FD-Autotune: Domain-Specific Configuration using Fast Downward

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

Malte Helmert
Albert-Ludwigs-Universität Freiburg
helmert@informatik.uni-freiburg.de

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

Erez Karpas
Technion
karpase@technion.ac.il

Gabriele Röger
Albert-Ludwigs-Universität Freiburg
roeger@informatik.uni-freiburg.de

Jendrik Seipp
Albert-Ludwigs-Universität Freiburg
seipp@informatik.uni-freiburg.de

Abstract

In this work, we present the FD-Autotune learning planning system, which is based on the idea of domain-specific configuration of the latest, highly parametric version of the Fast Downward Planning Framework by means of a generic automated algorithm configuration procedure. We describe how the extremely large configuration space of Fast Downward was restricted to a subspace that, although still very large, can be managed by a state-of-the-art automated configuration procedure. Additionally, we give preliminary results obtained from applying our approach to the nine domains of the IPC-2011 learning track, using the well-known ParamILS configurator and the recently developed HAL experimentation environment.

Introduction

Developers of state-of-the-art, high-performance algorithms for combinatorial problems, such as planning, are frequently faced with many interdependent design choices. These choices can include the heuristics to use during search, options controlling the behaviour of these heuristics, as well as which search techniques to use and in what combination.

Recent work in other combinatorial problem domains such as satisfiability (SAT) and mixed-integer programming (MIP) suggests that by exposing these design choices as parameters, developers can leverage generic tools for automated algorithm configuration to find performance-optimizing configurations of the resulting highly parameterised algorithm (Hutter et al. 2007; Hutter, Hoos, and Leyton-Brown 2010). In fact, the configurations resulting from this process often perform substantially better than those found manually through exploration by human experts.

These results suggest the following new approach to building a learning planner. Given a highly-parametric, general purpose planner P, a representative set I of planning instances from a specific domain, and a performance metric m to be optimised, we can obtain a configuration of the parameters of P optimised for performance on I with respect to m using a generic automated algorithm configuration tool.

For this submission, we apply the above approach using a new, highly-parameterised version of the Fast Downward planning system (Helmert 2006) and the state-of-the-art automated algorithm configuration tool ParamILS (Hutter, Hoos, and Stützle 2007; Hutter et al. 2009), creating domain-specific planning algorithms FD-Autotune.s (*speed*) and FD-Autotune.q (*quality*). FD-Autotune.s refers to the specific configuration of Fast Downward resulting from using mean runtime to find an initial satisficing plan as the optimisation metric, and FD-Autotune.q is the configuration obtained when using mean plan cost after a fixed runtime as the optimisation metric. Due to the highly structured and potentially infinite configuration space of Fast Downward, we carefully limited the number of parameters in order to comply with the requirements of ParamILS and to retain as many potential planner configurations as possible. Our learning approach was implemented to take advantage of HAL, a recently released tool for automating the specification and execution of common empirical algorithm design and analysis tasks (Nell et al. 2011).

The remainder of this paper is organised as follows. First, we describe the Fast Downward Planning Framework, as well as the configuration spaces used for both FD-Autotune.s and FD-Autotune.q. Next, we give a brief overview of both recent work in automated algorithm configuration and of the HAL experimentation environment. We then describe the experimental design of our IPC-2011 learning track submission and give preliminary results for the nine learning track domains. Finally, we briefly discuss some avenues for further work in this area.

The Fast Downward Planning Framework

In this section, we describe the capabilities of the IPC-2011 version of the Fast Downward planning system. Since Fast Downward incorporates many different algorithms and approaches, which have each been published separately in peer-reviewed conferences and/or journals, we will simply list the available components with pointers to further information for the interested reader.

The Fast Downward planning system (Helmert 2006) is composed of three main parts: the translator, the preprocessor, and the search component, which are run sequentially in this order. The translator (Helmert 2009) is responsible for translating the given PDDL task into an equivalent one in SAS$^+$ representation. This is done by finding groups of propositions which are mutually exclusive and combining them into a single SAS$^+$ variable. The preprocessor performs a relevance analysis and precomputes some data structures that are used by the search component and certain heuristics. The search component, whose capabilities we will describe in detail here, searches for a solution to the given SAS$^+$ task.

