Configurable Heuristic Adaptation for Improving Best First Search in AI Planning

Ivan Serina
Department of Information Engineering
University of Brescia
Brescia, Italy
ivan.serina@unibs.it

Mauro Vallati
School of Computing and Engineering
University of Huddersfield
Huddersfield, United Kingdom
m.vallati@hud.ac.uk

***Abstract*—Automated planning is one of the most prominent AI challenges. In the last few decades, there has been a great deal of activity in designing planning techniques and planning engines, with a focus on forward state-space search. Despite the ubiquitous use of heuristics in AI planning, these techniques are susceptible to being easily trapped by undetected dead ends and huge search plateaus.**

In this paper we introduce a highly configurable heuristic adaptation process based on the idea of dynamically penalising unpromising actions when an inconsistency in the heuristic evaluation is detected; its aim is to reduce the bias affecting specific actions, thereby encouraging exploration by the search process and adding diversity in the neighbourhood selection process. Our extensive experimental analysis demonstrates that the proposed heuristic can be configured to improve significantly the performance of best first search planning on a range of benchmark domains.

***Index Terms*—Automated Planning, Configurable Heuristics, Automated Configuration.**

I. Introduction

Automated planning is a prominent Artificial Intelligence challenge; it has been studied extensively for several decades and led to many real-world applications. Example application domains include drilling [1], transport [2], smart grid [3], and mining [4].

Given the complexity of planning problems, state-of-the-art AI planning techniques heavily rely on heuristic functions in order to guide their search process. An ideal heuristic can provide an inexpensive means to help the search process identifying a sequence of actions that allows to solve the problem at hand, i.e. to reach a goal state starting from a given initial state, and a set of applicable actions.

Unfortunately, existing heuristics are not perfect, and for the sake of minimising their computationally costs can get trapped in dead ends or huge search plateaus that drastically reduce their effectiveness. A plateau is an area of the search space where the heuristic value remains the same, and therefore the heuristic can not provide any guidance to the search. Dead-ends indicate states of the search space from which it is not possible to reach a goal state, and requires to backtrack. These issues are particularly true for domain-independent heuristics, that can not exploit domain-specific knowledge. To minimise their detrimental impact, a number of techniques have been developed in order to identify dead ends in advance and to differentiate the search process in order to avoid plateaus. Some approaches try to diversify the search by occasionally expanding nodes which do not have the lowest heuristic value, providing an opportunity to expand nodes that are mistakenly overlooked due to heuristic errors; see for example DBFS [5], ϵ-GBFS [6], LPG [7], [8] and Type-GBFS [9]. Other approaches try to mitigate the issue related to nodes that are labelled as promising although they are far from the goal (high optimal heuristic value); these techniques include plateau escaping [10], local exploration [11], intra-plateau exploration [12] or tie-breaking [13].

With the aim of improving the effectiveness of domain-independent heuristics, in this paper we explore the dynamic penalisation of actions when an inconsistency in the heuristic evaluation is detected. The underlying idea is to quickly identify inconsistent evaluations, which can be due to the relaxations made by the heuristic, and accordingly modify the way in which the search space is explored. The proposed approach is implemented as a configurable adaptation, that can be optimised, via a number of exposed parameters, in order to control the way in which it dynamically affects the heuristic evaluation. This grants maximal flexibility, as the proposed approach can be used in a domain-independent fashion –by fixing the value of parameters– or finely tuned to a specific domain, or a specific planning problem.

We experimentally evaluated the usefulness of the proposed approach on a range of well-known benchmarks, and using the SMAC tool [14] for identifying both domain-specific and general configurations that allows to improve the performance of a Best-First-Search-based planning engine. The empirical evaluation demonstrates that the exploitation of the introduced heuristic adaptation, properly configured, can lead to (in some cases statistically) significant performance improvements. Furthermore, our analysis provides insights into the most important aspects that affect performance, thus providing helpful insights for further improvements.

The remainder of this paper is organised as follows. First, we provide the necessary background. Second, we introduce the proposed heuristic adaptation approach. Third, we describe the performed experimental analysis, and we discuss the importance of the configurable parameters. Finally, conclusions are given.

II. Background

This section is devoted to provide the required background with regards to automated planning, delete-relaxation heuristic, and algorithm configuration.

A. Automated Planning

Similarly to Bylander's work [15], we define an instance of propositional planning as:

Definition 1: A **propositional STRIPS planning problem** is a tuple $\Pi = \langle \mathcal{P}_r, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ where:

- $\mathcal{P}_r$ is a finite set of ground atomic propositional formulae;
- $\mathcal{I} \subseteq \mathcal{P}_r$ is the initial state;
- $\mathcal{G} \subseteq \mathcal{P}_r$ is the goal specification;
- $\mathcal{A}$ is a finite set of actions, where each action $a \in \mathcal{A}$ has the form $pre(a) \Rightarrow add(a), del(a)$ such that
  - $pre(a) \subseteq \mathcal{P}_r$ are the propositional preconditions,
  - $add(a) \subseteq \mathcal{P}_r$ are the positive postconditions (add list),
  - $del(a) \subseteq \mathcal{P}_r$ are the negative postconditions (delete list)

  and $add(a) \cap del(a) = \emptyset$.

