Discrepancy Search with Reactive Policies for Planning

SungWook Yoon
Electrical & Computer Engineering
Purdue University
West Lafayette, IN 47907
sy@purdue.edu

Abstract

We consider a novel use of mostly-correct reactive policies. In classical planning, reactive policy learning approaches could find good policies from solved trajectories of small problems and such policies have been successfully applied to larger problems. Often, due to the inductive nature, the learned reactive policies are mostly correct but commit errors on some portion of the states, rendering them useless in solving every problem of the domain. When the reward is only at goal states, the well known policy rollout approach cannot improve the performance of such faulty policies. Discrepancy Search has been developed in search to leverage the structural information of the heuristic functions which tends to be mostly-correct due to the human support. In this paper, we use reactive policies in discrepancy search for planning, in place of the heuristic functions. In our initial experiments, our proposed approach is effective in improving the performance of the given faulty reactive policies. The proposed approach outperformed the policy rollout as well as the reactive policies themselves. We will conclude with our research plan in the extension of the current proposal.

Introduction

Machine Learning (ML) has been successfully applied to many real life application domains, from image classification to natural language processing. AI Planning is no exception and ML techniques have been applied to AI planning and has shown some successes (Khardon 1999; Martin & Geffner 2000; Yoon, Fern, & Givan 2002). The role of ML here is producing a classifier that classifies an optimal or good action conditioned on the current state and the goal. Then, the sequential application of a good classifier to any problem in the target domain, will result in a goal state with high probability. The most successful approach of ML to planning was learning from sampled solution trajectories (Khardon 1999; Martin & Geffner 2000; Yoon, Fern, & Givan 2002) and the classifier produced by the ML techniques took the form of ordered list of rules (Khardon 1996). The resulting classifier is called (reactive) *policy* (Martin & Geffner 2000; Yoon, Fern, & Givan 2002) following the Markov Decision Process (MDP) framework or *control knowledge* (Estlin & Mooney 1996; Huang, Selman, & Kautz 2000; Aler, Borrajo, & Isasi 2002; Fern, Yoon, & Givan 2004) following search control framework. The classifier is called *reactive* control or policy when the only input to the classifier is the current state and goal information, without any information on the history of the action choices or state sequences that led to the current state.

To be useful in planning, a classifier for planning must be highly accurate. One wrong-selection among any of the state sequence of long plan trajectory could result in the failure on the whole planning problem, even if the classifier made correct choices all along the trajectory except in one state. As reported by (Khardon 1999; Martin & Geffner 2000; Yoon, Fern, & Givan 2002), ML technique found good reactive policies that achieve high success ratio on some planning domains like Blocksworld. Due to the nature of induction in ML techniques, often automatically found reactive policies, are prone to have some faults, typically caused by overfitting. *Ensemble* (Breiman 1996) can overcome overfitting but Ensemble itself can be faulty and it takes more time to train and get a set of reactive policies (Yoon, Fern, & Givan 2002). From decision theoretic planning, iterative updating of policy has been developed and one can use approximate policy iteration (API) (Bertsekas & Tsitsiklis 1996) in correcting the faulty policies. API (Fern, Yoon, & Givan 2003) has been shown effective in correcting faulty policies for planning, but it needs time in updating the policies and there is no guarantee that API always will produce improved policy over the current policy.

Ensemble and API techniques both need the assumption that the hypothesis space for the classifier is complete in every state. When the hypothesis space for the classifier cannot choose the correct action in every state of the target domain, either of the technique has no chance in selecting the correct actions on every state in the target domain.

In this proposal, we would like to answer the following question, "how to use reactive policies found by an ML technique when there is some possibility that the policies can be faulty". Or what if the hypothesis space for the ML is not complete, means that the knowledge representation (KR) for the hypothesis is good for most of the states in the target domain but for small portion of the states. We seek a deductive technique in using possibly faulty policies. Policy rollout (Bertsekas & Tsitsiklis 1996) is one such technique that can improve the performance of the given policy by sequentially choosing the actions considered best in one-step

look ahead policy evaluation. Although policy rollout is a powerful technique that can improve upon the given policy, when the reward is limited to a specific state like goal state and the faulty choice of the given policy is sporadic across the trajectory of the policy, then policy rollout may not be able to improve the performance of the given policy. Even multi-level policy rollout (Xiang Yan & Van Roy 2004) cannot fix the problem unless the faulty choices are limited to the initial part of the trajectories. In this proposal, we seek an answer to this sporadic distribution of faulty choices of a policy, in search techniques.

