REPLICA: Relational Policies Learning in Planning

Rocío García-Durán, Fernando Fernández and Daniel Borrajo
Universidad Carlos III de Madrid
Avda de la Universidad 30, 28911-Leganés (Madrid), Spain
rgduran@inf.uc3m.es, ffernand@inf.uc3m.es and dborrajo@ia.uc3m.es

Abstract

REPLICA is a relational instance based learning module for solving STRIPS planning problems described in PDDL. REPLICA learns a reduced policy represented by a set of pairs <meta-state,action>. The meta-state represents the current planning state and the goal; the action represents the operator to execute in such meta-state. Both are described in terms of predicate logic. The next action to execute by the policy is computed as the action associated to the closest meta-state in that set. First, we extract an initial policy composed of a set of tuples (meta-state, action) from a set of solution plans. Second, we reduce this policy to obtain a subset of these tuples that generalizes the complete set. This learning proccess is done using relational nearest prototype classification. Finally, we use this policy for ordering the actions of the relaxed plans in a lookahead strategy for heuristic and forward search planning.

Introduction

Heuristic search has been the most successful approach for suboptimal domain-independent planning (Hoffmann and Nebel 2001; Gerevini and Serina 2003; Vidal 2004). Its success is due to the implementation of automated heuristic techniques based on relaxed plans, which ignore the delete effects of actions. These plans provide good estimations in many planning domains, as it can be seen since IPC-2004. However, this technique does not provide good results in other domains. In these domains, other automated techniques (Martin and Geffner 2000; Fern, Yoon, and Givan 2004) have been used to learn policies that are able to scale up well on these cases. Again, learned policies are imperfect and other thecniques, such as policy rollout (Bertsekas and Tsitsiklis 1996) and limited discrepancy search (Harvey and Ginsberg 1995) have been implemented. Other work combines both techniques together: heuristics search and policies (Yoon, Fern, and Givan 2007) using the policy during node expansion in the heuristic search.

In this work we present REPLICA.[1] It is a relational instance-based learning technique for solving STRIPS planning problems. REPLICA learns a reduced policy that is able to decide the next action to apply in any current meta-state. Each meta-state corresponds to the current search state and the pending goals. The policy is represented by a set of $<$meta-state$_i$,action$_i>$ tuples in predicate logic and a distance metric. This policy is obtained from a set of solution plans after letting the planner solve a set of simple and random problems. One problem with instance-based techniques is that the size of the set of tuples can increase as we solve more problems, suffering from the utility problem (Minton 1988): the time to retrieve the right policy can increase up to a point in which the time to make the decision based on the stored policy is more than the one needed to search for a good alternative. To solve this problem, we use a reduction algorithm to select the most representative set of tuples. The algorithm, called RNPC (Relational Nearest Prototype Classification) (García-Durán, Fernández, and Borrajo 2008), selects the most relevant instances of the training set. The selected prototypes, together with the distance metric, compose the policy. Finally, we use the reduced policy to order the actions of the relaxed plans in a lookahead strategy for heuristic and forward search planning (Vidal 2004).

The next section describes the complete process, which is explained in three subsections: the planning policy and distance metric, the learning process and how we use it.

The Complete Process of REPLICA

The complete process can be seen in Figure 1. We describe it in three steps: first, we extract the examples and generate the policy; second, we learn the reduced policy that generalizes the previous one; and, finally, we use this policy in a lookahead strategy to solve test problems. The three steps are explained in the next three subsections.

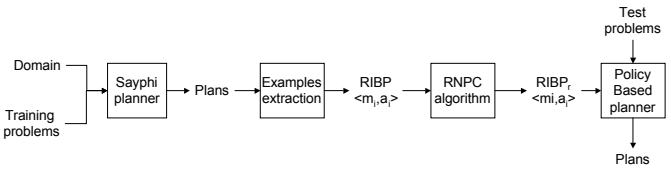

Figure 1: Scheme of the learning process.

[1] It stands for RElational Policy Learning.