A plan π is an ordered sequence of actions $\pi = (a_1, \ldots, a_m)$, where a_i is an action of π and a precondition f of a plan action a'_i is *supported* if *(i)* f is added by an earlier action a'_j and not deleted by an intervening action a'_k with $j \leq k < i$ or *(ii)* f is true in the initial state and $\nexists a'_k$ with $k < i$ s.t. $f \in del(a'_k)$. The result of applying an action a to a state s is defined as usual:

$$\gamma(s,a) = \begin{cases} (s \cup add(a))/del(a), & \text{if } pre(a) \subseteq s \\ s, & \text{otherwise} \end{cases}$$

A propositional planning task is associated with its state space (S, E), which is a graph structure where S are all states that are reachable from the initial state, and E is the set of all pairs $(s, s') \in S \times S$ of states where there is an action that leads to s' when executed in s.

The complexity of STRIPS planning problems has been studied extensively in the literature. Bylander [15] has defined PLANSAT as the decision problem of determining whether an instance Π of propositional STRIPS planning has a solution or not. PLANMIN is defined as the problem of determining if there exists a solution of length k or less, i.e., it is the decision problem corresponding to the search problem of generating plans with minimal length. Based on this framework, he has analysed the computational complexity of a general propositional planning problem and a number of generalisations and restricted problems. In its most general form, both PLANSAT and PLANMIN are PSPACE-complete. Severe restrictions on the form of the operators are necessary to guarantee polynomial time or even NP-completeness [15].

B. Delete-Relaxation Heuristic

Given the complexity of the problems afforded, a number of heuristic techniques have been developed in order to effectively guide the search process of planning engines. A common approach for deriving a heuristic is to relax the problem $\mathcal{P}$ at hand into a simpler problem $\mathcal{P}'$, which can be solved efficiently. In particular, in automated planning an extremely effective approach is to the ignore negative effects of the actions [16]–[18]. The planners based on this relaxation approximate in different ways the optimal relaxation heuristic which itself is NP-hard to compute; in the following we consider the h_{ff} heuristic [17], but other heuristics can be used as well [19].

More formally, let $\Pi = \langle \mathcal{P}_r, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ a propositional STRIPS planning problem, the corresponding relaxed planning problem $\Pi^+ = \langle \mathcal{P}_r, \mathcal{I}, \mathcal{G}, \mathcal{A}^+ \rangle$ is obtained by Π ignoring the delete effects of its actions $\mathcal{A}$; i.e., the relaxed version of actions $a = (pre(a), add(a), del(a)) \in A$ is defined as $a^+ = (pre(a), add(a), 0)$ and $A^+ = \{a^+ | a \in A\}$. A relaxed plan for a planning problem Π from an initial state s ($\pi_{ff}(s)$) is a solution of the relaxed planning problem Π^+ an corresponds to an action sequence achieving $\mathcal{G}$ when allowing to transition from fact set s to fact set s' via action $a^+ \in A^+$ if $pre(a^+) \subseteq s$ and $s' = s \cup add(a^+)$.

Specifically, first a planning graph for the relaxed planning task is constructed by a GraphPlan-like algorithm [17]. Then a non-optimal relaxed plan, a solution to the relaxed task, can be extracted from the planning graph in polynomial time. The *relaxed planning graph* (RPG) is a layered graph alternating between proposition levels and action levels,

$$F_i = \begin{cases} s, & \text{if } i = 0 \\ F_{i-1} \cup \{add(a^+) | pre(a^+) \subseteq F_{i-1}\}, & \text{if } 0 < i \leq k \end{cases}$$

$$A_i = \{a^+ \in A^+ | pre(a^+) \subseteq F_i\}, \quad 0 \leq i < k$$

where k corresponds to the first index of the relaxed planning graph such that $\mathcal{G} \subseteq F_k$.

After constructing the RPG for a search state s, FF starts at the final layer of RPG and uses a backtrack-free procedure that extracts a sequence of actions that corresponds to a successful relaxed plan for s. During the extraction process, two kinds of sets are maintained: the sequence of sub-goal sets $G_k = \mathcal{G}, G_{k-1} ..., G_1$ that represents the sub-goals first appearing in the respective levels $F_k, ..., F_1$ and the sequence of solution action set $O_{k-1}, ..., O_0$ that represents the actions achieving the sub-goals in $G_k, ..., G_1$. The heuristic value $h_{ff}(s) = \sum_{i=0}^{k-1} |O_i|$ is the number of actions of the relaxed plan $\pi_{ff}(s) = \bigcup_i O_i$, or $h_{ff}(s) = \infty$ if no relaxed plan exists (see [17], [20] for details).

