Improving Domain-Independent Heuristic State-Space Planning via Plan Cost Predictions

Francesco Percassi[a,b] and Alfonso E. Gerevini[a] and Enrico Scala[a] and Ivan Serina[a] and Mauro Vallati[b]

[a]Dipartimento d'Ingegneria dell'Informazione , Universitá degli Studi di Brescia
Brescia, Italy
[b]Department of Informatics, University of Huddersfield, Huddersfield, UK

ARTICLE HISTORY
Compiled June 21, 2021

ABSTRACT
Automated planning is a prominent Artificial Intelligence (AI) challenge that has been extensively studied for decades, which has led to the development of powerful domain-independent planning systems. The performance of domain-independent planning systems are strongly affected by the structure of the search space, that is dependent on the application domain and on its encoding.

This paper proposes and investigates a novel way of combining machine learning and heuristic search to improve domain-independent planning. On the learning side, we use learning to predict the plan cost of a good solution for a given instance. On the planning side, we propose a bound-sensitive heuristic function that exploits such a prediction in a state-space planner. Our function combines the input prediction (derived inductively) with some pieces of information gathered during search (derived deductively). As the prediction can sometimes be grossly inaccurate, the function also provides means to recognise when the provided information is actually misguiding the search. Our experimental analysis demonstrates the usefulness of the proposed approach in a standard heuristic best-first search schema.

KEYWORDS
Automated Planning; Plan Cost Prediction; Heuristic Search; Classical Planning.

1. Introduction

Automated planning is one of the most prominent AI challenges; it has been studied extensively for several decades and has led to a large number of real-world applications. In recent years, there has been considerable progress in developing powerful domain-independent planners, in no small part spurred on by the International Planning Competitions.

The performance of current planning systems are affected by the structure of the search space, which depends on the application domain and its encoding. In many cases, the planning performance can be improved by deriving and exploiting knowledge about the structure of the problem and its potential solutions that is not explicitly given in the input formalisation, and that can be used for optimising the planner performance. Well-known examples include portfolio configuration (Gerevini, Saetti, & Vallati, 2014; Howe, Dahlman, Hansen, Von Mayrhauser, & Scheetz, 1999; Roberts, Howe, Wilson, & desJardins, 2008; Seipp, Sievers, Helmert, & Hutter, 2015), reformulation approaches such as macro-operators (Botea, Enzenberger, Müller, & Schaeffer, 2005; Chrpa & Vallati, 2019; Korf, 1985; Scala, 2014; Scala & Torasso, 2015; Vallati, Chrpa, & Serina, 2020), entanglements (Chrpa, Vallati, & McCluskey, 2019), action schema splitting (Areces, Bustos, Dominguez, & Hoffmann, 2014), model configuration (Vallati, Hutter, Chrpa, & McCluskey, 2015; Vallati & Serina, 2018), case-based planning (Bonisoli, Gerevini, Saetti, & Serina, 2015; Borrajo, Roubícková, & Serina, 2014; Serina, 2010), and state constraints (Gerevini & Schubert, 2000). For an extensive overview of the field, the interested reader is referred to the work of Celorrio, de la Rosa, Fernández, Fernández, and Borrajo (2012).

Another kind of knowledge that could be derived by analysing a planning problem is the expected (predicted) cost of its solutions, and in particular of an optimal or good quality solution. Taking inspiration from the significant amount of work devoted at handling solution quality bounds (see, for instance, (Stern et al., 2014; Thayer & Ruml, 2011)), the question that we address in this paper is the following: *How can we exploit a prediction of the plan cost of a (good) solution to improve search performance?* Predictions on the plan costs can come from different sources: they can be made by human experts, or computed automatically through, e.g., inductive methods based on machine learning approaches. The use of such a prediction during the planning search poses a number of interesting challenges about how to fruitfully exploit it to improve planning performance. This is because predicted values can sometimes be grossly inaccurate and either under- or over-estimating the cost of the optimal plan, or of the best-quality plan that can be found by the considered planning approach within a certain time limit. Due to this, such predictions can not be straightforwardly used as bounds, but there is a need for appropriately designed approaches.

To address the above question, and building on our preliminary results (Percassi, Gerevini, Scala, Serina, & Vallati, 2020), in this paper we first introduce a domain-independent approach for predicting the plan cost of a "good" solution of a planning instance; then, to exploit the knowledge provided by these predictions, we introduce a set of best-first search that can choose the nodes to visit considering the predicted plan cost combined with pieces of information collected during search (heuristic costs to reach the goal, current cost from the root, number of expanded nodes). In particular, we define and introduce *unidirectional* and *bidirectional* heuristics function. The former is guiding the search towards the provided prediction, aiming at not exceeding it. The latter approach works on both sides of the predicted plan cost value, affecting the search space exploration accordingly. The heuristics can mitigate the impact of the prediction when the search recognises that such a prediction is actually misguiding the search because too inaccurate.

To evaluate the proposed approaches we perform an extensive experimental analysis using well-known benchmarks from the International Planning Competitions. This analysis shows that our techniques for exploiting the learned knowledge can be highly beneficial in the context of a best-first search approach to domain-independent planning.

The rest of this paper is organised as follows. Section 2 provides the necessary background and describes existing planning features. Section 3 introduces our domain-independent approach to predicting plan cost of good quality solutions. Section 4 describes the designed best-first search. Section 5 provides the results of our experimental analysis. Section 6 discusses related work. Finally, Section 7 gives the conclusions.

2. Essential Background

This section is devoted to introduce the required background in terms of representation, heuristic search, and predictive models for classical planning. Classical planning is concerned with finding a (partially or totally ordered) sequence of actions transforming the static, deterministic and fully observable environment from the given initial state to a desired goal state (Ghallab, Nau, & Traverso, 2004).

2.1. Representation

We focus on (propositional) classical planning, also called STRIPS (Fikes & Nilsson, 1971) with cost. A STRIPS planning task is defined by the tuple $\Pi = \langle F, A, I, G \rangle$ where:

- F is a finite set of propositional facts;
- A is a finite set of actions; each $a \in A$ is a triple $\langle \mathit{Pre}(a), \mathit{Eff}^-(a), \mathit{Eff}^+(a) \rangle$ where $\mathit{Pre}(a) \subseteq F$ is a set of preconditions, $\mathit{Eff}^-(a) \subseteq F$ is a set of delete effects and $\mathit{Eff}^+(a)$ is a set of positive effects; each action $a \in A$ is associated to a non-negative cost, i.e $\mathit{cost}(a) \in \mathbb{N}$;
- $I \subseteq F$ is the initial state;
- $G \subseteq F$ is the goal specification.

Facts in F can be interpreted as the Boolean state variables of our problem. A state s in STRIPS is a subset of F; any fact $f \in F$ that belongs to s is true in s, or, equivalently, it is assigned the value of $\top$; by closed-world assumption, if a fact $f \in F$ does not belong to s, f is said to be false in s, or equivalently f is assigned to value of $\bot$. An action a is said to be applicable in a state s iff $\mathit{Pre}(a) \subseteq s$. The application of a in s generates a new state $s[a] = (s \setminus \mathit{Eff}^-(a)) \cup \mathit{Eff}^+(a)$.

A plan π is a sequence of actions $\langle a_0, a_1, \cdots, a_{n-1} \rangle$. Let $\langle s_0 = I, s_1, \cdots, s_n \rangle$ be the sequence of states generated by iteratively applying actions from I, i.e., for all $0 < i \leq n$, $s_i = s_{i-1}[a_{i-1}]$, a plan is said to be applicable in I iff all actions are executable across the generated sequence of states, i.e. $\mathit{Pre}(a_0) \subseteq I$ and for all $0 \leq i < n$, $\mathit{Pre}(a_i) \subseteq s_i$. A plan is said to be a solution plan for a given planning problem iff it is applicable in I and the generated sequence of states ends in a state s_n such that $G \subseteq s_n$. The cost of a plan π is the sum of the cost of each action in π, i.e., $\sum_{a \in \pi} \mathit{cost}(a)$. A plan is said to be optimal if there is no solution with inferior plan cost.

2.2. Heuristic Search

A well established approach to solving planning tasks is forward heuristic search over a transition system induced by the grounding of Π.[1] That is, start from the initial state, and navigate the state space by the domain actions applied using a best-first search policy until a goal state is found. Best-first search usually organises the exploration by expanding from a frontier the node n that minimises a given function $f(n)$. Let $g(n)$ be the length or the plan cost of the prefix leading to node n, $h(n)$ a heuristic function predicting the remaining cost to get to the goal, and w a constant weight. With $f(n) = g(n) + w \cdot h(n)$ the best-first search becomes the well known weighted A*

[1]The grounding of a planning task is a transformation of all operators in a set of action instances, one for each possible instantiate of its parameters. Powerful techniques to do this automatically and in a way to focus only on actually reachable actions is described by Helmert (2003).

algorithm (Pohl, 1970). A critical aspect in such a setting is to devise a heuristic that well balances the precision of the $h(n)$ estimating the actual distance to the goal (h^*) and its computational cost. A popular way to devise a suitable heuristic function for a STRIPS problem is by exploiting its delete-free relaxation. The delete-free relaxation of a STRIPS Problem Π is a new STRIPS problem Π^+ which is identical to Π except that all the actions have empty negative effects. This relaxation has been proved effective to devise heuristics with a good trade-off between precision and computational cost. Several delete-free relaxation heuristics have been defined in the literature (e.g., (Bonet & Geffner, 2001; Domshlak, Hoffmann, & Katz, 2015; Hoffmann & Nebel, 2001)), but for the scope of this work it is only important to discern when such heuristics are admissible and when they are not. A heuristic is said to be *admissible* if it does not overestimate the actual solution cost, *inadmissible* otherwise. Admissible heuristics with $w = 1$ can be used to make A* always produce an optimal solution; inadmissible heuristics trade this admissibility for more information of the relaxed problem, and some of them are quite convenient approximations from a practical standpoint, especially when optimality is not a major concern. In this work, we will use three heuristics, all exploiting the delete-free relaxation principle: $h^{\max}$ (Bonet & Geffner, 2001), h^{FF} (Hoffmann & Nebel, 2001) and h^{LM} (Richter & Westphal, 2010). $h^{\max}$ is admissible, while both h^{FF} and h^{LM} only provide inadmissible estimates.

2.3. Predictive Models

Predictive models in classical planning have been traditionally designed to predict the performance of a given planning engine on a previously unseen planning task. Predictions are possible by exploiting Empirical Performance Models (EPMs) which are built by: (i) observing performance of solvers on a large set of training tasks; (ii) extracting task-specific features from each training problem; and (iii) learning a predictive model that maps features' value with observed performance (Hutter, Xu, Hoos, & Leyton-Brown, 2014).

Each feature is either a number or a categorical value that represents a property of the domain or problem model (e.g., the number of objects). In a sense, a feature summarises a potentially important property of the considered task. The whole set of features can be seen as the "fingerprint" of the planning instance at hand.

In the past decade, there has been a significant amount of work dedicated to identify large and informative set of features for classical planning. Given the good results achieved by Fawcett et al. (2014) in predicting planners' performance, in this work we decided to consider the set of features they exploited. They considered a large set of features, that included features proposed by Roberts et al. (2008) and Cenamor, de la Rosa, and Fernández (2012, 2013). The final set contains 311 instance features, classified into seven groups. Below, we provide a short description for each class, pointing out the type and number of features.

PDDL. By considering the PDDL domain and problem files, 49 features are extracted. These features include information about the use and number of object types, the language requirements, the number of operators, etc.

Fast Downward. This class includes 87 features that are extracted by exploiting the translation and preprocessing tools of Fast Downward (Helmert, 2006). The features are gathered by analysing the translation process (such as the number of removed operators and propositions and number of implied preconditions added), the finite domain representation created (such as the number of variables, number of mutex groups, and statistics over operator effects), the output of the preprocessing process (such as the percentage of variables, mutex groups and operators deemed necessary); a number of additional features about causal and domain-transition graphs are extracted by analysing the finite domain representation generated by Fast Downward.

