Received: 24 January 2017 | Revised: 30 April 2018 | Accepted: 29 March 2019

DOI: 10.1111/coin.12211

ORIGINAL ARTICLE

WILEY Computational Intelligence

On the predictability of domain-independent temporal planners

Isabel Cenamor[1] | Mauro Vallati[2] | Lukáš Chrpa[3,4]

[1]Department of Computer Science and Engineering, Universidad Carlos III de Madrid, Getafe, Spain

[2]School of Computing and Engineering, University of Huddersfield, Huddersfield, UK

[3]Department of Computer Science, Czech Technical University in Prague, Prague, Czech Republic

[4]Department of Theoretical Computer Science and Mathematical Logic, Charles University in Prague, Prague, Czech Republic

Correspondence
Mauro Vallati, School of Computing and Engineering, University of Huddersfield, Queensgate, Huddersfield HD1 3DH, UK.
Email: m.vallati@hud.ac.uk

Funding information
Czech Science Foundation, Grant/Award Number: 18-07252S; Operational Programme for Research, Development and Education (OP VVV), Grant/Award Number: CZ.02.1.01/0.0/0.0/16_019/0000765

Abstract

Temporal planning is a research discipline that addresses the problem of generating a totally or a partially ordered sequence of actions that transform the environment from some initial state to a desired goal state, while taking into account time constraints and actions' duration. For its ability to describe and address temporal constraints, temporal planning is of critical importance for a wide range of real-world applications. Predicting the performance of temporal planners can lead to significant improvements in the area, as planners can then be combined in order to boost the performance on a given set of problem instances.

This paper investigates the predictability of the state-of-the-art temporal planners by introducing a new set of temporal-specific features and exploiting them for generating classification and regression empirical performance models (EPMs) of considered planners. EPMs are also tested with regard to their ability to select the most promising planner for efficiently solving a given temporal planning problem.

Our extensive empirical analysis indicates that the introduced set of features allows to generate EPMs that can effectively perform algorithm selection, and the use of EPMs is therefore a promising direction for improving the state of the art of temporal planning, hence fostering the use of planning in real-world applications.

KEYWORDS

automated planning, predicting performance, temporal planning

1 | INTRODUCTION

Predicting performance of solvers is an important research direction boosting performance via per-instance solver selection as well as providing interesting insights into aspects that affect solvers' behavior. Prominent examples of the successful application of performance-predicting techniques can be found in combinatorial search,[1] especially in Satisfiability (SAT),[2] answer set programming (ASP),[3] classical planning,[4] and abstract argumentation.[5]

Predictions are possible by exploiting *empirical performance models* (EPMs),[6] which are built by (i) observing the performance of solvers on a large set of training instances, (ii) extracting instance-specific features from each training problem, and (iii) learning a predictive model that maps features' value with the observed performance. Each feature is either a number or a categorical value that represents a property of the domain or problem model (eg, the number of objects). Predictions can then be exploited for selecting promising algorithms or for combining algorithms into a portfolio.[7]

EPMs are well established in artificial intelligence and have been considered in the planning literature since the 1990s. Fink[8] exploited the problem size feature for predicting runtime through linear regression, Howe et al[9] used five features for predicting the performance of six planners, and the subsequent work by Roberts et al[10] and Roberts and Howe[11] provided a larger set of features focused on problem model—written in the Planning Domain Definition Language (PDDL)—statistics and increased the number of considered planners. Most recently, Cenamor et al[12,13] further expanded the feature set by including information about the causal graph (CG) and the domain transition graph (DTG).[14] Fawcett et al[4] also considered features computed by encoding the planning problem as a SAT formula and by analyzing the search space topology. On slightly different tasks, Gerevini et al[15] exploited planning features for predicting the length of a makespan-optimal solution plan of a given problem, whereas Vallati et al[16] provided a features-based approach for improving the efficiency of case-base planning systems. State-of-the-art planning EPMs are focused on classical planning, where actions are executed instantly and no numerical or temporal aspects are considered, and they do not guarantee the ability to predict planners' performance on more expressive planning models. Real-world planning applications, however, usually require to reason also in terms of time constraints; actions are not executed instantly, and it might be necessary to run some actions concurrently. Hence, improvements in temporal planning can have a significant impact on most of the planning applications and foster the use of planning in real-world scenarios.

In this paper, we

- introduce a new set of features that are specific to problems dealing with durative actions and temporal constraints,
- combine the introduced features with existing "classical" (propositional) features,[17]
- use the features to generate classification EPMs that predict whether a planner solves a given problem or not,
- use the features to generate regression EPMs that predict the runtime of a planner on a given problem,
- extract a small subset of the representative set of features, and
- exploit the EPMs for algorithm selection, ie, selecting the appropriate planning engine for a given problem.

Our extensive empirical analysis aims at demonstrating that (i) the generated EPMs are accurate, (ii) the selected subset of features is representative, and (iii) the algorithm selection method

based on the generated EPMs outperforms basic planning engines. Our analysis also provides insights on the state of the art of temporal planning systems that could be fruitfully exploited for improving future planning engines.

The remainder of this paper is organized as follows. First, we discuss related work. We then provide the relevant background on automated planning. Section 4 introduces the set of exploited features. After that, we describe the experimental settings and the framework exploited for the analysis. Then, we analyze the performance of EPMs based on classification and regression and how the EPMs can be exploited for algorithm selection. Finally, we give conclusions.

2 | RELATED WORK

EPMs can be used to predict the performance of algorithms on previously unseen inputs such as problem instances or parameters settings. One of the early applications of EPMs was in SAT, where EPMs have been used for predicting how much time a given algorithm will need to find a solution to a given formula.[6,18]

Gomes and Selman[19] conducted a theoretical and experimental study on the parallel run of stochastic algorithms for solving computationally hard search problems. Their work shows under what conditions running different stochastic algorithms in parallel can give a computational gain over running multiple copies of the same stochastic algorithm in parallel. The empirical hardness of combinatorial problems, which refers to how difficult it is to solve a given problem for a given algorithm, has then been studied by Leyton-Brown et al.[20] More recently, the work of Leyton-Brown et al was extended to create models that are able to predict the runtime of algorithms solving uniform random 3-SAT problems, and the resulting framework was called SATzilla.[2] SATzilla, which has then been extended for dealing with many different SAT problems, is one of the most successful portfolios at the state of the art, and it has been awarded in many tracks and editions of the SAT competition.* By extracting information from SAT instances, under the form of features, it predicts the runtime of algorithms by using EPMs; on the basis of such predictions, SATzilla selects the most promising solvers to be executed on the given SAT instance.

Another successful portfolio-based approach for SAT is instance-specific algorithm configuration (ISAC),[21] which exploits a pool of different configurations of the same solver. Given a previously unseen instance, ISAC exploits EPMs for selecting the most suitable configuration, in order to minimize the expected runtime.

Another area in which EPMs and portfolio approaches have been extensively studied is ASP. A prominent example is claspfolio,[3] which exploits regression-based EPMs for selecting, among a range of predefined configurations of the well-known ASP solver *clasp*,[22] the best configuration to minimize the runtime on a given ASP instance. Predictions are made according to a set of features that are extracted from the considered ASP problem. An improved version of claspfolio, called claspfolio 2,[23] provides a modular architecture that extends the provided set of techniques by integrating new approaches for extracting features, predicting solvers' performance, and combining solvers into a portfolio.

Portfolio approaches have been studied and exploited also in classical planning. BUS[9] is the first approach in which a static portfolio has been tested and implemented for solving planning problems. The authors tested the performance of six planners on over 200 problems

*http://www.satcompetition.org

(all the available benchmarks at that time). According to the observed performance, they then identified a suitable control strategy for combining weaknesses and strengths of the considered planners. Other well-known examples of static portfolios for classical planning include PbP,[24] Fast Downward Stone Soup,[25] and Cedalion.[26] These approaches, after observing the performance of a set of planners on training instances, generate a single portfolio that is then used for solving any (previously unseen) planning problem.

EPMs in classical planning have been exploited also for dynamic planning portfolios that combine the most promising planners into portfolios according to a given planning instance. IBaCoP2,[27] which is a good example of a dynamic planning portfolio approach, exploits EPMs for selecting the most promising planners (from a given set) for maximizing the quality of the solution plans. IBaCoP2 took part in the 2014 edition of the International Planning Competition (IPC) and won the sequential satisficing track.[28] Another dynamic portfolio approach, AllPACA (all planners automatic choice algorithm),[29] took part in the optimal track of the same competition. AllPACA is a portfolio that selects the most promising optimal planner to run on a given planning task. A comparison of static and dynamic portfolio techniques, focused on optimal planning, has been recently done by Rizzini et al.[30]

3 | AUTOMATED PLANNING

Automated planning deals with finding a (partially or totally ordered) sequence of actions transforming the environment from a given initial state to a desired goal state.[31]

3.1 | Classical planning

Classical planning assumes a static, deterministic, and fully observable environment where action effects are instantaneous.

In the classical representation, the environment is specified via first-order logic *predicates*. *States* of the environment are represented as sets *atoms*, fully grounded predicates. A *planning operator* $o = (\text{name}(o), \text{pre}(o), \text{eff}^-(o), \text{eff}^+(o))$ is specified such that $\text{name}(o) = \text{op\_name}(x_1, \ldots, x_k)$ (op_name is a unique operator name, and $x_1, \ldots, x_k$ are variable symbols (arguments) appearing in the operator), $\text{pre}(o)$ is a set of predicates representing the operator's preconditions, and $\text{eff}^-(o)$ and $\text{eff}^+(o)$ are sets of predicates representing the operator's negative and positive effects. *Actions* are fully grounded instances of planning operators. An action $a = (\text{pre}(a), \text{eff}^-(a), \text{eff}^+(a))$ is *applicable* in a state s if and only if $\text{pre}(a) \subseteq s$. The application of a in s (if possible) results in a state $(s \setminus \text{eff}^-(a)) \cup \text{eff}^+(a)$.

A *planning domain* is specified via sets of predicates and planning operators. A *planning problem* is specified via a planning domain, an initial state, and a set of goal atoms. A *solution plan* is a sequence of actions such that a consecutive application of the actions in the plan (starting in the initial state) results in a state that satisfies the goal.

