Identifying and Exploiting Features for Effective Plan Retrieval in Case-Based Planning

Mauro Vallati
University of Huddersfield, UK
m.vallati@hud.ac.uk

Ivan Serina and **Alessandro Saetti** and **Alfonso E. Gerevini**
University of Brescia, Italy
{ivan.serina,alessandro.saetti,alfonso.gerevini}@unibs.it

Abstract

Case-Based planning can fruitfully exploit knowledge gained by solving a large number of problems, storing the corresponding solutions in a plan library and reusing them for solving similar planning problems in the future. Case-based planning is extremely effective when similar reuse candidates can be efficiently chosen. In this paper, we study an innovative technique based on planning problem features for efficiently retrieving solved planning problems (and relative plans) from large plan libraries. Since existing planning features are not always able to effectively distinguish between problems within the same planning domain, we introduce a new class of features. Our experimental analysis shows that the proposed features-based retrieval approach can significantly improve the performance of a state-of-the-art case-based planning system.

Introduction

In this paper, we focus on the planning approach known as Case-Based Planning (CBP), or *planning by reuse* (Spalazzi 2001; Borrajo, Roubíková, and Serina 2015). The main observation in CBP is that in many of the real-world domains in which planning is applied, the typology of problems that should be solved remains similar. Therefore, it is expected that solutions of previously analysed problems can be useful when solving new problems within the same domain. In these cases, it can be more efficient to adapt an existing plan, rather than replanning from scratch. Intuitively, a case-based system is greatly dependent on the level of reusability of already solved instances. The useful level of dependency is fulfilled when problems tend to recur, and similar problems have similar solutions.

In CBP, a critical task is to efficiently identify, within a large library of already solved problems, those which are most similar to the new problem to solve. In this paper, we describe an innovative and efficient features-based approach for retrieving planning problems from large planning libraries, in order to improve the performance of CBP systems.

Recently, a large set of features has been exploited in planning for predicting the performance of planners (Fawcett et al. 2014; Cenamor, de la Rosa, and Fernández 2012; 2013; Howe et al. 1999; Roberts et al. 2008; Roberts and Howe 2009). Such features are either categorical or numerical, and they summarise specific properties of the planning instance. Typically, feature values are computed using a piece of software that efficiently analyses a given characteristic of the considered problem (Hutter et al. 2014). Features in planning have been mainly used for building predictive models of planners' performance (Roberts et al. 2008; Roberts and Howe 2009; Fawcett et al. 2014) or for selecting and combining planning engines in portfolios (Cenamor, de la Rosa, and Fernández 2012; 2013).

We observed that the existing largest set of planning features (Fawcett et al. 2014) is not always able to effectively distinguish between different problems from the same domain. Therefore, in this paper we introduce a new class of planning features, and demonstrate their usefulness in the CBP context.

Case-Based Planning

Following the formalisation proposed by Liberatore (2005), a *planning case* is a pair $\langle \Pi_0, \pi_0 \rangle$, where Π_0 is a planning problem and π_0 is a plan for it, while a plan library is a set of cases $\{\langle \Pi_i, \pi_i \rangle | 1 \leq i \leq m\}$.

In general the following steps are executed when a new planning problem is solved by a CBP system. **Plan Retrieval**: to retrieve cases from memory that are analogous to the current (*target*) problem and to evaluate their solution plans by execution, simulated execution, or analysis in order to choose one of them. **Plan Adaptation**: to repair any faults found in the retrieved plan to produce a new valid plan π. **Plan Revision**: to test π for success and repair it if a failure occurs during execution. **Plan Storage**: to eventually store π as a new case in the case base.

In order to exploit the benefits of remembering and reusing past plans, a CBP system needs efficient methods for retrieving analogous cases and for adapting retrieved plans together with a case base of sufficient size and coverage to yield useful analogues. The ability of the system to search in the library for a plan suitable to adaptation depends both on the efficiency/accuracy of the implemented retrieval algorithm and on the data structures used to represent the elements of the case base.