Search

The search component features three main types of search algorithms:

- Eager Best-First Search — the classic best-first search. The same search code is used for greedy best-first search, A^*, and weighted A^* by plugging in different f functions. The multi-path-dependent *LM-A** (Karpas and Domshlak 2009) is also implemented here.
- Lazy Best-First Search — this is best-first search with deferred evaluation (Richter and Helmert 2009). Here as well, the same search code is used for lazy greedy best-first search and lazy weighted A^* by using a different f function.
- Enforced Hill-Climbing (Hoffmann and Nebel 2001) — an incomplete local search technique. This has been slightly generalised from classic EHC to allow preferred operators from multiple heuristics, as well as enabling or disabling preferred operator pruning.

Each of these search algorithms can take several parameters and use one or more heuristics (heuristic combination methods will be discussed next). In addition, these searches can be run in an iterated fashion. This can be used, for example, to produce *RWA** (Richter, Thayer, and Ruml 2010), the search algorithm used in LAMA (Richter and Westphal 2010).

Heuristic Combination

As mentioned previously, the search algorithms described above can work with multiple heuristic evaluators. There are several heuristic combination methods available in the Fast Downward planning system, which are implemented as different kinds of *open lists*.

Some of these combination methods amount to simple arithmetic combinations of heuristic values and can use a standard (“regular”) open list implementation, while others treat the different heuristic estimates $\langle h_1(s), \dots, h_n(s)\rangle$ as a vector that is not reduced to a single scalar value (Röger and Helmert 2010).[1] As a result, some of these latter methods do not necessarily induce a total order on the set of open states. The following combination methods are available in Fast Downward, in addition to performing a regular search using a single heuristic:

- Max — takes the maximum of several heuristic estimates: $\max\{h_1(s), \dots, h_n(s)\}$.
- Sum — takes the sum or weighted sum of several heuristic estimates: $w_1 h_1(s) + \cdots + w_n h_n(s)$.

[1]To simplify discussion, this description assumes that search algorithm behaviour only depends on heuristic values, but all these algorithms can also take into account path costs, as in A^* or weighted A^*.

- Selective Max (Domshlak, Karpas, and Markovitch 2010) — a learning-based method which chooses one heuristic to evaluate at each state: $h_i(s)$ where i is chosen on a per-state basis using a naive Bayes classifier trained on-line.
- Tie-breaking — considers the heuristics in fixed order: first consider $h_1(s)$; if ties need to be broken, consider $h_2(s)$; and so on.
- Pareto-optimal — considers all states whose heuristic value vector is not Pareto-dominated by another heuristic value vector as candidates for expansion, with selection between multiple candidates performed randomly.
- Alternation (Dual Queue) — uses heuristics in a round-robin fashion: the first expansion uses $h_1(s)$, the second uses $h_2(s)$, and so on until $h_n(s)$ and then continuing again with $h_1(s)$. Alternation can also be enhanced by *boosting* (Richter and Helmert 2009).

Each combination method can take several parameters. One important parameter is whether the open list contains only states which have been reached via preferred operators, or all states.

Moreover, wherever this makes sense, instead of using different *heuristics* as their components, these combination methods can also combine the results of different *open lists* which can themselves employ combination methods, and this nesting can even be performed recursively. For example, it is possible to use alternation over one regular heuristic, one Pareto-based open list, and one open list that uses tie-breaking over various weighted sums.

Such combinations allow us to build the “classic” boosted dual queue of Fast Downward: use an alternation approach, which combines two standard open lists, one of which holds all states, and the other only preferred states, both of which are based on a single heuristic estimate. To use two heuristic estimates as in Fast Diagonally Downward (Helmert 2006) or LAMA (Richter and Westphal 2010), alternation over four open lists would be used (for each heuristic, one holding all states and one holding only preferred states).

Heuristics

So far, we have discussed the search algorithms and heuristic combination methods available in the Fast Downward planning system. We now turn our attention to the heuristics available in Fast Downward. Due to the number of heuristics, we simply list the available heuristics, with pointers to relevant literature.