3.2 | Temporal planning

Temporal planning extends classical planning by incorporating the notion of time. Action application (or execution) takes time, and thus, action effects might not be instantaneous. In this

paper, we consider the restricted form of temporal planning supported in PDDL 2.1[32] since it is supported by a range of planning engines. Alternatively, temporal planning tasks can be modeled, for instance, in the New Domain Definition Language[33] and solved by using the EUROPA framework.[34]

A *durative planning operator* $o = (\text{name}(o), \text{dur}(o), \text{pre}_S(o), \text{pre}_E(o), \text{pre}_A(o), \text{eff}_S^-(o), \text{eff}_S^+(o), \text{eff}_E^-(o), \text{eff}_E^+(o))$ is specified such that $\text{name}(o) = \text{op\_id}(x_1, \ldots, x_k)$ (op_id is a unique operator name, and $x_1, \ldots, x_k$ are variable symbols (arguments) appearing in the operator); $\text{dur}(o)$ represents the duration of o's application; $\text{pre}_S(o)$, $\text{pre}_E(o)$, $\text{pre}_A(o)$ are sets of predicates representing "at start," "at end," and "over all" conditions, respectively; and $\text{eff}_S^-(o)$, $\text{eff}_S^+(o)$, $\text{eff}_E^-(o)$, $\text{eff}_E^+(o)$ are sets of predicates representing "at start" negative and positive effects and "at end" negative and positive effects, respectively. *Durative actions* are fully grounded instances of durative planning operators. A durative action a is *applicable* in a state s and time t if and only if $\text{pre}_S(a) \in s$ in t, $\text{pre}_E(a) \in s$ in $t + \text{dur}(a)$, and $\text{pre}_A(a) \in s$ in $[t, t + \text{dur}(a)]$. The result of the application (or execution) of a in s and t (if possible) is such that $\text{eff}_S^-(a)$ becomes false in s and t, $\text{eff}_S^+(a)$ becomes true in s and t, $\text{eff}_E^-(a)$ becomes false in s and $t + \text{dur}(a)$, and $\text{eff}_E^+(a)$ becomes true in s and $t + \text{dur}(a)$.

Solution plan is a list of pairs ⟨action,time⟩ such that each (durative) action is applicable in a current state (starting in the initial state) at time and the result of application of all the actions is a state satisfying the goal.

An example of a temporal operator from the `Driver-Log` domain is provided in Figure 1. The operator (LOAD-TRUCK) represents loading of an object ?obj into a truck ?truck at a location ?loc.

4 | PROBLEM CHARACTERIZATION

Each planner's performance is predicted by using *planning features*, which are extracted from the domain and problem specifications. In a nutshell, a feature is a numerical value (either integer or real) that summarizes a specific property of a considered specification. A vector of planning features, which provides a succinct yet informative description of a problem instance, is provided to a predictive model. The predictive model, which is learned accordingly to the observed performance of the given planner on a training set of problem instances, maintains information about what features are beneficial or detrimental for the given planner and, thus, is able to predict its runtime on a previously unseen problem instance.

```
(:durative-action LOAD-TRUCK
  :parameters (?obj - obj ?truck - truck ?loc - location)
  :duration (= ?duration 2)
  :condition (and
                (over all (at ?truck ?loc))
                (at start (at ?obj ?loc))
            )
  :effect (and
              (at start (not (at ?obj ?loc)))
              (at end (in ?obj ?truck))
          )
)
```

FIGURE 1 An example of a durative operator encoded in Planning Domain Definition Language 2.1

In this work, we build on existing features introduced for classical planning, and we introduce 71 new features that are specific for temporal planning problems. In total, 139 features are extracted for each problem. The following types of features are extracted:

- *PDDL features* that are extracted directly from a PDDL domain and problem specification;
- *SAS+ features*[35] that are extracted from a SAS+ translation of a PDDL domain and problem specification provided by Fast Downward and its temporal version, ie, Temporal Fast Downward (TFD)[36];
- *SAT features* that are extracted by ITSAT,[37] which translates a PDDL domain and problem specification into a single SAT formula.

Other approaches such as Torchlight[38] could be a valuable source of features. However, they do not support models that include temporal reasoning and cannot be exploited in this work.

The considered types of features divided into *propositional* and *temporal* are described in detail in the following subsections.

4.1 | Propositional PDDL

We consider eight features, listed in Table 1, that are extracted by considering both domain and problem specifications in PDDL. They are a subset of features proposed by Roberts et al,[10] namely, number of PDDL requirements, number of types, objects, predicates, facts in the initial state, number of (nondurative) actions, and axioms. Such features can be extracted from classical planning problems and, thus, are not temporal specific.

4.2 | Temporal PDDL

This class of features, listed in Table 2, considers PDDL elements that appear in temporal models only. For instance, we consider the presence of numeric fluents representing the duration of actions; the minimum, maximum, and average and the standard deviation of arity of these fluents; and the number of conditions and effects that should be fulfilled at the start of, in the end of, or during action execution (`at_start`, `at_end`, and `over_all`). By considering the temporal aspects of PDDL models, it can be derived, for example, if some actions have to be run in parallel (one action achieves an effect at the start of its execution and removes it after its execution finishes while another action requires that "effect" during its execution). In total, we consider 31 features in this class.

TABLE 1 Propositional Planning Domain Definition Language (PDDL) features

Name	Type	Description
`Requirements`	Integer	Number of PDDL features that are included in the domain definition
`Types`	Integer	Number of types in the domain definition
`Objects`	Integer	Number of declared objects in the problem definition
`Predicates`	Integer	Number of predicates in the domain definition
`Facts`	Integer	Number of predicates included in the initial state of the problem definition
`Nondurative Actions`	Integer	Number of nondurative actions included in the domain definition
`Axioms`	Integer	Number of axioms included in the domain definition

TABLE 2 Temporal Planning Domain Definition Language features

Name	Type	Description
`Assignment`	Integer	Number of numeric assignments in the problem
`Num durative actions`	Integer	Number of durative actions included in the domain definition
`numeric duration`	Integer	Number of durative actions with numeric duration
`function duration`	Integer	Number of durative actions with a numeric fluent representing the duration
`Avg numeric duration`	Double	Average, minimum, and maximum duration of durative actions with numeric duration
`Functions`	Double	Number of numeric fluents included in the domain definition
`Avg arity`	Double	Average, minimum, and maximum of the arity of numeric fluents included in the domain
`At_start condition`	Double	Average, minimum, maximum, and standard deviation of "at start" conditions
`Over_all condition`	Double	Average, minimum, maximum, and standard deviation of "over all" conditions
`At_end condition`	Double	Average, minimum, maximum, and standard deviation of "at end" conditions
`At_start effect`	Double	Average, minimum, maximum, and standard deviation of "at start" effects
`At_end effect`	Double	Average, minimum, maximum, and standard deviation of "at end" effects

Considering the example operator provided in Figure 1, it can be seen, for example, that it has one `at_start` effect and one `over_all` condition.

4.3 | General SAS+

Many state-of-the-art domain-independent planners exploit SAS+ representation,[35] which can be obtained from PDDL models by the Fast Downward framework.[14] Hence, we considered features that can be derived from SAS+ encoding, which, contrary to predicate-centric PDDL, is object centric.

The object-centric property of SAS+ encoding can be exploited to derive a CG and a DTG. The CG encodes information about dependencies between values of state variables, whereas the DTG—generated for each variable—encodes how actions can affect the value of the specific variable.

In total, 49 features belong to this class. The nontemporal SAS+ features have already been investigated by Cenamor et al[12,13] and are considered by IBaCoP2.[17,27] Fawcett et al[4] also considered a subset of these features in their investigation.

Table 3 shows the list of features extracted from the CG of a problem instance. Table 4 provides the list of the features extracted from the DTGs.

4.4 | Temporal SAS+

The SAS+ formalism, originally designed for encoding classical planning problems, has been recently extended for temporal problems.[36] The main difference is in DTGs—called temporal DTGs—that store information about temporal conditions and effects. As previously introduced, in temporal planning problems, conditions can be required to be satisfied at_start, overall, or at_end of action execution. In total, 30 features are extracted from the temporal SAS+ encoding obtained

TABLE 3 General SAS+ features extracted by considering the causal graph (CG)

Name	Type	Description
Num_VariablesCG	Integer	Number of variables in the CG
high Level VariablesCG	Integer	Number of variables that have at least one goal
total EdgesCG	Integer	Number of edges that connect the nodes in the CG
total WeightCG	Integer	Sum of the weight of the edges in the CG
veRatio	Double	Ratio between variables and edges in the CG
weRatio	Double	Ratio between weight and edges in the CG
wvRatio	Double	Ratio between weight and variables in the CG
hvRatio	Double	Ratio between high-level variables and the other variables
input Edge	Double	Maximum, average, and standard deviation of the input edges at the CG
output Edge	Double	Maximum, average, and standard deviation of the output edges at the CG
input Weight	Double	Maximum, average, and standard deviation of the weight of the input edges at the CG
output Weight	Double	Maximum, average, and standard deviation of the weight of the output edges at the CG
input EdgeHV	Double	Maximum, average, and standard deviation of the input edges at the high level
output EdgeHV	Double	Maximum, average, and standard deviation of the output edges at the high level
input WeightHV	Double	Maximum, average, and standard deviation of the weight of the input edges at the high level
output WeightHV	Double	Maximum, average, and standard deviation of the weight of the output edges at the high level

by TFD.[36] The features are listed in Tables 5 and 6. Several features are "auxiliar" variables, which TFD needs for preprocessing purposes: it uses multivalued state variables and handles logical dependencies and arithmetic subterms via axioms.

4.5 | SAT size

This class of features contains information about the size of a problem encoded in SAT. The only SAT-based solver that is able to handle temporal planning problems is ITSAT.[37] However, for the sake of runtime optimization, ITSAT,[37] which is, so far, the only SAT-based solver handling temporal planning problems, generates a file that includes considered SAT variables and some basic relations between them. By using techniques from SATzilla,[2] we can extract from that file information about the problem size in SAT. In total, 13 features are considered in this class. Details are given in Table 7.

4.6 | Feature extraction

Feature extraction cutoff time was set to 100 seconds, and the RAM has been set to 4 GB. Using too much central processing unit (CPU) time for extracting features reduces their usefulness. In the light of the fact that planners tend to solve problems quickly or not at all,[39] it might be better to select a not-so-good planner than spending too much time to extract all features (and select a better planner).