In search community, due to the involvement of human, many search application domains have a good set of well-designed features and have good heuristic functions. For some search application domains, heuristic functions are mostly-correct in guiding the search path but though infrequent, they can be wrong. *Discrepancy Search* (William D. Harvey 1995) is developed to leverage such structural information about heuristic function, or when the heuristic function is mostly-correct. In this proposal, we will employ discrepancy search in using mostly correct reactive policies.

The remainder of this proposal is structured as follows. We will describe the proposed technique. Then, we will show initial experimental results. Finally, we will give research direction following this proposal.

Proposed Algorithm

Figure 1 shows our proposed algorithm. The big picture of our algorithm is the same as the Discrepancy Search (DS). One difference is the consideration of the nature of the planning. Unlike the typical applications of the traditional DS, the depth of the search for planning problems can be arbitrarily long. And one cannot follow the given policy indefinitely. To address this issue, in the **Neighbors** function of figure 1, in every expansion of a node, we assign weight. Weight can be understood as the discrepancy from a state to another state with regard to the input policy. The paths favored by the policy will be weighted lower, enabling deeper search following those paths. The depth of the search is increased by the amount of the weight calculated by the assign-weight function. In this proposal, we suggested to use 1 - (the probability that the policy selects the path) + ϵ, where $\epsilon > 0$. Investigating alternative techniques like logarithmic use of the probability of the choice for the assign-weight function is in our agenda for the research. The parameter ϵ gives a natural horizon to the paths that follows the input policy, preventing paths with arbitrary depth.

Figure 1 also suggests a natural way of using stochastic policy. The assign-weight function is designed to consider a stochastic policy. During machine learning, based on the purity of the coverage in training data, one can easily assign the weight of the policy or the weight of each rule of the policy. And this information can be effectively used in stochastic search and in our algorithm. Weighting the selection of the heuristic function in DS has been studied (Walsh 1997; Bedrax-Weiss 1999). The weight setting in these works were on the level or depth of the tree, and not on the intrinsic stochastic nature of the heuristic function or the policy. So, our suggestion of using stochastic policy in DS will be an interesting extension of the weighted version of the DS technique.

Discrepancy-Search $(\mathbb{S}, \pi, D)$
// D: depth limit, $\mathbb{S}$: problem or state, π: input policy
$Q \leftarrow \{(\mathbb{S}, 0)\}$
// search queue: stores pairs of states and depths
$s \leftarrow$ **first**(Q)
repeat until **goal-reached**$(\text{first}(s))$
  Neighbors$(s, \pi, 0)$
  $s \leftarrow$ **first**(Q)
return Plan(s)

Neighbors (s, π, d)
// d: neighbor depth, s: state, π input policy
if **second**$(s) > D$; **return** // over the discrepancy limit
if $d > 1$; **return** // over the neighbor discrepancy limit
$N \leftarrow$ **Next-States**(**first**(s))
// enumerate direct neighbors
for-each n *in* N
  $w \leftarrow$ **assign-weight**(s, n, π)
  $Q \leftarrow$ **add**$(Q, (n, w +$ **second**$(s))$
  // add to search Q a new pair of state and depth
  Neighbors$((n, w +$ **second**$(n)), \pi, d + w)$

assign-weight (s', n, π)
// *example assign weight function*
//s': state, n: next state, π: input policy
return $(1 - p(\pi, s', n) + \epsilon)$
// $p(\pi, s', n)$: probability that the given policy moves to state n from the current state s'

Figure 1: Discrepancy Search with a Reactive Policy: The search algorithm is the same as that of limited discrepancy search. The discrepancy limit is D. Note that the discrepancy depth is not increased by one. The discrepancy depth is increased with proportional to the probability that the path is chosen by the input policy.

Preliminary Experiments

To test the performance of discrepancy search with reactive policies, we conducted experiments on Blocksworld and Driverlog. We randomly selected a policy learned from our previous research (Yoon, Fern, & Givan 2005) then we compared the performance of the policy in 3 techniques, policy as it is, rollout policy and discrepancy search with the policy. The figure 2 shows the performance of the each technique on the corresponding planning domains. Column labeled P shows the success ratio (SR) of the policy as it is. The success ratio here is measured as the number of solved problems in 100 randomly generated problems. Column labeled PR shows the SR of rollout policy of the given policy. Column labeled DS(n) shows the SR of discrepancy search with the policy, where we limit the number of the discrepancies to n.