LAMA probing. The LAMA (version 2011) planner (Richter & Westphal, 2010) is run for 1 CPU second in order to extract some features from the resulting planning trajectory, such as the number of reasonable orders removed, landmark discovery and composition, and the number of graph edges. In total, 16 features belong to this class.

LPG preprocessing. 6 features are extracted by running the pre-processing phase of LPG (Gerevini, Saetti, & Serina, 2003, 2008, 2011a). This includes features such as the number of facts, the number of "significant" instantiated operators and facts, and the number of mutual exclusions between facts.

Torchlight. Torchlight (Hoffmann, 2011) is a tool for analysing local search topology under h^+. From its analysis it is possible to extract 10 features by considering success (sample state proved not to be a local minimum) and dead-end percentages, and statistics over exit distance bounds and preprocessing results.

SAT representation. A planning task can be translated into a propositional logic formula in CNF (Conjunctive Normal Form), to support the use of SAT solvers. The Madagascar-p (Rintanen, 2012) system is used to perform such a translation into a CNF formula with a planning horizon of 10 time steps. From the CNF formula, up to 115 features can be extracted leveraging the SATZilla framework (Xu, Hutter, Hoos, & Leyton-Brown, 2008). The interested reader is referred to (Hutter, Xu, Hoos, & Leyton-Brown, 2014) for a detailed description of these features.

Success and timing. For each of the aforementioned six extraction procedures, the CPU time required for extraction is recorded, as well as the success (or failure) of the process and of the involved subcomponents. In total, 28 features belong to this class.

3. Predicting Plan Cost

This section is devoted to describe the approach that we use to predict the cost of a plan solving a given planning task. Our aim is to provide a domain-independent predictor, i.e., a predictor that can be used for any (unseen) planning task that is represented using the PDDL language.

3.1. Training Instances

Designing a model that predicts the cost of a solution plan raises the question of what a good cost to be predicted is. We decided to train the predictive model to predict the cost of the best known solution of a training planning instance. For each planning instance, we considered the best known quality (lowest cost) of a plan, when available, by looking at the data stored in "Planning.Domains".[2] Otherwise, we considered the best solution generated by a set of selected state-of-the-art planners using 1800 CPU-time seconds for each run. Such planners are: LAMA (Richter & Westphal, 2010), Fast

[2]This can be downloaded from the `http://planning.domains/` website.

Table 1.: Average CPU-time (and standard deviation) needed to extract features from the considered classes, on the training instances.

Class	**Avg. CPU-time**	**Std. Dev**
PDDL	0.2	1.0
Fast Downward	10.0	30.1
LAMA probing	1.5	0.5
LPG preprocessing	1.2	4.7
Torchlight	1.3	17.0
SAT representation	286.8	623.6

Downward (Helmert, 2006), LPG (Gerevini, Saetti, & Serina, 2006, 2011b), Madagascar (Rintanen, 2012), FF (Hoffmann & Nebel, 2001), Arvand (Nakhost, Müller, Valenzano, & Xie, 2011), Probe (Lipovetzky & Geffner, 2011). They were chosen because they exploit very different planning techniques, and can therefore potentially provide solution plans of significantly different quality – increasing the possibility of obtaining good quality plans on the considered instances.

The data used to train the predictive model come from a large set of planning domains. Indeed, we collected as many PDDL classical planning instances as possible with the only restriction being that they were supported (but not necessarily solved) by at least one of the planning engines used for training purposes. We included instances from the following sources: (i) International Planning Competition (IPC)-98 and IPC-00; (ii) IPC deterministic tracks from 2002 to 2011; (iii) learning tracks of IPC-08 and IPC-11; (iv) FF benchmark library; (v) Fast Downward benchmark library; (vi) UCPOP Strict benchmarks; and (vii) Sodor and Stek domains used by Roberts et al. (2008). From the above-mentioned benchmark sets, we removed the instances of domains used in the testing set, that includes the IPC-14 and IPC-18 deterministic track benchmarks. In total, more than $7,000$ planning instances were used to derive the training data set.

3.2. Planning Features

Table 1 shows the average CPU time and standard deviation that was needed to extract the features from the corresponding class on the training instances. For instance, all the features from the PDDL class are extracted usually in 0.2 CPU-time seconds. It is easy to observe that the SAT features require a significant amount of CPU time (as well as memory resources, not shown in the table) to be extracted. Further, in a large number of cases the extraction process failed for this class of features. Therefore, we decided to remove this class of features, and the corresponding features from the *Success and timing* set, from the rest of our analysis. Thus, the final set we consider consists of 182 features that can be extracted on average in a matter of few CPU-time seconds.

In order to understand the informativeness of features, and to highlight relevant features for the task of predicting the cost of a solution plan, we used the RELIEF algorithm (Robnik-Sikonja & Kononenko, 1997) provided as part of the well-known WEKA tool (Hall et al., 2009). RELIEF uses a filter-based method approach to rank features according to their informativeness with regards to the predictive task considered. The approach highlighted that features from the PDDL and the *Fast Downward* sets are among the most informative. In particular, for the PDDL set, the number of goals and the number of objects are deemed to be very relevant. For the *Fast Down-*

```
best classifier: weka.classifiers.meta.AdditiveRegression
arguments: [-S, 1, -I, 6, -W,
    weka.classifiers.trees.RandomForest, --, -I, 22, -K, 0, -depth, 11]
metric: rootMeanSquaredError
```

Figure 1.: The classifier configuration identified by Auto-WEKA for predicting the quality of plans, on the considered training instances.

ward set, the maximum and mean SAS+ variable domain size, and the total mutex groups size play an important role, according to the RELIEF analysis.

3.3. Building the Predictor

For generating the predictive model, we used the WEKA tool. We used the Auto-WEKA tool (Kotthoff, Thornton, Hoos, Hutter, & Leyton-Brown, 2017) for selecting the best technique and the best configuration of the corresponding parameters on the training data. Auto-WEKA is an approach that, given a training set and a predictive task, explores all the classifiers provided by WEKA, and their parametrisation, in order to identify the configuration that leads to the best performance. Auto-WEKA was run for 2 CPU-time days in a 10-fold cross-validation on all the considered training instances, with the goal of identifying the best predictor that minimises the root mean squared error. The resulting predictive model that we obtained uses additive regression models based on random forests, and showed better performance (on the training instances) than those of the manually assessed predictive models. We therefore decided to use it for the rest of our analysis. The specific configuration identified by Auto-WEKA is shown in Figure 1, and can be straightforwardly exploited in WEKA.

The resulting predictor, assessed on the training instances using 10-fold cross validation, had a correlation coefficient of 0.921, a mean absolute error of 20.5, and a root mean squared error of 167.8. The results on the testing set, that is used also to assess the usefulness of the proposed adaptive heuristics, will be discussed in Section 5.

Regarding the CPU time needed by the identified model for generating a prediction, we observe that it is negligible. In our experiments it has never taken more than 0.1 CPU-time seconds to provide a prediction, given the extracted features.

4. Exploiting Potentially Inaccurate Predictions

Consider a planning task Π for which we know, thanks for instance to the predictor previously described, an approximation of the cost (or the length) of a solution plan solving Π. The question is: how can we make the search for a plan in a planning engine capable of exploiting such an estimate? At same time though, how can we take into account that such an estimate can potentially mislead the search for a solution?

We approach these questions through the lens of heuristic forward search, and study ways to make a A* search sensitive to an estimate B approximating the cost of a plan.

Similarly to the standard functioning of A*, we collect the cost $g(n)$ to the node n as the cost of the plan to go from the search tree root to n. Then we collect two heuristic estimates of the cost from n to the goal, namely $h_\delta(n)$ and $h_s(n)$, respectively. Both $h_\delta(n)$ and $h_s(n)$ are functions that return an estimate of the actual cost, but do so with different purposes.

- $h_\delta(n)$ is used for assessing whether the prefix under exploration has trespassed the estimate B given as an input. We call this heuristic the *anchor estimate*.
- $h_s(n)$ is used to effectively guide the search. We call this heuristic the *guiding estimate*.

As we will see in the following, $h_\delta(n)$ is used (in combination with $g(n)$) as a means to approximate when the plan prefix will likely exceed B, and for this reason, we use an estimate that computes a lower bound of the the cost to reach a goal for a given state, i.e., we adopt an admissible heuristic. Instead, for $h_s(n)$, we admit an estimate that is not guaranteed to be admissible, but potentially more informed, i.e., we adopt an inadmissible heuristic.

We use these elements in a weighted A* setting (Pohl, 1970), and, much as it happens in any best-first search, organise the search for a solution by expanding the node n in the frontier that minimises the cost function $f(n) = g(n) + w \cdot h^{Bound}(n)$. As a difference w.r.t. traditional use of heuristics in search, our function $h^{Bound}(n)$ is not a proper heuristic function in that it is not aimed at predicting the distance to the goal, but it is devoted at *steering the search in a way that is dependent on the input bound B whilst taking into account the distance to the goal.* We call this function the *adaptive heuristic.* This section discusses four variants of the aforementioned adaptive heuristic: two of these act on the guiding estimate in an additive way; the third and the fourth variants act by modifying the guiding estimate in a more profound manner. These can indeed act both by amplifying or decreasing the contribution of the guiding estimate. Because of this, we classify our variants into unidirectional and bidirectional adaptive heuristics.

4.1. Unidirectional $h^{Bound}(n)$

The main idea at the basis of the unidirectional $h^{Bound}(n)$ variants is that of modifying the value of the guiding heuristic by adding a positive amount when the search seems to go towards a direction that does not satisfy the input bound. Both variants make use of the anchor estimate $h_\delta(n)$ to weigh this modification differently. This is summarised in what we call $\Delta(n)$.

More precisely, let n be a node in the search, $\Delta(n)$ measures the discrepancy between the evaluation of the search node n against the input bound B, i.e., the difference between B and the sum $g(n) + h_\delta(n)$ of the cost $g(n)$ accumulated so far, and the expected cost $h_\delta(n)$ to the goal.

$$\Delta(n) = B - (g(n) + h_\delta(n)) \tag{1}$$

The first adaptive heuristic that we present is called $h_B^{penalty}(n)$, it is formulated as follows:

$$h_B^{penalty}(n) = h_s(n) + \begin{cases} |\Delta(n)| & \text{if } B < (g(n) + h_\delta(n)) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Heuristic $h_B^{penalty}(n)$ aims at favouring exploration of all those search nodes that can lead to a solution which is better than the given bound. Intuitively, this heuristic is assuming that the provided bound is an overestimation of the cost of a good quality solution plan. To obtain the described behaviour, for whatever is beyond the expected bound, $h_B^{penalty}(n)$ sums the $\Delta(n)$ value to the heuristic. As this search will discourage the explorations of nodes beyond a given threshold, we expect that this will steer the search on shallower solutions. Note that, this measures will not endanger the quality of the plan if the estimate is not good enough. At worst, it will do more search if the predicted bound is too small.

The second adaptive heuristic that we propose is called $h_B^{delta}(n)$:

$$h_B^{delta}(n) = h_s(n) + |\Delta(n)| \times \max\left(1 - \frac{g(n)}{B}, 0\right) \tag{3}$$

The main objective of h_B^{delta} is to focus the search towards those nodes whose distance to the initial state is close to the bound. It does so by discouraging with a higher penalty those nodes with lower g-values. Once the distance from the initial state saturates the bound, the discrepancy is basically de-activated. This policy is more reactive than $h_B^{penalty}(n)$ in the sense that the value of the heuristic can be modified for a larger number of situations. This may result in a more profound impact on the performance of the system. If the bound is a large overestimation of the actual solution quality, then it is likely that more expensive solutions are returned.

4.2. Bidirectional $h_B^{discount}(n)$

Our third adaptive heuristic proceeds along another dimension since it considers a multiplicative in place of an additive factor, and it has effect on both sides of the bound. As the other adaptive heuristics, this variant also keeps a handle on the anchor estimate, but uses it to discount the weight of the heuristic in ranking nodes in the frontier. This adaptive heuristic comes in a basic and extended version; the extended version makes use of a smoothing parameter. This parameter aims at attenuating the role of the adaptive heuristic when the performed exploration of the search space seems to indicate that the input bound is too tight.