TABLE 4 General SAS+ features derived from domain transition graphs (DTGs)

Name	Type	Description
`total Edges`	Double	Number of edges of all DTGs
`total Weight`	Double	Total weight of the edges of all DTGs
`edVa Ratio DTG`	Double	Ratio between edges and variables
`weEd Ratio DTG`	Double	Ratio between weight and edges
`weVa Ratio DTG`	Double	Ratio between weight and variables
`input Edge DTG`	Double	Maximum, average, and standard deviation of the input edges of the DTG
`output Edge`	Double	Maximum, average, and standard deviation of the output edges of the DTG
`input Weight`	Double	Maximum, average, and standard deviation of the weight of the input edges of the DTG
`output Weight`	Double	Maximum, average, and standard deviation of the weight of the output edges of the DTG

TABLE 5 Temporal SAS+ features—part I

Name	Type	Description
`Durative actions`	Numeric	Number of durative actions identified by TFD
`Action counter`	Numeric	Number of different actions from the SAS+ translation
`Function symbols`	Numeric	Number of symbols identified by TFD
`Generated rules`	Numeric	Number of rules generated by TFD in the translation process
`Final queue`	Numeric	Number of the elements that appear in the planning queue
`Translator variables`	Numeric	Number of temporal variables identified by TFD
`Translator derived variables`	Numeric	Number of temporal derived variables identified by TFD
`Translator facts`	Numeric	Number of temporal facts identified by TFD
`Mutex key`	Numeric	Number of mutexes
`Strips to sas`	Numeric	Number of auxiliary variables used in a temporal SAS+ encoding
`Ranges`	Numeric	Number of different numeric variables with different ranges
`Goal list`	Numeric	Number of elements in the goal state of the temporal task
`Task init`	Numeric	Number of elements in the initial state of the temporal task
`Translator durative act`	Numeric	Number of actions in the preprocess phase
`Translator axiom`	Numeric	Number of axioms in the translation phase
`Translator num axioms`	Numeric	Number of simplified axioms in the translation phase
`Translator num axioms by layer`	Numeric	Number of actions per level
`Translator max num layer`	Numeric	Maximum number of layers

Abbreviation: TFD, Temporal Fast Downward.

TABLE 6 Temporal SAS+ features—part II

Name	Type	Description
`Translator num axiom map`	Numeric	Number of axioms that appear throughout the process
`Translator const num axioms`	Numeric	Minimum number of necessary axioms
`Translator reachable`	Numeric	Number of variables that are reachable in the initial state
`Translator mutex group`	Numeric	Number of mutex groups
`Translation key`	Numeric	Auxiliary value of TFD
`Avg level`	Numeric	Average number of levels
`Std level`	Numeric	Standard deviation of the number of levels
`Global num type start`	Numeric	Number of transitions that are labeled at at_start
`Global num type end`	Numeric	Number of transitions that are labeled at at_end
`Global min level`	Numeric	Minimum number of levels in DTGs
`Global max level`	Numeric	Maximum number of levels in DTGs
`Global total level`	Numeric	Total number of levels in DTGs
`Init`	Integer	Number of predicates that appear in the initial state
`Goals`	Integer	Number of predicates that appear in the goal
`Function administrator`	Integer	Auxiliary number of functions in TFD
`Final queue length`	Integer	Size of the queue in the translation process
`Translator operators`	Integer	Number of operators that appear in the translation process
`Necessary operators`	Integer	Number of operators at the preprocessing phase
`Uncovered facts`	Integer	Number of facts included in the preprocessing phase
`Necessary variables`	Integer	Number of variables that appear in the translation process
`Relation axioms`	Integer	Number of axioms that are relational in TFD
`Functional axioms`	Integer	Number of axioms that are functional in TFD
`True axioms`	Integer	Number of axioms that are true in the translation process

Abbreviations: DTGs, domain transition graphs; TFD, Temporal Fast Downward.

Table 8 shows the average and maximum time required for extracting the different sets of features as well as the percentage of problems in which the extraction was successfully completed (ie, within the time and memory bounds). Whereas propositional PDDL feature extraction requires negligible time, temporal PDDL feature extraction requires around 10 seconds. On the other hand, extracting SAS+ features is usually more expensive in tens of seconds. SAT size feature extraction, on the other hand, takes about 1-2 seconds. SAS+ features as well as SAT size features have not been computed, due to timeout or running out of memory, in approximately 20% of the problems considered in our experimental analysis.

5 | EXPERIMENTAL SETTINGS

Our experimental analysis aims at assessing how classification and regression approaches can cope with the problem of algorithm selection for temporal planning problems.

- Classification approaches classify planning problems into a single category, according to the fact whether the planner will solve the problem or not.
- Regression techniques model each planner's runtime.

TABLE 7 SAT size features extracted by considering the SAT-based encoding exploited by ITSAT

Name	Type	Description
`Ratio relevant actions`	Double	Ratio between the number of final and initial actions
`Num action`	Integer	Number of final actions
`Num propositions`	Integer	Number of all propositions
`Num relevant actions`	Integer	Number of the final instantiated actions
`Num relevant propositions`	Integer	Number of propositions that are included in the relevant actions
`Variables end`	Integer	Created variables in the SAT formulation
`Propositions end`	Double	Number of propositions that are included in the instantiated actions
`Actions end`	Integer	Instantiated actions in the SAT formulation after simplification
`Total Mutex clauses`	Double	Number of mutex clauses
`Ratio end`	Integer	Ratio of the number of variables to the number of clauses
`Event clauses`	Double	Number of clauses in the original formula
`TClauses`	Integer	Number of simplification clauses
`Number Files`	Integer	Number of temporal files needed by ITSAT

Abbreviation: SAT, Satisfiability.

TABLE 8 Average and maximum central processing unit time needed to extract features, the number of features per group (No.), and the percentage of successful feature extraction (Succ)

		Average	Maximum	No.	Succ %
PDDL	Prop	0.01	0.15	8	100
	Temp	5.06	10.00	28	100
SAT size		0.89	2.00	13	80
SAS+		28.89	50.00	90	80
Total		33.96	60.15	139	–

Abbreviations: PDDL, Planning Domain Definition Language; SAT, Satisfiability.

When dealing with EPMs, a number of decisions have to be taken. First, it is pivotal to select a number of suitable planners; such planners will be used for evaluating the predicting capabilities of classification and regression approaches. Second, benchmarks have to be gathered for both training and testing purposes. Third, features should be extracted on which EPMs perform predictions. Finally, appropriate metrics have to be considered for measuring the planners' performance. In the next sections, we describe the decisions taken on the mentioned regards. The experimental framework exploited in this analysis is shown in Figure 2. It includes the relevant input and the two main steps, namely, training and testing.

Planners and feature extractors were run on a cluster with Intel XEON 2.93-GHz nodes with 8 GB of RAM each, using Linux Ubuntu 12.04 LTS. Planners had a cutoff time of 1800 seconds and a maximum of 4 GB of RAM, whereas feature extractors had a cutoff time of 100 seconds and a maximum of 4 GB of RAM.

5.1 | Planners

Planning systems that can deal with temporal problems are not as numerous as classical planning solvers. Initially, 12 planners were considered; however, those with very poor performance on

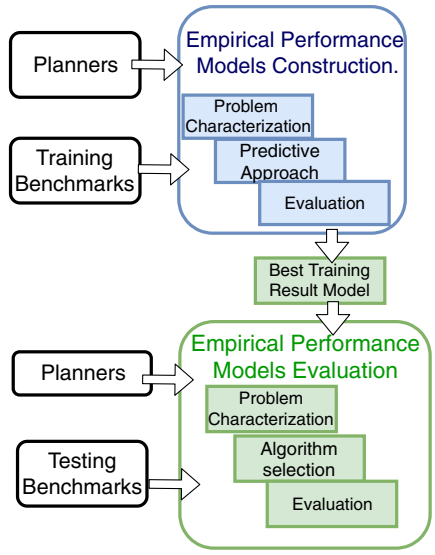

FIGURE 2 The architecture of the proposed system [Color figure can be viewed at wileyonlinelibrary.com]

training problems (in terms of coverage) were removed. Models for planners with poor coverage on training instances result in a trivial "always negative" EPM, which does always predict that the planner will not solve a given problem, which is usually built (and is accurate). Such an EPM just never considers these planners in the algorithm selection process. Hence, for our experiments, we have considered eight state-of-the-art temporal planners that accommodate various techniques, as follows.

- **LPG-td**[40] exploits stochastic local search in the space of planning graphs and is able to generate solutions of increasingly good quality. For the sake of this analysis, as we are interested in runtime performance, LPG was stopped after the first solution was found, and seed was fixed.
- **POPF2**[41] is a Forward-Chaining Partial Order Planner that exploits forward-chaining search, expanding nodes according to a partial order rather than the conventional total order.
- **Yahsp2** and **Yahsp2-MT**[42] compute look-ahead plans from delete-relaxed plans and use them in the state-space heuristic search.
- **TFD**[36] is based on the Fast Downward planning system and uses an adaptation of the context-enhanced additive heuristic to guide the search in the temporal state space induced by the given planning problem.
- **ITSAT**[37] translates the problem into a sequence of SAT instances, corresponding to different time horizons considered for solving the problem instance.
- **Yahsp3** and **Yahsp3-MT**[43] are the latest versions of the Yahsp planner, which took part in IPC 2014.

Two different versions (four planning engines) of Yahsp have been included, because it performed well in both IPC 2011 and IPC 2014 (Yashp 3-MT won the temporal track of IPC 2014). Due to the fact that an EPM is built for each planner, in order to predict its performance, we do not expect the selection of four different engines based on the same planner having an impact on the experimental evaluation. Instead, it may shed some light on the progress of the field.

TABLE 9 Training domains categorized according to the planning competition in which they were used

Training Domains	
IPC 2008	**IPC 2011**
Crewplanning	Crewplanning
Elevators-N	Elevators
Elevators	Floortile
Modeltrain	MatchCellar
Openstacks-adl	Openstacks
Openstacks-N	Parcprinter
Openstacks-N ADL	Parking
Openstacks	
Parcprinter	Pegsol
Pegsol	Sokoban
Sokoban	Storage
Transport	Temporal Machine Shop
Woodworking	Turn and Open

Abbreviation: IPC, International Planning Competition.

