ParLPG: Generating Domain-Specific Planners through Automatic Parameter Configuration in LPG

Mauro Vallati
University of Brescia
mauro.vallati@ing.unibs.it

Chris Fawcett
University of British Columbia
fawcettc@cs.ubc.ca

Alfonso E. Gerevini
University of Brescia
gerevini@ing.unibs.it

Holger H. Hoos
University of British Columbia
hoos@cs.ubc.ca

Alessandro Saetti
University of Brescia
saetti@ing.unibs.it

Abstract

The ParLPG planning system is based on the idea of using a generic algorithm configuration procedure – here, the well-known ParamILS framework – to optimise the performance of a highly parametric planner on a set of problem instances representative of a specific planning domain. This idea is applied to LPG, a versatile and efficient planner based on stochastic local-search with 62 parameters and over 6.5×10^{17} possible configurations. A recent, large-scale empirical investigation showed that the approach behind ParLPG yields substantial performance improvements across a broad range of planning domains.

Introduction

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 several other features of the search algorithm. 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.

The planning system ParLPG that we propose in this work is based on the idea of automatically configuring a generic, parameterized planner using a set of training planning problems in order to obtain planners that perform especially well in the domains of these problems. ParLPG uses the FocusedILS variant of the off-the-shelf, state-of-the-art automatic algorithm configuration procedure ParamILS (Hutter, Hoos, and Stützle 2007; Hutter et al. 2009), and LPG (Gerevini, Saetti, and Serina 2003; 2008), 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. Moreover, this work exploits several meta-algorithmic procedures provided by the empirical algorithmics environment HAL (Nell et al. 2011).

We developed two variants of the proposed planner: ParLPG.s refers to the planners resulting from domain-specific configuration aiming to minimize the runtime needed to find an initial satisficing plan, whereas ParLPG.q refers to the planners resulting from configuration for optimizing plan cost.

In a large experimental study we found that the approach underlying ParLPG yields substantial performance gains over the default configuration of LPG on 11 domains of planning problems used in previous international planning competitions (IPC-3–6), as well as on the benchmark problems considered in the learning track of IPC-6, where we have shown that in terms of speed and usefulness of the learned knowledge it outperforms the respective IPC-6 winners PbP.s (Gerevini, Saetti, and Vallati 2009) and ObtuseWedge (Yoon, Fern, and Givan 2008).

The Parameterized Planner LPG

In this section, we provide a very brief description of LPG and its parameters. LPG is a versatile system that can be used for plan generation, plan repair and incremental planning in PDDL2.2 domains (Hoffmann and Edelkamp 2005). The planner is based on a stochastic local search procedure that explores a space of partial plans represented through *linear action graphs* (Gerevini, Saetti, and Serina 2003), which are variants of the very well-known planning graph (Blum and 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 possible local minima, LPG is a randomized procedure.

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** σ 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.

The 62 configurable parameters of LPG control the possible settings of different components in the system, and can be grouped into seven distinct categories (each of which corresponds to a different component of LPG):

- *Preprocessing information* (e.g., mutually exclusive relations between actions).
- *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).
- *Flaw selection strategy* (i.e., different heuristics for deciding which flaw should be repaired first).
- *Search neighborhood definition* (i.e., different ways of defining/restricting the basic search neighborhood).
- *Heuristic function E* (i.e., a class of possible heuristics for weighting the neighborhood elements, with some variants for each of them).
- *Reachability information* used in the heuristic functions and in neighborhood definitions.
- *Search randomization* (i.e., different ways of statically and dynamically setting the noise value).

The Parameter Configurator ParamILS

While ParLPG could in principle utilize any sufficiently powerful automatic configuration procedure, we have chosen the FocusedILS variant of the off-the-shelf, state-of-the-art automatic algorithm configuration procedure ParamILS (Hutter, Hoos, and Stützle 2007; Hutter et al. 2009). 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 also avoids wasting CPU-time on low-performance configurations by adaptively limiting the amount of runtime allocated to each algorithm run using knowledge of the best-performing configuration found so far.

ParamILS has previously been applied to configure state-of-the-art solvers for SAT (Hutter et al. 2007) and mixed integer programming (MIP) (Hutter, Hoos, and Leyton-Brown 2010). This resulted in a version of the SAT solver Spear that won the first prize in one category of the 2007 Satisfiability Modulo Theories Competition (Hutter et al. 2007); it further contributed to the SATzilla solvers that won prizes in 5 categories of the 2009 SAT Competition and led to large improvements in the performance of CPLEX on several types of MIP problems (Hutter, Hoos, and Leyton-Brown 2010). Differently from SAT and MIP, in planning, explicit domain specifications are available through a planning language, which creates more opportunities for planners to take problem structure into account in parameterized components (e.g., specific search heuristics). This can lead to more complex systems, with greater opportunities for automatic parameter configuration, but also greater challenges (bigger, richer design spaces can be expected to give rise to trickier configuration problems).