For Blocksworld, we used 20 blocks problems and for Driverlog, we used 3 links, 4 drivers, 4 trucks and 8 package problems. As indicated in the figure 2, the rollout policy does not improve the performance of the faulty reactive policy. The faulty selections of actions happen sporadically across the trajectory, and the rollout policy does not address these faults. Rather the discrepancy search cures the faulty choices of the policy efficiently and improves the performance of the given policy. Even with discrepancy limit 1, the DS performs better than PR, and DS effectively correct the faulty choices while PR could not.

Domains	P	PR	DS(1)	DS(2)	DS(4)
Blocksworld	0.7	0.7	0.8	0.9	1
Driverlog	0.4	0.4	0.7	0.8	1

Figure 2: Using Reactive Policy

Research Direction

As an extension of the current proposal, we would develop discrepancy search techniques for stochastic planning domains and multi-agent planning domains, like Hearts or WarCraft domains. For the stochastic planning domains, the problem is the outcome distribution of the actions. We will assume that the reactive policy can designate not only the desired action choices but also the desired outcomes of the actions among possible outcomes of the actions. Similar idea of designating the outcome of an action is used in different purpose by (Boutilier, Dearden, & Goldszmidt 2000). Here the idea was used in symbolic value iteration. In our proposed work, the idea will reduce the branching factor to be considered in the AND-OR or ExpectiMax search. For the multi-agent domains, we would assume a similar situation, where the reactive policy can designate the expected action choices of the other agents as well as the current agent. This will again reduce the branching factor of the MIN-MAX search tree and will result in an effective and efficient algorithm.

Also we are planning to extend the proposed algorithm in heuristic based forward search setting in deterministic planning domains. In the heuristic search, the discrepancy search will be used in enumerating neighbors of a node. In this extension, we consider the nodes that are within one discrepancy from the current node with respect to the given policy, as neighbors of the current node. First, this will make the planner faster when the policy favors the same actions as the heuristics, since usually reactive policy's execution is faster than heuristic based action choices calculations. Second, when the heuristic is in a local minimum, and the policy favors orthogonal actions to the heuristic's favorites, this approach might help in escaping the local minimum, resulting in improvement on success ratio.

Note

Many of the ideas of this paper derive from joint work with Robert Givan

Acknowledgement

I thank Alan Fern for the helpful comments on this work.

References

Aler, R.; Borrajo, D.; and Isasi, P. 2002. Using genetic programming to learn and improve control knowledge. *AIJ* 141(1-2):29–56.

Bedrax-Weiss, T. 1999. *Optimal Search Protocols*. Ph.D. Dissertation, University of Oregon.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *AIJ* 121(1-2):49–107.

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2):123–140.

Estlin, T. A., and Mooney, R. J. 1996. Multi-strategy learning of search control for partial-order planning. In *AAAI*.

Fern, A.; Yoon, S.; and Givan, R. 2003. Approximate policy iteration with a policy language bias. In *Proceedings of the 16th Conference on Advances in Neural Information Processing*.

Fern, A.; Yoon, S.; and Givan, R. 2004. Learning domain-specific control knowledge from random walks. In *ICAPS*.

Huang, Y.-C.; Selman, B.; and Kautz, H. 2000. Learning declarative control rules for constraint-based planning. In *ICML*, 415–422.

Khardon, R. 1996. Learning to take actions. In *AAAI/IAAI, Vol. 1*, 787–792.

Khardon, R. 1999. Learning action strategies for planning domains. *AIJ* 113(1-2):125–148.

Martin, M., and Geffner, H. 2000. Learning generalized policies in planning domains using concept languages. In *KRR*.

Walsh, T. 1997. Depth-bounded discrepancy search. In *IJCAI*, 1388–1395.

William D. Harvey, M. L. G. 1995. Limited discrepancy search. In Mellish, C. S., ed., *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95); Vol. 1*, 607–615. Montréal, Québec, Canada: Morgan Kaufmann, 1995.

Xiang Yan, Persi Diaconis, P. R., and Van Roy, B. 2004. Solitaire: Man versus machine. In *NIPS*.

Yoon, S.; Fern, A.; and Givan, R. 2002. Inductive policy selection for first-order MDPs. In *UAI*.

Yoon, S.; Fern, A.; and Givan, R. 2005. Learning measures of progress for planning domains. In *AAAI*.