A *flat path* is a path in (S, E) on which the heuristic value does not change. A *plateau of level* l is a set of states that have the same heuristic value l, and that form a strongly connected

component in (S, E). An exit of a plateau is a state s that can be reached from the plateau on a flat path, and that has a better evaluated neighbour, i.e., $(s, s') \in E$ with $h_{ff}(s') < h_{ff}(s)$. A path in S is called *monotone* iff there exist no two consecutive states s_1 and s_2 on it so that $h_{ff}(s_1) < h_{ff}(s_2)$. A *local minimum* is a plateau of level $0 < l < \infty$ that has no exits [21]. Problems related to local minima have been extensively studied by different authors [22]–[25]. In [22], the author defines the exit distance for a local minima and proves that in many domains there are no local minima at all and are proved to be easily solvable by enforced hill climbing with h_{ff}. We say that s is a local minimum if there exists no monotone path to an exit.

Unfortunately, heuristics search techniques can easily get trapped in local minima and plateaus, and this could represent the main reason for their ineffectiveness in specific planning problems. In Section III, we propose an heuristic adaptation process based on the idea of dynamically penalising unpromising actions when an inconsistency in the heuristic evaluation is detected; its aim is to reduce the bias affecting specific actions, thereby encouraging exploration by the search process and adding diversity in the neighbourhood selection process.

C. Algorithm Configuration

Most algorithms have several free parameters that can be adjusted to optimise performance (e.g., solution cost, or runtime to solve a set of instances). Formally, this *algorithm configuration* problem can be stated as follows: given a parameterised algorithm with possible configurations $\mathcal{C}$, a benchmark set Π, and a performance metric $m(c, \pi)$ that measures the performance of configuration $c \in \mathcal{C}$ on instance $\pi \in \Pi$, find a configuration $c \in \mathcal{C}$ that minimises m over Π , i.e., that minimises

$$f(c) = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} m(c, \pi). \quad (1)$$

The AI community has developed dedicated algorithm configuration systems to tackle this problem [14], [26]–[28]. In this work we employ the sequential model-based algorithm configuration method SMAC [14], which represents the state of the art of configuration tools and, in contrast to ParamILS [26], can handle continuous parameters. SMAC uses predictive models of algorithm performance [29] to guide its search for good configurations. More precisely, it uses previously observed ⟨configuration, performance⟩ pairs $\langle c, f(c) \rangle$ and supervised machine learning (in particular, random forests [30]) to learn a function $\hat{f} : \mathcal{C} \rightarrow \mathbb{R}$ that predicts the performance of arbitrary parameter configurations (including those not yet evaluated). The performance data to fit these models is collected sequentially. In a nutshell, after an initialisation phase, SMAC iterates the following three steps: (1) use the performance measurements observed so far to fit a random forest model $\hat{f}$; (2) use $\hat{f}$ to select a promising configuration $c \in \mathcal{C}$ to evaluate next, trading off exploration in new parts of the configuration space and exploitation in parts of the space known to perform well; and (3) run c on one or more benchmark instances and compare its performance to the best configuration observed so far.

SMAC is an anytime algorithm that interleaves the exploration of new configurations with additional runs of the current best configuration to yield both better and more confident results over time. As all anytime algorithms, SMAC improves performance over time, and for finite configuration spaces it is guaranteed to converge to the optimal configuration in the limit of infinite time. Notably, SMAC has shown to be able to successfully configure various aspects of Automated Planning approaches [31]–[33].

III. Heuristic Adaptation

We are now in the position to introduce our heuristic adaptation approach. The underlying idea of the approach is to quickly identify the inconsistencies that can be present in a heuristic evaluation process, and to fix the behaviour of the heuristic by changing the cost associated to the actions selected via penalisation. In a nutshell, the point is to allow the heuristic to "learn" from the past, to improve efficiency.

Firstly, we provide our definition of inconsistent heuristic evaluation, that is the key point of our approach:

Definition 2: Given a generic heuristic function h based on the computation of a relaxed plan π, we say that an action a determines an *inconsistent evaluation* of a state s if, given $s_a = \gamma(s, a)$, $a \in \pi(s)$ and the heuristic evaluation of $h(s_a) > h(s)$ or $a \notin \pi(s)$ and the heuristic evaluation of $h(s_a) < h(s)$.

Inconsistent evaluations are manifest in plateaus or local minima, but they can also happen in other situations during the search process; obviously, they represent circumstances that are not desirable and the aim of the proposed approach is try to avoid these occurrences as much as possible. In order to avoid them, we associate a non negative penalisation cost to each action a which is primarily modified when an inconsistent evaluation associated to a occurs.

