FD-Autotune: Automated Configuration of 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

The FD-Autotune submissions for the IPC-2011 sequential tracks consist of three instantiations of the latest, highly parametric version of the Fast Downward Planning Framework. These instantiations have been automatically configured for performance on a wide range of planning domains, using the well-known ParamILS configurator. Two of the instantiations were entered into the sequential satisficing track and one into the sequential optimising track. 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 state-of-the-art automated configuration procedures, and how ParamILS was then used to obtain performance-optimised configurations.

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-optimising 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 planning algorithms. Given a highly-parametric, general purpose planner P, a representative set I of planning instances from one or more domains, 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 applied 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 configurator ParamILS (Hutter, Hoos, and Stützle 2007; Hutter et al. 2009) to obtain three configurations of Fast Downward: FD-Autotune-satisficing.1, FD-Autotune-satisficing.2, and FD-Autotune-optimizing. FD-Autotune-satisficing.1 was optimised using mean plan cost after a fixed runtime as the optimisation metric, while the FD-Autotune-optimizing configurations were obtained by using mean runtime to find an optimal plan. FD-Autotune-satisficing.2 is a hybrid planner obtained by inserting a phase at the beginning of search that has been configured to find satisficing plans as quickly as possible. 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. All of our configuration and analysis experiments were performed using HAL, a recently developed software environment for work in empirical algorithmics (Nell et al. 2011).

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 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 algorithms 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 $\langle h_1(s), \ldots, h_n(s)\rangle$ 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 — the maximum of several heuristic estimates: $\max\{h_1(s), \ldots, h_n(s)\}$.
- Sum — the sum or weighted sum of several heuristic estimates: $w_1 h_1(s) + \cdots + w_n h_n(s)$.
- 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 — all states whose heuristic value vector is not Pareto-dominated by another heuristic value vector are candidates for expansion, with selection between multiple candidates performed randomly.
- Alternation (Dual Queue) — heuristics are used in round-robin fashion: the first expansion uses $h_1(s)$, the second uses $h_2(s)$, etc., up to $h_n(s)$, followed again by $h_1(s)$. 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
- $h^{\text{M\&S}}$ (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
- $h^{\text{LM-cut}}$ (Helmert and Domshlak 2009) — the landmark-cut heuristic

[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^*.

Algorithm	Categorical	Numeric	Total	Configurations
FD-Autotune-satisficing	64	13	77	1.935×10^{26}
FD-Autotune-optimizing	20	6	26	6.99×10^{8}

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

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 instruct 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 an automated configurator: 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 neither ParamILS (Hutter et al. 2007) nor any other configuration procedure we are aware of handles such structured parameters, we had to limit the configuration space somewhat.

The configuration spaces used for the competition (as shown in Tables 2, 3, and 4) 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 theoretical configuration space is much more complex, since combination methods such as alternation and weighted sums introduce an infinite set of possibilities.

To keep this 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-optimizing and FD-Autotune-satisficing have many parameters, with 6.99×10^8 and 1.935×10^{26} distinct configurations, respectively. The configuration space of FD-Autotune-satisficing is one 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).

Experiments and Configurations

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 ParamILS algorithm configurator 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. All experiments were performed using a cluster of identical linux-based machines, each with an Intel Xeon E5450 quad-core processor and 6GB of RAM.

For optimising planning, we used HAL to run ten independent runs of ParamILS on a set of 2000 training instances

sampled uniformly at random from the IPC-2008 optimisation track instances located in the Fast Downward benchmark set [2], using a maximum runtime cutoff of 600 CPU seconds for each run of Fast Downward and a total configuration time limit of four CPU days. The objective function used in ParamILS was mean runtime to find an optimal plan, with timed-out runs penalised at 10 times the runtime cutoff. In this case, we were also able to leverage support in ParamILS for adaptive runtime capping to reduce the runtime required for each run of Fast Downward. Out of the ten incumbent configurations produced by ParamILS, we selected the configuration with the best reported training quality to be FD-Autotune-optimizing. The exact parameter values for this configuration of Fast Downward are shown in Table 2.

