A preliminary experimental analysis shows that the learned knowledge is useful for speeding up the computation of the optimal solution of the planning problem.

For further details, please see (Gerevini, Saetti & Vallati 2010).

Portfolio-Based Planner

PbP (*Portfolio-based Planner*) is a 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 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.

An experimental analysis about an improved implementation of the competition version of PbP.s and about PbP.q confirms 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 a more detailed description about PbP, the interested reader can see (Gerevini, Saetti & Vallati 2009).

ParLPG

When designing state-of-the-art, domain-independent planning systems, many decisions have to be made with respect to the domain analysis or compilation performed during preprocessing, the heuristic functions used during search, and other features of the search algorithm (e.g., the search neighborhood definition). These design decisions can have a large impact on the performance of the resulting planner.

By providing many alternatives for these choices and exposing them as parameters, highly flexible domain-independent planning systems are obtained, which then, in principle, can be configured to work well on different domains, by using parameter settings specifically chosen for solving planning problems from each given domain. However, usually such planners are used with default configurations that have been chosen because of their good average performance, based on limited exploration within a potentially vast space of possible configurations. The hope is that these default configurations will also perform well on domains and problems beyond those for which they were tested at design time.

ParLPG uses a different approach, based on the idea of *automatically* configuring a generic, parametrized planner using a set of training problems for domain D in order to obtain planners that perform especially well in this domain.

Automated configuration of heuristic algorithms has been an area of intense research focus in recent years, producing tools that have improved algorithm performance substantially in many problem domains. These techniques have not yet been applied to the problem of planning. While the proposed framework could utilize any of these automatic configuration procedures, the FocusedILS variant of the off-the-shelf, state-of-the-art automatic algorithm configuration procedure ParamILS (Hutter et al. 2009) has been chosen for this work.

At the core of the ParamILS framework lies Iterated Local Search (ILS), a well-known and versatile stochastic local search method that iteratively performs phases of a simple local search, such as iterative improvement, interspersed with so-called perturbation phases that are used to escape from local optima. The FocusedILS variant of ParamILS uses this ILS procedure to search for high-performance configurations of a given algorithm by evaluating promising configurations, using an increasing number of runs in order to avoid wasting CPU-time on poorly-performing configurations. ParamILS is able to adaptively limit the amount of runtime allocated to each algorithm run using knowledge of the best-performing configuration found so far, which helps to further limit the CPU-time wasted on low-performance configurations.

```
1. Set $\mathcal{A}$ to the action graph containing only $a_{start}$ and $a_{end}$;
2. *While* the current action graph $\mathcal{A}$ contains a flaw or
          a certain **number of search steps** is not exceeded *do*
3.    **Select a flaw** $\sigma$ in $\mathcal{A}$;
4.    Determine the search **neighborhood** $N(\mathcal{A}, \sigma)$;
5.    Weight the elements of $N(\mathcal{A}, \sigma)$ using a **heuristic function** $E$;
6.    Choose a graph $\mathcal{A}' \in N(\mathcal{A}, \sigma)$ according to $E$ and **noise** $n$;
7.    Set $\mathcal{A}$ to $\mathcal{A}'$;
8. *Return* $\mathcal{A}$.
```

Figure 1: High-level description of LPG's search procedure.

Recently, ParamILS was used to configure several solvers for mixed integer programming (MIP) problems (Hutter, Hoos, & Leyton-Brown 2010) and for propositional satisfiability (SAT) problems (Hutter *et al.* 2007).

These previous applications of ParamILS, while yielding impressive results, were limited to optimizing the performance of algorithms designed to solve a single problem (SAT and MIP, respectively). The application of algorithm configuration techniques to planning differs in this respect, as each planning domain can be thought of as an independent problem. Given that end-users of planning tools tend to focus their attention on a single domain or group of related domains; being able to automatically configure a domain-independent planner to optimize performance on a given domain of interest should have great utility to the planning community.

In ParLPG, ParamILS is used to configure the well-known, domain-independent, satisficing planner LPG (Gerevini, Saetti, and Serina 2005).

LPG is a versatile system that can be used for plan generation, plan repair and incremental planning. The planner is based on a stochastic local search procedure that explores a space of partial plans represented through *linear action graphs*, which are variants of the very well-known planning graph (Blum & Furst 1997).

Starting from the initial action graph containing only two special actions representing the problem initial state and goals, respectively, LPG iteratively modifies the current graph until there is no *flaw* in it or a certain bound on the number of search steps is exceeded. Intuitively, a flaw is an action in the graph with a precondition that is not supported by an effect of another action in the graph. LPG attempts to resolve flaws by inserting into or removing from the graph a new or existing action, respectively.