5.2 | Benchmarking

We considered temporal planning problems gathered from the temporal tracks of the last editions of the IPC,[†] namely, 2002, 2004, 2006, 2008, 2011, and 2014. Problems not solved by at least one planner were not included in the training set. EPMs have been trained on benchmarks from IPCs 2008 and 2011: in total, 25 domain models and 630 problems have been considered, as seen in Table 9. In IPC 2008, there are two domains having *Architecture Description Language (ADL) features (in blue)* and three domains with numeric fluents (in green). IPC 2011 does not include any domain with numeric fluents or *ADL features*.

For testing purposes, we designed three different testing sets that are described in Table 10.

- The *IPC 2014* testing set, which includes all the benchmarks from the temporal track of IPC 2014.
- The *Known* testing set, which considers domains that are also included in the training set. Testing problem instances are different from training ones.
- The *Unknown* testing set, which includes domains that are not present in the training set.

The IPC 2014 set aims at providing a general overview of the performance of the trained models. The other two testing sets have been designed for evaluating the generalization ability of trained models on either completely new domain models (Unknown) or new problem instances from already seen domain models (Known).

Whenever possible, we considered different encodings of the same domain. Specifically, we considered domain models encoded using Standford Research Institute Problem Solver (STRIPS) features only, including numerical constraints, and exploiting ADL features.

[†]http://icaps-conference.org/index.php/main/competitions

TABLE 10 The considered domains divided into the Unknown, Known, and International Planning Competition (IPC) 2014 sets. Planning Domain Definition Language requirements per each considered domain

No.	IPC	Domain	IPC 2014	Unknown	Known	ADL	Numeric	Durative Actions
1	2002	Depots-simple-T		✓				✓
2		Depots-T		✓				✓
3		DriverLog-time		✓			✓	✓
4		DriverLog-simpleTime		✓				✓
5		ZenoTravel-simpleTime		✓				✓
6		ZenoTravel-time		✓			✓	✓
7		satellite		✓				✓
8		Rovers-mt		✓			✓	
9		Rovers-time		✓			✓	✓
10		UMLS-flaw		✓			✓	✓
11		UMLS-fluents		✓			✓	✓
12	2004	Airport-adl		✓		✓		✓
13		Airport-str		✓				✓
14		Pipesworld-mt		✓			✓	
15		Pipesworld-mtc		✓			✓	✓
17		Satellite-adl		✓		✓		
18		NOTANKAGE		✓			✓	✓
19		TANKAGE		✓			✓	✓
20	2006	Temporal Machine Shop (TMS)			✓			
21		Openstacks-strips			✓	✓		
22		Openstacks-time			✓	✓	✓	✓
23		Openstacks-mt			✓		✓	✓
24		Openstacks			✓		✓	✓
25		Storage-time			✓			✓
26		Storage			✓			
27		Trucks-adl		✓	✓	✓	✓	✓
28		Trucks-time		✓			✓	✓
29		Rovers		✓			✓	✓
30		PipesWorld		✓			✓	✓
31	2014	DriverLog	✓					✓
32		Floortile	✓					✓
33		Map-Analyzer	✓					✓
34		MatchCellar	✓					✓
35		Parking	✓					✓
36		RTAM	✓					✓
37		Satellite	✓					✓
38		Storage	✓					✓
39		Temporal Machine Shop (TMS)	✓					✓
40		Turn and Open	✓					✓
Total			10 (4/6)	23	7			

Abbreviation: ADL, Architecture Description Language.

Having specified the training and testing benchmarks, in this analysis, we compare the performance of EPMs using

- a standard 10-fold cross-validation approach on a uniform random permutation of the training instances and
- the three different testing sets: IPC 2014, Unknown, and Known.

TABLE 11 An overview of the sets of features considered in our experimental analysis. The check mark indicates that the group of features (column) includes the corresponding set (row). The **Sel** set does not include all the features of involved groups

Group	PDDL	SAS+	Temporal (T)	Classical(nT)	Selection (Sel)	All
Propositional PDDL	✓			✓	✓	8
Temporal PDDL	✓		✓		✓	28
General SAS+		✓		✓	✓	49
Temporal SAS+		✓	✓		✓	30
TFD			✓			11
SAT size				✓		13
Total	49	90	71	68	11	139

Abbreviations: PDDL, Planning Domain Definition Language; SAS, Satisfiability; TFD, Temporal Fast Downward.

5.3 | Groups of features

In order to evaluate how different features affect the ability to predict planners' performance, we consider different groups of features. Features have been grouped according to either the encoding they refer to or their temporal specificity and are summarized in Table 11. **All** indicates the whole set of computed features (139). **PDDL** refers to the 49 features, including propositional PDDL, temporal PDDL, and problem size. **SAS+** considers the 90 features that are extracted by considering SAS+ encoding only. **nT** (non-Temporal) refers to the 68 features that are typical of classical planning. Features are gathered from propositional PDDL and general SAS+ sets. The **T** (Temporal) set considers the 71 features that are extracted by considering temporal PDDL and temporal SAS+ encoding. We also consider the **Sel** set, which includes a small number of relevant features that have been automatically selected. Feature selection was done by looking at a J48 decision tree,[44] which is built for predicting the solvability of the training instances, by considering planners as input information. Given the model, we select the features used in nodes placed in the top fifth of the decision tree. They are believed to be important since, according to the J48 algorithm, they provide the best information gain.[44] This can be seen as a supervised method for feature selection. Considering top nodes avoids potential overfitting, as it may arise in lower-level leaves of the tree that are used for classifying a very few instances. The accuracy of the EPM generated by the J48 algorithm is good, approximately 91%. Therefore, we believe that information extracted from such a model is relevant. The resulting automatically generated set of features, **Sel**, includes 11 features: one from the propositional PDDL set, seven from temporal PDDL, two from general SAS+, and one from temporal SAS+. In particular, the selected features are as follows: the number of predicates included in the domain definition (propositional PDDL); the number of durative actions, the number of actions that use numeric fluents for representing their duration, the average arity of these fluents, the minimum number of conditions that have to hold `at_start` of action execution, the maximum number of conditions that have to hold during action execution (`over_all`), and the minimum and maximum numbers of effects that become true after action execution finishes (`at_end`) (temporal PDDL); the maximum number of outgoing edges of the CG, the maximum number of incoming edges in the DTG (general SAS+); and the number of translated durative actions (temporal SAS+). The selection process emphasizes the importance of temporal features (8 out of 11 features are taken from temporal sets); they tend to appear earlier in the J48 decision tree and are thus deemed as being more informative. On the other hand, this distribution of selected features across SAS+ and PDDL sets requires extracting both PDDL and SAS+ sets of features (the latter is more computationally expensive).

6 | EXPERIMENTAL RESULTS

First, we assessed the performance of various classification and regression models (45 different algorithms in total), using the WEKA tool.[45] We considered linear regression, neural networks, Gaussian processes, decision trees, regression methods, clustering, support vector machine, and rule-based techniques.

6.1 | Classification

For exploiting a classification approach, a different predictive model is built per planner. Such a predictive model has to classify the problem instance according to the fact whether the planner will find its solution or not. Rotation Forest[46] performed best among the considered classification approaches on the training instances and is exploited hereinafter. Results are presented in terms of accuracy: it is the number of correct predictions made divided by the total number of predictions made, multiplied by 100 to obtain percentage.

Table 12 shows the results of the trained predictive models on training instances. As expected, the performance on training instances is good, regardless of the considered set of features. Usually, any set of features achieves an accuracy level of approximately 90%. We conjecture that each class includes at least a few informative features and that some of the included domains have a large number of corresponding problem instances. A larger number of problem instances can positively influence the performance of predictive models because, on a limited and generally coherent set of instances from the same domain, a given planner tends to perform uniformly. It is therefore easier, under such circumstances, for a predictive model to predict the planner's behavior.

The two considered classes (solved and unsolved) have been balanced among all the planners on the training instances; the maximum difference is 40%-60%. In order to achieve this class balance, we assessed the initial distribution between classes and, in imbalanced cases, randomly oversampled the minority class. This approach is common practice in machine learning.[47] The exploitation of training sets with very imbalanced classes will lead to the generation of trivial EPMs that classify all the instances as members of the most represented class.

TABLE 12 Accuracy (higher is better) of the classification empirical performance models predicting whether a planner will solve a problem or not on the training instances. Bold indicates the best results (also considering hidden decimals)

	Training Instances					
Planner	**All**	**PDDL**	**SAS+**	**nT**	**T**	**Sel**
LPG	92.6	88.5	88.6	**92.7**	91.9	88.4
POPF2	88.6	87.2	84.9	**88.7**	88.2	87.7
Yahsp2	89.6	91.0	89.1	87.9	89.9	**91.4**
Yahsp2-MT	**95.5**	91.9	89.3	93.9	95.3	89.8
ITSAT	**94.1**	88.2	88.4	93.6	94.1	89.1
TFD	94.1	87.5	84.9	93.5	**94.2**	88.8
Yahsp3	91.0	90.8	89.0	89.7	91.2	**93.1**
Yahsp3-MT	**93.9**	93.4	90.7	92.2	93.8	90.7

Abbreviations: PDDL, Planning Domain Definition Language; POPF2, Forward-Chaining Partial Order Planner; SAS, Satisfiability; TFD, Temporal Fast Downward.

TABLE 13 Accuracy (higher is better) of the classification empirical performance models predicting whether a planner will solve a problem or not on the testing instances. Bold indicates the best results (also considering hidden decimals)

	IPC 2014					
Planner	**All**	**PDDL**	**SAS+**	**nT**	**T**	**Sel**
LPG	76.5	**81.5**	73.0	75.0	74.5	76.0
POPF2	**87.0**	77.5	83.5	86.5	80.5	68.5
Yahsp2	74.5	**76.0**	67.5	57.0	59.5	56.5
Yahsp2-MT	63.5	**80.5**	65.0	72.5	57.0	68.0
ITSAT	**89.0**	88.5	73.0	84.5	88.5	74.5
TFD	67.0	67.0	69.5	**71.0**	67.0	67.0
Yahsp3	60.0	**74.0**	61.5	59.0	57.0	56.0
Yahsp3-MT	75.0	**82.0**	73.0	65.5	57.0	78.0
	Known					
	All	**PDDL**	**SAS+**	**nT**	**T**	**Sel**
LPG	42.5	**81.2**	33.3	31.2	76.3	53.8
POPF2	65.6	72.0	67.20	45.7	**77.4**	51.6
Yahsp2	43.6	75.8	74.19	76.9	**78.0**	78.0
Yahsp2-MT	**80.1**	57.5	76.9	76.3	79.0	79.0
ITSAT	97.3	**100**	76.3	86.0	98.4	92.5
TFD	42.5	39.3	71.5	**75.3**	44.6	41.4
Yahsp3	71.5	**77.4**	65.1	74.7	57.5	77.4
Yahsp3-MT	53.2	78.5	76.9	75.8	**78.5**	78.5
	Unknown					
	All	**PDDL**	**SAS+**	**nT**	**T**	**Sel**
LPG	62.6	48.9	**64.4**	55.4	55.7	56.8
POPF2	59.8	57.0	67.3	**77.1**	42.1	57.1
Yahsp2	48.9	80.3	**84.7**	84.5	73.2	75.7
Yahsp2-MT	75.0	71.4	79.4	**82.2**	74.7	72.0
ITSAT	92.4	86.2	90.0	75.0	89.3	**92.4**
TFD	57.6	53.7	**70.8**	68.0	40.8	30.6
Yahsp3	62.2	73.7	**78.2**	71.9	78.2	55.7
Yahsp3-MT	**86.4**	65.0	78.3	72.7	73.6	57.1

Abbreviations: IPC, International Planning Competition; PDDL, Planning Domain Definition Language; POPF2, Forward-Chaining Partial Order Planner; SAS, Satisfiability; TFD, Temporal Fast Downward.