Implementation of the Parameter Configuration Process using HAL

Empirical algorithm analysis and design techniques are often used in an ad-hoc fashion, relying upon informal experimentation. Furthermore, the techniques used in practice are often not correct, as many researchers and practitioners do not have sufficient knowledge of, or easy access to, more sophisticated techniques. Even when the best available techniques are used, the implementations of these techniques are often difficult to use, if they are publicly available at all.

HAL (the High-performance Algorithm Laboratory) was developed to address this need for easy access to powerful empirical techniques (Nell et al. 2011). HAL was created to support both the computer-aided design and the empirical analysis of high-performance algorithms, by means of ready-to-use, state-of-the-art procedures. It should be noted that HAL was designed to support these procedures for a wide range of problem domains, and was not designed for or limited to planning in any way.

In this work, we use several meta-algorithmic procedures provided by HAL, primarily the algorithm configuration tool ParamILS and the plugin providing support for empirical analysis of a single algorithm. We also leverage the robust support for compute clusters and data management.

For each given planning domain, the version of ParLPG competing in IPC-2011 uses HAL to run ten independent runs of ParamILS on a provided training set of instances, using a maximum runtime cutoff of 900 CPU seconds for each run of LPG and a total configuration time limit of five CPU days. In the case of ParLPG.s, we can leverage support in ParamILS for adaptive runtime capping to drastically reduce the runtime required for each run of LPG.

After all ten configuration runs have completed, we run LPG with a runtime cutoff of 900 CPU seconds on each instance in the training set in order to evaluate the so-called training score for each of the ten incumbent configurations. For ParLPG.s, this score is the mean runtime required to find a satisficing solution, and for ParLPG.q, it represents

mean plan cost, with timeouts assigned a plan cost of $2^{31}-1$. $2^{31}-1$ is the default value for solution quality output in HAL. The incumbent configuration with the best training score is returned as the learned knowledge for the given domain.

Conclusions

We believe that the generic approach underlying ParLPG represents a promising direction for the future development of efficient planning systems. In particular, we suggest that it is worth including many different variants and a wide range of settings for the various components of a planning system, instead of committing at design time to particular choices and settings, and to use automated procedures for finding configurations of the resulting highly parameterized planning systems that perform well on the problems arising in a specific application domain under consideration.

We note that our approach naturally benefits from future improvements in planning systems (and in particular, from new heuristic ideas that can be integrated, in the form of parameterised components, into existing, flexible planning systems or frameworks) as well as from progress in automated algorithm configuration procedures.

We also see much potential in testing new heuristics and algorithm components, based on measuring the performance improvements obtained by adding them to an existing highly-parameterized planner followed by automatic configuration for specific domains. The results may not only reveal to which extent new design elements are useful, but also under which circumstances they are most effective – something that would be very difficult to determine manually.

References

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1-2):281 – 300.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *J. Artif. Int. Res.* 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. An automatically configurable portfolio-based planner with macro-actions: Pbp. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS-09)*, 350–353.

Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of ipc-4: An overview. *J. Artif. Int. Res.* 24:519–579.

Hutter, F.; Babic, D.; Hoos, H. H.; and Hu, A. J. 2007. Boosting verification by automatic tuning of decision procedures. *Formal Methods in Computer Aided Design* 0:27–34.

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Stützle, T. 2009. Paramils: an automatic algorithm configuration framework. *J. Artif. Int. Res.* 36:267–306.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2010. Automated configuration of mixed integer programming solvers. In Lodi, A.; Milano, M.; and Toth, P., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 186–202.

Hutter, F.; Hoos, H. H.; and Stützle, T. 2007. Automatic algorithm configuration based on local search. In *Proceedings of the 22nd national conference on Artificial intelligence*, 1152–1157. AAAI Press.

Nell, C.; Fawcett, C.; Hoos, H. H.; and Leyton-Brown, K. 2011. HAL: A framework for the automated analysis and design of high-performance algorithms. In *LION-5*, (to appear).

Yoon, S.; Fern, A.; and Givan, R. 2008. Learning control knowledge for forward search planning. *J. Mach. Learn. Res.* 9:683–718.