Automatic Generation of Efficient Domain-Optimized Planners from Generic Parametrized Planners

Mauro Vallati
University of Huddersfield
m.vallati@hud.ac.uk

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

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. 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, planning systems can in principle be configured to work well on different domains. However, planners are typically used in default configurations that have been chosen because of their good average performance over a set of benchmark domains, with limited experimentation over the potentially huge range of possible configurations.

In this work, we propose a general framework for automatically configuring a parameterized planner, and show that substantial performance gains can be achieved. We apply the framework to the well-known LPG planner, which in the context of this work was expanded to 62 parameters and over 6.5×10^{17} possible configurations. By using this highly parameterized planning system in combination with the state-of-the-art automatic algorithm configuration procedure ParamILS, excellent performance on a broad range of well-known benchmark domains was achieved, as also witnessed by the results of the learning track of the 7th International Planning Competition.

Introduction

Automated planning is one of the most prominent AI challenges; it has been studied extensively for several decades and lead to many real-world applications (see, e.g., Ghallab, Nau, and Traverso 2004). 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. These design decisions can have a large impact on the performance of the resulting planner, and highly flexible domain-independent planning systems are obtained by providing many alternatives for these choices and exposing them as parameters. By using parameter settings specifically chosen for solving planning problems from each given domain, these planning systems can then be configured to work well on different domains. However, typically such domain-independent planners are used with default configurations that have been chosen based on their good average performance over a set of benchmark domains, based on limited exploration within a potentially vast space of possible configurations; often, these choices are hardwired into the code or take the form of undocumented parameters that remain fixed to their default values. 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.

In this work, we advocate a different approach, based on the idea of *automatically* configuring a generic, highly parameterized planner using a set of training planning problems in order to obtain planners that perform especially well in the domains of these training problems. 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. One such tool is the state-of-the-art automatic algorithm configuration procedure ParamILS (Hutter, Hoos, and Stützle 2007; Hutter et al. 2009). While our approach could in principle utilize any sufficiently powerful automatic configuration procedure, we have chosen the FocusedILS variant of ParamILS.

A well-known domain-independent highly parametrized planning system is LPG (Gerevini, Saetti, and Serina 2003; 2008; 2010). Based on a stochastic local search procedure, LPG is an efficient and versatile planner with many components.

In this work, we exposed 54 previously unused parameters in LPG and used ParamILS to automatically configure LPG on various propositional planning domains. This newly-expanded configuration space for LPG is one of the largest considered so far in applications of ParamILS.

We tested our approach on various types of randomly generated problem instances chosen from 9 problem domains used in previous international planning competitions (IPC-3..7), and on the official benchmark problem instances of IPC-6/7. Our results demonstrate that by using automatically determined, domain-optimized configurations (LPG.sd), substantial performance gains can be achieved compared to the default configuration (LPG.d). Using the same automatic configuration approach to optimize the performance of LPG on a merged set of benchmark instances from different domains also results in improvements over the default, but these are less pronounced than those obtained by automated configuration for single domains.

We also investigated to which extent the domain-optimized planners obtained by configuring the general-purpose LPG planner perform well compared to other state-of-the-art domain-independent planners. Our results indicate that, for the class of domains considered in the learning track of the Seventh International Planning Competition (IPC-7), LPG.sd is significantly faster than the top-performing propositional planners of the deterministic track of the last five International Planning Competitions (IPC-3..7).

While in this work we focus on the application of the proposed framework to the LPG planner, we are aware that similarly good results can be obtained for highly parameterized versions of other existing planning systems. This is the case of Fast Downward-Autotune (Fawcett et al. 2011), a system that is based on the same general approach proposed here, but the automated configuration process is performed for a planner with a very different design space than LPG, called Fast Downward (Helmert 2006). We note that the design of Fast Downward-Autotune was explicitly inspired by an earlier version of our work described here, which was later presented at the Third Workshop on Planning and Learning (PAL 2011). More recently, a technique for statically configuring a portfolio of tuned planners was proposed (Seipp et al. 2012). In this work, Seipp et al. used ParamILS for tuning Fast Downward on several different well-known benchmark domains, combining the resulting configurations in a static domain-independent portfolio.

In general, these related works and our results suggest that in the future development of efficient planning systems, it is worth including many different variants and a wide range of settings for the various components instead of committing at design time to particular choices and settings. Additionally, designers and users of planning systems will benefit from using automated procedures to find configurations of the resulting highly parameterized planning systems that perform well on the problems arising in a specific application domain under consideration.