Examples Extraction and Policy Generation

We provide a planner with a set of simple and random training problems to be solved. To obtain better plans, REPLICA solves them using the Enforced Hill Climbing algorithm (EHC) (Hoffmann and Nebel 2001), and then, it refines the solution with a Depth-First Branch and Bound algorithm (DFBnB). If EHC fails, an A* search is applied. Once we have the solution plans, we extract the examples to define the policy. A solution plan is an ordered set of instantiated actions, $< a_0, \ldots, a_{n-1} >$ such that, when executed, all goals are achieved. The execution of a plan generates state transitions that can be seen as tuples $< m_i, a_i >$ where $a_i \in A$ is an instantiated action of the plan, and $m_i \in M$ is a meta-state. A meta-state represents an instant on the search process containing relevant things about the search that allows making informed decisions (Veloso et al. 1995; Fernández, Aler, and Borrajo 2007). In our case, each m_i is composed of the state $s_i \in S$ and the pending goals $g_i \in S$. So, M is the set of all possible pairs (s, g). Other authors have included other features in the representation of meta-states as previously executed actions (Minton 1988), alternative pending goals in the case of backward search planners (Borrajo and Veloso 1997), hierarchical levels in the case of hybrid POP-hierarhical planners (Fernández, Aler, and Borrajo 2005), or deletes of the relaxed plan graph (Yoon, Fern, and Givan 2006). In the future, we would like to include some of these alternative features in the meta-states to understand the implications of the representation language of meta-states.

Relational Instance-Based Policies A *Relational Instance-Based Policy* (RIBP), π, is defined by a tuple $\pi =< L, P, d >$. P is a set of tuples, $t_1, \ldots t_n$ where each tuple t_i is defined as $< m, a >$, where $m \in M$ is a meta-state, and $a \in A$ is an instantiated action. Each t_i can be considered as an individual suggestion on how to make a decision, i.e. the action a that should be executed when the planner is in state s and tries to achieve the goals g. L defines the language used to describe the state and the action spaces. We assume that the state and action spaces are defined using PDDL. And, d is a distance metric that can compute the relational distance between two different meta-states. Thus, a *Relational Instance Based Policy*, $\pi : M \rightarrow A$ is a mapping from a meta-state to an action.

This definition of policy differs from the classical reinforcement learning definition, since the goal is also an input to the policy. Therefore, a Relational Instance-Based Policy can be considered an universal policy for the domain, since it returns an action to execute for any state and any goal of the domain. Given a meta-state, m, the policy returns the action to execute following an instance-based approach, by computing the closest tuple in P and returning its associated action. To compute the closest tuple, the distance metric d is used as defined in equation 1.

$$\pi(m) = \arg_a \min_{(<m', a> \in P)} dist(m, m') \qquad (1)$$

Next subsection describes the distance metric used in this work, although different distance metrics could be defined for different domains. In this work, the distance metric is based on previously defined metrics for Relational Instance-Based Learning approaches, the RIBL distance (Kirsten, Wrobel, and Horváth 2001).

The RIBL Distance To compute the distance between two meta-states, we follow a simplification of the RIBL distance metric, which has been adapted to our approach. Let us assume that we want to compute the distance between two meta-states, m_1 and m_2. Also, let us assume that there are K predicates in a given domain, $p_1, \ldots, p_K$. Then, the distance between the meta-states is a function of the distance between the same predicates in both meta-states, as defined in equation 2.

$$d(m_1, m_2) = \sqrt{\frac{\sum_{k=1}^{K} w_k d_k(m_1, m_2)^2}{\sum_{k=1}^{K} w_k}} \qquad (2)$$

Equation 2 includes a weight factor, w_i, for $i = 1, \ldots, K$, for each predicate. These weights modify the contribution of each predicate to the distance metric. And $d_k(m_1, m_2)$ computes the distance contributed by predicate p_k to the distance metric. For instance, in the Zenotravel domain, there are five different predicates that define the regular predicates of the domain, plus the ones referring to the goal ($K = 5$): *at, in, fuel_level, next, goal_at*. There is only one goal predicate, *goal_at*, since the goal in this domain is always defined in terms of the predicate *at*.