In our experiments, the penalised heuristic value of s (denoted as $h_p(s)$) is defined as the total number of actions in the relaxed plan (π_{ff}) plus their penalisation costs. More specifically, during the search process the most promising search state s is selected from the search queue and it is expanded, i.e. its sons are generated considering the applicable actions in s. For each of its sons $s_a = \gamma(s, a)$ obtained by an applicable action a in s we verify if $a \in \pi_{ff}(s)$ and an inconsistent evaluation occurs, in this case the penalisation cost a is modified according to Equation (2).

$$a.pcost = (1 - \lambda_k) \cdot a.pcost + \lambda_r \cdot log(h_p(s_a) - h_p(s) + c(a)) \quad (2)$$

where λ_k and λ_r are discount factors that can be set by the user associated respectively to the original penalisation value of a and to the inconsistencies in the evaluation function. In our implementation of the approach, their values can be set, respectively, using `k` and `r` parameters. Both ranges between $0.0 - 1.0$. $c(a)$ represents the penalisation cost associated to

a, and it is equal to $a.pcost$ when the C parameter is true, 1 when it false.

Moreover, if the action $a \notin \pi_{ff}(s)$ and $h_p(s_a) < h_p(s)$ then it means that a has not been used in the heuristic evaluation of s, but its selection would have determined a positive neighbourhood evaluation; in this case we update the penalisation cost trying to reduce the gap between the heuristic evaluation of s and s_a via Equation (3).

$$a.pcost = (1-\lambda_K) \cdot a.pcost + \lambda_R \cdot log(h_p(s) - h_p(s_a) + c(a)) \quad (3)$$

similarly to equation (2), λ_K and λ_R are penalisation and discount factors associated to the gap in the heuristic evaluation functions. In our implementation, their values are set, respectively, using K and R parameters, and range between $0.0 - 1.0$.

In addition, it is possible to set a maximum value associated to the penalisation cost which is proportional to the number of deletes effects of the action.

$$a.pcost = min(a.pcost, \mathtt{B} \cdot |del(a)|) \quad (4)$$

Equation (4) shows how the maximum value of penalisation is calculated. By default B is set to ∞, which means that no bound is applied. Other considered values are $0.5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$.

Note that the penalisation costs are not only used for defining the heuristic cost of a neighbour element but also during the relaxed plan computation for breaking ties in the selection process of the available actions, so they are also able to modify the structure of relaxed plans build during the search process.

Given a neighbour element associated to a state s, we consider two alternative possibilities: (i) it can be directly inserted in the (priority) search queue (set via a dedicated parameter $\mathtt{a} = 0$) using the heuristic value computed so far; or (ii) it can be inserted in the search queue using $h_{ff}(s)$ as primary index and using $h_p(s)$ for breaking ties (parameter $\mathtt{a} = 1$).

Summarising, the proposed heuristic adaptation approach aims at improving the heuristic evaluation by penalising actions that lead to inconsistent evaluations, and by favouring actions that would have had a beneficial impact on the evaluation. This is done via penalisation and discount factors, that can be tuned –together with additional parameters– using algorithm configuration techniques.

IV. Experimental Analysis

This section aims at investigating the usefulness of the heuristic adaptation technique introduced in this work.

A. Experimental Settings

The proposed heuristic adaptation approach has been implemented in the well-known FF planner [17]. Seven parameters have been exposed, and allow to control the behaviour of the heuristic adaptation via command line: r, R, k, K, C, B, and a. Details of their value ranges, and of the aspects of the adaptation that they control are given in the previous section.

We use SMAC, version 2.08 (publicly available at http://www.aclib.net/SMAC), for identifying a domain-wise configuration of heuristic adaptation's parameters that improves the PAR10 performance of FF, run using best first search and the h_{ff} heuristic. The Penalised Average Runtime (PAR10) is the average runtime where unsolved instances count as $10\times$cutoff time. PAR10 is a metric usually exploited in machine learning and algorithm configuration techniques, as it allows to consider coverage and runtime at the same time. In other words the configuration process aims to tune the system so that coverage is maximised, and solved instances are dealt with as quickly as possible.

We focused our study on domains that have been used in IPCs and for which a randomised problem generator is available: Blocksworld (4 ops version), Hiking, Depots, Logistics, Maintenance, Matching-Bw, Tetris, and ZenoTravel. For our experiments we created approximately 550 (solvable) random instances per domain, and split them randomly into roughly 500 training and 50 test instances.[1]

Experiments were performed on a Intel Xeon E5-2620 2.00GHz. Each configuration run was limited to a single core, and was given an overall runtime and memory limits of 5 days and 8GB, respectively. As in the Agile track of the IPC, the cutoff time for each instance, both for training and testing purposes, was 300 seconds.

TABLE I

Results in terms of PAR10, percentage of solved problems and IPC score, of the considered planning engine running without the heuristic adaptation (D), and with the automatically-configured heuristic adaptation (A). Bold indicates the best performance; bold red stands for statistically different PAR10 performance, according to a Wilcoxon signed rank test.