Figure 1 gives a high-level description of the general search process performed by LPG. Each search step *selects a flaw* σ in the current action graph $\mathcal{A}$, defines the elements (modified action graphs) of the *search neighborhood* of $\mathcal{A}$ for repairing σ, weights the neighborhood elements using a *heuristic function* E, and chooses the best one of them according to E with some probability n, called the *noise parameter*, and randomly with probability $1 - n$. Because of this noise parameter, which helps the planner to escape from

Domain Configuration	P1	P2	P3	P4	P5	P6	P7	Total
Blocksworld	1	1	2	1	5	1	2	13
Depots	2	2	1	1	2	2	2	12
Gold-miner	2	3	0	1	4	2	1	13
Matching-BW	1	2	2	1	3	0	2	11
N-Puzzle	4	5	3	2	14	5	2	35
Rovers	0	1	0	0	0	2	1	4
Satellite	2	7	3	1	11	5	3	32
Sokoban	0	1	1	1	1	1	2	7
Zenotravel	3	5	2	3	11	5	3	32
Number of parameters	6	15	8	6	17	7	3	62

Table 1: Number of parameters of LPG that are changed by ParamILS in the configurations computed for nine domains independently considered. Each P1–P7 column corresponds to a different parameter category (or planner component). The last line of the table indicates the number of parameters in each category.

possible local minima, LPG is a randomized procedure.

Many components of LPG can be configured very flexibly via 62 exposed configurable parameters, which jointly give rise to over 6.5×10^{17} possible configurations. These parameters can be grouped into seven distinct categories, each of which corresponds to a different component of LPG:

P1 *Preprocessing information* (e.g., mutually exclusive relations between actions).

P2 *Search strategy* (e.g., the use and length of a "tabu list" for the local search, the number of search steps before restarting a new search, and the activation of an alternative systematic best-first search procedure).

P3 *Flaw selection strategy* (i.e., different heuristics for deciding which flaw should be repaired first).

P4 *Search neighborhood definition* (i.e., different ways of defining/restricting the basic search neighborhood).

P5 *Heuristic function E* (i.e., a class of possible heuristics for weighting the neighborhood elements, with some variants for each of them).

P6 *Reachability information* used in the heuristic functions and in neighborhood definitions (e.g., the minimum number of actions required to achieve an unsupported precondition from a given state).

P7 *Search randomization* (i.e., different ways of statically and dynamically setting the noise value).

Table 1 shows, for each parameter category of LPG, the number of parameters that are changed from their defaults by ParamILS in the derived domain-specific configurations. Domain-specific configurations of LPG differ substantially from the default configuration. Moreover, usually the changed configurations are considerably different from each other.

ParLPG was tested on problem instances from eight known benchmark domains used in the last four international planning competitions (IPC-3–6), Depots, Gold-miner, Matching-BW, N-Puzzle, Rovers, Satellite, Sokoban, and Zenotravel, plus the well known domain Blocksworld.

For each domain, we used the respective random instance generator to derive three disjoint sets of instances: a training set with 2000 relatively small instances (benchmark T), a testing set with 400 middle-size instances (benchmark MS), and a testing set with 50 large instances (benchmark LS). The size of the instances in training set T was decided such that the instances may be solved by the default configuration of LPG in 20 to 40 CPU seconds *on average*. For testing sets MS and LS, the size of the instances was defined such the instances may on average be solved by the default configuration of LPG in 50 seconds to 2 minutes and in 3 minutes to 7 minutes, respectively. This does not mean that all problem instances can be solved by LPG; since the *size* of the instances was decided according to the performance of the default configuration, and then random generators were used for deriving the actual instances.

For all configuration experiments we used the FocusedILS variant of ParamILS version 2.3.5 with default parameter settings. Using the default configuration of LPG as the starting point for the automated configuration process, 10 independent runs of FocusedILS were performed concurrently per domain, using random orderings of the training set instances.[1] Each run of FocusedILS had a total CPU-time cutoff of 48 hours, and a cutoff time of 60 CPU seconds was used for each run of LPG performed during the configuration process. The objective function used by ParamILS for evaluating the quality of configurations was mean runtime, with timeouts and crashes assigned a penalized runtime of ten times the per-run cutoff. Out of the 10 configurations produced by these runs, we selected the configuration with the best training set performance (as measured by FocusedILS) as the final configuration of LPG for the respective domain.

Figure 2 provides results in the form of a scatterplot, showing the performance of automatically determined, domain-specific configurations (LPG.sd) and default configuration (LPG.d) of LPG on the individual benchmark instances. We consider all instances solved by at least one of these planners. Each cross symbol indicates the CPU time used by LPG.d and LPG.sd to solve a particular problem instance of benchmarks MS and LS. When a cross appears under (above) the main diagonal, LPG.sd is faster (slower) than LPG.d; the distance of the cross from the main diagonal indicates the performance gap (the greater the distance, the greater the gap). The results in Figure 2 indicate that LPG.sd performs almost always better than LPG.d, often by 1–2 orders of magnitude.

Future Work

There are several avenues for future work in this thesis.