The proposed approach was integrated into two systems that participated in the learning track of the latest international planning competition, IPC-7 (Celorrio, Coles, and Coles 2011): (i) a planning system called ParLPG, combining LPG.sd with HAL, a recently developed computational environment that supports the computer-aided design and empirical analysis of high-performance algorithms (Nell et al. 2011); (ii) a variant of ParLPG incorporated into PbP2, the new version of PbP, a portfolio-based system that automatically determines a series of planners to be run on the domain under consideration (Gerevini, Saetti, and Vallati 2009). PbP2 was the winner of the learning track of IPC-7. As we show here, ParLPG contributed substantially to the successful performance of PbP2.s in IPC-7 and ParLPG alone is competitive with other state-of-the-art planners with learning capabilities.

In the remainder of this paper, we first provide some background and further information on ParamILS and LPG. Next, we describe in detail our experimental analysis and results, followed by concluding remarks and a discussion of some avenues for future work.

Parameter Configuration using ParamILS

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 been shown to be very effective for configuration of algorithms with large numbers of both numerical and categorical parameters (for details, see Hutter et al. 2009).

ParamILS has previously been used to configure the parameters of Spear, a complete, DPLL-style algorithm for the propositional satisfiability (SAT) problem. The resulting configurations were shown to solve a given set of SAT-encoded software verification problems over 100 times faster than previous state-of-the-art solvers for these problems, winning first prize in one category of the 2007 Satisfiability Modulo Theories (SMT) Competition (Hutter et al. 2007). ParamILS has also been applied to SATenstein-LS, a highly parametric solver framework for SAT created by combining many prominent stochastic local search algorithms from the literature (KhudaBukhsh et al. 2009). Several automatically determined configurations of SATenstein-LS were used as part of a submission that won prizes in 5 of the 9 main categories of the 2009 SAT Competition.

ParamILS was also used to configure several prominent solvers for mixed integer programming (MIP) problems, including the widely used industrial CPLEX solver. Despite the fact that the default parameter settings for CPLEX are well-known for having been chosen very carefully and based on a large amount of carefully designed experimentation, substantial performance improvements were obtained for many prominent types of MIP instances (Hutter, Hoos, and Leyton-Brown 2010).