	PAR10		Solved		IPC score	
Domain	D	A	D	A	D	A
Blocksworld	1340.0	1340.0	56.0	56.0	28.0	28.0
Depots	2944.6	**2297.7**	2.0	**24.0**	1.0	**12.0**
Hiking	1423.1	**1013.9**	54.0	**68.0**	21.1	**33.6**
Logistics	2008.2	**1678.1**	40.0	**54.0**	19.0	**25.9**
Maintenance	2342.9	**712.7**	22.0	**78.0**	10.6	**34.2**
MatchingBw	1292.5	**825.6**	58.0	**74.0**	25.6	**33.1**
Tetris	1885.8	**1462.3**	38.0	**52.0**	16.4	**24.9**
ZenoTravel	2207.8	**539.2**	28.0	**86.0**	10.3	**42.9**
General	1593.7	**1291.4**	47.5	**57.5**	30.6	**46.3**

B. Experimental Results

Table I compares the performance, in terms of PAR10, coverage, and IPC score, achieved by FF when run using the automatically-configured (A) heuristic adaptation, and without it (D). Here the IPC score is calculated as in the Agile track

[1]The benchmarks will be made available via a link in the camera ready copy of this paper.

of the IPC 2014 [34]. For a planner $\mathcal{S}$ and a problem p, $score(\mathcal{S}, p)$ is defined as:

$$score(\mathcal{S}, p) = \begin{cases} 0 & \textit{if } p \textit{ is unsolved} \\ \frac{1}{1+\log_{10}\left(\frac{T_p(\mathcal{S})}{T_p^*}\right)} & \textit{otherwise} \end{cases}$$

where $T_p(\mathcal{S})$ is the CPU time needed by a planner $\mathcal{S}$ to solve the problem p and T_p^* is the CPU-time needed by the best considered planner, otherwise. The total IPC score is the sum the scores achieved on each considered instance.

The initial configuration considered by SMAC during the training process is the default (D). In other words, FF runs without the introduced heuristic adaptation is the baseline used by SMAC during the configuration for evaluating the expected performance improvement. Results in Table I indicate that the use of the introduced technique, when appropriately configured, can lead to significant performance improvement in terms of PAR10. In many of the considered domains, the differences (highlighted in bold in the table) are statistically significant, according to the Wilcoxon [35] signed rank test ($p = 0.05$). Even in cases where differences are not statistically significant, improvements can usually be observed. Performance are also generally improved in terms of coverage and IPC score.

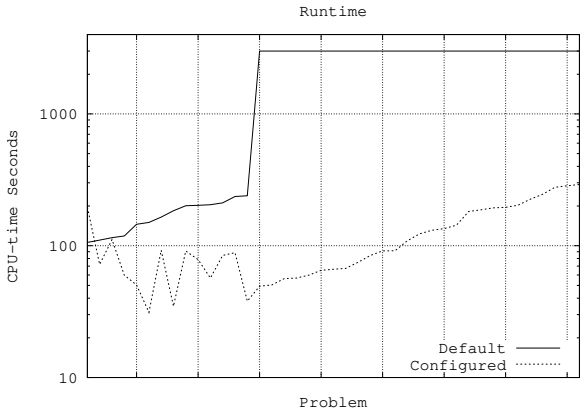

Fig. 1. CPU-time (log-scale) needed to solve the testing instances of the ZenoTravel domain by the considered planning engine running without the heuristic adaptation (Default), and with the automatically-configured heuristic adaptation (Configured). Results include only instances solved by at least one of the systems. As for the PAR10 score, a runtime of 3,000 is used to indicate that the instance is unsolved.

Figure 1 shows how the compared systems perform on the benchmark instances of the ZenoTravel domain. Instances are ordered according to the CPU-time needed by FF, when run without the introduced adaptation, to solve them. The default approach is able to solve only 14 instances, out of 50. On the contrary, the use of the configured heuristic adaptation allows the planning engine to quickly solve more than 40 instances. Generally, the adaptation of the heuristic allows FF to be an order of magnitude faster.

In only one domain, i.e. Blocksworld, SMAC was not able to find a configuration of the heuristic adaptation that allows to improve the performance of the baseline. Our intuition is that in Blocksworld the baseline approach is already delivering very good performance, that cannot be improved by modifying its behaviour.

The last row of Table I shows the performance that can be achieved using a general parametrisation of the heuristic adaptation, obtained by running the configuration process on instances from all the benchmark domains. The results suggest that it is possible to identify a single configuration of the proposed approach that can improve the performance of best first search planning on a wide range of domains. Of course, this does not directly imply that it is possible to identify a domain-independent configuration that will work well on every possible domain. Instead, this result indicates that there are aspects that can be configured in order to improve performance on a range of domains sharing some characteristics. Best performance are achieved via a domain-specific configuration process, but a general configuration can allow to obtain very good average performance. In fact, by carefully analysing the domain by domain results, it is possible to observe that the general configuration does improve the performance of the approach, with regards to the default configuration, in all domains but Blocksworld and Maintenance. Intuitively, this is because in the former the default configuration is already able to deliver very good performance; in Maintenance instead, it seems that due to the peculiar characteristics of the model (i.e., a single operator with conditional effects) a very specific configuration is needed.