For satisficing planning, we again performed ten independent runs of ParamILS on a set of 2000 training instances sampled from the entire Fast Downward benchmark set, with instances from the IPC-2008 satisficing track having twice the probability of being selected. We again used a runtime cutoff of 600 CPU seconds for each run of Fast Downward with a total configuration time limit of four CPU days. Here, the objective function used in ParamILS was mean plan cost, with runs that failed to find a satisficing solution assigned a (dummy) plan cost of $2^{31} - 1$ (the default value for solution quality in HAL). Again, we selected the configuration with the best reported training quality to be FD-Autotune-satisficing.1. The exact parameter values for this configuration are shown in Tables 3 and 4.

Finally, we performed an additional ten independent runs of ParamILS using the satisficing planning training set, on a reduced design space containing the parameters from Table 3 and only a single set of search parameters from Table 4. We again used a runtime cutoff of 600 CPU seconds for each run of Fast Downward with a total configuration time limit of four CPU days. The objective function used was mean runtime to find an initial satisficing plan, and Fast Downward was configured to terminate after this solution was found. The configuration with the best reported training quality was selected to form the basis of a hybrid version of the planner, where Fast Downward uses this configuration to find an initial satisficing plan, at which point the FD-Autotune-satisficing configuration with second-best training quality is used. We call this hybrid scheme FD-Autotune-satisficing.2. The parameter values for both phases of FD-Autotune-satisficing.2 are indicated in Tables 3 and 4.

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, and to 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 set of domains, under consideration. We plan to further investigate ways in which automated algorithm configurators, such as ParamILS, can deal more effectively with the highly structured and potentially infinite space of Fast Downward.

We note that our approach naturally benefits from future improvements in planning systems (and in particular, from new heuristic ideas that can be integrated, in the form of parameterised components, into existing, flexible planning systems or frameworks) as well as from progress in automated algorithm configuration procedures. 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. The results of such experiments should indicate to which extent new design elements are useful and also reveal under which circumstances they are most effective.

Acknowledgements The authors would like to thank WestGrid and Compute-Calcul Canada for providing access to the cluster hardware used in our experiments.

[2] This benchmark repository is part of the main Fast Downward repository, accessed from http://www.fast-downward.org

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.

Parameter name	Domain	Default	FD-Autotune-optimizing
blind_heuristic_enabled	$\{true, false\}$	$false$	$false$
hm_heuristic_enabled	$\{true, false\}$	$false$	$false$
hm_heuristic_m	$\{1, 2, 3\}$	2	–
hmax_heuristic_enabled	$\{true, false\}$	$false$	$true$
lm_heuristic_enabled	$\{true, false\}$	$true$	$false$
lm_heuristic_conjunctive_landmarks	$\{true, false\}$	$true$	–
lm_heuristic_disjunctive_landmarks	$\{true, false\}$	$true$	–
lm_heuristic_hm_m	$\{1, 2, 3\}$	1	–
lm_heuristic_no_orders	$\{true, false\}$	$false$	–
lm_heuristic_only_causal_landmarks	$\{true, false\}$	$false$	–
lm_heuristic_type	$\{lm\_rhw, lm\_zg, lm\_hm, lm\_exhaust, lm\_rhw\_hm1\}$	lm_rhw	–
lmcut_heuristic_enabled	$\{true, false\}$	$true$	$true$
mas_heuristic_enabled	$\{true, false\}$	$false$	$false$
mas_heuristic_max_states	$\{10\,000, 50\,000, 100\,000, 150\,000, 200\,000\}$	50 000	–
mas_heuristic_merge_strategy	$\{5\}$	5	–
mas_heuristic_shrink_strategy	$\{4, 7, 6, 12\}$	4	–
combine_heuristics	$\{true, false\}$	$true$	$true$
combine_with_smax	$\{true, false\}$	$true$	$true$
smax_alpha	$\{0.0, 0.1, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.5, 3.0, 4.0, 5.0\}$	1.0	4.0
smax_classifier	$\{0, 1\}$	0	0
smax_conf_threshold	$\{0.51, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.99\}$	0.9	0.85
smax_sample	$\{0, 2\}$	0	0
smax_training_set	$\{10, 50, 100, 500, 1\,000, 5\,000, 10\,000\}$	1 000	10
smax_uniform	$\{true, false\}$	$false$	$true$
a_star_use_mpd	$\{true, false\}$	$true$	$false$
a_star_use_pathmax	$\{true, false\}$	$false$	$true$

Table 2: Configuration space for the optimising planner, comprising 26 parameters. Heuristic parameters are only active if the corresponding heuristic is enabled; $smax\_*$ are only active if $combine\_heuristics$ and $combine\_with\_smax$ are both $true$. "–" indicates that the given parameter is not active.