In each state there may exist different instantiations of the same predicate. For instance, two literals of predicate *at*: *(at p0, c0)* and *(at pl0, c0)*. Then, when computing $d_k(m_1, m_2)$ we are, in fact, computing the distance between two sets of literals. Equation 3 shows how to compute such distance.

$$d_k(m_1, m_2) = \frac{1}{N} \sum_{i=1}^{N} \min_{p \in P_k(m_2)} d'_k(P_k^i(m_1), p) \qquad (3)$$

where $P_k(m_i)$ is the set of literals of predicate p_k in m_i, N is the size of the set $P_k(m_1)$, $P_k^i(m_i)$ returns the ith literal from the set $P_k(m_i)$, and $d'_k(p_k^1, p_k^2)$ is the distance between two literals, p_k^1 and p_k^2 of predicate p_k. Basically, this equation computes, for each literal p in $P_k(m_1)$, the minimal distance to every literal of predicate p_k in m_2. Then, the distance returns the average of all those distances. Finally, we only need to define the function $d'_k(p_k^1, p_k^2)$. Let us assume the predicate p_k has M arguments. Then,

$$d'_k(p_k^1, p_k^2) = \sqrt{\frac{1}{M} \sum_{l=1}^{M} \delta(p_k^1(l), p_k^2(l))} \qquad (4)$$

where $p_k^i(l)$ is the lth argument of literal p_k^i, and $\delta(p_k^1(l), p_k^2(l))$ returns 0 if both values are the same, and 1 if they are different.

In this approach the static predicates have not been taken into account. The information on these literals is not specially relevant and the time for computing the distance decreases significantly.

Given these definitions, the distance between two instances depends on the similarity between the names of both sets of objects. For instance, the distance between two meta-states that are exactly the same but with different object names is judged as maximal distance. To partially avoid this problem, the object names of every meta-state are renamed. Each object is renamed by its type name and an appearance index. The first renamed objects are the ones that appear as parameters of the action, followed by the objects that appear in the goals. Finally, we rename the objects appearing in literals of the state. Thus, we try to keep some kind of relevance level of the objects to find a better similarity between two instances.

The Learning Process

From each resulting plan, $\{a_0, a_1, \ldots, a_{n-1}\}$, we extract a set of tuples $< m_i, a_i >$. All these tuples from all solution plans compose a policy (RIBP). However, we must reduce the number of tuples $< m_i, a_i >$ of the policy to obtain a reduced one (RIBP$_r$). The higher the number of tuples is, more time will be needed to reuse the policy. If this time is too high, it would be better to use the planner search instead of the learned policy. To reduce the number of tuples, we use the Relational Nearest Prototype Classification algorithm (RNPC) (García-Durán, Fernández, and Borrajo 2008), which is a relational version of the original algorithm ENPC (Fernández and Isasi 2004). There are three main differences with that work: RNPC uses a relational representation; the prototypes are extracted by selection as in (Kuncheva and Bezdek 1998); and we can reduce the number of final prototypes by using an optional parameter. The goal is to obtain a reduced set of prototypes P that generalizes the data set, such that it can predict the class of a new instance faster than using the complete data set and with an equivalent accuracy. The RNPC algorithm is independent of the distance measure and different distance metrics could be defined for different domains. For the sixth IPC we have experimented with the RIBL distance described in the previous section.

Because the RNPC algorithm is stochastic, we execute it 10 times, generating 10 different classifiers or 10 different RIBP$_r$. In order to use only one of them, we select the best RIBP$_r$ using a validation set of problems; those with higher complexity in the learning phase of the competition. This step will return the best RIBP$_r$ that solves more problems in less time, following the metric used in the competition. The strategy we employed on how to use the policy is explained in the next subsection.

Using the RIBP$_r$ in a Lookahead Strategy for Relaxed Plan Heuristic Planners

In this section we explain how we use the learned RIBP$_r$ to order the actions of the relaxed plans in a lookahead strategy for heuristic and forward search planning. Although the relaxed plan heuristic can be extended to solution plans (Hoffmann and Nebel 2001), the relaxed plan for a given state is not always a solution plan. It ignores the negative effects during its computation and the execution of one of these actions can make the rest of actions not executable. To avoid this, REPLICA orders the actions of the computed relaxed plans according to the RIBP$_r$, generating a lookahead state. Because this lookahead state is closer to the goal than the direct descendants of the current state, it is added at the beginning of the open list as a new descendent.A lookahead strategy allows planners to reduce node evaluations, which is usually the most expensive operation.