Summarizing, the results in Table 12 clearly indicate that on training instances, the EPMs are able to identify relevant features and combine them for predicting solvability of problems.

Table 13 shows the performance of classification EPMs on the considered testing sets. The analysis of the results on the IPC 2014 set provides a number of interesting insights: (i) the PDDL set leads, in five out of eight cases, to the best prediction results; (ii) using either a temporal or a nontemporal set of features achieves similar prediction results; (iii) using all the features together, on the other hand, does not guarantee the best performance; (iv) TFD and Yahsp3 behaviors are hard to predict on testing instances; and (v) the set of selected features usually achieves good prediction results, particularly considering that only 11 features are considered

for a domain-independent prediction. We observed that TFD and Yahsp2/3 show a very different behavior on training and testing problems, possibly because of new domains and/or significantly larger instances used in the testing set. TFD translates the PDDL planning problem into SAS+ and then solves the SAS+ problem; the translation phase can be slow and, sometimes, requires a huge amount of memory. On large instances, as those used in the IPC 2014 set, it happens that the translation step fails due to lack of available memory (4 GB); this is clearly hard to predict for an EPM that has been trained on smaller instances, where this issue does rarely arise. Both planners have issues in dealing with problems that need to reason with concurrency in order to be solved. In fact, on the benchmarks of IPC 2014, TFD is not able to solve problems from five domains, whereas Yahsp3 is not able to provide any solution for instances from three domains.

Considering all the features at the same time is not always the best option. We believe this is mainly because of introduced "noise." Our hypothesis is supported by the results achieved using the 11 selected features: they represent a (hopefully) noise-free set of features, and their exploitation achieves results close to those achieved when using the **All** set. The considered sets have some overlap, and this partially explains why, in some cases, they show similar performance.

Table 13 also shows the results achieved by trained EPMs on the Known and Unknown test sets. We observed that on the Known set, performance is usually less accurate than those achieved on the IPC 2014 testing set. We believe this is due to the fact that the domain models are encoded using different sets of PDDL features. In many cases, features introduced in domains that are included in the testing set are not supported by planners. Therefore, predictions are less accurate because, although many features have values that are similar to some instances included in the training set, the final outcome is completely different. This is also reflected in the very different performance of the considered sets of features. The Known set is significantly smaller than the other sets: from this perspective, mistakes have a much larger impact on the overall evaluation.

ITSAT is the only planner that has very predictable performance on the Known testing set. On the contrary, LPG has quite unpredictable performance on the Known set: for instance, the use of SAS+ and nT feature sets leads to around 30% accuracy. This may be due to the intrinsic randomness of the planning approach exploited by LPG: it is based on stochastic local search. On the other hand, EPMs generated for predicting the performance of Yahsp2, Yahsp3, and TFD tend to have similar accuracy on all the considered testing sets.

To investigate how the importance of the features varies between training and testing problems, we applied our selection process on the EPMs built by considering only testing instances. Similarly to the selection process done on training problems, 11 features are selected. One of them is exactly the same: the minimum number of effects that become true when action execution finishes (`at_end`) (PDDL). The other six features selected according to the testing instances are strongly related to those extracted on training problems, as they consider similar aspects of the problem, but from a slightly different perspective: maximum arity of numeric fluents (PDDL), minimum number of `at_start` conditions (PDDL), minimum duration of an action (PDDL), standard deviation of incoming edges of the DTG (SAS+), number of variables (SAS+), and number of relevant actions (SAS+). Finally, the remaining features are completely different from those included for the EPMs built considering training instances. This is the case of the following features: number of PDDL requirements (PDDL), number of facts in the initial state (PDDL), ratio between the weight and the edges in the CG (SAS+), and ratio between edges and variables of the DTG (SAS+).

Overall, considering also the results achieved by the EPMs exploiting the **Sel** set of features, this analysis confirms their informativeness. It also indicates that the technique we designed

for selecting informative features is reasonably accurate, in the sense that it selects features that generalize on different benchmarks.

6.2 | Regression

Regression EPMs predict the runtime a planner needs to solve a given problem instance. The runtimes of considered planners on selected benchmarks vary between 0 and 1800 CPU seconds. Given the large variations in CPU times, we trained our regression models to predict the log runtime rather than absolute time: this has been demonstrated effective in similar circumstances.[6] To predict when a planner will not be able to solve a given problem instance, we assigned a default value of 2000 CPU-time seconds to unsolved instances. In this way, any predicted value between 1800 and 2000 CPU-time seconds will be considered as that the EPM identified that the given instance will not be solved.

Performance is measured in terms of the root-mean-square error (RMSE). Experimentally, we observed that the decision tables algorithm[48] generates, on average, the most accurate predictive models, and we will exploit this approach for the remainder of this experimental analysis.

Table 14 shows the results, in terms of RMSE, of the best regression models with 10-fold cross validation on a uniform random permutation of the 630 training instances. First, we noticed that predicting algorithm runtime is challenging, according to the RMSE values. On the other hand, it is well known that RMSE is sensitive to occasional large errors (eg, predicting an instance as unsolvable although it can be solved quickly); thus, actual predictions can be better, on average.

Table 15 shows the RMSE results achieved by the regression predictive models on the three considered testing sets. Differently from the results of classification EPMs, regression models are providing the most accurate predictions on the Known test set. On the other test sets, regression models tend to perform similarly. However, as for the classification models, ITSAT's performance is the easiest to predict. On the Known testing set, the RMSE goes below 1 because ITSAT does not solve the vast majority of the problems, and therefore, the EPM tends to predict very poor performance.

We noticed that the regression approach shows similar RMSE performance for the TFD planner on training and testing instances. This was not the case for the classification model. On the

TABLE 14 Root-mean-square error (lower is better) of the regression empirical performance models built by using decision tables on training instances. Bold indicates the best performance (also considering hidden decimals)

	Training Instances					
Planner	**All**	**PDDL**	**SAS+**	**nT**	**T**	**Sel**
LPG	1.49	1.57	1.84	1.54	**1.48**	1.49
POPF2	2.12	2.27	2.53	2.23	2.11	**2.05**
Yahsp2	1.76	1.45	2.07	1.86	1.65	**1.27**
Yahsp2-MT	1.41	1.45	2.25	1.84	1.33	**1.30**
ITSAT	1.45	1.58	1.68	1.41	1.42	**1.38**
TFD	2.18	2.32	2.56	2.19	2.16	**2.02**
Yahsp3	1.61	1.60	2.04	1.81	1.43	**1.41**
Yahsp3-MT	1.42	1.28	1.29	1.55	1.21	**1.17**

Abbreviations: PDDL, Planning Domain Definition Language; POPF2, Forward-Chaining Partial Order Planner; SAS, Satisfiability; TFD, Temporal Fast Downward.

TABLE 15 Root-mean-square error (lower is better) of regression empirical performance models built by using decision tables on testing instances. Bold indicates the best performance (also considering hidden decimals)

	IPC 2014					
Planner	**All**	**PDDL**	**SAS+**	**nT**	**T**	**Sel**
LPG	3.29	3.56	2.61	2.60	3.44	**2.20**
POPF2	2.49	2.43	2.22	2.84	**2.48**	2.76
Yahsp2	2.76	2.55	3.22	2.76	**2.37**	3.63
Yahsp2-MT	**2.83**	3.05	3.36	3.08	2.89	2.86
ITSAT	2.06	2.28	2.54	2.42	2.36	**1.87**
TFD	2.51	2.73	2.87	2.80	2.83	**2.19**
Yahsp3	2.60	3.33	3.23	2.85	2.79	**2.20**
Yahsp3-MT	2.99	2.85	3.12	3.27	2.65	**2.64**
	Known					
	All	**PDDL**	**SAS+**	**nT**	**T**	**Sel**
LPG	3.02	**3.02**	3.54	3.53	**3.02**	**3.02**
POPF2	2.86	2.46	2.53	2.67	**2.43**	2.46
Yahsp2	1.73	**1.57**	2.03	2.06	**1.57**	1.62
Yahsp2-MT	3.18	3.16	2.12	2.13	3.16	**1.54**
ITSAT	1.61	1.61	**0.87**	0.99	2.15	2.29
TFD	3.47	3.47	2.98	**2.94**	3.45	3.09
Yahsp3	1.46	**1.51**	2.12	2.12	1.47	1.57
Yahsp3-MT	2.42	1.99	1.95	1.97	1.68	**1.49**
	Unknown					
	All	**PDDL**	**SAS+**	**nT**	**T**	**Sel**
LPG	2.86	2.86	**2.13**	2.86	2.86	2.31
POPF2	2.26	2.35	2.15	**2.06**	2.35	2.36
Yahsp2	2.18	**2.15**	2.39	2.37	**2.15**	**2.15**
Yahsp2-MT	2.80	2.78	2.40	2.35	2.78	**2.02**
ITSAT	2.70	2.70	2.45	2.56	2.84	**2.85**
TFD	3.86	3.86	2.81	2.78	3.86	**2.80**
Yahsp3	2.15	2.15	2.46	2.44	**2.12**	2.32
Yahsp3-MT	2.13	2.03	2.16	2.20	2.02	**1.88**

Abbreviations: IPC, International Planning Competition; PDDL, Planning Domain Definition Language; POPF2, Forward-Chaining Partial Order Planner; SAS, Satisfiability; TFD, Temporal Fast Downward.

other hand, we observed that Yahsp-based systems show a very different behavior on the training and testing instances as in the classification case. In particular, the behavior of the MT versions is the most challenging to predict. Since Yahsp-MT exploits a multithreaded approach, it is possibly more sensitive to small changes of the execution environment (eg, operative system calls and input/output delays). This has a limited impact on the ability of the planner in solving instances but makes the actual runtime harder to predict. A similar explanation can be provided for the high error in the LPG predictions: LPG exploits a randomized search algorithm that, in the presence of domain models that are similar to those used in training instances, leads the predictive model to make inaccurate estimations.