Limits of Existing Features in the CBP Context

As a matter of fact, in case-based planning selecting the problems of the case base which are mostly similar to the new given problem is of critical importance. In particular, the retrieval step deals with this process. Current techniques are either expensive or imprecise. In the former case, spending too much CPU-time in the retrieval, dramatically reduces the CPU usage of subsequent steps. In the latter case, the number of similar problems provided can be extremely large and/or not including the most similar problem.

Given the results achieved by Fawcett et al. (2014) in predicting planners' performance, we initially decided to consider all the features they exploited. Thus, for each planning problem of the case base, 311 features are extracted. Such features represent the largest set of planning features ever considered. The features that Fawcett et al. exploited come from a number of different sources. They computed features by (a) considering different encodings of a planning problem (PDDL, SAT, SAS+) (Bäckström and Nebel 1995), (b) extracting pre-processing statistics, (c) analysing the search space topology (Hoffmann 2011b; 2011a), and (d) considering probing features – brief runs of a planner on the considered problem, in order to extract information from its search trajectories. Their computation can take up to few minutes, according to the domain and the size of the considered problem.

Interestingly, we observed that this large set of features is often not accurate enough to distinguish between similar problems from the same domain. In some domains, when problems have the same number of objects, but differences in terms of goals and/or initial predicates, the values of all the features remain exactly the same. Hence, regardless to the function used for evaluating the similarity, such problems are indistinguishable. In fact, Fawcett-et-al features were designed for inter-domain exploitation, thus they are not capable of distinguish, under some conditions, similar problems within the same domain. We noticed that this is the case in Logistics-like domains.

Consider two Logistics problems involving four objects "`obj1`, `obj2`, `obj3`, `obj4`", four airports "`Aport1`, `Aport2`, `Aport3`, `Aport4`" and one airplane "`plane1`". In both problems the goal requires to have all the objects at "`Aport2`", while their corresponding initial state is:

Problem 1: `(at obj1 Aport1), (at obj2 Aport3), (at obj3 Aport3), (at obj4 Aport4), (at plane1 Aport4)`

Problem 2: `(at obj1 Aport1), (at obj2 Aport3), (at obj3 Aport3), (at obj4 Aport1), (at plane1 Aport4)`

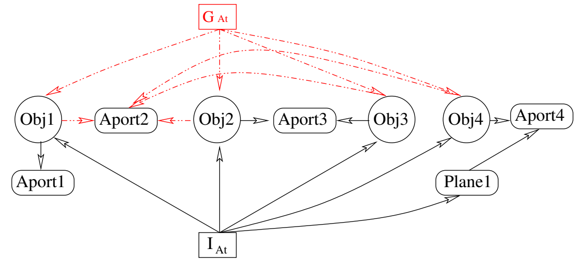

Figure 1: PEG of **Problem1**.

These problems are indistinguishable using the Fawcett et al. features, but they are significantly different from a CBP perspective. Clearly, this example involves trivially small instances, and the different adaptation cost between plans solving the two problems is irrelevant. Adaptation cost becomes relevant when indistinguishable problems involve hundreds of objects and numerous predicates in the initial or goal states are different.

Identifying features for CBP

Since in case-based planning it is critical to identify and order problems also on the basis of differences that are not revealed by existing planning features – this is of paramount importance for minimising the cost of evaluation and adaptation steps – we had to consider other features. We focused on investigating information that can allow a finest intra-domain problem characterisation. In OAKPlan (Serina 2010), each case in the case base is encoded as a Planning Encoding Graph (PEG). The underlying idea of the PEG is to provide a description of the "topology" of a planning problem without making assumptions regarding the importance of specific problem features for the encoding. The PEG of a planning problem Π is the union of the directed labelled graphs encoding the initial and goal facts.