Admissible Heuristics

- Blind — 0 for goal states, 1 (or cheapest action cost for non-unit-cost tasks) for non-goal states
- $h^{\max}$ (Bonet, Loerincs, and Geffner 1997; Bonet and Geffner 1999) — the relaxation-based maximum heuristic
- h^m (Haslum and Geffner 2000) — a very slow implementation of the h^m heuristic family
- $h^{\mathrm{M\&S}}$ (Helmert, Haslum, and Hoffmann 2007; 2008) — the merge-and-shrink heuristic
- h^{LA} (Karpas and Domshlak 2009; Keyder, Richter, and Helmert 2010) — the admissible landmark heuristic

Algorithm	Categorical	Numeric	Total	Configurations
FD-Autotune.s	40	5	45	2.99×10^{13}
FD-Autotune.q	64	13	77	1.94×10^{26}

Table 1: The number of categorical and numeric parameters in the reduced configuration space for both FD-Autotune.s and FD-Autotune.q, as well as the total number of distinct configurations for each.

- $h^{\text{LM-cut}}$ (Helmert and Domshlak 2009) — the landmark-cut heuristic

Inadmissible Heuristics

- Goal Count — number of unachieved goals
- h^{add} (Bonet, Loerincs, and Geffner 1997; Bonet and Geffner 1999) — the relaxation-based additive heuristic
- h^{FF} (Hoffmann and Nebel 2001) — the relaxed plan heuristic
- h^{cg} (Helmert 2004) — the causal graph heuristic
- h^{cea} (Helmert and Geffner 2008) — the context-enhanced additive heuristic (a generalisation of h^{add} and h^{cg})
- h^{LM} (Richter, Helmert, and Westphal 2008; Richter and Westphal 2010) — the landmark heuristic

Apart from Goal Count, all heuristics listed above are cost-based versions (that is, they support non-unit cost actions). This also allows another option for these heuristics: action-cost adjustment. It is possible to tell the heuristics (as well as the search code) to treat all actions as unit-cost (regardless of their true cost) or to add 1 to all action costs. This has been found to be helpful in tasks with 0-cost actions (Richter and Westphal 2010).

Configuration Space

The configuration space of Fast Downward poses a challenge in formulating the parameter space to be explored by a parameter-tuning algorithm: structured parameters. For example, it is possible to configure an alternation open list that alternates between two internal alternation open lists, each of which alternates between their own internal alternation open lists, and so on. Since ParamILS (Hutter et al. 2007) does not handle structured parameters, we had to limit the configuration space somewhat.

The configuration spaces used in this work (as shown in Table 2, located in the appendix) contain a Boolean parameter for each heuristic (all heuristics for satisficing planning, only admissible heuristics for optimal planning), indicating whether that heuristic is in use or not. The other parameters of the heuristic (if any) are conditional on the heuristic being used.

For optimal planning, the search algorithm is predetermined (A^*), and so our only other choice is, when more than one heuristic is used, how the heuristics are combined (the relevant options are Max and Selective Max). This is controlled by another parameter, which is conditional on more than one heuristic being chosen.

For satisficing planning, the setting that applies to the planning and learning competition, the theoretical configuration space is much more complex, since combination methods such as alternation and weighted sums introduce an infinite set of possibilities.

To keep the configuration space manageable, we only allow one layer of alternation, and its components must be standard open lists (sorted by scalar ranking values), one for each heuristic that was selected, and possibly more if preferred operators are used. In addition, we can combine search algorithms using iterated search as in *RWA**. Here, we limit the number of searches to a maximum of 5, in order to avoid an infinitely large structured configuration space. As shown in Table 1, FD-Autotune.s and FD-Autotune.q have many parameters, with 2.99×10^{13} and 1.94×10^{26} distinct configurations, respectively. (The difference is due to the fact that iterated search is not very useful for the "speed" setting, and hence is not enabled there.) These configuration spaces are some of the largest ever experimented with using automated algorithm configuration tools.

Automated Configuration

For the configuration task faced in the context of this work, we chose to use the FocusedILS variant of ParamILS (Hutter, Hoos, and Stützle 2007; Hutter et al. 2009), because it is the only procedure we are aware of that has been demonstrated to perform well on algorithm configuration problems as hard as the one encountered here. ParamILS is fundamentally based on Iterated Local Search (ILS), a well-known, general stochastic local search method that interleaves phases of simple local search – in particular, iterative improvement – with so-called perturbation phases that are designed to escape from local optima.