With regard to the different classes of features, using the **Sel** set often results in the best regression EPMs since, very likely, this set is noise free and very informative. We also observed that the

features from the temporal set are very informative and achieve prediction performance that is usually very close to the best.

6.3 | Exploiting EPMs for algorithm selection

After evaluating the prediction performance of the classification and regression EPMs, we are in a position to exploit them for performing online algorithm selection. In particular, we tested the capability of EPMs as mechanisms for selecting the most promising planner to exploit on a given (and previously unseen) testing instance. A single planner is selected for solving each planning instance, and a cutoff time of 1800 seconds is allocated to the selected planner.

Classification EPMs are able to predict whether a given planner will solve a given problem instance or not. Therefore, they can be used to select planners in order to maximize coverage, ie, the number of solved instances. As a different classification EPM is generated for each planning engine, the selection is performed as follows. Among all the planners that are predicted to solve a given problem, the selected planner corresponds to the EPM that showed the best accuracy on training instances.

Regression EPMs predict, for each planner, the runtime needed to solve a given planning instance. The planner selected is the one predicted to be the fastest.

We compare the approaches by considering the IPC runtime score and the coverage. The IPC score is defined as in the Agile track of IPC 2014. For a planner C and a problem p, Score(C, p) is 0 if p is unsolved, and $1/(1 + \log_{10}(T_p(C)/T_p^*))$, where $T_p(C)$ is the CPU time needed by planner C to solve problem p and T_p^* is the CPU time needed by the best considered planner, otherwise. The IPC score on a set of problems is given by the sum of the scores achieved on each considered problem.

In terms of basic planners' performance on the IPC 2014 testing set, Figure 3 shows the corresponding number of solved problems, with regard to CPU time. Most of the planners are usually either solving instances quickly or not at all. Exceptions are ITSAT and TFD, which are able to solve a few instances in about 600 seconds and a few more instances in about 1400 seconds.

Table 16 shows the results in terms of the number of solved problems and IPC runtime score achieved on the IPC 2014 test set by the classification and regression EPMs using different sets of features. In this analysis, we ignore the CPU time needed for extracting features, as the main goal of this section is to evaluate the ability of the generated EPMs to effectively select a suitable planner for a given problem.

We focus on four groups of features: **All**, **Sel**, **Temporal**, and **non-Temporal**. For algorithm selection, we are particularly interested in assessing the usefulness of temporal-specific features and in evaluating the effectiveness of the small set of selected features.

For the sake of comparison, Table 16 includes the performance of the virtual best solver (VBS), which represents an Oracle that selects always the best possible planner for solving the specific problem, the two best basic solvers accordingly to (C)overage (LPG) and IPC (S)core (Yahsp2), and a static portfolio (B4P), which includes the best four planners according to coverage performance on testing instances: LPG, Yashp2, Yashp3, and TFD. The solvers are ordered according to their coverage (descending order), and each planner runs for one-fourth of the cutoff time (ie, 450 seconds per planner). Considering these additional systems—VBS, B4P, and the best basic solvers—provides a better and more complete understanding of the performance of algorithm selection across the EPMs.

Both classification and regression EPMs achieve better coverage results than the best basic solver (+11% and +32.5%, respectively). The performance achieved using the regression EPMs is very close to the performance of the VBS and better than that of the B4P.

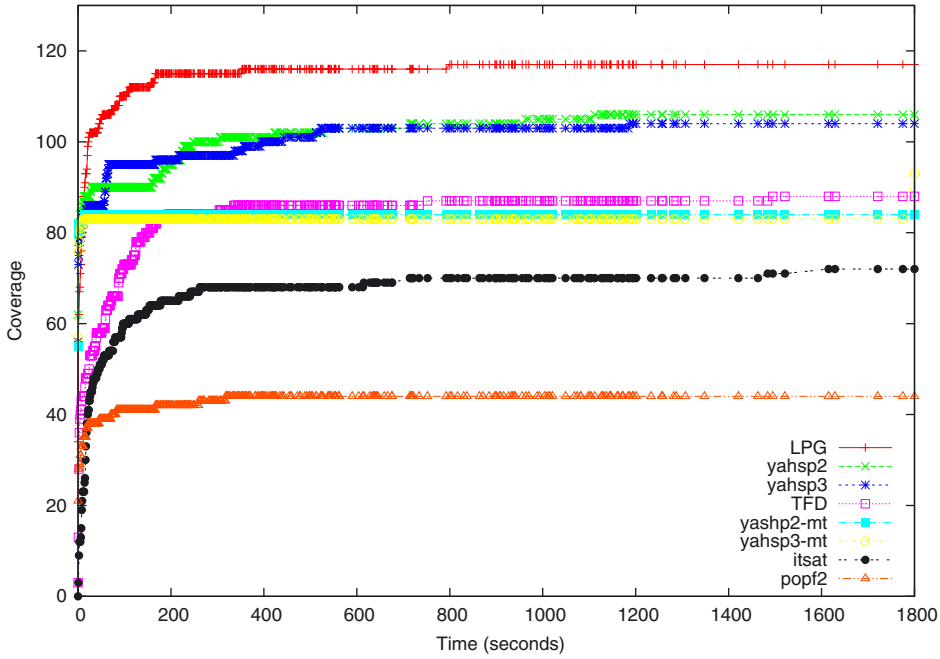

FIGURE 3 The number of solved instances over time of the considered planners on the benchmarks from the International Planning Competition 2014 temporal track. TFD, Temporal Fast Downward [Color figure can be viewed at wileyonlinelibrary.com]

It is useful to remind that the B4P has been configured by considering the performance of planners on testing instances, whereas both regression and classification EPMs have been trained on a different set of instances. From this perspective, the proposed EPMs demonstrate the ability to generalize, since they provide useful prediction for performing algorithm selection on unseen instances, although we observed that the regression EPMs outperform the classification EPMs, both in terms of coverage and IPC score. This is due to the fact that the classification EPMs do not estimate the performance difference between solvers; hence, an error in the prediction might result in selecting a planner that will not solve the problem. The regression EPMs consider planners' runtime. Therefore, a mistakenly selected planner usually needs a longer execution runtime, whereas a planner with extremely poor performance is very rarely selected.

With regard to the considered sets of features, we noticed a very different behavior of the classification and regression EPMs. Classification achieves the best coverage performance when using the selected set of 11 features; the IPC score on that set is close to the best one, which is achieved by using Temporal features. On the other hand, the **Sel** set is not the most informative for algorithm selection through regression; using the whole set of features—or even the set including only temporal/nontemporal features—achieves better performance.

In Table 16, domains are listed according to the *difficulty* of their instances. In this context, the smaller is the number of planners that can solve all the problems, the more difficult is the domain. According to this intuitive definition, the less difficult (easier) domain is Parking, since six planners solve all the problems, and all the considered planners solve at least six instances. The two more difficult domains are TMS (Temporal Machine Shop), because only one planner is able to solve all its benchmark problems, and TurnAndOpen, where three planners solve about 10 problems each. We conjecture that the difficulty of domains plays a pivotal role in algorithm

TABLE 16 Coverage and total International Planning Competition (IPC) score of the regression and classification empirical performance models exploited for algorithm selection, of the best basic solver according to coverage (Best-C), of the best basic solver according to the IPC score (S-Best), of the virtual best solver (VBS), and of a static portfolio including four planners (B4P). The rows in gray indicate the domains that are not included in the training set. Bold indicates the best performance

	Classification				Regression				Best		VBS	B4P
Domain	**All**	**Sel**	**nT**	**T**	**All**	**Sel**	**nT**	**T**	**C**	**S**		
TMS	18	18	16	18	18	18	18	18	0	0	18	0
TurnAndOpen	12	12	14	15	17	17	17	17	0	0	17	15
Storage	17	17	17	17	17	17	17	17	17	9	17	17
DriverLog	7	2	6	0	13	0	13	13	13	9	13	12
Floortile	20	20	20	20	20	20	20	20	20	8	20	20
MatchCellar	19	20	20	20	20	20	20	20	0	0	20	20
MapAnalyzer	10	14	9	10	7	7	7	7	7	20	20	20
RTAM	0	6	0	3	20	20	20	20	20	20	20	20
Satellite	12	3	6	2	20	20	20	20	20	20	20	20
Parking	14	20	20	20	20	20	20	20	20	20	20	20
Coverage	129	132	128	125	**172**	159	**172**	**172**	117	106	185	164
IPC score	91.8	102.4	95.1	105.8	**129.3**	126.6	**129.3**	**129.3**	62.1	86.2	185	72.5

Abbreviation: TMS, Temporal Machine Shop.

selection. If a difficult domain is included in the training set, it is easier for the EPM to correctly identify the planner(s) to exploit on the corresponding testing instances. On the other hand, if the domain is not considered in the training set, the capability of the EPM-based approach in selecting the good planner depends only on the informativeness of features and generalization.

Figure 4 provides an overview of the empirical difficulty of the testing domains used in IPC 2014, from both the planning and instances perspectives. The red line (solved problems) represents the proportion of problems solved per domain. A value of 1 indicates that all the planners are able to solve all the testing problems; on the contrary, the value of 0 means that no planner can solve any of the testing problems. Similarly, the green line (planners) reports the planners' perspective, as the proportion of planners that can solve all the problems of a domain. Figure 4 clearly shows that out of the considered domains, four are extremely difficult for the state-of-the-art domain-independent planners. The difficulty of TMS and TurnAndOpen derives from the fact that their problems need actions to be executed concurrently in order to be solved.