More formally, the new adaptive heuristic is defined as follows:

$$h_B^{discount}(n) = h_s(n) \cdot \left(\frac{B}{g(n) + h_\delta(n)}\right) \tag{4}$$

The work done by $h_B^{discount}(n)$ is more disruptive than the work of the other proposed heuristics, because it can also nullify the heuristic value. From an abstract standpoint, it accelerates the search towards the nodes that are close to B (the provided predicted cost), and from that point on-wards, it promotes an exploration that is more sensible to the g-values (the heuristic contribution is diminished in the minimisation). In other words, $h_B^{discount}(n)$ *discounts the cost of all plan prefixes of cost less than the input bound* B. This is obtained by amplifying the contribution of the heuristic for all those search nodes n with $g(n) + h_\delta(n) < B$, and diminishing the heuristic value of those nodes n with $g(n) + h_\delta(n) > B$. Note that, since $h_\delta(n)$ is an admissible function, $g(n) + h_\delta(n)$ plays the role of an optimistic predictor for the cost of the solution plan that the search would obtain through the candidate node n.

It is easy to see that the overall behaviour of the search heavily depends on how bound B relates to the solutions space of the problem. When B is much lower than the optimal cost, the second factor in (4) gets very small, and therefore we expect the search to expand more nodes; most of the search will be indeed done favouring nodes with lower g-values. That can have a very negative impact on runtime performance. Instead, when B is too large (much larger than the optimal cost), the second factor of (4) makes the search greedier, because it favours the indication provided by our guiding heuristic. In general this can reduce the number of expanded nodes, but it can also compromise the quality of the obtained solution. Nevertheless, we experimentally observed that having B over-estimating the optimal cost does not worsen solution quality significantly, especially when B is only slightly larger than the cost of an optimal plan.

To overcome the issue related to using a too tight bound B, we devise a further exponential modifier in the definition of $h_B^{discount}(n)$, namely p_{rate}. The aim of this modifier is to alleviate the effect of the bound on the heuristic by taking advantage of the information acquired whilst searching. More precisely, p_{rate} is defined as the fraction of expanded nodes that have a value for $g(n) + h_\delta(n)$, where h_δ is admissible, that exceeds bound B:

$$p_{rate} = \frac{\#\{n \in \{expanded\ nodes\ \} \mid\ g(n) + h_\delta(n) > B\}}{\#\{expanded\ nodes\}}$$

This way, the more expanded nodes exceeded B, the closer the second factor of (4) gets to one. This leads to the following variant:

$$h_B^{discount\text{-}pr}(n) = h_s(n) \cdot \left(\frac{B}{g(n) + h_\delta(n)}\right)^{(1-p_{rate})} \tag{5}$$

We observe that when p_{rate} increases, the exponent of the second factor tends to zero, and the second factor itself becomes irrelevant (i.e., close to 1).

The described adaptive heuristic functions exploiting B can inform the search in a way that is in contrast with the heuristic evaluation of the original function h_s. For instance, consider $B = 7$, $w = 1$ and suppose that there are two nodes on the search frontier $n1$ and $n2$ such that $h_s(n1) = 5$, $h_\delta(n1) = 3$, $g(n1) = 1$, $h_s(n2) = 9$, $h_\delta(n2) = 7$, and $g(n2) = 1$. Algorithm wA* using h_s prefers $n1$ to $n2$, while wA* with $h_B^{discount}(n)$ prefers $n2$ because $h_B^{discount}(n)(n1) = 5 \cdot \frac{7}{3+1} = 8.75$ and $h_B^{discount}(n)(n2) = 9 \cdot \frac{7}{7+1} = 7.875$. A situation like this could happen, for instance, in a logistic problem where we are at a location where we can reach the target via two paths; one of such paths appears to be better according to h_s, but it is actually not viable to the end because, along this path and differently from the other longer path, there is no refuel station, and the target location can be reached only if at least one refuel is done before a maximum number of moves. The predicted cost B could alter the h_s-values towards the longer but safer path.

5. Experimental Analysis

The aim of our experimental analysis is to assess the usefulness of the proposed adaptive heuristics in a state-of-the-art classical planning system. In what follows we introduce the experimental setting and the evaluation metrics, and then report on our experimental findings.

```
lazy_wastar([hff],preferred=[hff],w=5)          # Single heuristic baseline
lazy_wastar([hlm],preferred=[hlm],w=5)          # Single heuristic baseline
lazy_wastar([hff,hlm],preferred=[hff,hlm],w=5)  # Pair heuristics baseline
```

Figure 2.: Fast Downward code for the three considered baselines.

5.1. Experimental Settings

We implemented the adaptive heuristic described in the previous section within the Fast Downward (FD) planning system (Helmert, 2006). We focus on the setting of FD where the search is carried on using wA* with $w = 5$, and up to two heuristics are used in alternation. This is an approach commonly exploited within classical planners; it is also exploited as one of the search episodes of the LAMA planning system (Richter & Westphal, 2010).

As our anchor heuristic, we use the admissible heuristic $h_\delta = h^{\max}$. As a guiding heuristic, we use one of the following inadmissible heuristics: $\{h^{\mathrm{FF}}, h^{\mathrm{LM}}\}$ (Hoffmann & Nebel, 2001; Richter & Westphal, 2010). In our experiments we compare the effect of using any combination of the aforementioned inadmissible heuristics by comparing the performance obtained with the same search without the adaption enabled. Figure 2 details the exact configuration of the FD system with the different inadmissible heuristics. For the configuration with more inadmissible heuristics, we run experiments using the adaptive heuristic to either one of them, or both of them. A first set of experiments is devoted to investigate the best set of heuristics to use, and the best heuristic to be modified using the provided prediction B. Then we focus our analysis on the best heuristic and report on a domain by domain evaluation.

Our benchmarks involve 26 domains from the satisficing track of the 2014 and 2018 International Planning Competitions (IPC). For each considered domain, we have 20 instances giving us a total of 520 instances. For each instance of the considered benchmarks (that are different from the problems used to train the predictor), our predictor is provided with features extracted from the problem, and it returns an estimate of a the cost of a valid solution solving such an instance.[3]. This predicted cost plays the role of B in our adaptive heuristics. However, the plan cost prediction made on a given planning instance can at times be grossly wrong, and this can harm the search substantially. To overcome this problem, our system employs a simple preprocessing step that tests whether the predicted value B is lower than the value of the admissible heuristic $h^{\text{LM-Cut}}$ (Helmert & Domshlak, 2009) computed in the initial state. If B is lower, the prediction underestimates even the cost of a lower bound for an optimal solution of the considered problem, and so we do not use any adaptive heuristic (meaning we are in the baseline case). While if B is higher or equal to the value of $h^{\text{LM-Cut}}$, then an adaptive heuristic is used.

All experiments were run on an Intel Xeon Gold 6140M CPUs with 2.30 GHz. For each instance we set a cutoff time of 1800 seconds, and the memory was limited to 8 GB. To account for disturbance and randomness, every run has been repeated five times and the median result has been reported.

[3]The actual predicted costs and the model parameterisation can be found at `https://bitbucket.org/maurovallati/icaps-2020/`

5.2. Evaluation Metrics

Our experimental analysis considers a range of different metrics to assess the impact of the presented approaches. In particular, we consider:

PAR10. The Penalised Average Runtime is a metric usually exploited in algorithm configuration techniques, where average runtime is calculated by assigning to the runs that did not find a plan ten times the cutoff time (Hutter, Hoos, Leyton-Brown, & Stützle, 2009). Intuitively, PAR10 provides a good tradeoff information between runtime and coverage.

score-T. This is the *IPC runtime score*, as defined in the 2014 edition of the International Planning Competition (Vallati, Chrpa, & McCluskey, 2018). It is calculated as follows: for a planner $\mathcal{C}$ and a problem p, $\mathit{Score}(\mathcal{C}, p)$ is 0 if p is unsolved, and $1/(1 + \log_{10}(T_p(\mathcal{C})/T_p^*))$ otherwise (where T_p^* is the minimum time required by the compared systems to solve the problem). The IPC runtime score of a planner is given by the sum of the scores achieved on each considered instance.

score-Q. The *IPC quality score* of a planner $\mathcal{C}$ and a problem p is calculated as: $\mathit{Score}(\mathcal{C}, p)$ is 0 if p is unsolved, and $Q_p^*/Q_p(\mathcal{C})$ otherwise (where Q_p^* is the quality of the best solution found by the compared systems to solve the problem, and $Q_p(\mathcal{C})$ is the quality of the solution found by the $\mathcal{C}$ planner). The IPC quality score of a planner is given by the sum of the scores achieved on each considered instance.

Coverage. This reports the number of problem instances solved within the given cutoff time.

avg-T. The average runtime needed by the planner to solve the benchmark instances. The average is calculated considering instances solved by all the compared approaches.

avg-C. The average plan cost of generated solutions. The average is calculated considering instances solved by all the compared approaches.

The IPC score of a given planner is relative to the performance of the other considered planners; instead, PAR10 is an absolute metric value and it does not take the performance of other competitors into account.

In order to understand the impact of the proposed approaches, in domain-by-domain comparisons, the following metrics are also considered:

- δ. This is an indicative measure of how close/distant the predicted value is with regards to the found solution. It is calculated as $(B - S')/S'$, where S' is the quality of the solution found by the baseline system. This metric indicates the accuracy of the provided predictions, with regards to the solution that the approach would have found. Closer to 0.0, more accurate the predictions are.
- $\frac{\mathbf{exp(X)}}{\mathbf{exp(Y)}}$. The expanded note ratio between a system X and a system Y. This gives an indication of the impact of the considered techniques on the exploration of the search space.

It is hard to define what the *perfect* prediction should be. One may argue that the cost of the optimal plan is a perfect prediction, but it depends on the way in which the prediction is exploited. Our intuition is that the value of a prediction has to be related to the planning approach that has to exploit such prediction: it may be the case that using the optimal plan cost could make finding a solution significantly harder for a given search technique. Also, the value of a prediction cannot be directly related to the cost of the solution that the approach would have found *without* using such prediction:

Table 2.: Performance achieved by each search configuration with or without the use of the estimate. Each search configuration uses wA* with $w = 5$ using the heuristic(s) denoted by their aliases where $h(B)$, with $h \in \{h^{\text{FF}}, h^{\text{LM}}\}$, denotes the adaptive heuristic obtained using an approach between *discount*, *discount-pr*, *delta* and *penalty* in which $h_s = h$. Results are presented in terms of coverage. Bold results denotes the best configuration for each adaptive approach.

Coverage	Search configuration				
	$h^{\text{FF}}(B)$	$h^{\text{LM}}(B)$	$h^{\text{FF}}(B)$,h^{LM}	h^{FF},$h^{\text{LM}}(B)$	$h^{\text{FF}}(B)$,$h^{\text{LM}}(B)$
discount	183	40	**252**	234	187
discount-pr	214	163	**264**	246	223
delta	216	171	**261**	243	247
penalty	179	170	220	**242**	224
Baseline	214	170	234	234	234

this is because the use of a prediction modifies the way in which the search space is explored. Since our aim is to provide a prediction that helps the search approach in findings solutions, we decided to evaluate the quality of a prediction according to how close it is to the cost of a plan that will then be found by the approach that is exploiting such prediction.

5.3. Results

We are now turning our attention to the results of the performed experiments.

5.3.1. Choosing the Guiding Heuristic

As hinted at above, our first set of experiments evaluates the effect of our adaptive heuristics by varying the guiding heuristic. To this end, we run wA* across all possible configurations and compare its performance against the very same search with the adaptation disabled. Table 2 summarises the coverage for each considered configuration. (B) indicates which heuristic is modified through the given adaptive schema. Overall, it is easy to observe that the best performance are achieved when two heuristics are used.

