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

Mauro Vallati[1], Chris Fawcett[2], Alfonso E. Gerevini[1],
Holger H. Hoos[2], and Alessandro Saetti[1]

[1] Dipartimento di Ingegneria dell'Informazione
Università di Brescia, Italy
{mauro.vallati,gerevini,saetti}@ing.unibs.it
[2] Computer Science Department
University of British Columbia, Canada
{fawcettc,hoos}@cs.ubc.ca

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, usually planners are used in default configurations that have been chosen because of their good average performance over a set of benchmark domains, with limited experimentation of the potentially huge range of possible configurations. In this work, we propose a general framework for automatically configuring a parameterized planner, showing that substantial performance gains can be achieved. We apply the framework to the well-known LPG planner, which has 62 parameters and over 6.5×10^{17} possible configurations. We demonstrate that by using this highly parameterized planning system in combination with the off-the-shelf, state-of-the-art automatic algorithm configuration procedure ParamILS, the planner can be specialized obtaining significantly improved performance.

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

Proceedings of the 18th RCRA workshop on *Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion* (RCRA 2011).
In conjunction with IJCAI 2011, Barcelona, Spain, July 17-18, 2011.

from each given domain. However, usually such planners are used with default configurations that have been chosen because of their good average performance over a set of benchmark domains, 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.

In this work, we advocate a different approach, 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 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. To our knowledge, however, these techniques have not yet been applied to the problem of planning.

While our approach 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 [8]. 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 [7] and mixed integer programming (MIP) [9]. 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 [7]; 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 [9]. 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).

One such planning system is LPG (e.g., [3, 4]). Based on a stochastic local search procedure, LPG is a well-known efficient and versatile planner 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. The default settings of these parameters have been chosen to allow the system to work well on a broad range of domains. In this work, we used ParamILS to automatically configure LPG on various propositional domains; LPG's configuration space is one of the largest considered so far in applications of ParamILS.

We tested our approach using ParamILS and LPG on 11 domains of planning problems used in previous international planning competitions (IPC-3–6). 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 our analysis, LPG.sd is significantly faster than LAMA [10], the top-performing propositional planner of the last planning competition (IPC-6).[3]

Moreover, in order to understand how well our approach works compared to state-of-the-of-art systems in automated planning with learning, we have experimentally compared LPG.sd with the planners of the learning track of IPC-6, showing that in terms of speed and usefulness of the learned knowledge our system outperforms the respective IPC-6 winners PbP.s [5] and ObtuseWedge [11].

While in this work, we focus on the application of the proposed framework to the LPG planner, we believe that similarly good results can be obtained for highly parameterized versions of other existing planning systems. In general, 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, 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.

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

The Generic 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 [6]. 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 [1].

Starting from the initial action graph containing only two special actions representing the problem initial state and goals, respectively, LPG iteratively modifies the

[3] The version of LAMA used in the competition has only four Boolean parameters exposed, which its authors recommend to leave unchanged; it is therefore not suitable for studying automatic parameter configuration. A newer, much more flexibly configurable version of LAMA has become available very recently, as part of the Fast Downward system, which we are studying in ongoing work.

```
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 $\mathcal{A}$.
```

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

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.

LPG is a highly parameterized planner with 62 exposed configurable parameters, which control the possible settings of different components in the system. 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).

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
Merged set	0	1	0	1	5	2	2	11
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 (2nd–10th lines) and jointly considered ("merged set" line). Each P1–P7 column corresponds to a different parameter category (or planner component).

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

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. While our analysis is focused on planning speed, we also report preliminary results on plan quality.

Benchmark domains and instances

In our first set of experiments, we considered 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`. 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 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 our problem instances can be solved by LPG, since we have just decided the *size* of the instances according to the performance of the default configuration, and then we have used random generators for deriving the actual instances.

For the experiments comparing automatically determined configurations of LPG against the planners that entered the learning track of IPC-6, we employed the same instance sets as those used in the competition.

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, using random orderings of the training set instances.[4] 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.

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 noticed that usually the changed configurations are considerably different from each other.

[4] 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`.