Table 17 shows the results in terms of the number of solved problems and IPC runtime score of the considered classification and regression EPMs, using different sets of features, on the Unknown and Known testing sets. On these testing sets, the best basic solver according to either coverage or runtime is LPG. LPG provides better coverage results than the proposed classification and regression–based algorithm selection approaches. This is true also for the B4P static portfolio that, in fact, includes LPG as well. The best basic planner and the static portfolio are selected (configured) according to the performance of considered planners on the testing instances; hence, they are exploiting information that is not available to the algorithm selection approaches and that is not available before having the instances solved. Algorithm selection approaches rely on a single (selected) planner for generating a solution for a given planning problem; instead, the B4P can fully exploit the available CPU time for running four planners for a considerable amount of time (LPG, Yashp2, Yashp3, and TFD).

Algorithm selection techniques aim to select planners that solve the given problem instances in minimum time. In the case of regression techniques, predicted-to-be-fastest planners are

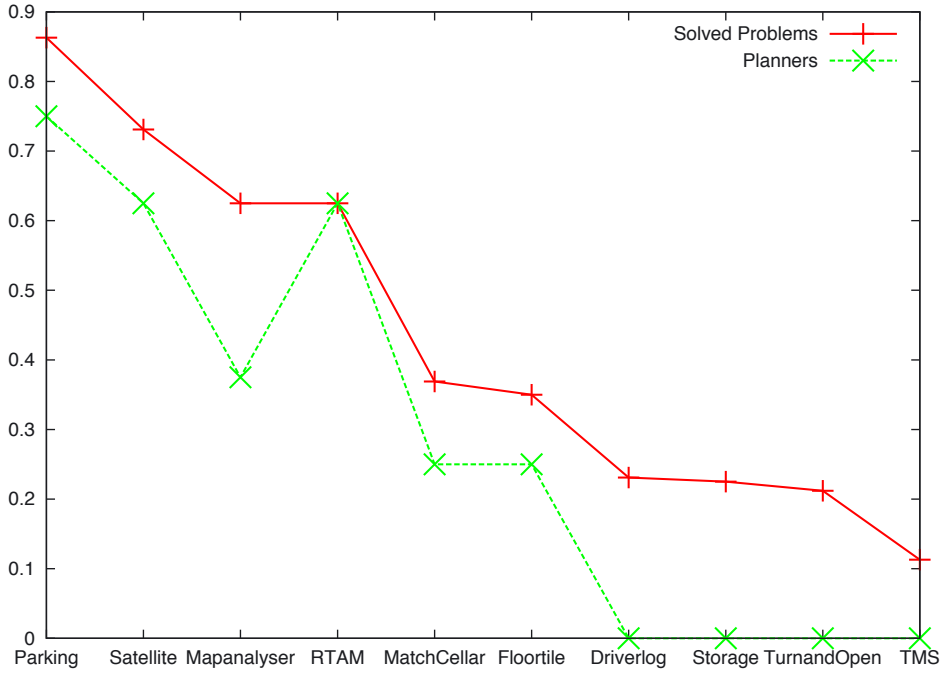

FIGURE 4 The red line (Solved Problems) is the proportion of the problems solved by all the planners. The green line (Planners) is the proportion of the planners that solved all the problems in the particular domain [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 17 Coverage and total International Planning Competition (IPC) score of the regression and classification empirical performance models exploited for algorithm selection, of the best basic solver according to coverage (Best), of the virtual best solver (VBS), and of a static portfolio configured on the testing problems (B4P). Bold indicates the best performance

	Unknown Test Set										
		Classification				**Regression**			**Best**	**VBS**	**B4P**
	All	**Sel**	**nT**	**T**	**All**	**Sel**	**nT**	**T**			
Total	238	167	260	229	309	309	301	309	360	437	**404**
IPC score	175.1	177.9	183.0	168.4	**277.0**	**277.0**	269.2	**277.0**	242.6	437	249.6
	Known Test Set										
		Classification				**Regression**			**Best**	**VBS**	**B4P**
	All	**Sel**	**nT**	**T**	**All**	**Sel**	**nT**	**T**			
Coverage	79	78	71	78	115	111	115	114	143	162	**153**
IPC score	50.0	57.3	64.2	54.8	**113.1**	110.2	110.2	108.6	88.7	162.0	90.0

selected in order to solve the given problem instance; classification-based selection instead tries to identify a planner that will solve the problem regardless of the expected runtime. However, as observed in our experiments, the classification-based approaches underperform the regression approaches.

On the Unknown testing set, we observed that the algorithm selection approaches are struggling with domains in which very few planners are able to solve some instances. This is the case, for example, of the TMS domain: the regression approaches tend to select LPG, which is not able to solve any problem. On such a test set, we noticed that using—instead of a portfolio—a single

planner that shows the best coverage on training instances does not necessarily lead to the best results.

Analyzing the importance of each feature set, we made several observations. The **Sel** set achieves good performance in the classification approach (see Table 16). The only exception can be observed on the Unknown test set: in that case, despite a remarkably high IPC score, the number of solved instances is significantly lower than that achieved when exploiting different sets of features (see Table 17). On the Unknown testing set, the use of non-Temporal features leads to the best performance of the classification EPM approaches (see Table 17). Temporal features, on the other hand, are useful for the regression approaches on the Unknown test sets (see Table 17).

The results shown in Table 17 confirm that the regression EPMs are able to effectively select planners for solving previously unseen instances and show that the very small set of selected features (**Sel**) is a valuable source of information for performing algorithm selection on either previously seen or previously unseen domains and problem instances.

6.4 | Discussion

The algorithm selection approaches presented in the previous section exploit EPMs for selecting a single planner for solving a given planning problem. In this section, we shed some light on the selected planners.

As Table 16 shows, on the four "unseen" domains in the IPC 2014 set (highlighted in gray in the Table), the regression approaches tend to provide better prediction performance on average; hence, they are able to better generalize on previously unseen domains. On the other hand, the classification approaches are unable to select a good planner for the Road Traffic Accident Management (RTAM) domain but are able to identify a suitable planner for the MapAnalyzer domain. Table 18 shows the planners selected by the particular EPMs using the different sets of features. The classification approaches usually exploit more different planners per domain. In every domain except Floortile, the regression approaches select one single planner per set of features (in MatchCellar and DriverLog, different planners were selected while considering a different set of features). This, in combination with results shown in Table 16, supports the observation that a single planner usually performs well on problems from the same domain. However, we conjecture that this is due to the fact that benchmarks for IPCs are usually selected from a homogeneous distribution and are generated using a single problem generator. This can lead to structurally similar problem instances, on which a single planner can excel.

By analyzing the results shown in Table 18, we can derive that the difference in performance between the regression EPMs using the selected set of features, and the other sets, mainly arises in the DriverLog domain. In that domain, TFD does not solve any problem; thus, selecting it has a detrimental effect on performance. The winner of the IPC 2014 temporal track—Yahsp3-MT—is never selected by the regression EPMs and is selected only in one domain by the classification EPMs. Similarly, the previous version of that planner is rarely used. This is possibly due to the fact that these planners show impressive performance on a very limited number of domains, particularly RTAM and MapAnalyzer, which are not included in the training set. We also noticed the remarkable performance of the LPG planner; although it has been developed more than a decade ago, it is competitive with the current state of the art of temporal planning. Finally, Table 19 summarizes the number of times that each planner was selected by the considered EPMs.

TABLE 18 Planners selected by the classification or regression empirical performance models, with different sets of features on the International Planning Competition 2014 benchmarks

		Classification				Regression			
		All	Sel	nT	T	All	sel	nT	T
DriverLog	LPG	0	0	2	0	20	0	20	20
	POPF2	0	3	0	1	0	0	0	0
	TFD	3	14	3	19	0	20	0	0
	Y2	17	3	13	0	0	0	0	0
	ITSAT	0	0	2	0	0	0	0	0
Floor	ITSAT	10	0	20	20	15	20	15	20
	LPG	10	20	0	0	5	0	5	0
Map	LPG	0	0	0	0	20	20	20	20
	POPF2	5	0	0	0	0	0	0	0
	TFD	15	20	14	14	0	0	0	0
	ITSAT	0	0	6	6	0	0	0	0
MatchCellar	ITSAT	15	0	9	9	0	0	0	0
	POPF2	0	0	4	4	0	20	0	20
	TFD	5	20	7	7	20	0	20	0
Park.	POPF2	11	0	0	0	0	0	0	0
	Y2	0	20	0	0	0	0	0	0
	Y2-MT	9	0	0	0	20	20	20	20
	Y3-MT	0	0	20	20	0	0	0	0
RTAM	LPG	0	6	0	0	20	20	20	20
	TFD	20	14	17	17	0	0	0	0
	Y3-MT	0	0	3	3	0	0	0	0
Satellite	LPG	0	0	0	0	20	20	20	20
	POPF2	7	20	0	0	0	0	0	0
	TFD	13	0	2	2	0	0	0	0
	ITSAT	0	0	18	18	0	0	0	0
Stor.	LPG	20	20	20	20	20	20	20	20
TMS	ITSAT	20	20	20	20	20	20	20	20
T&O	ITSAT	3	0	0	0	0	0	0	0
	POPF2	6	11	2	2	0	0	0	0
	TFD	11	9	18	18	20	20	20	20

Abbreviations: POPF2, Forward-Chaining Partial Order Planner; T&O, TurnAndOpen; TFD, Temporal Fast Downward; TMS, Temporal Machine Shop.

TABLE 19 Number of times each planner has been selected by the classification or regression empirical performance models exploiting different sets of features. nT and T refer to non-Temporal and Temporal sets of features, respectively

	Classification				Regression			
	All	Sel	nT	T	All	Sel	nT	T
LPG	30	46	22	20	105	80	100	105
Yahsp2	17	23	13	0	0	0	0	0
Yahsp2-MT	9	0	0	0	20	20	20	20
POPF2	29	34	6	7	0	20	20	0
ITSAT	48	20	57	55	35	40	40	35
TFD	67	77	61	77	40	40	20	40
Yahsp3	0	0	0	0	0	0	0	0
Yashp3-MT	0	0	23	23	0	0	0	0

Abbreviations: POPF2, Forward-Chaining Partial Order Planner; TFD, Temporal Fast Downward.