More precisely:

- if –$h^{\text{FF}}(B)$,h^{LM}– is used, then each adaptive approach performs better with respect to the baseline, except for *penalty*;
- if –h^{FF},$h^{\text{LM}}(B)$– is used, then each adaptive approach performs better with respect to the baseline;
- if –$h^{\text{FF}}(B)$,$h^{\text{LM}}(B)$– is used, then each adaptive approach performs worse, with respect to the baseline, except for *delta*;

Given the presented results, we focus our attention on the –$h^{\text{FF}}(B)$,h^{LM}– configuration in what follows. For the sake of conciseness, we simplify our notation:

- *discount-pr* denotes wA*, with $w = 5$ and the pair –$h^{\text{FF}}(B)$,h^{LM}– as heuristics, where $h^{\text{FF}}(B) = h_B^{\textit{discount-pr}}$ is configured in a way that $h_s = h^{\text{FF}}$ and $h_\delta = h^{\max}$;
- *discount* denotes wA*, with $w = 5$ and –$h^{\text{FF}}(B)$,h^{LM}– as heuristics, where $h^{\text{FF}}(B) = h_B^{\textit{discount}}$ is configured in a way that $h_s = h^{\text{FF}}$ and $h_\delta = h^{\max}$;
- *delta* denotes wA* with $w = 5$ and –$h^{\text{FF}}(B)$,h^{LM}– as heuristics, where $h^{\text{FF}}(B) = h_B^{\textit{delta}}$ is configured in a way that $h_s = h^{\text{FF}}$ and $h_\delta = h^{\max}$;

Table 3.: Performance achieved by the Baseline and by the adaptive heuristics on the complete set of considered benchmarks.

Approach	**Coverage**	**avg-C**	**score-Q**	**PAR10**	**score-T**
discount-pr	**264**	388.8	**243.2**	**8948.8**	**232.9**
discount	252	389.6	230.5	9356.0	213.3
delta	261	391.6	236.8	9055.2	225.5
penalty	220	**378.5**	214.8	10452.4	182.8
Baseline	234	381.6	223.9	9970.0	207.5

- *penalty* denotes $w\mathrm{A}^*$ with $w = 5$ and $–h^{\mathrm{FF}}(B)$,$h^{\mathrm{LM}}–$ as heuristics, where $h^{\mathrm{FF}}(B) = h_B^{penalty}$ is configured in a way that $h_s = h^{\mathrm{FF}}$ and $h_\delta = h^{\max}$.

As a baseline, we consider a $w\mathrm{A}^*$ search, with $w = 5$, using $–h^{\mathrm{FF}}$,$h^{\mathrm{LM}}–$ as heuristics in a multi-queue fashion.

5.3.2. Overall

Table 3 summarises the results achieved by our adaptive heuristics and the Baseline. Compared to the baseline, the performance of the search engine improves by exploiting the predicted plan costs. This happens in particular with the two variants of *discount* and with *delta*. Instead, *penalty* reduces the coverage and the speed substantially. This is not surprising though: as we have seen in the previous section, *penalty* gives lower priority to those nodes leading to prefixes of higher predicted cost (i.e., those where $B < (g(n) + h_\delta(n)))$. On the other hand, while *discount* and *delta* cause a slight increase on the average cost of the plans, *penalty* tends to produce plans of higher quality. This is reflected in the average cost of plans (avg-C in Table 3): it is worth noting that the IPC quality score (score-Q in the Table) of *penalty* is the worst among the considered approaches, because such metric takes coverage into account as well.

Both variants of *discount*, and the *delta*, push the algorithm deeper in the search tree, as suggested by the generally higher average plan cost. On the other hand, this allows *discount-pr* to improve coverage and also runtime of about +12%. The negative impact on the average cost seems to remain very limited on average (we have an increase of approximately 0.02%).

5.3.3. penalty versus Baseline

Table 4 shows the performance by domain of Baseline compared to *penalty*. In this table, and in all the subsequent tables where domain-by-domain results are presented, we use P to refer to the system exploiting the given prediction B, and we use $w\mathrm{A}^*$ to indicate the baseline approach. By looking at the results presented in Table 4, it is apparent that the *penalty* does not behave particularly well in terms of coverage and speed, yet, when the predicted cost is rather close to the cost of the found solution, the *penalty* approach tends to produce solutions of good quality. In many cases, it outperforms the baseline in terms of quality of the generated solutions. Conversely, for all the domains where the difference between the predicted plan cost and the actual plan cost is more pronounced, for instance in Caldera and Spider, the quality of the solutions identified by *penalty* is severely affected.

Table 4.: Comparison by domain of the *penalty* performance compared to Baseline. Results are presented in terms of number of instances where the predictions have been used (Used). Domains where predictions are never used are omitted.

Domain	**Used**	**Cov. P**	**Cov. *w*A***	**Solved P**	**Solved *w*A***	δ	$\frac{\exp(P)}{\exp(wA^*)}$	**score-Q P**	**score-Q *w*A***	**avg-C P**	**avg-C *w*A***	**score-T P**	**score-T *w*A***	**avg-T P**	**avg-T *w*A***
Barman	20	**5**	**5**	**0**	**0**	0.15	1.0	**5.0**	**5.0**	**148.6**	**148.6**	5.0	**5.0**	813.9	**799.5**
caldera	20	**19**	15	**4**	0	5.95	1.7	**18.8**	15.0	27.1	**26.5**	**17.9**	14.5	89.9	**38.9**
caldera-split	20	4	**6**	0	**2**	0.69	**0.9**	4.0	**5.8**	**85.5**	91.5	3.9	**5.9**	**5.9**	6.6
CaveDiving	13	**7**	**7**	**0**	**0**	-0.75	1.0	**7.0**	**7.0**	**110.0**	**110.0**	6.9	**7.0**	49.1	**48.4**
Childsnack	20	1	**2**	0	**1**	-0.29	1288.9	1.0	**2.0**	**49.0**	**49.0**	0.3	**2.0**	1304.9	**6.2**
CityCar	20	6	**7**	0	**1**	0.18	**0.4**	6.0	**6.6**	**114.3**	127.2	5.4	**6.5**	**144.5**	272.8
data-network	19	**0**	**0**	**0**	**0**	na	na	**0.0**	**0.0**	na	na	**0.0**	**0.0**	na	na
flashfill	4	**11**	**11**	**0**	**0**	-0.97	1.0	**11.0**	**11.0**	**436.2**	**436.2**	11.0	**11.0**	63.3	**60.2**
Floortile	5	**2**	**2**	**0**	**0**	-0.6	1.0	**2.0**	**2.0**	**76.0**	**76.0**	2.0	**2.0**	3.8	**3.5**
GED	20	**20**	**20**	**0**	**0**	3.05	1.0	**20.0**	**20.0**	**40.0**	**40.0**	19.9	**20.0**	10.9	**10.8**
Hiking	20	**20**	**20**	**0**	**0**	-0.25	1.9	**20.0**	15.6	**43.1**	55.9	17.6	**18.3**	19.9	**9.6**
Maintenance	20	**0**	**0**	**0**	**0**	na	na	**0.0**	**0.0**	na	na	**0.0**	**0.0**	na	na
nurikabe	20	**11**	**11**	**0**	**0**	1.2	1.0	10.9	**11.0**	75.1	**74.5**	10.9	**11.0**	125.3	**116.2**
Openstacks	20	**20**	**20**	**0**	**0**	0.01	7.1	**20.0**	19.7	**190.4**	193.2	13.0	**20.0**	95.5	**9.0**
organic-synthesis	20	**3**	**3**	**0**	**0**	64.75	1.0	**3.0**	**3.0**	**4.0**	**4.0**	**3.0**	**3.0**	0.1	**0.1**
organic-synthesis-split	10	9	**11**	0	**2**	0.2	**0.4**	9.0	**11.0**	**284.2**	**284.2**	8.9	**10.1**	**29.3**	95.2
Parking	20	**20**	**20**	**0**	**0**	0.32	1.0	**20.0**	20.0	**94.6**	94.8	19.8	**19.8**	**142.0**	142.3
settlers	20	0	**7**	0	**7**	na	na	0.0	**7.0**	na	na	0.0	**7.0**	na	na
snake	20	**3**	**3**	**0**	**0**	0.77	1.0	**3.0**	**3.0**	**45.7**	**45.7**	**3.0**	3.0	**72.6**	72.7
spider	20	**18**	16	**3**	1	2.92	2.3	**16.7**	15.8	51.2	**47.1**	13.2	**15.4**	435.2	**150.9**
termes	20	2	**4**	0	**2**	-0.47	3.7	2.0	**3.8**	**103.0**	113.0	1.4	**4.0**	287.4	**76.5**
Tetris	20	**4**	**4**	**0**	**0**	0.78	1.0	**4.0**	**4.0**	**115.5**	**115.5**	4.0	**4.0**	584.0	**578.4**
Thoughtful	20	**18**	15	**3**	0	0.11	4.1	**17.9**	14.4	**90.6**	94.9	**15.7**	14.5	8.7	**2.7**
Transport	20	**2**	**2**	**0**	**0**	1.19	1.0	**2.0**	**2.0**	**2478.5**	**2478.5**	2.0	**2.0**	644.6	**636.5**
Visitall	20	5	**13**	0	**8**	0.08	**0.5**	5.0	**12.9**	**1691.2**	1725.0	5.0	**12.7**	**49.0**	132.1
TOTAL	451	220	**234**	10	**24**	1.9	1.46	218.2	**227.5**	**374.4**	377.2	199.4	**228.4**	152.9	**118.5**

Table 5.: Comparison by domain of the *delta* performance compared to Baseline. Results are presented in terms of number of instances where the predictions have been used (Used). Domains where predictions are never used are omitted.

Domain	**Used**	**Cov. P**	**Cov. *w*A***	**Solved P**	**Solved *w*A***	δ	$\frac{\exp(P)}{\exp(wA^*)}$	**score-Q P**	**score-Q *w*A***	**avg-C P**	**avg-C *w*A***	**score-T P**	**score-T *w*A***	**avg-T P**	**avg-T *w*A***
Barman	20	**19**	5	**14**	0	-0.05	**0.1**	**18.3**	5.0	177.6	**148.6**	**19.0**	2.4	**74.9**	799.5
caldera	20	13	**15**	0	**2**	6.73	8.9	12.6	**14.7**	**24.1**	**24.1**	9.0	**15.0**	237.9	**14.8**
caldera-split	20	**6**	**6**	**0**	**0**	0.47	3.4	**5.8**	5.8	**105.0**	**105.0**	5.1	**5.6**	384.1	**116.6**
CaveDiving	13	**7**	**7**	**0**	**0**	-0.75	1.0	**7.0**	**7.0**	**110.0**	**110.0**	6.9	**7.0**	50.1	**48.4**
Childsnack	20	**5**	2	**5**	2	-0.41	na	**5.0**	2.0	na	na	**5.0**	2.0	na	na
CityCar	20	**8**	7	**2**	1	-0.24	**0.3**	6.1	**7.0**	190.0	**120.5**	**7.9**	5.4	**89.6**	229.2
data-network	19	**1**	0	**1**	0	0.21	na	**1.0**	0.0	na	na	**1.0**	0.0	na	na
flashfill	4	**11**	**11**	**0**	**0**	-0.97	1.0	**11.0**	**11.0**	**436.2**	**436.2**	**11.0**	**11.0**	60.8	**60.2**
Floortile	5	**2**	**2**	**0**	**0**	-0.6	1.0	**2.0**	**2.0**	**76.0**	**76.0**	2.0	**2.0**	3.8	**3.5**
GED	20	**20**	**20**	**0**	**0**	2.83	**0.1**	18.3	**19.7**	43.6	**40.0**	**19.9**	16.3	**1.3**	10.8
Hiking	20	**20**	**20**	**0**	**0**	-0.46	3.3	18.8	**19.8**	59.4	**55.9**	17.9	**18.0**	30.4	**9.6**
Maintenance	20	**1**	0	**1**	0	3.91	na	**1.0**	0.0	na	na	**1.0**	0.0	na	na
nurikabe	20	**11**	**11**	**2**	**2**	1.4	1.6	**10.8**	10.7	**68.0**	68.6	10.6	**10.8**	**1.2**	1.2
Openstacks	20	**20**	**20**	**0**	**0**	-0.0	2.0	**20.0**	**20.0**	**193.2**	**193.2**	17.7	**19.8**	25.7	**9.0**
organic-synthesis	20	**3**	**3**	**0**	**0**	64.75	2.5	**3.0**	**3.0**	**4.0**	**4.0**	**3.0**	**3.0**	0.2	**0.1**
organic-synthesis-split	10	**11**	**11**	**0**	**0**	0.09	**0.8**	**11.0**	**11.0**	**324.2**	**324.2**	**10.9**	10.6	**301.8**	310.9
Parking	20	**20**	**20**	**0**	**0**	0.03	1.2	15.8	**20.0**	121.0	**94.8**	18.0	**18.9**	172.1	**142.3**
settlers	20	**7**	**7**	**0**	**0**	-0.84	**0.7**	6.8	**6.9**	596.4	**582.1**	**6.8**	6.0	**167.4**	288.4
snake	20	**7**	3	**4**	0	0.11	**0.0**	**6.3**	3.0	62.0	**45.7**	**7.0**	1.2	**2.3**	72.7
spider	20	**16**	**16**	**1**	**1**	3.08	3.4	14.6	**15.8**	51.2	**46.9**	11.5	**15.3**	435.1	**98.2**
termes	20	**4**	**4**	**0**	**0**	-0.53	**0.7**	**4.0**	3.9	**131.3**	133.3	**4.0**	3.5	**42.3**	60.8
Tetris	20	**5**	4	**2**	1	0.15	1.0	**4.2**	4.0	166.0	**119.0**	**4.9**	3.8	**349.3**	493.3
Thoughtful	20	**19**	15	**4**	0	0.07	3.0	**18.6**	14.5	**93.7**	94.9	**17.2**	14.4	5.9	**2.7**
Transport	20	**2**	**2**	**1**	**1**	-0.07	**0.1**	1.4	**2.0**	6295.0	**2466.0**	**2.0**	1.5	**47.2**	330.9
Visitall	20	**13**	**13**	**2**	**2**	-0.16	1.1	12.6	**12.8**	3200.7	**3139.2**	**11.9**	11.8	569.1	**487.8**
TOTAL	451	**261**	234	**39**	12	1.4	0.61	**246.0**	231.7	512.9	**485.6**	**241.2**	215.0	160.3	**137.1**