In the FocusedILS variant of ParamILS, ILS is used to search for high-performance configurations of a given target algorithm (here: Fast Downward) by evaluating promising configurations. To avoid wasting CPU time on poorly-performing configurations, FocusedILS carefully controls the number of target algorithm runs performed for candidate configurations; it also adaptively limits the amount of runtime allocated to each algorithm run using knowledge of the best-performing configuration found so far. Further information on ParamILS can be found in earlier work by Hutter, Hoos, and Stützle (2007) and Hutter et al. (2009), and interesting applications have been reported by Hutter et al. (2007), and Hutter, Hoos, and Leyton-Brown (2010).

Implementation using HAL

For realising our learning planning system as well as for all experiments performed in this work, we took advantage of the features in HAL, a recently developed tool to support both the computer-aided design and the empirical analysis of high-performance algorithms (Nell et al. 2011). We used several meta-algorithmic procedures provided by HAL, primarily the algorithm configuration tool ParamILS and the plug-ins providing support for empirical analysis of one or two algorithms. We also leveraged the robust support in

HAL for data management and run distribution on compute clusters.

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

After all ten configuration runs have completed, we run Fast Downward 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 FD-Autotune.s, this score is the mean runtime required to find a satisficing solution, and for FD-Autotune.q it represents the mean plan cost, with timeouts assigned a (dummy) cost of $2^{31} - 1$. The incumbent configuration with the best training score is returned as the learned knowledge for the given domain.

Preliminary IPC-2011 Learning Track Results

We have applied the framework introduced in this work to the domains used for the learning track of the 7th International Planning Competition (IPC-2011), currently in progress at the time of this writing. The training sets for each domain used for configuration consisted of 60 randomly generated instances, selected such that the default configurations of Fast Downward could find an initial satisficing solution in less than 3 minutes of CPU time. Target instance distributions were provided by the competition organizers, and our test sets for each domain contained 30 randomly generated instances from the same distribution.

The FD-Autotune.s configurations for each domain are shown in Table 3 (located in the appendix), and performance comparisons between the FD-Autotune.s default configuration and the optimised configurations on each domain are shown in Figures 1 and 2. From these results, it is clear that the configuration of FD-Autotune.s is very successful in all domains, although neither the default nor the optimised configuration for the Spanner domain can solve any instances from the test set within the given CPU time limits.

Unfortunately, this process did not result in adequate performance from FD-Autotune.q, as the tuned configurations never outperformed FD-Autotune.s and in many cases could not solve the instances in our test sets. We believe that this is because the tuned configurations were optimised for producing plans of high quality on the (easier) training sets, without any regard to the speed with which they found a solution. Additionally, due to the fixed runtime cutoff of 900 CPU-seconds and the lack of adaptive capping when configuring for solution quality with ParamILS, much fewer runs of Fast Downward could be performed in the time allocated for configuration. As a result, the performance of these solvers on the training sets did not scale to the much harder test sets.

Conclusions and Future Work

We believe that the generic approach underlying our work on FD-Autotune 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. Algorithm developers can then use automated procedures for finding configurations of the resulting highly parameterised planning systems that perform well on the problems arising in a specific application domain (or domains) under consideration. We plan to further investigate framing the highly structured and potentially infinite space of Fast Downward in ways that permit the effective use of automated algorithm configuration procedures, such as ParamILS.

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 developing automated algorithm configuration procedures. In principle, planning systems developed in this way can also be used in combination with techniques for automated algorithm selection, giving even greater performance than any single configuration alone (Xu et al. 2008; 2009; Xu, Hoos, and Leyton-Brown 2010). 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-parameterised 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.

Acknowledgements The authors would like to thank WestGrid and Compute-Calcul Canada for providing access to the cluster hardware used in our experiments, and Chris Nell for providing support for HAL. HH acknowledges funding through the MITACS NCE and through an NSERC Discovery Grant.

References

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *ECP*, 360–372.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *AAAI*, 714–719.

Domshlak, C.; Karpas, E.; and Markovitch, S. 2010. To max or not to max: Online learning for speeding up optimal planning. In *AAAI*, 1071–1076.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *AIPS*, 140–149.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS*, 162–169.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In *ICAPS*, 140–147.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, 176–183.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2008. Explicit-state abstraction: A new method for generating heuristic functions. In *AAAI*, 1547–1550.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *ICAPS*, 161–170.