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). 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 planning 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).

```
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.

The Generic Parameterized Planner LPG

LPG (Gerevini, Saetti, and Serina 2003; 2008; 2010) is a versatile system that can be used for plan generation, plan repair and incremental planning (Fox et al. 2006) 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*, which are variants of the very well-known planning graph (Blum and Furst 1997). A linear action graph is a directed acyclic leveled graph that alternates between a proposition level, i.e., a set of domain propositions, and an action level, i.e., one ground domain action and a set of special dummy actions, called "no-ops", each of which propagates a proposition of the previous level to the next one. If an action is in the graph, then its preconditions and positive effects appear in the corresponding proposition levels of the graph.

A pair of propositions, no-ops or actions can be marked as mutually exclusive at every graph level where the pair appears (in LPG only "permanent" mutex relations are considered; for a detailed description, see Gerevini, Saetti, and Serina 2003; 2008; 2010). If a proposition appears at a level of the action graph, then its no-op appears at that level and at every successive graph level until a level containing an action that is marked mutually exclusive with it is reached (if any).

The initial and last levels of every action graph contain the special actions a_{start} and a_{end}, where the effects of a_{start} are the facts of the problem initial state and the preconditions of a_{end} are the problem goals.

The plan represented by an action graph is a valid plan if, and only if, the graph contains no *flaw*, where, intuitively, a flaw is an action in the graph with a precondition that is not supported by the propagation of an effect of another action appearing at a previous graph level.

Starting from the initial action graph containing only the two actions a_{start} and a_{end}, 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. 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.

Domain	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
Merged set	0	1	0	1	5	2	2	11
# parameters	6	15	8	6	17	7	3	62

Table 1: Number of parameters of LPG changed by ParamILS in the configurations computed for our nine benchmark domains considered independently (2nd–10th lines) and together ("Merged set" line). The columns P1-P7 correspond to various categories of parameters (see text), and the last line indicates the total number of LPG parameters in each category.

LPG is a highly parameterized planner. In previous work, the default settings of these parameters had been chosen manually to allow the system to work well on a broad range of domains. For the work presented here, we exposed 4 new parameters that were previously implemented as "magic constants", and documented 50 parameters that were hidden in the code and not exposed to users. This lead to a highly parametrized version of LPG that can be configured very flexibly via 62 exposed configurable parameters which jointly give rise to over 6.5×10^{17} possible configurations. The 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).

The last line of Table 1 shows the number of LPG's parameters that fall into each of these seven components categories.

Experimental Analysis

In this section, we present the results of a large experimental study examining the effectiveness of the automated approach outlined in the introduction.

Benchmark domains and instances

In our first set of experiments, we considered problem instances from nine well-known benchmark domains used in previous international planning competitions, `Blocksworld` (IPC-2/7), `Depots` (IPC-3/7), `Rovers` (IPC-3/5/7), `Satellite` (IPC-3/7), `Zenotravel` (IPC-3), `Gold-miner` (IPC-6), `Matching-BW` (IPC-6), `N-Puzzle` (IPC-6) and `Sokoban` (IPC-6). These domains were selected because they are not trivially solvable and random instance generators are available for them, such that large training and testing sets of instances can be obtained.

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 defined such that the instances were solvable 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 that the instances were solvable by the default configuration of LPG in an average of 50 seconds to 2 minutes and in 3 minutes to 7 minutes, respectively. This does not mean that all of our problem instances can be solved by LPG, since we decided only the *size* of the instances according to the performance of the default configuration, and then used the random generators for deriving the actual instances.