Parameter name	Domain	Default	FD-Autotune-satisficing.1	FD-Autotune-satisficing.2	
				$Phase1$	$Phase2$
add_heuristic_enabled	$\{true, false\}$	$false$	$false$	$false$	$true$
add_heuristic_cost_type	$\{0, 1, 2\}$	2	–	–	0
add_heuristic_pref_ops	$\{true, false\}$	$true$	–	–	$false$
blind_heuristic_enabled	$\{true, false\}$	$false$	$false$	$false$	$false$
cea_heuristic_enabled	$\{true, false\}$	$false$	$true$	$false$	$true$
cea_heuristic_cost_type	$\{0, 1, 2\}$	2	2	–	0
cea_heuristic_pref_ops	$\{true, false\}$	$true$	$true$	–	$true$
cg_heuristic_enabled	$\{true, false\}$	$false$	$true$	–	$true$
cg_heuristic_cost_type	$\{0, 1, 2\}$	2	1	–	2
cg_heuristic_pref_ops	$\{true, false\}$	$true$	$false$	–	$false$
ff_heuristic_enabled	$\{true, false\}$	$false$	$true$	$true$	$false$
ff_heuristic_cost_type	$\{0, 1, 2\}$	2	0	1	–
ff_heuristic_pref_ops	$\{true, false\}$	$true$	$false$	$true$	–
goalcount_heuristic_enabled	$\{true, false\}$	$false$	$true$	$false$	$true$
goalcount_heuristic_cost_type	$\{0, 1, 2\}$	2	2	–	0
goalcount_heuristic_pref_ops	$\{true, false\}$	$true$	$true$	–	$true$
hm_heuristic_enabled	$\{true, false\}$	$false$	$false$	$false$	$false$
hm_heuristic_m	$\{1, 2, 3\}$	2	–	–	–
hmax_heuristic_enabled	$\{true, false\}$	$false$	$false$	$false$	$false$
lm_ff_synergy	$\{true, false\}$	$true$	–	–	–
lm_heuristic_enabled	$\{true, false\}$	$true$	$false$	$false$	$false$
lm_heuristic_admissible	$\{true, false\}$	$false$	–	–	–
lm_heuristic_conjunctive_landmarks	$\{true, false\}$	$true$	–	–	–
lm_heuristic_cost_type	$\{0, 1, 2\}$	2	–	–	–
lm_heuristic_disjunctive_landmarks	$\{true, false\}$	$true$	–	–	–
lm_heuristic_hm_m	$\{1, 2, 3\}$	1	–	–	–
lm_heuristic_no_orders	$\{true, false\}$	$false$	–	–	–
lm_heuristic_only_causal_landmarks	$\{true, false\}$	$false$	–	–	–
lm_heuristic_pref_ops	$\{true, false\}$	$true$	–	–	–
lm_heuristic_reasonable_orders	$\{true, false\}$	$true$	–	–	–
lm_heuristic_type	$\{lm\_rhw, lm\_zg, lm\_hm, lm\_exhaust, lm\_rhw\_hm1\}$	lm_rhw	–	–	–
lmcut_heuristic_enabled	$\{true, false\}$	$true$	$false$	$false$	–
lmcut_heuristic_cost_type	$\{0, 1, 2\}$	0	$false$	–	–
mas_heuristic_enabled	$\{true, false\}$	$false$	$false$	$false$	–
mas_heuristic_max_states	$\{10\,000, 50\,000, 100\,000, 150\,000, 200\,000\}$	50 000	–	–	–
mas_heuristic_merge_strategy	$\{5\}$	5	–	–	–
mas_heuristic_shrink_strategy	$\{4, 7, 6, 12\}$	4	–	–	–

Table 3: Parameters controlling heuristics in the configuration space for the satisficing planner, comprising 37 parameters. As with the optimising planner configuration space, the parameters for each heuristic are only active if the corresponding heuristic is enabled. "–" indicates that the given parameter is not active.