Results on specific domains

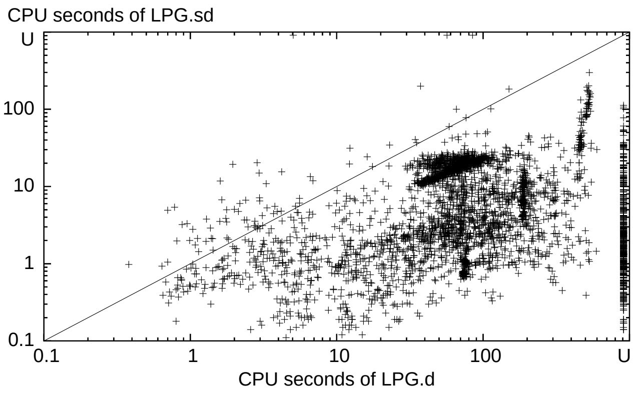
The performance of each configuration was evaluated using the performance score functions adopted in IPC-6 [2]. The *speed score* of a configuration $\mathcal{C}$ is defined as the sum of the speed scores assigned to $\mathcal{C}$ over all test problems. The speed score assigned to $\mathcal{C}$ for a planning problem p is 0 if p is unsolved and $T_p^*/T(\mathcal{C})_p$ otherwise, where T_p^* is the lowest measured CPU time to solve problem p among those of the compared solvers, and $T(\mathcal{C})_p$ denotes the CPU time required by $\mathcal{C}$ to solve problem p. Higher values for the speed score indicate better performance.

Table 2 shows the results of the comparison between LPG.d and LPG.r, which we conducted to assess the performance of the default configuration on our benchmarks.

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

Specifically, LPG.r solves very few problems in 6 of the 9 domains we considered, while LPG.d solves most of the considered problems in all but one domain. This observation 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 starting point. As expected, the resulting configurations of LPG perform much worse than LPG.sd, and even sometimes perform worse than LPG.d.

Figure 2 provides results in the form of a scatterplot, showing the performance of LPG.sd and LPG.d 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.



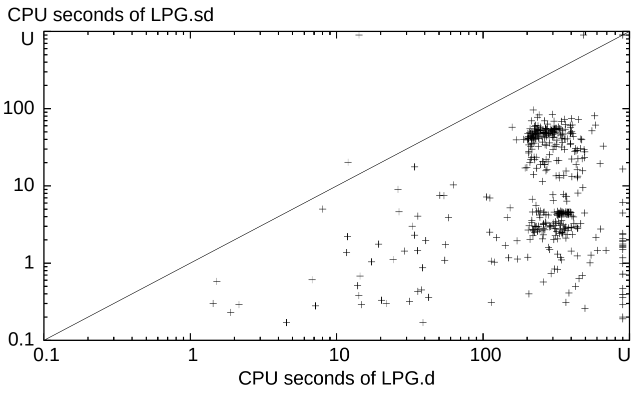


Fig. 2. CPU time (log. scale) of LPG.sd with respect to LPG.d for problems of benchmarks MS (upper plot) and LS (bottom plot). U corresponds to runs that timed out with the given runtime cutoff.

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 sometimes the maximum score (i.e., 400 points for benchmark MS, and 50 for benchmark LS).[5]

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

Interestingly, the results in Figure 2 and Table 3 also indicate that, for larger test problems, the performance gap between LPG.sd and LPG.d tends to increase: For ex-

[5] Additional results (not detailed here for lack of space) 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	MS problems					
	Speed score (% solved)			Average CPU time		
	LPG.d	LPG.md	LPG.sd	LPG.d	LPG.md	LPG.sd
`Blocksworld`	21.3 (98.8)	74.8 (100)	400 (100)	105.3	28.17	4.29
`Depots`	124 (90.3)	164 (99)	345 (98.5)	78.1	42.4	5.7
`Gold-miner`	18.5 (90.5)	232 (100)	374 (100)	94.4	7.4	1.6
`Matching-BW`	9.74 (15.8)	72.5 (55.3)	375 (97.8)	93.8	42.3	5.6
`N-Puzzle`	20.1 (85)	27.0 (86.3)	347 (86.8)	321.0	247	31.20
`Rovers`	131 (100)	162 (100)	400 (100)	72.2	52.9	21.2
`Satellite`	104 (100)	111 (100)	400 (100)	64.0	59.2	1.3
`Sokoban`	26.7 (75.8)	191 (94.8)	335 (96.5)	24.6	6.15	1.19
`Zenotravel`	49.1 (100)	97.2 (99.8)	397 (100)	103.7	57.6	11.1
All above	280.3 (83.3)	304.3 (91.5)	–	115.4	38.8	–