5.3.4. delta versus Baseline

Table 5 shows the domain-by-domain performance comparison between *delta* and the baseline. Overall, *delta* has a very positive impact on coverage, and its use can lead to a substantial reduction of node expansions (the ratio is well below 1). The reduction is well spread across all benchmark domains and, as it should be expected, it bumps up only when the δ value is particularly large (for instance in organic-synthesis and caldera). However, it seems that this adaptive heuristic is quite robust in handling grossly inaccurate predictions. Also the results concerning the cost of the found so-

Table 6.: Comparison by domain of the *discount* performance compared to Baseline. Results are presented in terms of number of instances where the predictions have been used (Used). Domains where predictions are never used are omitted.

Domain	**Used**	**Cov. P**	**Cov. wA***	**Solved P**	**Solved wA***	δ	$\frac{\exp(P)}{\exp(wA^*)}$	**score-Q P**	**score-Q wA***	**avg-C P**	**avg-C wA***	**score-T P**	**score-T wA***	**avg-T P**	**avg-T wA***
Barman	20	**20**	5	**15**	0	-0.08	**0.1**	**18.7**	5.0	203.0	**148.6**	**20.0**	2.4	**63.4**	799.5
caldera	20	13	**15**	1	**3**	6.93	8.4	12.7	**14.6**	**22.7**	23.0	9.4	**15.0**	139.7	**9.4**
caldera-split	20	**6**	**6**	**0**	**0**	0.61	3.8	**6.0**	5.4	**95.0**	105.0	4.0	**5.7**	483.2	**116.6**
CaveDiving	13	**7**	**7**	**0**	**0**	-0.75	1.0	**7.0**	**7.0**	**110.0**	**110.0**	7.0	**7.0**	**48.1**	48.4
Childsnack	20	**3**	2	**2**	1	-0.39	**0.1**	**2.9**	2.0	82.0	**71.0**	**3.0**	1.5	**4.4**	35.6
CityCar	20	**9**	7	**2**	0	-0.39	**0.5**	6.3	**7.0**	230.4	**129.3**	**8.7**	5.9	**183.9**	281.5
data-network	19	**1**	0	**1**	0	0.45	na	**1.0**	0.0	na	na	**1.0**	0.0	na	na
flashfill	4	**11**	**11**	**0**	**0**	-0.97	1.0	**11.0**	**11.0**	**436.2**	**436.2**	11.0	**11.0**	62.0	**60.2**
Floortile	5	**2**	**2**	**0**	**0**	-0.6	1.0	**2.0**	**2.0**	**76.0**	**76.0**	2.0	**2.0**	3.6	**3.5**
GED	20	**20**	**20**	**0**	**0**	3.1	**0.2**	**18.9**	18.6	**39.0**	40.0	**20.0**	16.7	**1.8**	10.8
Hiking	20	19	**20**	0	**1**	-0.48	2.7	17.3	**19.6**	59.6	**55.3**	14.7	**18.9**	16.1	**9.7**
Maintenance	20	**1**	0	**1**	0	3.91	na	**1.0**	0.0	na	na	**1.0**	0.0	na	na
nurikabe	20	**12**	11	**3**	2	1.49	5.2	**11.8**	10.7	**67.3**	68.6	**11.5**	10.7	1.7	**1.2**
Openstacks	20	**20**	**20**	**0**	**0**	-0.0	1.1	**20.0**	**20.0**	**193.2**	**193.2**	18.9	**20.0**	10.4	**9.0**
organic-synthesis	20	**3**	**3**	**0**	**0**	64.75	2.4	**3.0**	**3.0**	**4.0**	**4.0**	**3.0**	**3.0**	0.3	**0.1**
organic-synthesis-split	10	9	**11**	0	**2**	0.2	2.0	9.0	**11.0**	**284.2**	**284.2**	8.6	**10.8**	105.4	**95.2**
Parking	20	**20**	**20**	**0**	**0**	0.0	1.5	15.5	**19.9**	125.0	**94.8**	16.6	**19.4**	227.6	**142.3**
settlers	20	0	**7**	0	**7**	na	na	0.0	**7.0**	na	na	0.0	**7.0**	na	na
snake	20	**7**	3	**4**	0	0.19	**<0.1**	**6.6**	3.0	54.0	**45.7**	**7.0**	1.2	**2.7**	72.7
spider	20	13	**16**	1	**4**	3.44	4.9	12.3	**15.9**	45.6	**43.3**	9.1	**15.5**	473.9	**163.8**
termes	20	**4**	**4**	**0**	**0**	-0.53	6.0	**4.0**	3.8	**128.8**	133.3	3.0	**4.0**	355.2	**60.8**
Tetris	20	**8**	4	**5**	1	0.37	1.4	**7.3**	4.0	133.7	**103.0**	**7.7**	4.0	428.4	**366.4**
Thoughtful	20	**16**	15	**2**	1	0.1	2.9	**15.8**	14.7	**91.5**	92.6	13.7	**15.0**	7.2	**2.7**
Transport	20	**2**	**2**	**2**	**2**	-0.04	na	**2.0**	**2.0**	na	na	**2.0**	**2.0**	na	na
Visitall	20	**16**	13	**4**	1	-0.2	**0.5**	**15.7**	12.8	3180.2	**3174.0**	**15.5**	11.0	**220.8**	479.4
TOTAL	451	**252**	234	**43**	25	1.5	1.35	**237.6**	230.0	504.1	**495.7**	**228.3**	219.5	141.6	**127.1**

Table 7.: Comparison by domain of the *discount-pr* performance compared to Baseline. Results are presented in terms of number of instances where the predictions have been used (Used). Domains where predictions are never used are omitted.

Domain	**Used**	**Cov. P**	**Cov. wA***	**Solved P**	**Solved wA***	δ	$\frac{\exp(P)}{\exp(wA^*)}$	**score-Q P**	**score-Q wA***	**avg-C P**	**avg-C wA***	**score-T P**	**score-T wA***	**avg-T P**	**avg-T wA***
Barman	20	**20**	5	**15**	0	-0.08	**0.1**	**19.1**	5.0	184.0	**148.6**	**20.0**	2.2	**50.1**	799.5
caldera	20	14	**15**	1	**2**	6.87	7.8	13.8	**14.7**	**24.1**	24.3	10.5	**15.0**	190.2	**17.8**
caldera-split	20	**6**	**6**	**0**	**0**	0.5	1.2	**6.0**	5.8	**102.0**	105.0	4.9	**5.8**	141.7	**116.6**
CaveDiving	13	**7**	**7**	**0**	**0**	-0.75	1.0	**7.0**	**7.0**	**110.0**	**110.0**	6.9	**7.0**	49.5	**48.4**
Childsnack	20	**3**	2	**2**	1	-0.39	**0.1**	**2.9**	2.0	82.0	**71.0**	**3.0**	1.5	**3.2**	35.6
CityCar	20	**12**	7	**5**	0	-0.36	**0.4**	**9.2**	7.0	229.0	**129.3**	**11.8**	5.6	**161.3**	281.5
data-network	19	**1**	0	**1**	0	0.54	na	**1.0**	0.0	na	na	**1.0**	0.0	na	na
flashfill	4	**11**	**11**	**0**	**0**	-0.97	1.0	**11.0**	**11.0**	**436.2**	**436.2**	**11.0**	**11.0**	60.2	**60.2**
Floortile	5	**2**	**2**	**0**	**0**	-0.6	1.0	**2.0**	**2.0**	**76.0**	**76.0**	2.0	**2.0**	3.9	**3.5**
GED	20	**20**	**20**	**0**	**0**	3.14	**0.2**	**19.0**	18.5	**38.6**	40.0	**19.9**	16.8	**2.1**	10.8
Hiking	20	**20**	**20**	**0**	**0**	-0.43	7.3	19.1	**19.4**	57.0	**55.9**	17.1	**18.8**	68.4	**9.6**
Maintenance	20	**1**	0	**1**	0	3.91	na	**1.0**	0.0	na	na	**1.0**	0.0	na	na
nurikabe	20	**12**	11	**3**	2	1.49	9.3	**11.8**	10.7	**67.1**	68.6	**11.4**	10.6	2.6	**1.2**
Openstacks	20	**20**	**20**	**0**	**0**	-0.0	1.1	**20.0**	**20.0**	**193.2**	**193.2**	19.4	**19.8**	9.4	**9.0**
organic-synthesis	20	**3**	**3**	**0**	**0**	64.75	2.4	**3.0**	**3.0**	**4.0**	**4.0**	**3.0**	**3.0**	0.2	**0.1**
organic-synthesis-split	10	**11**	**11**	**0**	**0**	0.09	1.1	**11.0**	**11.0**	**324.2**	**324.2**	10.6	**10.8**	**305.9**	310.9
Parking	20	**20**	**20**	**0**	**0**	0.0	1.5	15.4	**19.9**	124.9	**94.8**	16.7	**19.4**	225.8	**142.3**
settlers	20	5	**7**	0	**2**	-0.83	**0.9**	5.0	**7.0**	572.0	**570.0**	4.7	**6.9**	**96.6**	102.8
snake	20	**7**	3	**4**	0	0.22	**0.0**	**6.6**	3.0	54.0	**45.7**	**7.0**	1.2	**3.0**	72.7
spider	20	**16**	**16**	**2**	**2**	3.57	1.5	15.7	**15.8**	43.5	**42.8**	14.1	**15.7**	187.4	**109.6**
termes	20	**4**	**4**	**0**	**0**	-0.54	**0.9**	**4.0**	4.0	**132.3**	133.3	**4.0**	3.8	**59.1**	60.8
Tetris	20	**8**	4	**5**	1	0.34	1.1	**7.3**	4.0	133.0	**103.0**	**7.8**	3.9	393.2	**366.4**
Thoughtful	20	**15**	**15**	**0**	**0**	0.09	**0.8**	14.8	**14.9**	**94.7**	94.9	**14.8**	14.5	**2.0**	2.7
Transport	20	1	**2**	1	**2**	0.19	na	1.0	**2.0**	na	na	1.0	**2.0**	na	na
Visitall	20	**15**	13	**4**	2	-0.18	**0.9**	**14.6**	12.8	3098.2	**3081.3**	**14.3**	11.8	**390.8**	405.4
TOTAL	451	**264**	234	**44**	14	1.5	0.61	**251.3**	230.5	478.8	**470.8**	**247.7**	219.1	133.1	**127.5**

lutions are good, making exceptions for those situations where the adaptive heuristic causes the system to behave in a much more greedy fashion. For instance the increase of solution cost in the Transport domain is significant.