Parameter name	Domain	Default	FD-Autotune-satisficing.1	FD-Autotune-satisficing.2	
				$Phase1$	$Phase2$
search_0_cost_type	$\{0, 1\}$	0	0	1	1
search_0_eager_pathmax	$\{true, false\}$	$false$	$-$	$-$	$-$
search_0_ehc_preferred_usage	$\{0, 1\}$	0	0	$-$	$-$
search_0_search_boost	$\{0, 100, 200, 500, 1\,000, 2\,000, 5\,000\}$	0	$-$	2000	1000
search_0_search_open_list_tb	$\{true, false\}$	$false$	$-$	$false$	$false$
search_0_search_reopen	$\{true, false\}$	$false$	$-$	$false$	$false$
search_0_search_w	$\{1, 1.125, 1.25, 1.5, 2, 3, 5, 7, 10, -1\}$	-1	$-$	10	7
search_0_type	$\{none, ehc, eager, lazy\}$	$lazy$	ehc	$lazy$	$lazy$
search_1_cost_type	$\{0, 1\}$	0	1	$-$	0
search_1_eager_pathmax	$\{true, false\}$	$false$	$-$	$-$	$-$
search_1_ehc_preferred_usage	$\{0, 1\}$	0	$-$	$-$	$-$
search_1_search_boost	$\{0, 100, 200, 500, 1\,000, 2\,000, 5\,000\}$	0	200	$-$	5000
search_1_search_open_list_tb	$\{true, false\}$	$false$	$false$	$-$	$true$
search_1_search_reopen	$\{true, false\}$	$false$	$false$	$-$	$false$
search_1_search_w	$\{1, 1.125, 1.25, 1.5, 2, 3, 5, 7, 10, -1\}$	-1	1.5	$-$	3
search_1_type	$\{none, ehc, eager, lazy\}$	$lazy$	$lazy$	$-$	$lazy$
search_2_cost_type	$\{0, 1\}$	0	0	$-$	0
search_2_eager_pathmax	$\{true, false\}$	$false$	$-$	$-$	$true$
search_2_ehc_preferred_usage	$\{0, 1\}$	0	$-$	$-$	$-$
search_2_search_boost	$\{0, 100, 200, 500, 1\,000, 2\,000, 5\,000\}$	0	5000	$-$	500
search_2_search_open_list_tb	$\{true, false\}$	$false$	$false$	$-$	$true$
search_2_search_reopen	$\{true, false\}$	$false$	$true$	$-$	$true$
search_2_search_w	$\{1, 1.125, 1.25, 1.5, 2, 3, 5, 7, 10, -1\}$	-1	5	$-$	10
search_2_type	$\{none, ehc, eager, lazy\}$	$lazy$	$lazy$	$-$	$eager$
search_3_cost_type	$\{0, 1\}$	0	$-$	$-$	$-$
search_3_eager_pathmax	$\{true, false\}$	$false$	$-$	$-$	$-$
search_3_ehc_preferred_usage	$\{0, 1\}$	0	$-$	$-$	$-$
search_3_search_boost	$\{0, 100, 200, 500, 1\,000, 2\,000, 5\,000\}$	0	$-$	$-$	$-$
search_3_search_open_list_tb	$\{true, false\}$	$false$	$-$	$-$	$-$
search_3_search_reopen	$\{true, false\}$	$false$	$-$	$-$	$-$
search_3_search_w	$\{1, 1.125, 1.25, 1.5, 2, 3, 5, 7, 10, -1\}$	-1	$-$	$-$	$-$
search_3_type	$\{none, ehc, eager, lazy\}$	$lazy$	$none$	$-$	$none$
search_4_cost_type	$\{0, 1\}$	0	1	$-$	$-$
search_4_eager_pathmax	$\{true, false\}$	$false$	$-$	$-$	$-$
search_4_ehc_preferred_usage	$\{0, 1\}$	0	$-$	$-$	$-$
search_4_search_boost	$\{0, 100, 200, 500, 1\,000, 2\,000, 5\,000\}$	0	1000	$-$	$-$
search_4_search_open_list_tb	$\{true, false\}$	$false$	$false$	$-$	$-$
search_4_search_reopen	$\{true, false\}$	$false$	$true$	$-$	$-$
search_4_search_w	$\{1, 1.125, 1.25, 1.5, 2, 3, 5, 7, 10, -1\}$	-1	2	$-$	$-$
search_4_type	$\{none, ehc, eager, lazy\}$	$lazy$	$lazy$	$-$	$none$

Table 4: Parameters controlling search in the configuration space for the satisficing planner, comprising 40 parameters. 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.