7 | CONCLUSION

In this paper, we filled the gap between classical and temporal planning in terms of predicting planners' performance. Our work establishes a new extensive set of features that can be extracted from temporal planning problems. In particular, we introduced 71 new temporal-specific features and merged them with "classical" (propositional) features that can be extracted also from temporal problems; in total, 139 planning-specific features have been considered for generating both classification and regression EPMs, which are exploited to select online the planner for solving a given planning task. The large empirical analysis performed in this work (i) demonstrates that the performance of many temporal planners can be accurately predicted using EPMs; (ii) gives insights into the motivations that make planners hard to predict, particularly running out of memory and the concurrency requirements; (iii) provides a valuable and informative set of 11 features that can be used for effectively predicting the performance of temporal planners; (iv) shows that both temporal-specific and nontemporal features are useful for predicting planners' performance; and (v) demonstrates that using EPMs for algorithm selection can significantly improve the current state of the art of temporal planning. Our work also highlights a worrying evidence: in terms of coverage, planners that have been introduced more than a decade ago are able to achieve performance comparable—and often better than—to that of the most recent planning systems. LPG results emphasized this idea; in many cases, it works better than the more recent planners.

Future work includes the extension of the current set of features by considering probing features—information gained by short runs of different solvers—and the integration of different planners' configurations obtained by using algorithm configuration tools, such as the sequential model-based algorithm configuration.[49] Finally, we plan to test the suitability of deep learning approaches for generating EPMs.

ACKNOWLEDGMENTS

This research was funded by the Czech Science Foundation through project no. 18-07252S and by the Operational Programme for Research, Development and Education (OP VVV) through project no. CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics." The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work.

ORCID

Mauro Vallati https://orcid.org/0000-0002-8429-3570
Lukáš Chrpa https://orcid.org/0000-0001-9713-7748

REFERENCES

1. Kotthoff L. Algorithm selection for combinatorial search problems: a survey. *AI Magazine*. 2014;35(3):48-60.
2. Xu L, Hutter F, Hoos HH, Leyton-Brown K. SATzilla: portfolio-based algorithm selection for SAT. *J Artif Intell Res*. 2008;32:565-606.
3. Gebser M, Kaminski R, Kaufmann B, Schaub T, Schneider MT, Ziller S. A portfolio solver for answer set programming: preliminary report. In: Delgrande JP, Faber W, eds. *Logic Programming and Nonmonotonic Reasoning: 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2011:352-357. https://doi.org/10.1007/978-3-642-20895-9_40

4. Fawcett C, Vallati M, Hutter F, Hoffmann J, Hoos HH, Leyton-Brown K. Improved features for runtime prediction of domain-independent planners. In: Proceedings of Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS); 2014; Portsmouth, NH.
5. Cerutti F, Giacomin M, Vallati M. Algorithm selection for preferred extensions enumeration. In: *Computational Models of Argument: Proceedings of COMMA 2014*. Amsterdam, The Netherlands: IOS Press BV; 2014:221-232. *Frontiers in Artificial Intelligence and Applications*; vol. 266.
6. Hutter F, Xu L, Hoos HH, Leyton-Brown K. Algorithm runtime prediction: methods & evaluation. *Artificial Intelligence*. 2014;206:79-111.
7. Rice JR. The algorithm selection problem. *Adv Comput*. 1976;15:65-118.
8. Fink E. How to solve it automatically: selection among problem-solving methods. In: Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS); 1998; Pittsburgh, PA.
9. Howe A, Dahlman E, Hansen C, von Mayrhause A, Scheetz M. Exploiting competitive planner performance. In: Proceedings of the 5th European Conference on Planning: Recent Advances in AI Planning (ECP); 1999; Durham, UK.
10. Roberts M, Howe AE, Wilson B, desJardins M. What makes planners predictable? In: Proceedings of the Eighteenth International Conference on International Conference on Automated Planning and Scheduling (ICAPS); 2008; Sydney, Australia.
11. Roberts M, Howe A. Learning from planner performance. *Artificial Intelligence*. 2009;173(5-6):536-561.
12. Cenamor I, de la Rosa T, Fernández F. Mining IPC-2011 results. In: Proceedings of the 3rd Workshop on the International Planning Competition (ICAPS); 2012; São Paulo, Brazil.
13. Cenamor I, de la Rosa T, Fernández F. Learning predictive models to configure planning portfolios. In: Proceedings of the 4th Workshop on Planning and Learning (ICAPS-PAL); 2013; Rome, Italy.
14. Helmert M. The Fast Downward planning system. *J Artif Intell Res*. 2006;26:191-246.
15. Gerevini A, Saetti A, Vallati M. Exploiting macro-actions and predicting plan length in planning as satisfiability. In: Proceedings of the 12th International Conference on Artificial Intelligence Around Man and Beyond (AI*IA); 2011; Palermo, Italy.
16. Vallati M, Serina I, Saetti A, Gerevini AE. Identifying and exploiting features for effective plan retrieval in case-based planning. In: Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling (ICAPS); 2015; Jerusalem, Israel.
17. Cenamor I, de la Rosa T, Fernández F. The IBaCoP planning system: instance-based configured portfolio. *J Artif Intell Res*. 2016;56:657-691.
18. Hutter F, Hoos HH, Stützle T. Automatic algorithm configuration based on local search. *AAAI*. 2007;7:1152-1157.
19. Gomes CP, Selman B. Algorithm portfolios. *Artificial Intelligence*. 2001;126(1-2):43-62.
20. Leyton-Brown K, Nudelman E, Andrew G, McFadden J, Shoham Y. A portfolio approach to algorithm selection. In: Proceedings of the 18th international joint conference on Artificial intelligence (IJCAI); 2003; Acapulco, Mexico.
21. Malitsky Y. Instance-specific algorithm configuration. *Instance-Specific Algorithm Configuration*. Cham, Switzerland: Springer International Publishing; 2014:15-24.
22. Gebser M, Kaufmann B, Neumann A, Schaub T. *clasp*: a conflict-driven answer set solver. In: *Logic Programming and Nonmonotonic Reasoning: 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2007:260-265.
23. Hoos H, Lindauer MT, Schaub T. claspfolio 2: advances in algorithm selection for answer set programming. *Theory Pract Log Program*. 2014;14(4-5):569-585.
24. Gerevini A, Saetti A, Vallati M. Planning through automatic portfolio configuration: the PbP approach. *J Artif Intell Res*. 2014;50:639-696.
25. Helmert M, Röger G, Karpas E. Fast Downward Stone Soup: a baseline for building planner portfolios. Paper presented at: 3rd Workshop on Planning and Learning (ICAPS); 2011; Freiburg, Germany.
26. Seipp J, Sievers S, Helmert M, Hutter F. Automatic configuration of sequential planning portfolios. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI); 2015; Austin, TX.
27. Cenamor I, de la Rosa T, Fernández F. IBaCoP and IBaCoP2 planner. In: Proceedings of the 8th International Planning Competition; 2014.
28. Vallati M, Chrpa L, Grześ M, McCluskey TL, Roberts M, Sanner S. The 2014 international planning competition: progress and trends. *AI Magazine*. 2015;36(3):90-98.

29. Malitsky Y, Wang D, Karpas E. The AllPACA planner: all planners automatic choice algorithm. Paper presented at: International Planning Competition (IPC); 2014.
30. Rizzini M, Fawcett C, Vallati M, Gerevini AE, Hoos HH. Static and dynamic portfolio methods for optimal planning: an empirical analysis. *Int J Artif Intell Tools*. 2017;26(1):1-27.
31. Ghallab M, Nau D, Traverso P. *Automated Planning: Theory and Practice*. San Francisco, CA: Morgan Kaufmann Publishers; 2004.
32. Fox M, Long D. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J Artif Intell Res*. 2003;20:61-124.
33. Bedrax-Weiss T, McGann C, Bachmann A, Edgington W, Iatauro M. *EUROPA2: User and Contributor Guide*. Technical report. Mountain View, CA: NASA Ames Research Center; 2005.
34. Frank J, Jónsson AK. Constraint-based attribute and interval planning. *Constraints*. 2003;8(4):339-364. https://doi.org/10.1023/A:1025842019552
35. Bäckström C, Nebel B. Complexity results for SAS^+ planning. *Computational Intelligence*. 1995;11:625-656.
36. Eyerich P, Mattmüller R, Röger G. Using the context-enhanced additive heuristic for temporal and numeric planning. In: *Towards Service Robots for Everyday Environments*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2012:49-64.
37. Rankooh MF, Mahjoob A, Ghassem-Sani G. Using satisfiability for non-optimal temporal planning. In: *Logics in Artificial Intelligence*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2012:176-188.
38. Hoffmann J. Analyzing search topology without running any search: on the connection between causal graphs and h+. *J Artif Intell Res*. 2011;41:155-229.
39. Howe A, Dahlman E. A critical assessment of benchmark comparison in planning. *J Artif Intell Res*. 2002;17:1-33.
40. Gerevini A, Saetti A, Serina I. Planning through stochastic local search and temporal action graphs in LPG. *J Artif Intell Res*. 2003;20:239-290.
41. Coles AJ, Coles AI, Fox M, Long D. Forward-chaining partial-order planning. In: Proceedings of the Twentieth International Conference on International Conference on Automated Planning and Scheduling (ICAPS); 2010; Toronto, Canada.
42. Vidal V. YAHSP2: keep it simple, stupid. In: Proceedings of the 7th International Planning Competition (IPC); 2011.
43. Vidal V. YAHSP3 and YAHSP3-MT in the 8th international planning competition. In: Proceedings of the 8th International Planning Competition; 2014.
44. Quinlan JR. *C4. 5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann; 1993.
45. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH. The WEKA data mining software: an update. *ACM SIGKDD Explor Newsl*. 2009;11(1):10-18.
46. Rodriguez J, Kuncheva LI, Alonso CJ. Rotation Forest: a new classifier ensemble method. *IEEE Trans Pattern Anal Mach Intell*. 2006;28(10):1619-1630.
47. He H, Garcia EA. Learning from imbalanced data. *IEEE Trans Knowl Data Eng*. 2009;21(9):1263-1284.
48. Kohavi R. The power of decision tables. In: *Machine Learning: ECML-95*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 1995:174-189.
49. Hutter F, Hoos HH, Leyton-Brown K. Sequential model-based optimization for general algorithm configuration. In: *Learning and Intelligent Optimization*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2011:507-523.

How to cite this article: Cenamor I, Vallati M, Chrpa L. On the predictability of domain-independent temporal planners. *Computational Intelligence*. 2019;1–29. https://doi.org/10.1111/coin.12211