Domain	LS problems					
	Speed score (% solved)			Average CPU time		
	LPG.d	LPG.md	LPG.sd	LPG.d	LPG.md	LPG.sd
`Blocksworld`	5.12 (100)	11.1 (100)	50 (100)	320.9	144.8	30.8
`Depots`	3.91 (100)	17.4 (100)	44.1 (98)	326.6	181.1	25.7
`Gold-miner`	1.54 (100)	32.6 (100)	35.9 (100)	327	21.0	21.2
`Matching-BW`	1.51 (86)	15.2 (94)	47.4 (100)	225	72.3	1.90
`N-Puzzle`	0.66 (100)	1.41 (100)	50 (100)	344	158	4.44
`Rovers`	9.61 (100)	48.5 (100)	45.6 (100)	248	48.3	52.7
`Satellite`	9.43 (100)	28.8 (100)	50 (100)	263	85.4	48.9
`Sokoban`	4.55 (62)	24.0 (82)	38.7 (94)	70.8	7.00	4.23
`Zenotravel`	0.52 (100)	4.26 (100)	50 (100)	294	42.9	2.90
All above	12.6 (96)	49.7 (100)	–	309.7	81.3	–

Table 3. Speed score, percentage of solved problems, and average CPU time of LPG.d, LPG.md and LPG.sd for 400 `MS` and 50 `LS` instances in each of 9 domains, independently considered, and in all domains (last line).

ample, 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 usually 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.

Domain	LPG.sd vs. LAMA		LPG.sd vs. PbP.s	
	Δ-speed	Δ-solved	Δ-speed	Δ-solved
`Blocksworld`	+377.4	+52	+361.7	±0
`Depots`	+393.9	+381	+211.1	+54
`Gold-miner`	+400	+400	+395.6	+319
`Matching-BW`	+227.8	+118	+40.7	+330
`N-Puzzle`	+255.7	+4	+279.8	−20
`Rovers`	+392.9	+14	+313.4	+9
`Satellite`	+388.1	+157	+253.6	+9
`Sokoban`	+340.1	+278	−41.6	+5
`Zenotravel`	+368.3	±0	−282.1	+8
Total	+3144	+1404	+1532	+714

Table 4. Performance gap between LPG.sd and LAMA (2nd-3rd columns) and LPG.sd and PbP.s (4-5th columns) for 400 `MS` problems in each of 9 domains in terms of speed score and number of solved problems.

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

Next, we compared our LPG configurations with state-of-the-art planning systems, namely, the winner of the IPC-6 classical track LAMA (configured to stop when the first solution is computed), and the winner of the IPC-6 learning track, PbP. The performance gap between LPG.sd and these planners for `MS` problems are shown in Table 4, where we report the speed score and the number of solved problems (positive numbers mean that LPG.sd performs better). These experimental results indicate clearly that our configurations of LPG are significantly faster and solve many more problems than LAMA.

Empirical result 6 LPG.sd *performs significantly better than* LAMA *on well-known non-trivial domains.*

Moreover, LPG.sd outperforms PbP.s in most of the selected domains: only for `Sokoban` and `Zenotravel` PbP.s obtains a better speed score (but performs slightly worse in terms of solved problems), and only for `N-Puzzle` it solves more problems (but it is generally slower). Interestingly, for these domains the multiplanner of PbP.s runs a single planner with an associated set of macro-actions; these macro-actions clearly help to significantly speed up the search phase of this planner.

Empirical result 7 *For the considered well-known benchmark domains,* LPG.sd *performs significantly better than* PbP.s.