In Fig. 1 we can see the PEG of the **Problem 1** of our running example where the edges associated to initial facts are depicted with black lines and the edges associated to goal facts are depicted with dashed red lines. The first and last level nodes correspond to initial and goal fact relation nodes, while the nodes of the intermediate levels correspond to concept nodes representing the objects of the initial and goal states. The PEG of **Problem 2** is not depicted for lack of space, but in this case it is characterized by an edge that connects `Obj4` to `Aport1` instead to `Aport4`.[1]

The way in which the graph is generated allows to identify and quantify even small differences between planning problems within the same domain. Furthermore, the graph generation is quick (usually it takes a few seconds). For characterising the graph, we considered three different versions: (i) the complete graph, (ii) the graph which considers only goal relations, and (iii) the graph which considers only initial state relations. Each version has been considered both

[1]The PEG of a problem includes also labels which are not considered by our features extractor. The interested reader can find a detailed description of the PEG in (Serina 2010).

as directed and undirected graph. From the directed graph versions, 19 features have been extracted, belonging to 4 classes. **Size**: number of vertices, number of edges, ratios vertices-edges and inverse, and graph density. **Degree**: average, standard deviation, maximum, minimum degree values across the nodes in the graph. **SCC**: number of Strongly Connected Components, average, standard deviation, maximum and minimum size. **Structure**: auto-loops, number of isolated vertices, flow hierarchy; results of tests on Eulerian and aperiodic structure of the graph.

The features extracted by considering the undirected graph versions are 18. **Size**: number of edges, ratios vertices-edges and inverse, and graph density. **Degree**: average, standard deviation, maximum, minimum degree values across the nodes in the graph. **Components**: number of connected components, average, standard deviation, maximum and minimum size. **Transitivity**: transitivity of the graph. **Triangle**: total number of triangles in the graph and average, standard deviation, maximum, minimum number of triangles per vertex.

In total, 115 features for each planning problem are computed: 111 from the different graphs, plus the value of degree sequences of the general graph (Ruskey et al. 1994), the number of instantiated actions, facts and mutex relations between facts. The original version of OAKplan uses a filtering approach based exclusively on the degree sequences of the PEG, while our version of OAKplan combines different features extracted from the PEG (including also the original degree sequences as additional feature) in order to define a more accurate similarity function. The original similarity function (based only on degree sequences) and our similarity function (based on the set of features) are used to filter the elements of the case base and provide a subset of them to the matching phase based on kernel functions.

Exploiting features for CBP

In our approach, a planning problem p is described by a sequence of features $F^p = \langle f_1^p, f_2^p, ..., f_n^p \rangle$. In order to quantify the similarity between two planning problems px and py, the following is done. First, for each pair of features f_m^{px}, f_m^{py} the *diff*(m) function is calculated as follows:

$$\textit{diff}(m) = \left| \frac{f_m^{px} - f_m^{py}}{max(f_m^{px}, f_m^{py})} \right| \tag{1}$$

Normalisation avoids that features with very different values have a higher impact. The similarity value *sim* between the problems is then computed as:

$$\textit{sim} = 1 - \frac{\sum_{i=1}^{n} \textit{diff}(i)}{n} \tag{2}$$

If $\textit{sim} = 1$, then the problems are estimated identical according to the considered features. The lower the similarity value, the higher the difference between the problems and, subsequently, also the cost of adaptation for a plan solving one of the two problems to become solution of the other problem. Given a new planning problem, the elements in the case-base are ordered according to their similarity to it, and the most similar elements are retrieved for the next steps of the planning process.[2]

Experimental Analysis

The benchmark domains considered in the experimental analysis are: Logistics, DriverLog, ZenoTravel, Satellite, Rovers and Elevators. The PDDL models are those used in the last IPC.