For our second set of experiments, we considered the same problem instances used in IPC-6/7. These were selected in order to compare the configured versions of LPG against the planners that entered the learning track of IPC-6/7, considering the official competition results.

Automated configuration using ParamILS

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, we concurrently performed 10 independent runs of FocusedILS per domain, each using a different random ordering 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 – the so-called PAR10 score. Out of the 10 configurations produced by these runs for each domain, we selected the configuration with the best training set performance (as measured by FocusedILS) as the final configuration of LPG for that domain.

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

Domain	LPG.d		LPG.r	
	Score	% solved	Score	% solved
`Blocksworld`	99.00	99	0.00	16
`Depots`	86.00	86	0.00	18
`Gold-miner`	91.00	91	0.00	19
`Matching-BW`	14.00	14	0.15	9
`N-Puzzle`	59.10	89	34.75	86
`Rovers`	85.81	100	31.21	53
`Satellite`	96.02	100	18.99	37
`Sokoban`	73.20	74	2.06	28
`Zenotravel`	98.70	100	2.47	24
Total	702.8	83.7	89.6	32.2

Table 2: Speed scores and percentage of problems solved by LPG.d and LPG.r for 100 problems in each of 9 domains of benchmark MS.

Additionally, we used FocusedILS for optimizing the configuration of LPG across all of the selected domains together. As with our approach for individual domains, we performed 10 independent runs of FocusedILS starting from the default configuration; again, the single configuration with the best performance on the merged training set as measured by FocusedILS was selected as the final result of the configuration process.

The final configurations thus obtained were then evaluated on the two testing sets of instances (benchmarks MS and LS) for each domain. We used a timeout of 600 CPU seconds for benchmark MS, and 900 CPU seconds for benchmark LS.

For convenience, we define the following abbreviations corresponding to configurations of LPG:

- *Default* (LPG.d): The default configuration of LPG.
- *Random* (LPG.r): Configurations selected independently at random from all possible configurations of LPG.
- *Specific* (LPG.sd): The specific configuration of LPG found by ParamILS for each domain.
- *Merged* (LPG.md): The configuration of LPG obtained by running ParamILS on the merged training set.

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-optimized configurations and in the configuration obtained for the merged training set.

Empirical result 1 *Domain-optimized configurations of* LPG *differ substantially from the default configuration.*

Moreover, we found that usually the changed configurations are considerably different from each other.

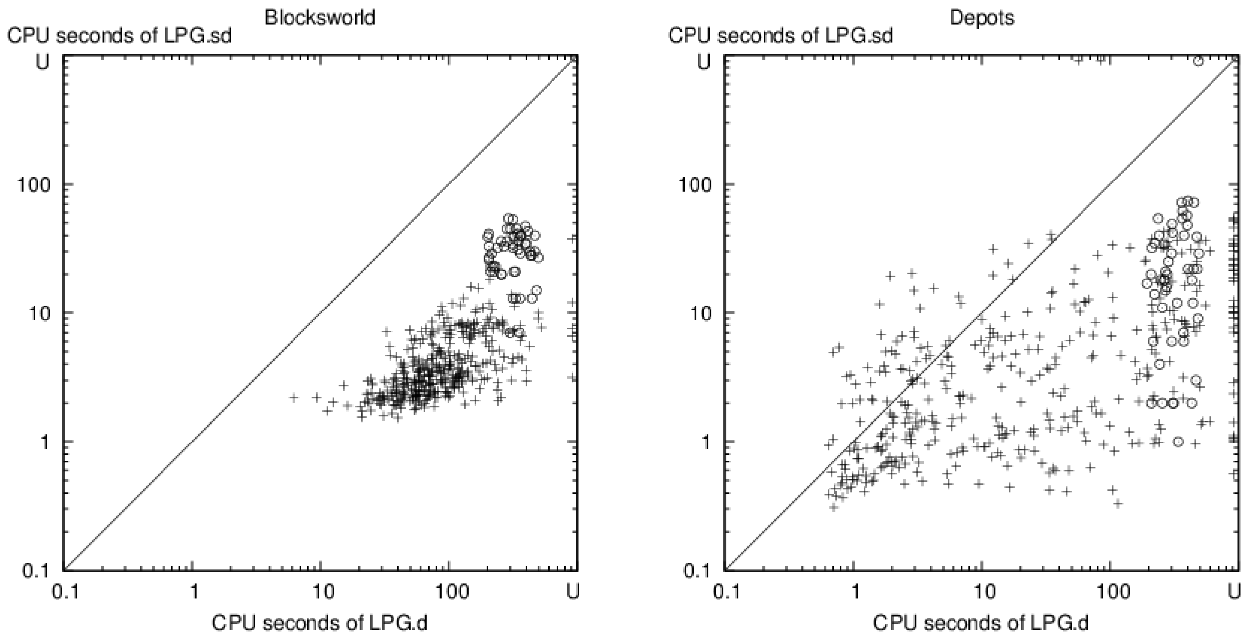

Figure 2: CPU time (log. scale) of LPG.sd w.r.t. LPG.d for problems of 2 domains (`Blocksworld` and `Depots`) of benchmarks `MS` (crosses) and `LS` (circles). The x-axis refers to CPU seconds of LPG.d; the y-axis to CPU seconds of LPG.sd; U corresponds to runs that timed out with the given runtime cutoff.

Results for selected known domains

The performance of each configuration was evaluated using the performance score functions adopted in IPC-6 (Fern, Khardon, and Tadepalli 2008), which is a widely used evaluation criterion in the planning community. The *speed score* of a configuration $\mathcal{C}$ is defined as the sum of the speed scores assigned to $\mathcal{C}$ over all problems in the chosen test set, divided by the total number of problems in the test set. For a given problem p, the speed score $\mathit{Score6}(\mathcal{C}, p)$ is 0 if p is unsolved by any solver (configuration), and $T^*_p / T_p(\mathcal{C})$ otherwise, where T^*_p is the lowest measured CPU time to solve problem p among the compared solvers (configurations), and $T_p(\mathcal{C})$ denotes the CPU time required by $\mathcal{C}$ to solve problem p. Higher values for the speed score indicate better performance.

In order to ascertain the quality of our given default configuration, we performed a comparison between LPG.d and LPG.r on 100 instances from each of the 9 domains of benchmark `MS`. The results of this experiment are shown in Table 2. Considering all domains together, LPG.d obtained a speed score of 702.8, whereas the speed score of LPG.r was 89.6. The best performance for LPG.r was on `N-Puzzle`, with a speed score of 34.75 against 59.10 for LPG.d. For the domains `Blocksworld`, `Depots`, and `Gold-miner`, LPG.r was dominated on every instance by LPG.d.