Concerning ParLPG, we intend to experimentally analyze the usefulness of the proposed framework for identifying configurations improving the planner performance in terms of plan quality. Moreover, we plan to apply the framework to metric-temporal planning domains, which LPG supports. Finally, it is important to investigate the use of other existing or forthcoming highly parameterized planners. In particular, preliminary results from ongoing work indicate that substantial performance gains can be obtained when apply-

[1]Multiple independent runs of FocusedILS were used, because this approach can help ameliorate stagnation of the configuration process occasionally encountered otherwise.

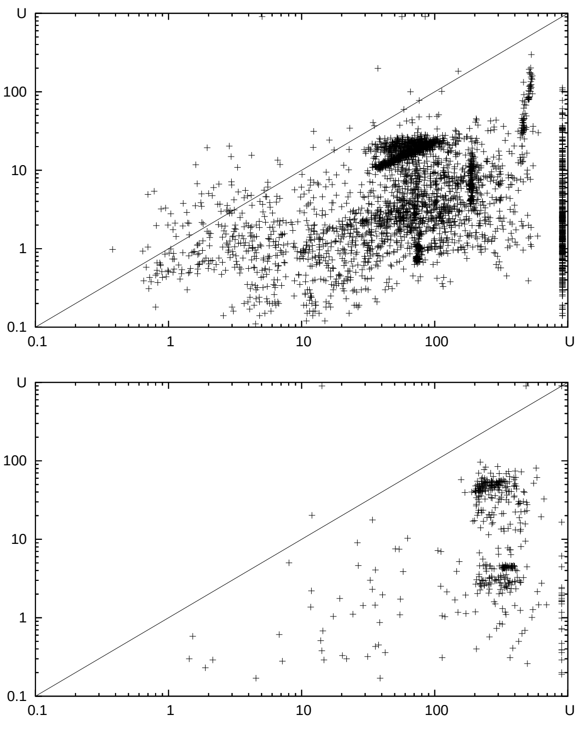

Figure 2: CPU time (log. scale) of LPG.sd with respect to LPG.d for problems of benchmarks MS (upper plot) and LS (bottom plot). The x-axis refers to CPU seconds of LPG.d; the y-axis to CPU seconds of the specific LPG.sd solvers; U corresponds to runs that timed out with the given runtime cutoff.

ing ParLPG approach to a very recent, highly parameterized version of the IPC-4 winner Fast Downward (Helmert 2006).

Concerning MacroSatPlan, future work includes further experiments, the evaluation of different ways for exploiting the extracted knowledge, and the integration of Wizard (Newton et al. 2007) as an alternative system for learning macros.

ParLPG and a new version of PbP entered the learning track of the seventh international planning competition (IPC7). The results will be announced in June 2011.

Acknowledgements

All the works described in this abstract are joint works with Alfonso E. Gerevini and Alessandro Saetti. ParLPG is the result of a joint work also involving Chris Fawcett and Holger H. Hoos.

References

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

Eèn, N.; Sörensson, N. 2003. An Extensible SAT-solver In *Proc. of SAT-03*.

Richter, S. Helmert, M., and Westphal, M. 2007. Landmarks revisited. In *Proc. of 23rd Conf. on Artificial Intelligence (AAAI-07)*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: fast plan generation through heuristic search. *JAIR*, 14:253–302.

Kautz, H.; Selman, B. 1992. Planning as satisfiability. In *Proc. of ECAI-92*.

Helmert, M. 2006 The Fast Downward Planning System *JAIR*, 26:191–246.

Hutter, F.; Hoos, H.; Leyton-Brown, K.; and Stutzle, T. 2009 ParamILS: An Automatic Algorithm Configuration Framework *JAIR*, 36:267–306.

Blum, A., and Furst, M., L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90:pp. 281–300.

Gerevini, A.; Saetti, A.; and Serina, I. 2005. Integrating Planning and Temporal Reasoning for Domains with Durations and Time Windows. In *Proceedings of IJCAI-05*.

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning macro-actions for arbitrary planners and domains. In *Proc. of ICAPS-07*.

Vallati, M.; Fawcett, C.; Gerevini, A.; Hoos, H.; and Saetti, A.. 2011. 7th IPC – Learning Track – Technical Report http://www.plg.inf.uc3m.es/ipc2011-learning/

Gerevini, A. E.; Saetti, A.; and Vallati, M. 2009. An Automatically Configurable Portfolio-based Planner with Macro-actions: PbP. In *Proc. of ICAPS09*.

Gerevini, A. E.; Saetti, A.; and Vallati, M. 2010. Optimal SAT-based Planning with Macro-actions and Learned Horizons. In *Proc. of PlanSIG10*.

Hutter, F.; Babić, D.; Hoos, H. H.; and Hu, A. J. 2007. Boosting verification by automatic tuning of decision procedures. In *Formal Methods in Computer-Aided Design*, 27–34. IEEE CS Press.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2010. Automated configuration of mixed integer programming solvers. In *Proc. of CPAIOR-10*.

Richter, S. Helmert, M., and Westphal, M. 2007. Landmarks revisited. In *Proc. of 23rd Conf. on Artificial Intelligence (AAAI-07)*.

Witten, I. H., and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition)*.