Results on learning track of IPC-6

To evaluate the effectiveness of our approach against recent learning-based planners, we compared our LPG.sd configurations with planners that entered the learning track

Planner	# unsolved	Speed score	Δ-score
LPG.sd	38	**93.23**	+**59.7**
ObtuseWedge	63	63.83	+33.58
PbP.s	**7**	69.16	−3.54
RFA1	85	11.44	–
Wizard+FF	102	29.5	+10.66
Wizard+SGPlan	88	38.24	+7.73

Table 5. Performance of the top 5 planners that took part in the learning track of IPC-6 plus LPG.sd, in terms of the number of unsolved problems, speed score and score gap with and without using the learned knowledge for the problems of the learning track of IPC-6.

of IPC-6, based on the same performance criteria as used in the competition. Table 5 shows performance in terms of the number of unsolved problems, speed score, and performance gap with and without using the learned knowledge (positive numbers mean that the planner performs better using the knowledge); the results in this table indicate that LPG.sd performs better than every solver that participated in the IPC-6 learning track, including the version of PbP.s which won the IPC-6 learning track. Although LPG.sd solves fewer problems than PbP (obtaining zero score for each unsolved problem), it achieves the best score as it is the fastest planner on 3 domains (`Gold-miner`, `N-Puzzle` and `Sokoban`), and it performs close to PbP.s on one additional domain (`Matching-BW`). Furthermore, the results in Table 5 indicate that the performance gap between LPG.sd and LPG.d is significant, and is greater than the gap achieved by ObtuseWedge, the planner recognised as best learner of the IPC-6 competition.

Empirical result 8 *According to the evaluation criteria of IPC-6,* LPG.sd *performs better than the winners of the learning track for speed and best-learning.*

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 3 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 cut-off of 2 CPU minutes, with the objective to minimise the best plan cost (number of actions) within that time limit (LPG is an incremental planner computing a sequence of plans with increasing quality). 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, LPG.sd always finds considerably better plans than LPG.d, unless small CPU-time limits are used, in which case they perform similarly.

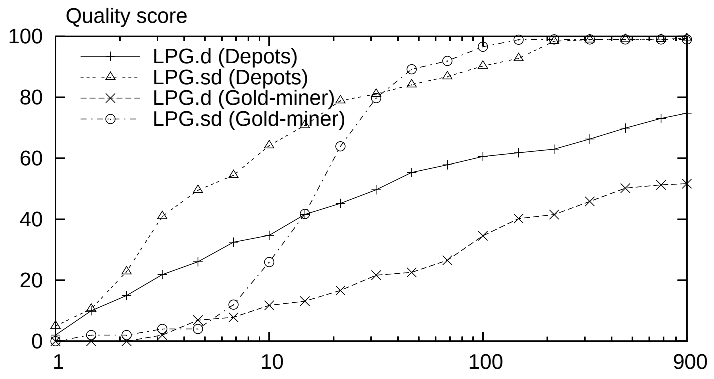


Fig. 3. Quality score of LPG.d and LPG using domain-optimized configurations for computing high-quality plans w.r.t. an increasing CPU-time limit (x-axis: ranging from 1 to 900 seconds) for domains `Depots` and `Gold-miner`.

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 several parameterized components to obtain specialized planners that work efficiently on given domains. 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.

Our work and results also 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, of which in this paper we have given preliminary results. Moreover, we plan to apply the framework to metric-temporal planning domains. Finally, we believe that our approach can yield good results for other planners that have been rendered highly configurable by exposing many parameters. In particular, preliminary results from ongoing work indicate that substantial performance gains can be obtained when applying our approach to a very recent, highly parameterized version of the IPC-4 winner Fast Downward.

References

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

2. Fern, A.; Khardon, R.; and Tadepalli, P. 2008. Learning track of the 6th international planning competition. In *http://eecs.oregonstate.edu/ipc-learn/*.
3. 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.
4. 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.
5. Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An automatically configurable portfolio-based planner with macro-actions: PbP. In *Proc. of ICAPS-09*.
6. Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research* 24:519–579.
7. 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.
8. 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.
9. Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2010. Automated configuration of mixed integer programming solvers. In *Proc. of CPAIOR-10*.
10. Richter, S. Helmert, M., and Westphal, M. 2007. Landmarks revisited. In *Proc. of AAAI-07*.
11. Yoon, S.; Fern, A.; and Givan, R. 2008. Learning control knowledge for forward search planning. *Journal of Machine Learning Research (JMLR)* 9:683–718.