The superior performance of LPG.d also suggests that the default configuration is a much better starting point for deriving configurations using ParamILS than a random configuration. In order to confirm this intuition, we performed an additional set of experiments using the random configuration as the starting point. As expected, the resulting configurations of LPG perform much worse than LPG.sd.

Empirical result 2 LPG.d *is significantly faster and solves many more problems than* LPG.r.

Figure 2 provides results in the form of a scatterplot, showing the performance of LPG.sd and LPG.d on the problem instances of two individual domains, `Blocksworld` and `Depots`. These domains were chosen as they are the best and worst, respectively, on benchmark set `MS` in terms of the difference in speed score *vs* LPG.d. Figure 3 compares the performance of the two planners over the full benchmarks `MS` and `LS`.

For this analysis, we considered all instances solved by at least one of these planners. In Figure 2, each cross symbol indicates the CPU time used by LPG.d and LPG.sd to solve a particular problem instance of benchmark `MS`. Each circle symbol is defined similarly for benchmark `LS`. When a point appears under (above) the main diagonal, LPG.sd is faster (slower) than LPG.d; the distance of the point from the main diagonal indicates the performance gap (the greater the distance, the greater the gap). The results in Figures 2 and 3 indicate that LPG.sd performs almost always better than LPG.d, often by 1–2 orders of magnitude.

Table 3 shows the performance of LPG.d, LPG.md, and LPG.sd for each domain of benchmarks `MS` and `LS` in terms of speed score, percentage of solved problems and average CPU time (computed over the problems solved by all the considered configurations). These results indicate that LPG.sd solves many more problems, is on average much faster than LPG.d and LPG.md, and that for some benchmark sets LPG.sd *always* performs better than or equal to the other configurations, as the IPC score of LPG.sd is some-

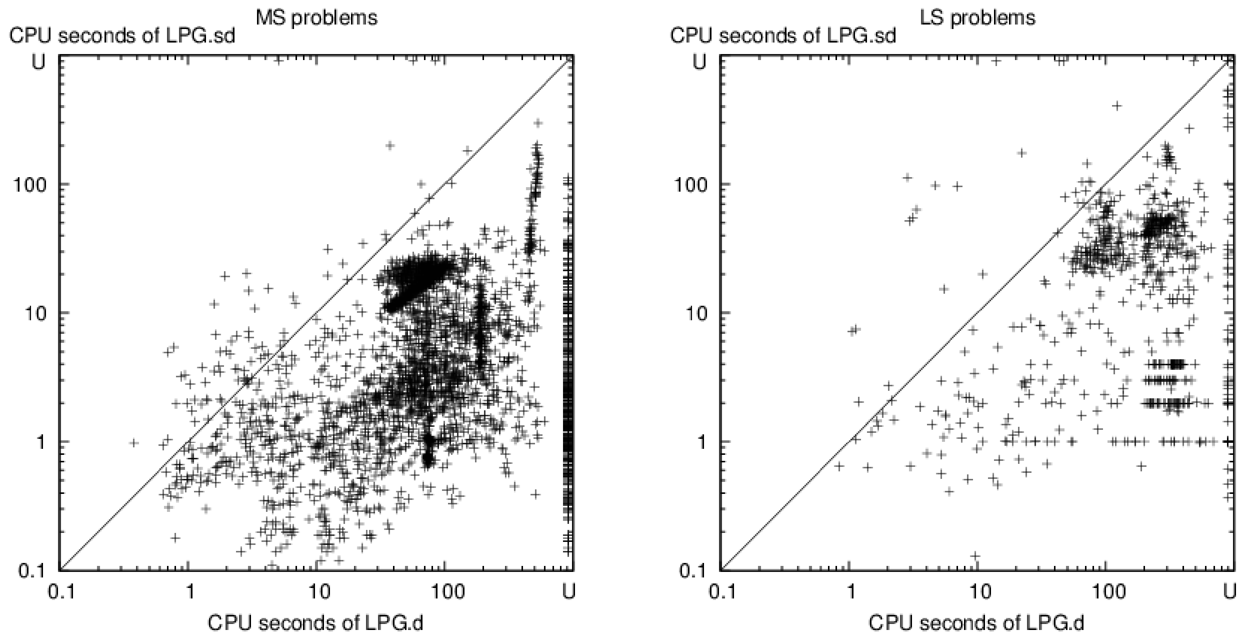

Figure 3: CPU time (log. scale) of LPG.sd with respect to LPG.d for problems of benchmarks `MS` and `LS`. U corresponds to runs that timed out with the given runtime cutoff.

Domain	MS problems						LS problems					
	Average speed score (% solved)			Average CPU time			Average speed score (% solved)			Average CPU time		
	LPG.d	LPG.md	LPG.sd	LPG.d	LPG.md	LPG.sd	LPG.d	LPG.md	LPG.sd	LPG.d	LPG.md	LPG.sd
`Blocksworld`	0.05 (98.8)	0.19 (100)	1.00 (100)	105.3	28.17	4.29	0.10 (100)	0.22 (100)	1.00 (100)	320.9	144.8	30.8
`Depots`	0.31 (90.3)	0.41 (99)	0.86 (98.5)	78.1	42.4	5.7	0.08 (100)	0.35 (100)	0.89 (98)	326.6	181.1	25.7
`Gold-miner`	0.05 (90.5)	0.58 (100)	0.94 (100)	94.4	7.4	1.6	0.03 (100)	0.65 (100)	0.71 (100)	327.2	21.0	21.2
`Matching-BW`	0.24 (15.8)	0.18 (55.3)	0.94 (97.8)	93.8	42.3	5.6	0.03 (86)	0.31 (94)	0.95 (100)	224.9	72.3	1.90
`N-Puzzle`	0.05 (85)	0.07 (86.3)	0.87 (86.8)	321.0	246.6	31.20	0.01 (100)	0.03 (100)	1.00 (100)	343.7	157.6	4.44
`Rovers`	0.33 (100)	0.41 (100)	1.00 (100)	72.2	52.9	21.2	0.19 (100)	0.97 (100)	0.91 (100)	247.8	48.3	52.7
`Satellite`	0.26 (100)	0.28 (100)	1.00 (100)	64.0	59.2	1.3	0.19 (100)	0.58 (100)	1.00 (100)	262.6	85.4	48.9
`Sokoban`	0.67 (75.8)	0.48 (94.8)	0.84 (96.5)	24.6	6.15	1.19	0.09 (62)	0.48 (82)	0.78 (94)	70.8	7.00	4.23
`Zenotravel`	0.12 (100)	0.24 (99.8)	0.99 (100)	103.7	57.6	11.1	0.01 (100)	0.09 (100)	1.00 (100)	293.9	42.9	2.90
All above	0.70 (83.3)	0.76 (91.5)	–	115.4	38.8	–	0.25 (96)	0.99 (100)	–	309.7	81.3	–

Table 3: Average speed score, percentage of solved problems, and average CPU time of LPG.d, LPG.md, and LPG.sd for each of 9 domains, independently considered, and in all domains (last line) of benchmarks `MS` and `LS`.

times the maximum score of 1.0. [2]

Empirical result 3 LPG.sd *performs much better than both* LPG.d *and* LPG.md*.*

Based on the previous results for ParamILS in the literature, we were expecting an improvement over the (already efficient) default configuration, but we were surprised about the magnitude of the improvement observed across all domains. Moreover, the results in Figure 3 and Table 3 also indicate that, for larger test problems, the performance gap between LPG.sd and LPG.d tends to increase: For example, on the middle-size instances of `Matching-BW`, LPG.sd is on average about one order of magnitude faster than LPG.d, while on the largest instances it has an average performance advantage of more than two orders of magnitude.