5.3.5. *discount versus Baseline*

Tables 6 and 7 report the performance of, respectively, *discount* and *discount-pr* with regards to the performance of the considered baseline.

discount-pr is the approach that allows to deliver the best coverage performance. Intuitively, considering also the performance of *discount*, this seems to be due to the penalty rate mechanism (the p_{rate}) that can limit some of the issues of *discount*. In particular, looking at the expanded nodes, *discount-pr* tends to expanded a lower number of nodes on average. This is easy to observe in domains such as caldera (both versions), termes, and spider. Further, in Settlers the use of the penalty rate restricts the difference of performance with the baseline to only 2 instances. Similarly in spider, *discount-pr* solves two instances that were not solvable using the baseline, while *discount* solves only one instance more.

5.3.6. Complementarity of adaptive heuristic

Our experimental findings highlight that the approaches complement each other. To quantify this aspect we consider the coverage that would be obtained by a Virtual Best Planner (VBP), i.e., by an oracle that for each instance is able to select the best adaptive heuristic to use among the considered ones. Figure 3 reports such an information comparing the VBP against all adaptive heuristics and the baseline. Figure 3 also shows how for very small instances solved in less than one CPU-time minute, there is no significant performance difference among the compared approaches; intuitively this is due to the fact that for quickly solved instances, the use of the predicted cost does not lead to the exploration of substantially different areas of the search space. When more CPU time is used, and larger areas of the search space are explored, the three best performing adaptive heuristics, i.e., *delta*, *discount*, and *discount-pr* obtain better coverage results than the baseline when more than approximately 100 CPU-time seconds are given to solve instances. In terms of coverage, the use of p_{rate} is clearly useful in *discount*, and the benefit is evenly spread with regards to the given runtime limits.

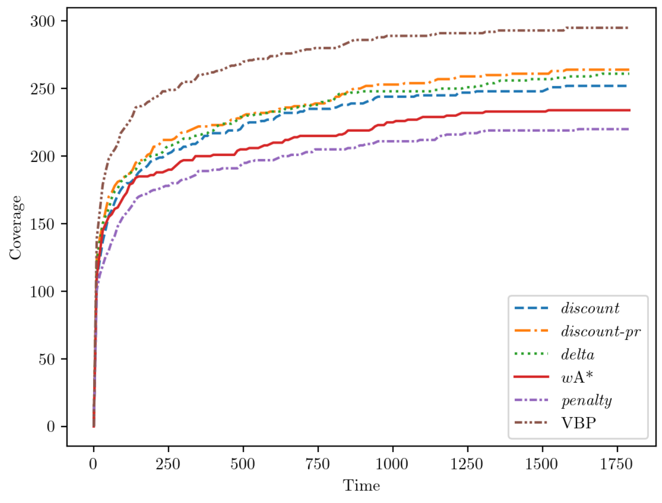

Figure 3.: Coverage over CPU time for wA* with h^{Bound} (with and without using p_{rate}) and the Baseline (wA*) on all the considered benchmark instances.

We conclude our experimental analysis with an experiment on the performance of *discount-pr* when used together with Greedy Best First Search (GBFS). This setting is sometimes exploited by anytime planners to incrementally improve on a solution. The baseline here is GBFS+wA*, where our approach is GBFS+ *discount-pr*. Note

Table 8.: Comparison of the performance of wA* using the best adaptive heuristic function (*discount-pr*) and the baseline wA* alone (top table) or after a run of GBFS (bottom table). Results are presented considering the metrics introduced in Section 5.2.

Approach	Coverage	avg-C	score-Q	PAR10	score-T
discount-pr	**264**	478.8	**251.3**	**8948.8**	**247.7**
wA*	234	**470.8**	230.5	9970.0	219.1
GBFS+*discount-pr*	**314**	845.1	307.6	**7472.9**	**299.4**
GBFS+wA*	**314**	**830.0**	**308.2**	7524.8	290.1

that the second episode of search does not consider the solution found by GBFS during search: GBFS is used only to help increasing instance coverage, since it can be the case that GBFS solves instances that wA* alone can not. In other words, the cost of the GBFS solution (if any) is not used instead of B in h^{Bound}. In both the considered configurations, if the second episode of wA* (with or without the considered adaptive heuristic) is not completed, we penalise the runtime by an amount equal to 1800 seconds. As shown in Table 8, GBFS+wA* tends to provide slightly better quality solutions (1.7% better), but GBFS+*discount-pr* reduces runtime. This suggests that there can be useful synergies between the proposed adaptive heuristic and the GBFS settings. Moreover, the results presented in Table 8 indicate that there may exist different ways for combining GBFS and wA* searches, different from the configuration exploited by state-of-the-art approaches like LAMA, that can lead to further performance improvement.

5.4. Discussion

The results of the performed extensive experimental analysis, indicate that the use of the adaptive heuristics can have a strong impact on the way in which the considered search framework works. Overall, Table 3 suggests that the *discount-pr* delivers the best performance in terms of coverage and runtime, while *penalty* tends to be significantly slower, but can generate plans of the average best quality. As it should be apparent, the focus of the proposed adaptive heuristics is on exploiting the prediction B in order to speed up the search process, and little consideration is given to the quality of the generated solution. To better understand the significance of the performance gain in terms of runtime, we ran a Wilcoxon signed-rank test (Wilcoxon & Wilcox, 1964), which has been commonly used in AI planning for similar purposes (see, for instance the work of Gerevini, Haslum, Long, Saetti, and Dimopoulos (2009); Roberts and Howe (2009); Vallati et al. (2018)). The test indicated that the *penalty*, with regards to runtime, is statistically worse ($p < 0.05$) than the considered baseline. There is instead not a statistically significant difference between the baseline, and the other introduced adaptive heuristics. This is despite a clear coverage gap between some of the proposed adaptive heuristics, particularly *discount-pr* and *delta*. To better understand this aspect, Figure 4 compares the number of nodes expanded by wA* and each adaptive heuristics. The figure highlights that, beside the mentioned *penalty* case, for all the other adaptive heuristics there is not a clear gain in terms of number of expanded nodes.

We performed the Wilcoxon test also for comparing the VBP and the baseline, and in this case the test indicates that the VBP is capable of delivering runtime

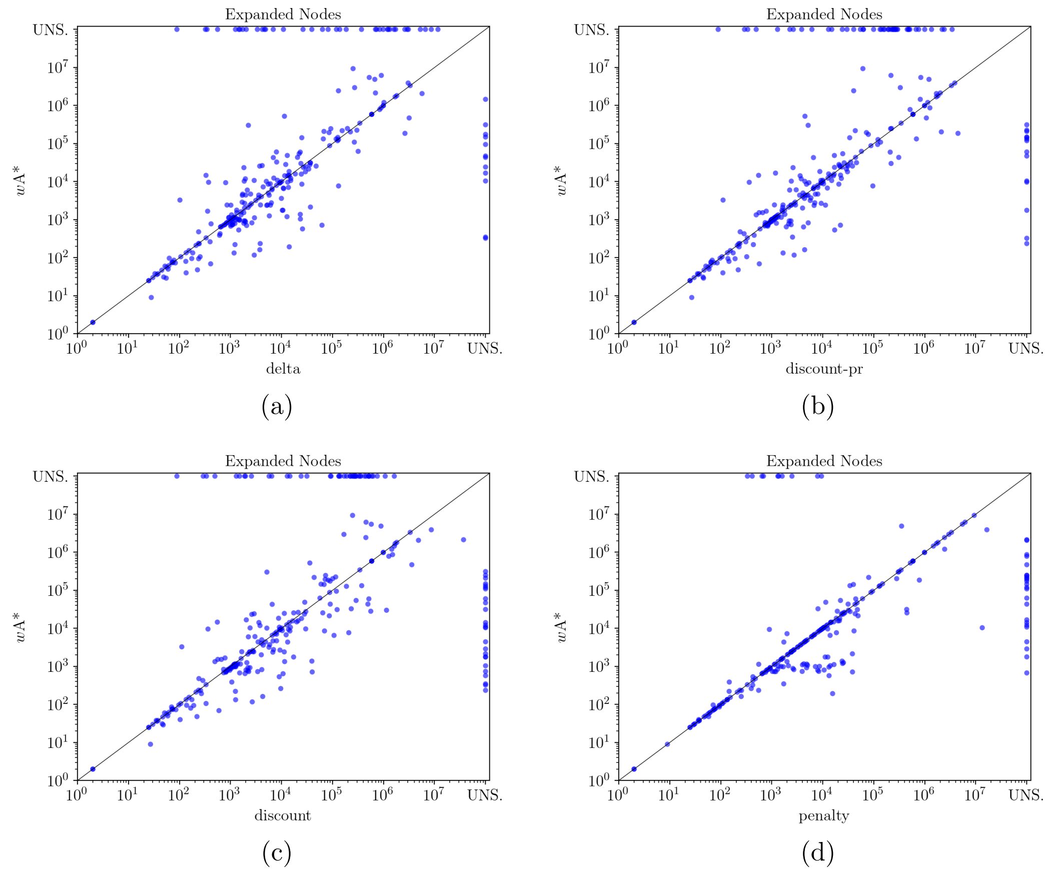

Figure 4.: Scatter plot of the expanded nodes for the proposed approaches compared to the baseline.

performance that are statistically better than those of the considered baseline. This confirms the high degree of complementarity between the proposed adaptive heuristics. An interesting point to explore in future work is therefore how to effectively combine the proposed adaptive heuristics.

With regards to the quality of generated plans, it is easy to observe from Table 3, and by the domain-by-domain analysis, that the use of the adaptive heuristics generally reduces the quality of solutions. Only *penalty* is capable of generating solutions that are, on average, better than those identified by the baseline approach.

Finally, regarding the relation between the predicted B and the actual found solution, we observed that for a large number of benchmark domains the predictions tend to be reasonably accurate. Focusing on the results achieved by the best performing adaptive heuristic *discount-pr*, and shown in Table 7: in 10 domains the prediction is proved wrong by less than 40%; in 5 domains, however, predictions overestimate by more than three times the cost of the solution found. Interestingly, the use of reformulation on a domain model can lead to dramatically different predictions. Notable examples are the Caldera and the Organic-synthesis benchmark domains: predictions on the original domain model tend to be significantly wrong, but on the split models are very accurate. Split models are generated by automatically performing the operator schema split approach (Areces, Bustos, Domínguez, & Hoffmann, 2014). Further, it seems that a large overestimation does not always lead to worse performance. In GED and Maintenance, where predictions largely overestimate, the use of the proposed adaptive heuristics is still beneficial. This is not the case in Spider, where the large overestimation leads instead to significantly worse performance. It may indeed be the case that the structure of the domain plays a pivotal role in the behaviour of the proposed adaptive heuristics, when B is a large over- or under- estimation. The usefulness of a provided prediction does not only depend on the fact that it accurately identifies the quality of a solution plan that the considered system would have found. As noted, the use of the prediction is modifying the way in which the search space is explored, so the usefulness of a prediction also depends on the structure of the search space for the considered domain, and on the way in which the search space is explored. Our intuition is that a good prediction should point the search towards a region of the search space that is "rich" of solutions, i.e. where many solutions can be found. In that way, it would effectively support the search process.

Summarising, the proposed approaches demonstrated the be capable of exploiting the knowledge provided under the form of predictions of the quality of a solution. Some approaches are able to generate solutions faster, while others are more focused on the quality of the generated plans. The results seem to indicate that having some information about the quality of a solution plan, even if grossly inaccurate, is better than having no information at all. Further, our analysis highlighted that given the high level of complementarity of the proposed approaches, they can be fruitfully combined in order to further extend the capabilities of a planning approach.

6. Related Work

In this section, we describe related work on (i) the use of feature-based models in automated reasoning, (ii) feature-based predictive models in automated planning, (iii) the use of bounds in automated planning, and (iv) the combination of inductive and deductive approaches, pointing out some important differences between our approach and the most related work.