The results presented in Table I give an overview of the performance of the considered techniques. However, one interesting aspect to understand is on which kind of instances the heuristic adaptation is more beneficial. *Does the configurable adaptation work better on easy to solve instance, where limited runtime is needed, or is it more beneficial on complex instances that would have required hundreds of seconds to be solved?* To provide some insights on this point, Figure 2 provides the cumulative IPC score over runtime of the the considered planning engine running without the heuristic adaptation (Default), and with the automatically-configured heuristic adaptation (Configured) on the General benchmarks. Considering only instances that are solved in less than 50 CPU-time seconds, the use of the configured heuristic does not lead to significant improvement. Instead, for instances that require more than 50 seconds to be solved, the use of the configured heuristic adaptation gives a remarkable boost to the performance of the planning engine. The major impact comes from instances solved in 50-100 CPU-time seconds, but the performance gap keeps widening after that, due to instances that are not solved by the default configuration. It is worth noting that instances solved in 50-100 CPU-time seconds do not come from a single domain, but are instead

a mix of instance from the considered benchmark domains. The results shown in Figure 2 suggest that the configurable heuristic adaptation is useful on complex instances, while it is not extremely beneficial on easy-to-solve problem instances.

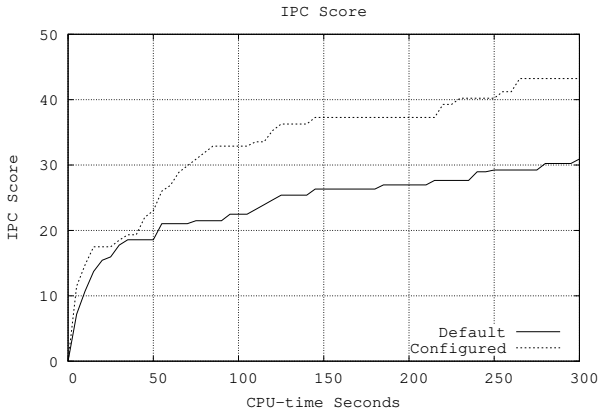

Fig. 2. Cumulative IPC-score over Runtime for the considered planning engine running without the heuristic adaptation (Default), and with the automatically-configured heuristic adaptation (Configured). The General benchmarks are considered.

To shed some light on the impact of the considered techniques on the quality of generated plans, here we focused on two domains in which the use of the learned configuration lead to improvements in terms of runtime: Logistics and Zenotravel. The comparison of generated plans highlighted that the exploitation of configured heuristic adaptations do not result in significant changes in the quality of plans. In the Logistics domain, quality of plans is on average improved by 1.5%, considering only problems solved by default and the configured version. In Zenotravel, instead, the use of the parametrised FF can lead to plans that are, on average, 8.9% worse. Of course, overall quality measured using the IPC quality score introduced in IPCs, is however increased due to the higher coverage performance given by the use of parametrised FF. Similar results have been observed in the other domains. Our intuition is that the proposed approach can help FF identifying a satisficing solution for the problem at hand, but it does not modify the solution that would have been identifying by default FF.

V. Discussion

One may wonder about the most important configurable aspects of the proposed heuristic adaptation. This is of course related to the specific planning engine used, and the search technique. However, their analysis can still shed some light on their relevance, and help to deploy this technique in different planning engines.

In order to understand the influence of the considered configurable aspects on the impact of the heuristic adaptation of the FF performance, we used the fAnova tool [36] to assess parameters' importance. As a general observation, in domains where the use of the configured adaptation lead to significant performance improvement, the penalisation cost type associated to parameter `C` and the maximum value of penalisation associated to parameter `B` tend to play a very significant role. According to the performed analysis, their value can contribute to between 5% and 10% of the observed performance improvement. By analysing the importance of parameters for generating the general configuration, we instead notice that the inconsistency discount factor of 2 associated to parameter `r` is the single most important, and it can contribute to up to 8% of the performance improvement. The other parameters, when considered singularly, show a limited impact on the overall performance. However, the fAnova tool also highlighted that the interaction between parameter `r` and the others can lead to significant synergies. It has also been observed that the pairwise interaction of the parameter `B` with the other parameters can play a significant role. Summarising, most of the considered parameters play an important role improving the performance of the planning engine, and the complex interactions between different parameters are important to boost the performance. However, the actual "best" value of parameters depends on the domain, even though there seems to be a general value that allows to improve the performance on all the domains at once.