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

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173(5–6):503–535.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

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* 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.; 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. 186–202.

Hutter, F.; Hoos, H. H.; and Stützle, T. 2007. Automatic algorithm configuration based on local search. In *AAAI*, 1152–1157.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *IJCAI*, 1728–1733.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In *ECAI*, 335–340.

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.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *ICAPS*, 273–280.

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

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *AAAI*, 975–982.

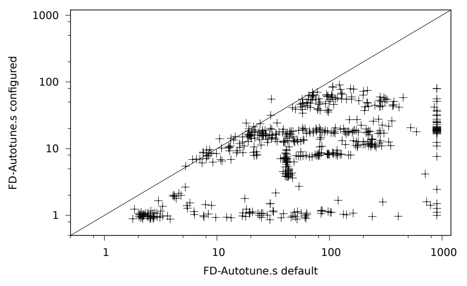
Richter, S.; Thayer, J. T.; and Ruml, W. 2010. The joy of forgetting: Faster anytime search via restarting. In *ICAPS*, 137–144.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In *ICAPS*, 246–249.

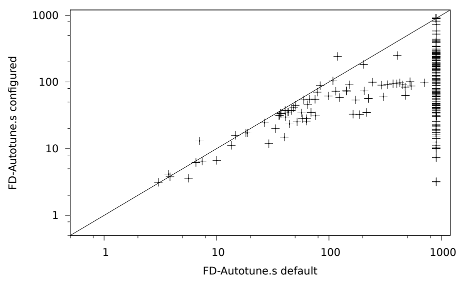
Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research* 32:565–606.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2009. SATzilla2009: an automatic algorithm portfolio for SAT. Solver description, SAT competition 2009.

Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In *AAAI*, 210–216.