Empirical result 4 LPG.sd *is faster than* LPG.d *also for instances considerably larger than those used for deriving the planner configurations.*

This observation indicates that the approach used for deriving configurations scales well with increasing problem instance size.

As can be seen from the last line of Table 3, LPG.md performs better than LPG.d on the individual domain test sets. Moreover, it performs better than LPG.d on the sets obtained by merging the test sets for all individual domains, which indicates that by using a merged training set, we successfully produced a configuration with good performance on average across all selected domains.

[2]Additional results using 2000 test problems for each of the nine considered domains of the same size as those used for the training indicate a performance behavior very similar to the one observed for the `MS` and `LS` instances considered in Table 3.

Domain	Speed score		% solved		Average time	
	LPG.d	LPG.sd	LPG.d	LPG.sd	LPG.d	LPG.sd
Barman	–	–	–	–	–	–
Blocksworld	9.5	30	70	100	253.2	30.3
Depots	6.5	16.9	43	63	326.4	88.6
Gripper	17.7	30	100	100	90.4	18.3
Parking	–	–	–	–	–	–
Rovers	15.9	27.7	93	93	107.0	17.3
Satellite	19.4	30	100	100	74.8	17.2
Spanner	14.6	30	100	100	245.3	19.8
Tpp	–	15	–	50	–	56.1

Table 4: Speed score, percentage of solved problems and average CPU time of LPG.d and LPG.sd for 30 instances from the test sets of IPC-7 domains. "–" indicates that the planner failed to solve every problem instance considered for a given domain.

Empirical result 5 LPG.md *performs better than* LPG.d.

Results for the competition benchmarks

In this section, we report experimental results for the benchmark problems of the learning track of IPC-6 and IPC-7 (the only planning competitions featuring a learning track), with two main aims: (i) to illustrate the performance gap between LPG.sd and LPG.d on *existing* benchmark problems, and (ii) to compare the performance of LPG.sd with state-of-the-art planners using learning techniques.

Table 4 shows the performance of LPG.sd and LPG.d for the IPC-7 benchmark problems in terms of percentage of solved problems, speed score and average CPU time. For this analysis, we determined speed score as defined for IPC-7, where for a planner $\mathcal{C}$ and a problem p, *Score7*$(\mathcal{C}, p)$ is 0 if p is unsolved, and $1/(1 + \log_{10}(T_p(\mathcal{C})/T_p^*))$ otherwise.

For each of the IPC-7 domains, the configuration used by LPG.sd was obtained by performing 10 independent FocusedILS runs on 60–70 randomly generated instances that could be solved by LPG.d within 900 CPU seconds. During training, a runtime cutoff of 900 CPU seconds was used for each run of LPG, while the overall time limit of the learning phase for deriving each domain configuration was set to 5 CPU days. From the LPG configurations obtained in the 10 independent runs of ParamILS, the one with the best reported training performance was chosen for subsequent evaluation. The CPU time limit for each run of LPG.sd during testing was 900 CPU seconds – the same as used in IPC-6 and IPC-7.

The results in Table 4 show that, with the exception of the IPC-7 problems from the Barman and Parking domains, LPG.sd achieved excellent performance: for the IPC-7 problems of Depots, Tpp and Blocksworld domains LPG.sd solved many more problems than LPG.d; moreover, it generally obtained speed scores better than those of LPG.d, and, on average, was found to be considerably faster.

Empirical result 6 *For the benchmark domains and problems used in IPC-7,* LPG.sd *performs significantly better than* LPG.d.

Best IPC-6 Planners	% solved	Speed score	Δ-score
LPG.sd	78.9	93.23	+59.7
ObtuseWedge	65.0	63.8	+33.6
PbP.s	96.1	69.16	−3.54
RFA1	52.8	11.4	–
Wizard+FF	43.3	29.5	+10.7
Wizard+SGPlan	51.1	38.2	+7.73
Best IPC-7 Planners	**% solved**	**Speed score**	**Δ-score**
Fast Downward-Autotune	77.0	115.6 (126.1)	+67.5
Fast Downward-Autotune.quality	33.8	35.3	+16.4
OALDAEYASHP	7.40	5.70	−18.0
LPG.sd (ParLPG)	57.0	105.1 (169.2)	+43.7
PbP2.s	88.2	189.8 (223.6)	+128.3
PbP2.q	85.2	71.3	+16.6

Table 5: Performance of the planners that took part in the learning track of IPC-6/7, in terms of the percentage of solved problems, IPC-6/7 speed score and score gap with and without using the learned knowledge for the problems of the learning track of IPC-6/7. The scores in brackets indicate the revised relative performances of the three best performing planners obtained considering the new version of ParLPG (corrected and revised after the competition) instead of the version that entered the competition.