We generated a plan library with 6000 cases for each domain. Specifically, each plan library contains a number of case "clusters" ranging from 34 (for Rovers) to 107 (for ZenoTravel), each cluster c is formed by either a large-size competition problem or a randomly generated problem $\overline{\Pi}_c$ (with a problem structure similar to the large-size competition problems) plus a random number of cases ranging from 0 to 99 are obtained by changing $\overline{\Pi}_c$. Problem $\overline{\Pi}_c$ was modified either by randomly changing at most 10% of the literals in its initial state and goal set, or adding/deleting an object to/from the problem. The solution plans of the planning cases were computed by planner TLPlan (Bacchus and Kabanza 2000).[3] In our case bases, plans have a number of actions ranging from 72 to 519, while problem objects range from 69 in Elevators to 309 in Satellite.

For each considered domain, we generated 25 test problems, each of which derived by (randomly) changing from 1 to 5 initial goal facts, and renaming all the objects, of a problem randomly selected among those in the case base.[4] The experimental tests were conducted using a Xeon$^{\text{TM}}$2.00 GHz CPU, with a limit of 4 Gbytes of RAM. The CPU-time limit for each run was 30 minutes.

The planners we used are: LAMA 2011 (Richter, Westphal, and Helmert 2011), LPG-td v 1.0.2 (Gerevini, Saetti, and Serina 2006), FF v2.3 (Hoffmann 2003), and SGplan IPC6 (Hsu and Wah 2008). The results of OAKplan and LPG-td are median values over five runs for each problem.

Before evaluating our set of features, we tested the use of those introduced by Fawcett et al. (2014). We observed that on average they require between tens and hundreds seconds to be computed, although on some domains (such as Satellite) the computation can require up to 20 minutes. In several domains, it happens that they are not informative enough for distinguishing effectively between problems, thus a too large number of problems is retrieved and passed to the following CBP step, which becomes the bottleneck of the CBP process. On the other hand, the 311 features extracted by the Fawcett et al. approach are useful on two considered domains, Rovers and Satellite. On these domains their performance, in terms of problem filtering (i.e. not considering the large features computation time) are similar to those achieved by using our set of 115 features.

Table 1 shows the domain-by-domain results of the comparison between OAKPlan using its original plan retrieval

[2] In our experiments we selected problems with a difference between their similarity value and the best similarity value over all problems in the case base that was ≤ 0.001.

[3] Except for Elevators where we used the best solution plan produced by the generative planners considered in Table 1.

[4] Using the same experimental setup of (Gerevini et al. 2013).

Planner & Domain	Solved	Speed Score	Match. Time	Quality Score	Stab.
OAKplan					
DriverLog	100.0 %	12.53	23.18	23.30	0.89
Elevators	100.0 %	13.27	52.36	19.35	0.76
Logistics	96.0 %	16.06	40.06	22.34	0.88
Rovers	100.0 %	12.16	240.37	23.81	0.99
Satellite	100.0 %	18.43	259.49	24.82	0.97
ZenoTravel	100.0 %	19.43	16.01	21.77	0.91
Total	99.3 %	95.38	107.44	**140.36**	**0.90**
OAKplan All Features					
DriverLog	100.0 %	21.39	1.17	23.43	0.89
Elevators	100.0 %	17.81	1.43	18.39	0.69
Logistics	96.0 %	20.79	3.66	21.87	0.78
Rovers	100.0 %	23.67	14.85	23.68	0.98
Satellite	100.0 %	23.43	7.89	24.60	0.95
ZenoTravel	100.0 %	23.28	2.14	22.34	0.89
Total	99.3 %	**135.12**	**5.27**	139.12	0.87
OAKplan-GraphC					
Total	99.3 %	127.55	7.73	137.72	0.88
OAKplan-GraphG					
Total	98.6 %	131.56	5.92	132.46	0.82
OAKplan-GraphI					
Total	99.3 %	130.18	7.80	140.03	0.89

Table 1: Results of OAKPlan using its original retrieval function versus using all the features, and summary results (all domains together) of OAKPlan using only features extracted from partial PEGs. Performance is shown in terms of: % of solved instances, IPC speed score, average matching CPU time seconds, IPC quality score and stability.