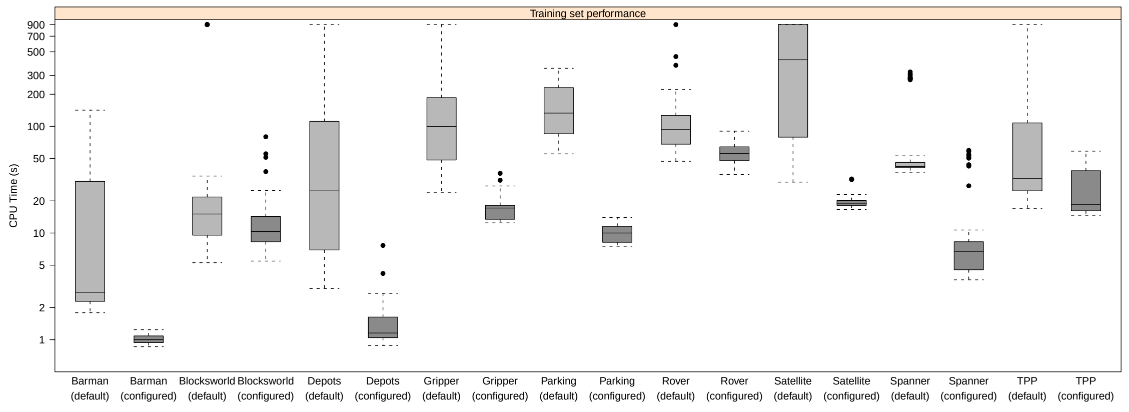


(a) Training set performance

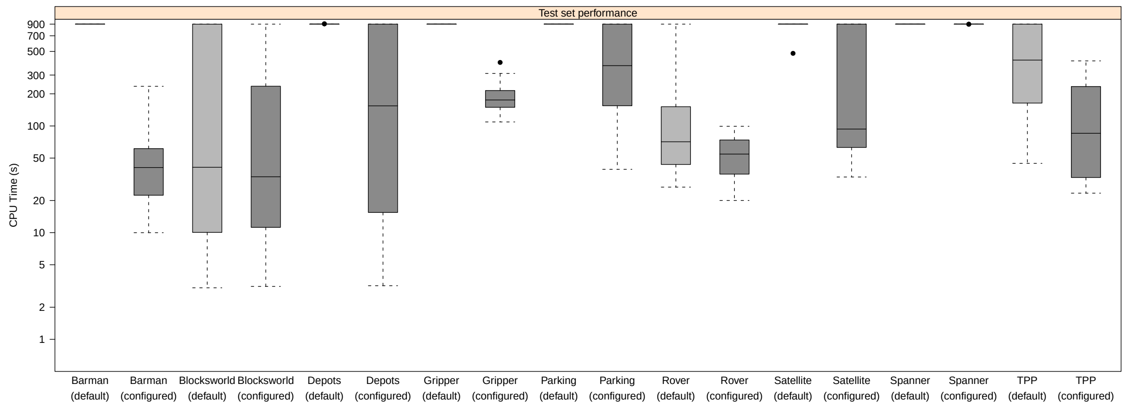


(b) Test set performance

Figure 1: These scatter plots show the performance increase realised by the configured FD-Autotune.s compared to the default, using runs of 900 CPU seconds on 540 (training) and 270 (test) instances obtained by combining our respective training and test sets for all nine IPC-2011 domains. Points below the main diagonal indicate instances where the configured FD-Autotune.s outperforms the default, and in this case this outperformance is often of several orders of magnitude.



(a) Training set performance



(b) Test set performance

Figure 2: Box plots for the CPU time used by the default and automatically configured FD-Autotune.s, on the training and test sets for each of the nine IPC-2011 domains. Each training (test) set was composed of 60 (30) instances, and each run of Fast Downward was allocated 900 CPU seconds of runtime for both the training and the test sets. Box plots for all default configurations are light grey, while the plots for the configured FD-Autotune.s are dark grey. Note that for 5 of these domains, the default configuration fails to solve all or nearly all of the instances in the test set for that domain.