6.1. Feature-based Predictive Models for Automated Reasoning

SATZilla is a prominent example of an algorithm portfolio designed for SAT (Xu et al., 2008). It exploits regression techniques to build a predictor of the runtime of a number of provided SAT solvers. For solving a new SAT problem, SATZilla computes the values of a large set of features, predicts the performance of the considered SAT solvers. The solvers are then ordered according to the predicted runtime, and executed for the predicted time, following the established ordering. It should be noted that, for the sake of overall performance, SATZilla exploits also pre- and backup solvers: the former is used before extracting features, in order to quickly solve trivial instances. The latter is run when all the selected solvers failed.

Similarly, Claspfolio (Gebser et al., 2011) exploits regression-based predictive models for selecting, among a range of predefined configurations of the well-known ASP solver Clasp (Gebser, Kaufmann, Neumann, & Schaub, 2007), the best configuration to minimise the runtime on a given ASP instances. Predictions are made according to a set of features that are extracted from the considered ASP problem. Claspfolio 2 (Hoos, Lindauer, & Schaub, 2014) is an improved version of Claspfolio, that provides a modular architecture that allows for integrating several different approaches and techniques for extracting features, predicting solvers' performance, and combine solvers into a portfolio.

Matos, Planes, Letombe, and Marques-Silva (2008) propose an algorithm portfolio solving the MaxSAT problem. According to the values of several features, it estimates the runtime of each incorporated solver, and then solves the instance with the estimated fastest solver. The estimation is done using a (linear) model configured by performing ridge regression (Marquardt & Snee, 1975).

Similarly, Pulina and Tacchella (2007) study an algorithm portfolio solving the QBF problem. They identify some features of the QBF problem, and investigate the usage of four inductive models to select the best solver to use according to the values of the identified features. Maratea, Pulina, and Ricca (2014) applied a similar approach for solving ASP problems: given a set of easy-to-compute features, extracted from the given instance, the authors propose a classification-based approach for selecting the most promising solver to run in order to minimise the required runtime.

In the abstract argumentation field, recent works (Cerutti, Thimm, & Vallati, 2020; Vallati, Cerutti, & Giacomin, 2019) demonstrate how predictive models, based on feature extracted by considering the directed graph representation of argumentation frameworks, can be used to predict the runtime of solvers, and to predict some interesting properties of the framework, such as the number of extensions of a given type.

6.2. Features-based Approaches for Automated Planning

Approaches based on features for performing predictions have been studied for decades in the planning field. Howe et al. (1999) build regression models based on five features to predict performance of six planners, in order to identify the best to use for solving a given classical planning instance. Subsequent work by Roberts et al. (2008) and Roberts and Howe (2009) includes comprehensive features regarding PDDL statistics, considered additional planners, and explored more complex models to predict the runtime of planners on previously unseen instances. Fawcett et al. (2014) introduce the set of features that is considered in this paper, and exploited them to generate a domain-independent approach to accurately predict the runtime of planners.

IBaCoP (Cenamor, de la Rosa, & Fernández, 2016) is a planning system that considers a large set of features, also including the SAS+ reformulation of a planning instance, to combine planners into an instance-specific portfolio. A version of IBaCoP took part in the 2014 edition of the International Planning Competition (IPC), and won the sequential satisficing track Vallati et al. (2018). On a similar note, Cenamor, Vallati, and Chrpa (2019) proposed a set of features to predict the performance of planners on temporal planning problems. Some work has also focused on designing features-based domain-independent predictors of the runtime of optimal planners (Rizzini, Fawcett, Vallati, Gerevini, & Hoos, 2017).

Aside from predicting the runtime of planners on instances, for the sake of performing algorithm selection or combining solvers into portfolios, features have also been used as a measure to compare planning instances, for supporting case-based planning (Vallati, Serina, Saetti, & Gerevini, 2016), and for assessing the informativeness and complexity of existing benchmarks (Cenamor & Pozanco, 2019).

Finally, Gerevini, Saetti, and Vallati (2015) propose a domain-specific approach, based on a very limited set of features, for predicting the make-span optimal length of a plan to boost SAT-based planning. Our approach is similar to the one proposed by Gerevini et al. (2015), but we introduce a domain-independent predictor, that leverages on a wide range of features.

A slightly different line of work introduced stratified sampling to predict the optimal cost of a solution in regular search spaces (Lelis et al., 2016). The designed approach can provide very accurate estimations, but is able to handle only search problems with single and fully defined goal states – i.e. with cases where the goal is unique and fully described and the search space is regular.

6.3. Bounds in Automated Planning

There has been a significant amount of work focusing on the problem of finding a solution constrained to satisfy a given bound, i.e. an hard constraint on the quality (cost) of the solution. Beside the anytime search approach implemented in a large number of the existing satisficing planning engines, Stern et al. (2014) introduce approaches to quickly find a plan whose cost is within a given bound. Thayer and Ruml (2011) propose an algorithm for dealing with the so-called Bounded suboptimal search: given an external bound $B \geq 1$, the task is to find a solution whose cost is lower than or equal to $B \times cost_{\text{opt}}$, where $cost_{\text{opt}}$ is the cost of the optimal solution. Percassi, Gerevini, and Geffner (2017) propose an extension of the well-know planner LAMA where, at each search episode, the cost of the incumbent solution is used as a bound to prune the search space through an admissible heuristic.

6.4. Combination of planning and learning approaches

With regards to the combination of inductive and deductive approaches for deriving implicit knowledge about the problem instance to be solved, an extensive overview of the area (and of the more general field of learning for planning) is provided by Celorrio et al. (2012). A notable approach that takes advantage of this combination include the use of decision trees (de la Rosa, Celorrio, Fuentetaja, & Borrajo, 2011), to learn some control knowledge that can be exploited during the exploration of the search space. Krajnanský, Hoffmann, Buffet, and Fern (2014) introduced a technique for learning rules to prune the search space, and Yoon, Fern, and Givan (2006) use learning to optimise a generalised heuristic approach for solving a given class of planning instances. There is also an increasing interest in trying to exploit neural network to improve on planning algorithms, (e.g., Shen, Trevizan, and Thiébaux (2019); Toyer, Thiébaux, Trevizan, and Xie (2020)). For example the work done by Toyer et al. (2020) does so by proposing the ASNET architecture, a layered graph representation whose scope is that of approximating the state space by using a planning graph representation (Blum and Furst (1997)). The training of the ASNET gives weights to propositional and action atoms and blends them together in a convolutions neural network approach. The authors show how this can be used to devise policies for generalising the problem of action selection from smaller to larger problems, for both deterministic and probabilistic planning tasks. The main difference with our work lies in the overall objective. Their solution is indented to predict information that is to be used in a domain specific way, while our predictor is domain independent; this introduces different challenges, but we do not exclude that there could be ways to exploit some synergy between the two approaches.

7. Conclusions

Automated planning is a prominent Artificial Intelligence challenge, as well as being a common capability requirement for intelligent autonomous agents. In this paper we have addressed the problem of exploiting plan cost predictions, computed at preprocessing through machine learning techniques, in order to improve planning performance for propositional domains with action costs.

An effective use of these predictions during planning should take into account that the predicted cost can be (even grossly) inaccurate with respect to the best quality plan that a planning approach can found within a certain time limit. We have developed a cost prediction model that is based on standard machine learning techniques using a large set of instance features, and we have proposed a set of methods to exploit the predictions made by our model in the context of wA*. The underlying idea is to dynamically adjust, during search, the input weight w of the heuristic used in wA*, taking into account the predicted plan cost, and preventing the search to be misguided when the prediction is severely inaccurate.

We have carried out a large experimental evaluation demonstrating the usefulness of the designed approaches. In particular, we observed that: (i) it is possible to exploit a prediction of the cost of a solution to improve the planning performance either in terms of runtime, or in terms of quality of the generated solutions, and (ii) the proposed methods to guide the search process using a given prediction lead to very different explorations of the search space, and this can be exploited by combining them into a very effective portfolio. We observed that it is challenging to clearly specify what is the most suitable value to be predicted: an accurate prediction of the cost of an optimal plan can drive the search towards an area of the search space where it is hard to find a solution; large under- or over- estimations can reduce the effectiveness of the approaches. Nevertheless, our results suggest that, even if the prediction is highly inaccurate, exploiting the prediction can be beneficial: in other words, some (potentially wrong) knowledge is better than no knowledge.

We are interested in exploiting the highlighted complementarity of the introduced techniques, possibly in a portfolio configuration, and we are interested in testing the predicting model and the introduced policies in a more controlled environment, using synthetically generated benchmarks (Roberts, Howe, & Ray, 2014). We are also interested in investigating an extension of our approach to predict and exploit makespan and plan cost in PDDL metric-temporal planning (Fox & Long, 2003), taking also into account potential issues of existing IPC benchmarks (Radzi, 2010), and in multi-agent planning (Gerevini, Lipovetzky, Peli, et al., 2019; Gerevini, Lipovetzky, Percassi, Saetti, & Serina, 2019; Nissim & Brafman, 2014). Moreover, we plan to explore novel predictive approaches, that can make state-dependent predictions of the distance from a goal state, rather a single than problem-dependent prediction of the cost of a good quality solution. Finally, we are looking at improving the quality of predictions by combining different predictors together.

Acknowledgements

We thank the anonymous reviewers for their useful comments. This work was supported by MIUR "Fondo Dipartimenti di Eccellenza 2018-2022" of the DII Department at the University of Brescia. Mauro Vallati was supported by a UKRI Future Leaders Fellowship [grant number MR/T041196/1].

References

Areces, C., Bustos, F., Dominguez, M., & Hoffmann, J. (2014). Optimizing planning domains by automatic action schema splitting. In *Proc. of icaps 2014.*

Areces, C., Bustos, F., Domínguez, M. A., & Hoffmann, J. (2014). Optimizing planning domains by automatic action schema splitting. In *Proceedings of the twenty-fourth international conference on automated planning and scheduling, ICAPS.*

Blum, A., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artif. Intell.*, *90*(1-2), 281–300.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, *129*(1-2), 5–33.

Bonisoli, A., Gerevini, A., Saetti, A., & Serina, I. (2015). Effective plan retrieval in case-based planning for metric-temporal problems. *J. Exp. Theor. Artif. Intell.*, *27*(5), 603–647.

Borrajo, D., Roubícková, A., & Serina, I. (2014). Progress in case-based planning. *ACM Comput. Surv.*, *47*(2), 35:1–35:39.

Botea, A., Enzenberger, M., Müller, M., & Schaeffer, J. (2005). Macro-FF: improving AI planning with automatically learned macro-operators. *J. Artif.Intell. Res. (JAIR)*, *24*, 581-621.

Celorrio, S. J., de la Rosa, T., Fernández, S., Fernández, F., & Borrajo, D. (2012). A review of machine learning for automated planning. *The Knowledge Engineering Review (KER)*, *27*(4), 433–467.

Cenamor, I., de la Rosa, T., & Fernández, F. (2012). Mining IPC-2011 results. In *Proceedings of the 3rd workshop on the international planning competition.*

Cenamor, I., de la Rosa, T., & Fernández, F. (2013). Learning predictive models to configure planning portfolios. In *Proceedings of the 4th workshop on planning and learning (ICAPS-PAL 2013)* (pp. 14–22).

Cenamor, I., de la Rosa, T., & Fernández, F. (2016). The ibacop planning system: Instance-based configured portfolios. *J. Artif. Intell. Res.*, *56*, 657–691.

Cenamor, I., & Pozanco, A. (2019). Insights from the 2018 ipc benchmarks. In *Proc. of the workshop of the ipc WIPC.*

Cenamor, I., Vallati, M., & Chrpa, L. (2019). On the predictability of domain-independent temporal planners. *Comput. Intell.*, *35*(4), 745–773.

Cerutti, F., Thimm, M., & Vallati, M. (2020). An experimental analysis on the similarity of argumentation semantics. *Argument Comput.*, *11*(3), 269–304.

Chrpa, L., & Vallati, M. (2019). Improving domain-independent planning via critical section macro-operators. In *The thirty-third AAAI conference on artificial intelligence, AAAI* (pp. 7546–7553). AAAI Press.