Quite interestingly, we can observe that different approaches adopted in order to improve planners performances have a strong relation with our techniques. In particular, the use of preferred actions/operators [21] are a key ingredient of different planning systems which use the actions in the relaxed plan to effectively reduce the branching factor. Preferred actions are indirectly used in our approach, since an evaluation of a preferred action is inconsistent only if it is determined by a penalisation associated to Equation (2). In addition, similarly to our approach, partial delete relaxation and in particular delete-relaxation with conjunctions techniques [23]–[25] aim to refine the delete relaxation heuristic by analysing when it makes mistakes in an online fashion. Moreover, in the case of learning conjunctions from the flaws in the relaxed plan, they have the guarantee that the heuristic value will eventually converge to a real plan and we plan to further elaborate on the connections with our approach.

VI. Conclusion

Heuristics play a pivotal role in AI planning. However, particularly domain-independent ones, can be easily trapped in local minima or plateaus. One way of dealing with such an issue is to allow heuristic to adapt to the characteristics of the search space. In this paper we proposed a heuristic adaptation approach that can deal with inconsistencies in heuristic evaluations. The underlying idea is to identify inconsistencies and penalise affected actions, in order to modify the way in which the search space is explored and potentially avoid plateaus and local minima. Specifically, we investigated how to dynamically adapt the evaluation of the h_{ff} heuristic using best-first search.

The analysis we performed in this work: (i) demonstrates that the proposed heuristic adaptation approach can be configured to deliver a statistically significant improvement on planner's performance; (ii) it is possible to exploit a single configuration of the approach that improves the performance on a range of domains; (iii) gives insights into the parameters (and therefore the elements) that have the strongest impact. It is also worth pointing out that the use of the configurable heuristic adaptation seems to be beneficial on complex instances, while its impact is limited on instances that are quickly solved, i.e. require less than 50 CPU-time seconds, by the default settings.

Heuristic adaptation opens up many possibilities for future work. Firstly, we plan to evaluate its performance when used with different heuristics and search strategies. We are interested in checking if it is possible to learn models that map characteristics of a planning problem (for instance, identified by features [37]) with a parameters' configuration of the heuristic adaptation approach, to obtain a per-instance tuning of performance. We further plan to check whether different configurations of the heuristic adaptation can be combined into a domain-independent portfolio, to improve the robustness of planning engines. In addition, our techniques could be used in multi-agent planning [38], [39] and in the context of plan adaptation [40] in order to improve the performance of the adaptation system or the adaptation phase of a Case-Based Planning system [41]–[43]. Finally, we believe this work could foster the exploitation of Reinforcement Learning techniques [44] in the context of classical planning.

REFERENCES

[1] M. Fox, D. Long, G. Tamboise, and R. Isangulov, "Creating and executing a well construction/operation plan," 2018, uS Patent App. 15/541,381.

[2] T. L. McCluskey and M. Vallati, "Embedding automated planning within urban traffic management operations," in *Proceedings of the International Conference on Automated Planning and Scheduling*, 2017.

[3] S. Thiébaux, C. Coffrin, H. Hijazi, and J. Slaney, "Planning with mip for supply restoration in power distribution systems," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2013.

[4] N. Lipovetzky, C. N. Burt, A. R. Pearce, and P. J. Stuckey, "Planning for mining operations with time and resource constraints," in *Proceedings of the International Conference on Automated Planning and Scheduling*, 2014.

[5] T. Imai and A. Kishimoto, "A novel technique for avoiding plateaus of greedy best-first search in satisficing planning," in *AAAI*, 2011.

[6] R. Valenzano and F. Xie, "On the completeness of best-first search variants that use random exploration," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 784–790.

[7] A. Gerevini, A. Saetti, and I. Serina, "An approach to efficient planning with numerical fluents and multi-criteria plan quality," *Artif. Intell.*, vol. 172, no. 8-9, pp. 899–944, 2008.

[8] ——, "An empirical analysis of some heuristic features for planning through local search and action graphs," *Fundam. Inform.*, vol. 107, no. 2-3, pp. 167–197, 2011.

[9] F. Xie, M. Müller, R. C. Holte, and T. Imai, "Type-based exploration with multiple search queues for satisficing planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2014.

[10] A. Coles and A. Smith, "Marvin: A heuristic search planner with online macro-action learning," *CoRR*, vol. abs/1110.2736, 2011. [Online]. Available: http://arxiv.org/abs/1110.2736

[11] F. Xie, M. Müller, and R. C. Holte, "Adding local exploration to greedy best-first search in satisficing planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2014.

[12] M. Asai and A. Fukunaga, "Exploration among and within plateaus in greedy best-first search," in *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS*, 2017, pp. 11–19.

[13] ——, "Tie-breaking strategies for cost-optimal best first search," *J. Artif. Intell. Res.*, vol. 58, pp. 67–121, 2017.

[14] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proceedings of the 5th Learning and Intelligent OptimizatioN Conference (LION)*, 2011, pp. 507–523.

[15] T. Bylander, "The computational complexity of propositional STRIPS planning," *Artificial Intelligence*, vol. 69, pp. 165–204, 1994.