Parameter name	Domain	FD-Autotune.s Default	FD-Autotune.q Default
add_heuristic_enabled	{*true*, *false*}	*false*	*true*
add_heuristic_cost_type	$\{0, 1, 2\}$	–	0
add_heuristic_pref_ops	{*true*, *false*}	–	*false*
blind_heuristic_enabled	{*true*, *false*}	*false*	*false*
cea_heuristic_enabled	{*true*, *false*}	*false*	*true*
cea_heuristic_cost_type	$\{0, 1, 2\}$	–	0
cea_heuristic_pref_ops	{*true*, *false*}	–	*true*
cg_heuristic_enabled	{*true*, *false*}	*false*	*true*
cg_heuristic_cost_type	$\{0, 1, 2\}$	–	2
cg_heuristic_pref_ops	{*true*, *false*}	–	*false*
ff_heuristic_enabled	{*true*, *false*}	*true*	*false*
ff_heuristic_cost_type	$\{0, 1, 2\}$	1	–
ff_heuristic_pref_ops	{*true*, *false*}	*true*	–
goalcount_heuristic_enabled	{*true*, *false*}	*false*	*true*
goalcount_heuristic_cost_type	$\{0, 1, 2\}$	–	0
goalcount_heuristic_pref_ops	{*true*, *false*}	–	*true*
hm_heuristic_enabled	{*true*, *false*}	*false*	*false*
hm_heuristic_m	$\{1, 2, 3\}$	–	–
hmax_heuristic_enabled	{*true*, *false*}	*false*	*false*
lm_ff_synergy	{*true*, *false*}	–	–
lm_heuristic_enabled	{*true*, *false*}	*false*	*false*
lm_heuristic_admissible	{*true*, *false*}	–	–
lm_heuristic_conjunctive_landmarks	{*true*, *false*}	–	–
lm_heuristic_cost_type	$\{0, 1, 2\}$	–	–
lm_heuristic_disjunctive_landmarks	{*true*, *false*}	–	–
lm_heuristic_hm_m	$\{1, 2, 3\}$	–	–
lm_heuristic_no_orders	{*true*, *false*}	–	–
lm_heuristic_only_causal_landmarks	{*true*, *false*}	–	–
lm_heuristic_pref_ops	{*true*, *false*}	–	–
lm_heuristic_reasonable_orders	{*true*, *false*}	–	–
lm_heuristic_type	{*lm_rhw*, *lm_zg*, *lm_hm*, *lm_exhaust*, *lm_rhw_hm1*}	–	–
lmcut_heuristic_enabled	{*true*, *false*}	*false*	*false*
lmcut_heuristic_cost_type	$\{0, 1, 2\}$	–	–
mas_heuristic_enabled	{*true*, *false*}	*false*	*false*
mas_heuristic_max_states	$\{10\,000, 50\,000, 100\,000, 150\,000, 200\,000\}$	–	–
mas_heuristic_merge_strategy	$\{5\}$	–	–
mas_heuristic_shrink_strategy	$\{4, 7, 6, 12\}$	–	–
search_0_cost_type	$\{0, 1\}$	1	1
search_0_eager_pathmax	{*true*, *false*}	–	–
search_0_ehc_preferred_usage	$\{0, 1\}$	–	–
search_0_search_boost	$\{0, 100, 200, 500, 1\,000, 2\,000, 5\,000\}$	2000	1000
search_0_search_open_list_tb	{*true*, *false*}	*false*	*false*
search_0_search_reopen	{*true*, *false*}	*false*	*false*
search_0_search_w	$\{1, 1.125, 1.25, 1.5, 2, 3, 5, 7, 10, \infty\}$	10	7
search_0_type	{*none*, *ehc*, *eager*, *lazy*}	*lazy*	*lazy*
search_1_cost_type	$\{0, 1\}$	–	0
search_1_eager_pathmax	{*true*, *false*}	–	–
search_1_ehc_preferred_usage	$\{0, 1\}$	–	–
search_1_search_boost	$\{0, 100, 200, 500, 1\,000, 2\,000, 5\,000\}$	–	5000
search_1_search_open_list_tb	{*true*, *false*}	–	*true*
search_1_search_reopen	{*true*, *false*}	–	*false*
search_1_search_w	$\{1, 1.125, 1.25, 1.5, 2, 3, 5, 7, 10, \infty\}$	–	3
search_1_type	{*none*, *ehc*, *eager*, *lazy*}	–	*lazy*
search_2_cost_type	$\{0, 1\}$	–	0
search_2_eager_pathmax	{*true*, *false*}	–	*true*
search_2_ehc_preferred_usage	$\{0, 1\}$	–	–
search_2_search_boost	$\{0, 100, 200, 500, 1\,000, 2\,000, 5\,000\}$	–	500
search_2_search_open_list_tb	{*true*, *false*}	–	*true*
search_2_search_reopen	{*true*, *false*}	–	*true*
search_2_search_w	$\{1, 1.125, 1.25, 1.5, 2, 3, 5, 7, 10, \infty\}$	–	10
search_2_type	{*none*, *ehc*, *eager*, *lazy*}	–	*eager*
search_3_cost_type	$\{0, 1\}$	–	–
search_3_eager_pathmax	{*true*, *false*}	–	–
search_3_ehc_preferred_usage	$\{0, 1\}$	–	–
search_3_search_boost	$\{0, 100, 200, 500, 1\,000, 2\,000, 5\,000\}$	–	–
search_3_search_open_list_tb	{*true*, *false*}	–	–
search_3_search_reopen	{*true*, *false*}	–	–
search_3_search_w	$\{1, 1.125, 1.25, 1.5, 2, 3, 5, 7, 10, \infty\}$	–	–
search_3_type	{*none*, *ehc*, *eager*, *lazy*}	–	*none*
search_4_cost_type	$\{0, 1\}$	–	–
search_4_eager_pathmax	{*true*, *false*}	–	–
search_4_ehc_preferred_usage	$\{0, 1\}$	–	–
search_4_search_boost	$\{0, 100, 200, 500, 1\,000, 2\,000, 5\,000\}$	–	–
search_4_search_open_list_tb	{*true*, *false*}	–	–
search_4_search_reopen	{*true*, *false*}	–	–
search_4_search_w	$\{1, 1.125, 1.25, 1.5, 2, 3, 5, 7, 10, \infty\}$	–	–
search_4_type	{*none*, *ehc*, *eager*, *lazy*}	–	*none*

Table 2: Parameters in the configuration space for the satisficing planner, comprising 45 parameters for FD-Autotune.s and 77 parameters for FD-Autotune.q. The parameters for each heuristic are only active if the corresponding heuristic is enabled. If *search_i_type* is *none* for some *i*, then that entry is left out of the iterated search in Fast Downward. "–" indicates that the given parameter is not active.