Table 5 shows the performance of LPG.sd in comparison to the planners that participated in the learning track of IPC-6 and IPC-7. Compared to the IPC-6 planners, LPG.sd solved many more problems, obtained better speed scores, and, on average, turned out to be much faster.[3]

LPG.sd directly participated in IPC-7 in the form of a planning system called ParLPG. ParLPG was the 3rd best performing planner in terms of speed, preceded only by PbP2.s and Fast Downward-Autotune.

The fact that PbP2.s performed better than ParLPG is not too surprising, because PbP2.s is a portfolio-based planner incorporating ParLPG as one of its constituent components, and as such had access to additional planning techniques that on some domains perform better than ParLPG alone. However, we observed that, interestingly, in most cases the high performance of PbP2.s was obtained by using ParLPG: for five out of the nine IPC-7 domains (Blocksworld, Gripper, Rovers, Satellite and Spanner), PbP2.s selected ParLPG as the most promising planner to run. For another of these domains (Depots), ParLPG was selected as one of the two best planners to run.

After the competition, we discovered that part of LPG.sd was implemented inefficiently. After revising that part of the code, we repeated the comparison with the other best-performing IPC-7 planners and obtained significantly better results for LPG.sd; in particular, we found PbP2.s still to

[3]Due to not having code from all competitors, the evaluation on the IPC-6 benchmark problems was performed using two different machines. For the planners other than LPG.sd, we used the reported competition runtimes, while LPG.sd was run on a slightly slower machine using information provided by the competition organizers. For the IPC-7 benchmarks, all compared planners were run on the same machine.

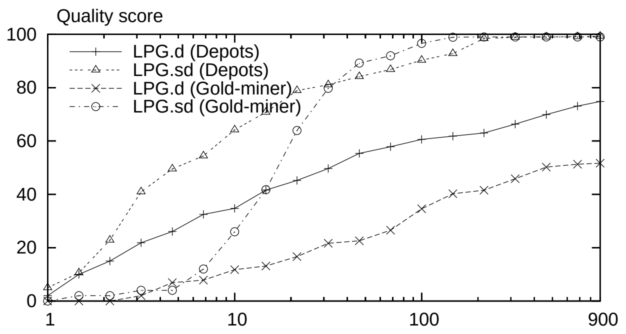

Figure 4: Quality score of LPG.d and LPG using domain-optimized configurations for computing high-quality plans w.r.t. an increasing CPU-time limit (x-axis: ranging from 1 to 900 seconds) for 100 MS problems of domains Depots and Gold-miner.

be the fastest planner, but ParLPG to move into the runner-up position. In our experiments, the IPC-7 speed scores of PbP2.s, ParLPG and Fast Downward-Autotune were 223.6, 169.2 and 126.1 (in parentheses in Table 5), respectively. We of course realize that the other competition planners could be similarly improved, but we include this result for the purpose of comparison.

Empirical result 7 LPG.sd *performs better than all the planners from the learning track of IPC-6 and is competitive with the planners from the learning track of IPC-7.*

The IPC-7 organizers compared the winner of the satisficing deterministic track, LAMA-2011 (Richter and Westphal 2010), with the planners that entered the learning track, showing that ParLPG performs much better in terms of both speed score and number of solved problems. In order to better understand the performance of ParLPG w.r.t. to the state-of-the-art generic planners, we have extended this comparison considering the deterministic track winners of the last four IPCs. For each IPC-7 benchmark problem of the learning track, we compared ParLPG CPU time and the best CPU time over these four planners. Overall, ParLPG obtained a much better speed score (181 against 100) and solved more problems (57% against 47.8%).

Empirical result 8 LPG.sd *performs significantly better than the state-of-the-art generic planners on the IPC-7 benchmarks.*

Further preliminary results on plan quality

Although the experimental analysis in this paper focuses on planning speed, we give some preliminary results indicating that automatic algorithm configuration is also promising for optimizing plan quality. Additional experiments to confirm this observation are in progress. Figure 4 shows results on two benchmark domains (100 problems each from the MS set) in terms of relative solution quality of LPG.sd and LPG.d over CPU time spent by the planner, where, in this context, LPG.sd refers to LPG configured for optimizing plan quality. Training was conducted based on LPG runs with a runtime cut-off of 2 CPU minutes, with the objective to minimize the best plan cost (number of actions) within that time limit. LPG is an incremental planner, and computes a sequence of plans with increasing quality within the specified CPU-time limit (Gerevini, Saetti, and Serina 2008). The quality score of a configuration is defined analogously to the runtime score previously described, but using plan cost instead of CPU time.