The lookahead state of a node is computed by iteratively selecting the best action to apply from the relaxed plan using the RIBP$_r$ as follows:

- The children with a helpful action, ha_j, are expanded.
- All the meta-states of the children are renamed following the binding of ha_j as we described at the end of the section .
- For all the children meta-states the smallest distance to a prototype in RIBP$_r$ with the same action as ha_j is computed.
- We select the helpful action that obtains the smallest distance.

The process goes on iteratively until it is not possible to advance any more in the relaxed plan or the goals have been achieved. The final state (lookahead state) is placed at the beginning of the open list.

The combination of the relaxed plan heuristics with the lookahead strategy guided by a policy offers us some advantages as: we reduced the number of evaluated nodes, and so the time, improving specially in domains with a strong interaction among the goals; and we can follow the heuristics even if the policy is not so good. On the other hand, the behavior of this strategy strongly depends on the distance metric, and sometimes it fails to capture the right decision. We are now working on finding a better distance metric.

Acknowledgements

This work has been partially supported by the Spanish MEC project TIN2005-08945-C06-05, a grant from the Spanish MEC, and regional CAM-UC3M project CCG06-UC3M/TIC-0831.

References

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. Neuro-dynamic programming. In *Athena Scientific*.

Borrajo, D., and Veloso, M. 1997. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning* 11(1-5):371–405. Also in the book "Lazy Learning", David Aha (ed.), Kluwer Academic Publishers, May 1997, ISBN 0-7923-4584-3.

Fern, A.; Yoon, S.; and Givan, R. 2004. Learning domain-specific control knowledge from random walks. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*.

Fernández, F., and Isasi, P. 2004. Evolutionary design of nearest prototype classifiers. *Journal of Heuristics* 10(4):431–454.

Fernández, S.; Aler, R.; and Borrajo, D. 2005. Machine learning in hybrid hierarchical and partial-order planners for manufacturing domains. *Applied Artificial Intelligence* 19(8):783–809.

Fernández, S.; Aler, R.; and Borrajo, D. 2007. Transferring learned control-knowledge between planners. In Veloso, M., ed., *Proceedings of IJCAI'07*. Hyderabad (India): IJCAI Press. Poster.

García-Durán, R.; Fernández, F.; and Borrajo, D. 2008. Prototypes based relational learniing. In *The 13th International Conference on Artificial Intelligence: Methodology, Systems, Applications*. To appear.

Gerevini, A., and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in lpg. *Journal of Artificial Intelligence Research* 20.

Harvey, W. D., and Ginsberg, M. L. 1995. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Inteligence (IJCAI-95)*, volume 1, 607–615.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Kirsten, M.; Wrobel, S.; and Horváth, T. 2001. *Relational Data Mining*. Springer. chapter Distance Based Approaches to Relational Learning and Clustering, 213–232.

Kuncheva, L., and Bezdek, J. 1998. Nearest prototype classfication: Clustering, genetic algorithms, or random search? *IEEE Transactions on Systems, Man, and Cybernetics*.

Martin, M., and Geffner, H. 2000. Learning generalized policies from planning examples using concept languages. In *Proc. 7th Int. Conf. on Knowledge Representation and Reasoning (KR 2000)*. Colorado: Morgan Kaufmann.

Minton, S. 1988. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Boston, MA: Kluwer Academic Publishers.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI* 7:81–120.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, 150–160.

Yoon, S.; Fern, A.; and Givan, R. 2006. Learning heuristic functions from relaxed plans. In *International Conference on Automated Planning and Scheduling (ICAPS-2006)*.

Yoon, S.; Fern, A.; and Givan, R. 2007. Using learned policies in heuristic-search planning. In *Proceedings of the 20th IJCAI*.