function based on degree sequences (first subtable), and OAKPlan using the proposed features-based plan retrieval techniques (the other 4 subtables); in particular OAKplan-All-Features exploits all the 115 features, while the other subtables consider the features extracted from: the complete PEG (GraphC), the graph which considers only goal relations (GraphG), and the graph which considers only initial state relations (GraphI). Interestingly, there is not a significant difference between the performance achieved by exploiting different sets. Although using all the features guarantee the best IPC speed score, using only the features computed by considering the PEG of initial state provides slightly better results in terms of plan quality and stability. Finally, using only PEG of goal relations worsen the solved instances. As defined in (Nguyen et al. 2012), the plan stability between two plans π and π' is equal to the plan distance between π' and π, i.e. the number of actions that are in π' and not in π plus the number of actions that are in π and not in π' (Fox et al. 2006), divided by the sum of the number of actions of π' and π.

As shown in Table 1, in every considered domain the features-based approach improves the OAKPlan performance in terms of both IPC speed score (used in IPC 2014 Agile Track; considering the CPU-time required by the whole planning process) and CPU-time seconds needed for matching, only. Figure 2 shows the CPU time needed for solving the instances of the Logistics domain by the two

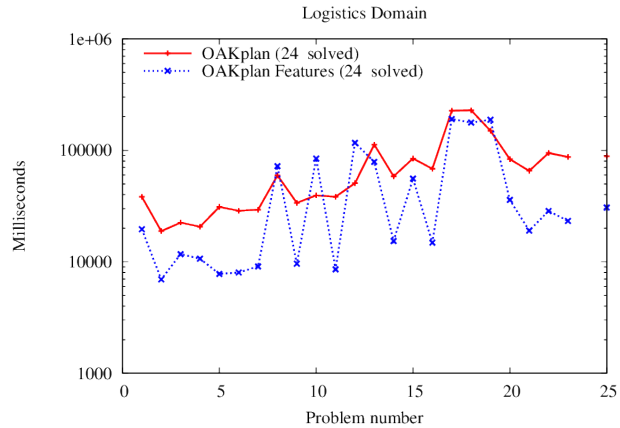

Figure 2: CPU time used by OAKPlan to solve problems in the Logistics domains with either our features-based retrieval or OAKPlan original retrieval.

Planner	% Sol.	Time (score)	Quality (score)
OAKplan	99.3 %	164.63 (82.93)	332.79 (116.38)
OAKplan All Features	99.3 %	91.33 (114.88)	334.99 (115.68)
FF v2.3	68.4 %	798.69 (57.56)	216.20 (100.87)
LAMA-2011	81.8 %	415.57 (65.37)	208.11 (104.31)
LPG-td	99.3 %	227.28 (111.57)	341.99 (114.85)
SGPlan	82.5 %	399.74 (94.48)	238.28 (104.24)

Table 2: Comparison in terms of % of solved problems, average CPU time (IPC speed score), and average quality (IPC quality score), between OAKPlan, OAKPlan using the proposed features for plan retrieval, LPG-td, FF and SGPlan.

OAKPlan versions, where we observe that in a few problems using the original retrieval function is faster than using the features-based approach. This is because the feature-based filtering is too selective, and considers only a very small set of cases, that do not include the best matching problem.

Table 2 shows the results of a comparison between OAKPlan exploiting the new features-based plan retrieval, the original OAKPlan, and the generative planners. On the considered benchmarks, the features-based approach demonstrated to be the fastest, and to provide good quality plans.

Conclusions

A critical step of Case-Based Planning is to efficiently identify, within a large library of already solved problems, those which are most similar to the new problem to solve. In this work we have proposed an efficient method that exploits problem features for effectively retrieving similar planning problems. Our experimental analysis demonstrated that (i) the filtering process is performed quickly, usually in a few seconds, and (ii) the proposed method can significantly speed-up the state-of-the-art case-based planner OAKPlan.