[16] B. Bonet, L. G., and H. Geffner, "A robust and fast action selection mechanism for planning," in *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI)*, 1997, pp. 714–719.

[17] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research (JAIR)*, vol. 14, pp. 253–302, 2001.

[18] S. Richter, M. Helmert, and M. Westphal, "Landmarks revisited," in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI*, 2008, pp. 975–982.

[19] M. Helmert, "The fast downward planning system," *J. Artif. Intell. Res.*, vol. 26, pp. 191–246, 2006.

[20] R. Liang, "A survey of heuristics for domain-independent planning," *JSW*, vol. 7, no. 9, pp. 2099–2106, 2012.

[21] J. Hoffmann, "Local search topology in planning benchmarks: An empirical analysis," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, 2001, pp. 453–458.

[22] ——, "Where ignoring delete lists works, part II: causal graphs," in *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS*, 2011.

[23] E. R. Keyder, J. Hoffmann, and P. Haslum, "Improving delete relaxation heuristics through explicitly represented conjunctions," *J. Artif. Intell. Res.*, vol. 50, pp. 487–533, 2014.

[24] M. Fickert, J. Hoffmann, and M. Steinmetz, "Combining the delete relaxation with critical-path heuristics: A direct characterization," *J. Artif. Intell. Res.*, vol. 56, pp. 269–327, 2016.

[25] M. Fickert, "Making hill-climbing great again through online relaxation refinement and novelty pruning," in *Proceedings of the Eleventh International Symposium on Combinatorial Search, SOCS*, 2018, pp. 158–162.

[26] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "Paramils: An automatic algorithm configuration framework," *J. Artif. Intell. Res. (JAIR)*, vol. 36, pp. 267–306, 2009.

[27] C. Ansótegui, M. Sellmann, and K. Tierney, "A gender-based genetic algorithm for the automatic configuration of algorithms," in *Proceedings of the Principles and Practice of Constraint Programming (CP)*, 2009, pp. 142–157.

[28] Z. Yuan, T. Stützle, and M. Birattari, "Mads/f-race: Mesh adaptive direct search meets f-race," in *Proceedings of the 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE)*, 2010, pp. 41–50.

[29] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artificial Intelligence*, vol. 206, pp. 79–111, 2014.

[30] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[31] M. Vallati and I. Serina, "A general approach for configuring PDDL problem models," in *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS*, 2018, pp. 431–436.

[32] J. Seipp, S. Sievers, M. Helmert, and F. Hutter, "Automatic configuration of sequential planning portfolios," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015, pp. 3364–3370.

[33] M. Vallati, F. Hutter, L. Chrpa, and T. L. McCluskey, "On the effective configuration of planning domain models," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, 2015, pp. 1704–1711.

[34] M. Vallati, L. Chrpa, and T. L. McCluskey, "What you always wanted to know about the deterministic part of the international planning competition (IPC) 2014 (but were too afraid to ask)," *Knowledge Eng. Review*, vol. 33, p. e3, 2018.

[35] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[36] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," in *Proceedings of The 31st International Conference on Machine Learning*, 2014, pp. 754–762.
[37] C. Fawcett, M. Vallati, F. Hutter, J. Hoffmann, H. H. Hoos, and K. Leyton-Brown, "Improved features for runtime prediction of domain-independent planners," in *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS*, 2014.
[38] A. Bonisoli, A. Gerevini, A. Saetti, and I. Serina, "A privacy-preserving model for the multi-agent propositional planning problem," *Frontiers in Artificial Intelligence and Applications*, vol. 263, pp. 973–974, 2014.
[39] A. E. Gerevini, N. Lipovetzky, F. Percassi, A. Saetti, and I. Serina, "Best-first width search for multi agent privacy-preserving planning," in *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*, J. Benton, N. Lipovetzky, E. Onaindia, D. E. Smith, and S. Srivastava, Eds. AAAI Press, 2019, pp. 163–171.
[40] M. Fox, A. Gerevini, D. Long, and I. Serina, "Plan stability: Replanning versus plan repair," in *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6-10, 2006*, D. Long, S. F. Smith, D. Borrajo, and L. McCluskey, Eds. AAAI, 2006, pp. 212–221.
[41] D. Borrajo, A. Roubícková, and I. Serina, "Progress in case-based planning," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 35:1–35:39, 2014.
[42] A. Gerevini, A. Saetti, and I. Serina, "Case-based planning for problems with real-valued fluents: Kernel functions for effective plan retrieval," *Frontiers in Artificial Intelligence and Applications*, vol. 242, pp. 348–353, 2012.
[43] I. Serina, "Kernel functions for case-based planning," *Artificial Intelligence*, vol. 174(16-17), pp. 1369–1406, 2010.
[44] M. Sewak, *Deep Reinforcement Learning: Frontiers of Artificial Intelligence*. Springer Singapore, 2019.