Overall, these results indicate that, at least for the domains considered here, our approach finds configurations which produce considerably better quality plans than LPG.d, unless small CPU-time limits are used, in which case they perform similarly.

Conclusions and Future Work

We have investigated the application of computer-assisted algorithm design to automated planning and proposed a framework for automatically configuring a generic planner with many parameterized components to obtain specialized planners that work efficiently on specific domains of interest. In a large-scale empirical analysis, we have demonstrated that our approach, when applied to the state-of-the-art, highly parameterized LPG planning system, effectively generates substantially improved domain-optimized planners. This is also confirmed by the excellent results obtained by the two planning systems using the proposed approach in the most recent planning competition, IPC-7.

Our work and results suggest a potential method for 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.

We see several avenues for future work. Concerning the automatic configuration of LPG, we are conducting an experimental analysis about the usefulness of the proposed framework for identifying configurations improving the planner performance in terms of plan quality. Initial results in this area are promising, and suggest the potential for significant performance improvements. Moreover, we plan to analyze the performance of our framework on metric-temporal planning domains. Finally, we are investigating approaches for determining which parameter settings are important to solver performance, along with investigating this parameter importance across domains. We believe that our approach can yield good results for other planners that have been rendered highly configurable by exposing many parameters, partially borne out by independent results for Fast Downward which strongly suggest that there is great value to creating highly-parameterized planners and applying our approach.

References

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

Celorrio, S. J.; Coles, A.; and Coles, A. 2011. Learning track of the 7th international planning competition. http://www.plg.inf.uc3m.es/ipc2011-learning.

Fawcett, C.; Helmert, M.; Hoos, H.; Karpas, E.; Röger, G.; and Seipp, J. 2011. Fd-autotune: Domain-specific configuration using fast-downward. In *Working notes of the Twenty-first International Conference on Automated Planning and Scheduling (ICAPS-11), Workshop on Planning and Learning*.

Fern, A.; Khardon, R.; and Tadepalli, P. 2008. Learning track of the 6th international planning competition. http://eecs.oregonstate.edu/ipc-learn.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, 212 – 221.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research* 20:239 – 290.

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

Gerevini, A.; Saetti, A.; and Serina, I. 2010. An empirical analysis of some heuristic features for planning through local search and action graphs. *Fundamenta Informaticae* 105:1 – 31.

Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An automatically configurable portfolio-based planner with macro-actions: PbP. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 350 – 353.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191 – 246.

Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research* 24:519 – 579.

Hutter, F.; Babić, D.; Hoos, H. H.; and Hu, A. J. 2007. Boosting verification by automatic tuning of decision procedures. In *Proceedings of Formal Methods in Computer Aided Design (FMCAD'07)*, 27 – 34.

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.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2010. Automated configuration of mixed integer programming solvers. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR-10)*, 186 – 202.

Hutter, F.; Hoos, H. H.; and Stützle, T. 2007. Automatic algorithm configuration based on local search. In *Proceedings of the Twenty-Second Conference on Artifical Intelligence (AAAI '07)*, 1152–1157.

KhudaBukhsh, A. R.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2009. SATenstein: Automatically building local search sat solvers from components. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09)*, 517–524.

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 *Learning and Intelligent Optimization (LION-5)*, 600 – 615.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127 – 177.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning portfolios of automatically tuned planners. In *Proceedings of the Twenty-second International Conference on Automated Planning and Scheduling (ICAPS-12)*.