This work does not only impact the CBP topic, but can also open new avenues of research in different areas of planning. Potential applications of an efficient and effective similarity function include: (i) in learning-based planning systems, the evolution of extracted knowledge when testing problems are very different from training ones, as done in (Malitsky, Mehta, and O'Sullivan 2013); and (ii) the identification of different clusters of problems, on which tuning planners' configurations, that can be exploited in portfolio algorithms (Seipp et al. 2015; 2012).

Future work includes analysing different functions for evaluating the similarity value of two problems, and identifying important features within the considered set.

References

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116(1):123–191.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11:625–656.

Borrajo, D.; Roubíkova, A.; and Serina, I. 2015. Progress in case-based planning. *ACM Computing Surveys (CSUR)* 47(2):35.

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2012. Mining IPC-2011 results. In *Proceedings of the 3rd workshop on the International Planning Competition*.

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

Fawcett, C.; Vallati, M.; Hutter, F.; Hoffmann, J.; Hoos, H. H.; and Leyton-Brown, K. 2014. Improved features for runtime prediction of domain-independent planners. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, (ICAPS-14)*, 355–359.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-06)*, volume 6, 212–221.

Gerevini, A.; Roubícková, A.; Saetti, A.; and Serina, I. 2013. On the plan-library maintenance problem in a case-based planner. In *Case-Based Reasoning Research and Development - 21st International Conference, (ICCBR-13)*, 119–133.

Gerevini, A.; Saetti, A.; and Serina, I. 2006. An approach to temporal planning and scheduling in domains with predictable exogenous events. *Journal Artificial Intelligence Research (JAIR)* 25:187–231.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal Artificial Intelligence Research (JAIR)* 20:291–341.

Hoffmann, J. 2011a. Analyzing search topology without running any search: On the connection between causal graphs and h+. *Journal of Artificial Intelligence Research* 41:155–229.

Hoffmann, J. 2011b. Where ignoring delete lists works, part II: causal graphs. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS-11)*, 98–105.

Howe, A.; Dahlman, E.; Hansen, C.; Von Mayrhauser, A.; and Scheetz, M. 1999. Exploiting competitive planner performance. In *Proceedings of the 5th European Conference on Planning (ECP-99)*, 62–72.

Hsu, C.-W., and Wah, B. W. 2008. The SGPlan planning system in ipc-6. In *The 6th International Planning Competition (IPC-6)*.

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

Liberatore, P. 2005. On the complexity of case-based planning. *Journal of Experimental & Theoretical Artificial Intelligence* 17(3):283–295.

Malitsky, Y.; Mehta, D.; and O'Sullivan, B. 2013. Evolving instance specific algorithm configuration. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search, (SOCS-13)*, 132–140.

Nguyen, T. A.; Do, M. B.; Gerevini, A.; Serina, I.; Srivastava, B.; and Kambhampati, S. 2012. Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence* 190:1–31.

Richter, S.; Westphal, M.; and Helmert, M. 2011. Lama 2008 and 2011. In *Booklet of the 7th International Planning Competition*.

Roberts, M., and Howe, A. 2009. Learning from planner performance. *Artificial Intelligence* 173(5-6):536–561.

Roberts, M.; Howe, A. E.; Wilson, B.; and desJardins, M. 2008. What makes planners predictable? In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS-08)*, 288–295.

Ruskey, F.; Cohen, R.; Eades, P.; and Scott, A. 1994. Alley cats in search of good homes. *Congressus Numerantium* 97–110.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning portfolios of automatically tuned planners. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, (ICAPS-12)*, 368–372.

Seipp, J.; Sievers, S.; Helmert, M.; and Hutter, F. 2015. Automatic configuration of sequential planning portfolios. In *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI-15)*.

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

Spalazzi, L. 2001. A survey on case-based planning. *Artificial Intelligence Review* 16(1):3–36.