Chrpa, L., Vallati, M., & McCluskey, T. L. (2019). Inner entanglements: Narrowing the search in classical planning by problem reformulation. *Computational Intelligence*, *35*(2), 395–429.

de la Rosa, T., Celorrio, S. J., Fuentetaja, R., & Borrajo, D. (2011). Scaling up heuristic planning with relational decision trees. *J. Artif.Intell. Res. (JAIR)*, *40*.

Domshlak, C., Hoffmann, J., & Katz, M. (2015). Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence*, *221*, 73–114.

Fawcett, C., Vallati, M., Hutter, F., Hoffmann, J., Hoos, H. H., & Leyton-Brown, K. (2014). Improved features for runtime prediction of domain-independent planners. In *Proc. of ICAPS 2014.*

Fikes, R., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, *2*(3/4), 189–208.

Fox, M., & Long, D. (2003). PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, *20*, 61–124.

Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., Schneider, M. T., & Ziller, S. (2011). A portfolio solver for answer set programming: Preliminary report. In *Proceedings of the 11th international conference logic programming and nonmonotonic reasoning (LPNMR)* (pp. 352–357). Springer.

Gebser, M., Kaufmann, B., Neumann, A., & Schaub, T. (2007). *clasp* : A conflict-driven answer set solver. In *Proceedings of the 9th international conference logic programming and nonmonotonic reasoning (LPNMR)* (pp. 260–265). Springer.

Gerevini, A., Haslum, P., Long, D., Saetti, A., & Dimopoulos, Y. (2009). Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, *173*(5-6), 619–668.

Gerevini, A., Lipovetzky, N., Peli, N., Percassi, F., Saetti, A., & Serina, I. (2019). Novelty messages filtering for multi agent privacy-preserving plannin. In *Proceedings of the twelfth international symposium on combinatorial search, SOCS* (pp. 79–87). AAAI Press.

Gerevini, A., Lipovetzky, N., Percassi, F., Saetti, A., & Serina, I. (2019). Best-first width search for multi agent privacy-preserving planning. In *Proceedings of the twenty-ninth international conference on automated planning and scheduling, ICAPS* (pp. 163–171). AAAI Press.

Gerevini, A., Saetti, A., & Serina, I. (2003). Planning through stochastic local search and temporal action graphs. *J. Artif.Intell. Res. (JAIR)*, *20*, 239–290.

Gerevini, A., Saetti, A., & Serina, I. (2006). An approach to temporal planning and scheduling in domains with predictable exogenous events. *J. Artif. Intell. Res.*, *25*, 187–231.

Gerevini, A., Saetti, A., & Serina, I. (2008). An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artif. Intell.*, *172*(8-9), 899–944.

Gerevini, A., Saetti, A., & Serina, I. (2011a). An empirical analysis of some heuristic features for planning through local search and action graphs. *Fundam. Inform.*, *107*(2-3), 167–197.

Gerevini, A., Saetti, A., & Serina, I. (2011b). Planning in domains with derived predicates through rule-action graphs and local search. *Ann. Math. Artif. Intell.*, *62*(3-4), 259–298.

Gerevini, A., Saetti, A., & Vallati, M. (2014). Planning through automatic portfolio configuration: The pbp approach. *J. Artif.Intell. Res. (JAIR)*, *50*, 639–696.

Gerevini, A., Saetti, A., & Vallati, M. (2015). Exploiting macro-actions and predicting plan length in planning as satisfiability. *AI Commun.*, *28*(2), 323–344.

Gerevini, A., & Schubert, L. K. (2000). Discovering state constraints in DISCOPLAN: some new results. In *Proceedings of the seventeenth national conference on artificial intelligence and twelfth conference on on innovative applications of artificial intelligence* (pp. 761–767).

Ghallab, M., Nau, D. S., & Traverso, P. (2004). *Automated planning - theory and practice.* Elsevier.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD explorations*, *11*(1), 10–18.

Helmert, M. (2003). Complexity results for standard benchmark domains in planning. *Artif. Intell.*, *143*(2), 219–262.

Helmert, M. (2006). The Fast Downward planning system. *J. Artif.Intell. Res. (JAIR)*, *26*, 191–246.

Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. of icaps 2009.*

Hoffmann, J. (2011). Analyzing search topology without running any search: On the connection between causal graphs and h+. *J. Artif.Intell. Res. (JAIR)*, *41*, 155–229.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *J. Artif.Intell. Res. (JAIR)*, *14*, 253–302.

Hoos, H., Lindauer, M. T., & Schaub, T. (2014). claspfolio 2: Advances in algorithm selection for answer set programming. *Theory and Practice of Logic Programming*, *14*(4-5), 569–585.

Howe, A., Dahlman, E., Hansen, C., Von Mayrhauser, A., & Scheetz, M. (1999). Exploiting competitive planner performance. In *Proc. of ecp-99* (pp. 62–72).

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

Hutter, F., Xu, L., Hoos, H. H., & Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods & evaluation. *Artif. Intell.*, *206*, 79–111.

Hutter, F., Xu, L., Hoos, H. H., & Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, *206*, 79–111.

Korf, R. E. (1985). Macro-operators: A weak method for learning. *Artif. Intell.*, *26*(1), 35–77. Retrieved from `https://doi.org/10.1016/0004-3702(85)90012-8`

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., & Leyton-Brown, K. (2017). Auto-weka 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, *18*, 25:1–25:5.

Krajnanský, M., Hoffmann, J., Buffet, O., & Fern, A. (2014). Learning pruning rules for heuristic search planning. In *Proceedings of the twenty-first european conference on artificial intelligence (ECAI 2014)* (pp. 483–488).

Lelis, L. H., Stern, R., Jabbari Arfaee, S., Zilles, S., Felner, A., & Holte, R. C. (2016). Predicting optimal solution costs with bidirectional stratified sampling in regular search spaces. *Artificial Intelligence*, *230*, 51 - 73.

Lipovetzky, N., & Geffner, H. (2011). Searching for plans with carefully designed probes. In *Proc. of ICAPS 2011* (pp. 154–161).

Maratea, M., Pulina, L., & Ricca, F. (2014). A multi-engine approach to answer-set programming. *Theory and Practice of Logic Programming*, *14*(6), 841–868.

Marquardt, W., D., & Snee, D. (1975). Ridge regression in practice. *The American Statistician*, *29(1)*, 3–20.

Matos, P., Planes, J., Letombe, F., & Marques-Silva, J. (2008). A MAX-SAT algorithm portfolio. In *Proceedings of the 18th european conference on artificial intelligence (ecai)* (pp. 911–912). IOS Press.

Nakhost, H., Müller, M., Valenzano, R., & Xie, F. (2011). Arvand: the art of random walks. In *Booklet of the seventh international planning competition.*

Nissim, R., & Brafman, R. I. (2014). Distributed heuristic forward search for multi-agent planning. *J. Artif. Intell. Res.*, *51*, 293–332.

Percassi, F., Gerevini, A., & Geffner, H. (2017). Improving plan quality through heuristics for guiding and pruning the search: A study using LAMA. In *Proceedings of the tenth international symposium on combinatorial search (SOCS 2017)* (pp. 144–148).

Percassi, F., Gerevini, A. E., Scala, E., Serina, I., & Vallati, M. (2020). Generating and exploiting cost predictions in heuristic state-space planning. In *Proceedings of the thirtieth international conference on automated planning and scheduling,* (pp.

569–573).

Pohl, I. (1970). Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, *1*(3), 193–204.

Pulina, L., & Tacchella, A. (2007). A multi-engine solver for quantified boolean formulas. In *Proceedings of the 13th international conference on principles and practice of constraint programming (cp)* (pp. 574–589). Springer.

Radzi, N. H. M. (2010). *Multi-objective planning using linear programming* (Unpublished doctoral dissertation). University of Strathclyde, Glasgow, UK.

Richter, S., & Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif.Intell. Res. (JAIR)*, *39*, 127–177.

Rintanen, J. (2012). Engineering efficient planners with SAT. In *Proceedings of the 20th european conference on artificial intelligence (ECAI)* (pp. 684–689).

Rizzini, M., Fawcett, C., Vallati, M., Gerevini, A., & Hoos, H. (2017). Static and dynamic portfolio methods for optimal planning: An empirical analysis. *International Journal on Artificial Intelligence Tools*, *26*(1), 1–27.

Roberts, M., & Howe, A. E. (2009). Learning from planner performance. *Artif. Intell.*, *173*(5-6), 536–561.

Roberts, M., Howe, A. E., & Ray, I. (2014). Evaluating diversity in classical planning. In *Proceedings of the twenty-fourth international conference on international conference on automated planning and scheduling* (p. 253–261).

Roberts, M., Howe, A. E., Wilson, B., & desJardins, M. (2008). What makes planners predictable? In *Proceedings of the 18th international conference on automated planning and scheduling (ICAPS 2008)* (pp. 288–295).

Robnik-Sikonja, M., & Kononenko, I. (1997). An adaptation of relief for attribute estimation in regression. In *Proceedings of the fourteenth international conference on machine learning* (p. 296–304). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Scala, E. (2014). Plan repair for resource constrained tasks via numeric macro actions. In *Proc. of ICAPS 2014*.

Scala, E., & Torasso, P. (2015). Deordering and numeric macro actions for plan repair. In *Proc. of IJCAI 2015* (pp. 1673–1681).

Seipp, J., Sievers, S., Helmert, M., & Hutter, F. (2015). Automatic configuration of sequential planning portfolios. In *Proc. of AAAI 2015* (pp. 3364–3370).

Serina, I. (2010). Kernel functions for case-based planning. *Artificial Intelligence*, *174(16-17)*, 1369–1406.

Shen, W., Trevizan, F. W., & Thiébaux, S. (2019). Learning domain-independent planning heuristics with hypergraph networks. *CoRR*, *abs/1911.13101*.

Stern, R., Felner, A., van den Berg, J., Puzis, R., Shah, R., & Goldberg, K. (2014). Potential-based bounded-cost search and anytime non-parametric A*. *Artificial Intelligence*, *214*, 1–25.

Thayer, J. T., & Ruml, W. (2011). Bounded suboptimal search: A direct approach using inadmissible estimates. In *Proc. of IJCAI 2011* (pp. 674–679).

Toyer, S., Thiébaux, S., Trevizan, F. W., & Xie, L. (2020). Asnets: Deep learning for generalised planning. *J. Artif. Intell. Res.*, *68*, 1–68.

Vallati, M., Cerutti, F., & Giacomin, M. (2019). Predictive models and abstract argumentation: the case of high-complexity semantics. *Knowl. Eng. Rev.*, *34*.

Vallati, M., Chrpa, L., & McCluskey, T. L. (2018). What you always wanted to know about the deterministic part of the international planning competition (IPC) 2014 (but were too afraid to ask). *The Knowledge Engineering Review (KER)*, *33*, e3.

Vallati, M., Chrpa, L., & Serina, I. (2020). Mevo: a framework for effective macro sets evolution. *J. Exp. Theor. Artif. Intell.*, *32*(4), 685–703.

Vallati, M., Hutter, F., Chrpa, L., & McCluskey, T. L. (2015). On the effective configuration of planning domain models. In *Proceedings of the twenty-fourth international joint conference on artificial intelligence, IJCAI* (pp. 1704–1711).

Vallati, M., & Serina, I. (2018). A general approach for configuring PDDL problem models. In *Proc. of ICAPS 2019* (pp. 431–436).

Vallati, M., Serina, I., Saetti, A., & Gerevini, A. (2016). Identifying and exploiting features for effective plan retrieval in case-based planning. *Fundam. Inform.*, *149*(1-2), 209–240.

Wilcoxon, F., & Wilcox, R. A. (1964). *Some rapid approximate statistical procedures.* Pearl River, N.Y.: American Cyanamid Co.

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2008). SATzilla: Portfolio-based Algorithm Selection for SAT. *Journal of Artificial Intelligence Research*, 565–606.

Yoon, S. W., Fern, A., & Givan, R. (2006). Learning heuristic functions from relaxed plans. In *Proceedings of the sixteenth international conference on automated planning and scheduling, (ICAPS 2006)* (pp. 162–171).