Parameter name	FD-Autotune.s Default	Barman	Blocksworld	Depots	Gripper	Parking	Rover	Satellite	Spanner	Tpp
add_heuristic_enabled	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*
add_heuristic_cost_type	–	–	–	–	–	–	–	–	–	–
add_heuristic_pref_ops	–	–	–	–	–	–	–	–	–	–
blind_heuristic_enabled	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*false*
cea_heuristic_enabled	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*false*
cea_heuristic_cost_type	–	–	–	–	–	–	–	–	1	–
cea_heuristic_pref_ops	–	–	–	–	–	–	–	–	*true*	–
cg_heuristic_enabled	*false*	*false*	*false*	*false*	*false*	*true*	*false*	*true*	*false*	*false*
cg_heuristic_cost_type	–	–	–	–	–	1	–	2	–	–
cg_heuristic_pref_ops	–	–	–	–	–	*true*	–	*true*	–	–
ff_heuristic_enabled	*true*	*true*	*true*	*false*	*true*	*false*	*true*	*false*	*false*	*true*
ff_heuristic_cost_type	1	2	1	–	0	–	1	–	–	1
ff_heuristic_pref_ops	*true*	*false*	*true*	–	*false*	–	*false*	–	–	*false*
goalcount_heuristic_enabled	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*false*	*false*
goalcount_heuristic_cost_type	–	–	–	–	–	–	–	2	–	–
goalcount_heuristic_pref_ops	–	–	–	–	–	–	–	*true*	–	–
hm_heuristic_enabled	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*
hm_heuristic_m	–	–	–	–	–	–	–	–	–	–
hmax_heuristic_enabled	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*
lm_ff_synergy	–	*true*	–	–	*true*	–	*true*	–	–	*true*
lm_heuristic_enabled	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*false*	*false*	*true*
lm_heuristic_admissible	–	*false*	–	*true*	*false*	*false*	*false*	–	–	*false*
lm_heuristic_conjunctive_landmarks	–	*true*	–	*false*	*true*	*true*	*true*	–	–	*true*
lm_heuristic_cost_type	–	2	–	0	2	0	0	–	–	2
lm_heuristic_disjunctive_landmarks	–	–	–	–	–	–	–	–	–	–
lm_heuristic_hm_m	–	1	–	1	1	1	1	–	–	1
lm_heuristic_no_orders	–	*true*	–	*true*	*true*	*false*	*false*	–	–	*true*
lm_heuristic_only_causal_landmarks	–	–	–	–	–	–	–	–	–	–
lm_heuristic_pref_ops	–	*true*	–	–	*true*	*false*	*true*	–	–	*true*
lm_heuristic_reasonable_orders	–	*false*	–	–	*true*	*false*	*false*	–	–	*true*
lm_heuristic_type	–	*lm_hm*	–	*lm_hm*	*lm_hm*	*lm_hm*	*lm_hm*	–	–	*lm_hm*
lmcut_heuristic_enabled	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*
lmcut_heuristic_cost_type	–	–	–	–	–	–	–	–	–	–
mas_heuristic_enabled	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*
mas_heuristic_max_states	–	–	–	–	–	–	–	–	–	–
mas_heuristic_merge_strategy	–	–	–	–	–	–	–	–	–	–
mas_heuristic_shrink_strategy	–	–	–	–	–	–	–	–	–	–
search_0_cost_type	1	1	1	0	1	0	0	0	1	0
search_0_eager_pathmax	–	–	–	–	–	–	–	–	*false*	–
search_0_ehc_preferred_usage	–	–	–	–	–	–	–	–	–	–
search_0_search_boost	2000	200	5000	200	2000	5000	500	0	5000	2000
search_0_search_open_list_tb	*false*	–	–	*false*	–	–	*true*	–	*false*	–
search_0_search_reopen	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
search_0_search_w	10	∞	∞	3	∞	∞	10	∞	1	∞
search_0_type	*lazy*	*lazy*	*lazy*	*lazy*	*lazy*	*lazy*	*lazy*	*lazy*	*eager*	*lazy*

Table 3: Results for FD-Autotune.s on the nine provided IPC-2011 learning track domains. "–" indicates that the given parameter is not active.