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 of the target domains. Often, due to the inductive nature, the learned reactive policies are mostly correct but commit errors on some portion of the states. Discrepancy search has been developed to explore the structure of the heuristic function when it is mostly-correct. In this paper, to improve the performance of machine learned reactive policies, we propose to use such policies in discrepancy search. In our experiments on benchmark planning domains, our proposed approach is effective in improving the performance of the machine learned reactive policies. The proposed approach outperformed the policy rollout with the learned policies as well as the machine learned policies themselves. As an extension, we consider using reactive policies in heuristic search. During a node expansion in a heuristic search, we added to the search queue all the states that occur along the trajectory of the given policy from the node. Experiments show that this approach greatly improves the performance of heuristic search on benchmark planning domains.

Introduction

Discrepancy search has been proposed and developed in search community (Harvey & Ginsberg 1995; Korf 1996; Walsh 1997). For some search domains, good heuristic functions can be defined and one can bound the search space with number of violations on the defined heuristic function, or *discrepancies*. When the heuristic function is good and mostly correct, discrepancy search (DS) with small number of discrepancies can be highly efficient, as shown in (Harvey & Ginsberg 1995; Korf 1996; Walsh 1997). In planning, best first search or (enforced) hill-climbing search (Hoffmann & Nebel 2001) have been applied successfully with the automatically calculated heuristics. But as reported by (Bonet & Geffner 1999), some other searches like discrepancy search did not produce good results partly because the heuristic, automatically calculated from problem and domain definition, is not accurate enough. [1] Although automatically calculated heuristic may not guide the search well for all the planning domains, it seems that fast enumeration of state space and fast calculation of such heuristics is more important than time-consuming and potentially more accurate heuristics, as can be seen in the results of recent planning competitions.

In planning, policy learning approach has been successful. From sampled solutions of small problems of the target planning domain, machine learning can find good policies that can work well on larger and unseen problems of the target domain (Khardon 1999; Martin & Geffner 2000; Yoon, Fern, & Givan 2002). Due to the inductive nature of machine learning, machine generated policies tend to be faulty (Yoon, Fern, & Givan 2002; Fern, Yoon, & Givan 2003). Such a policy is mostly-correct, suggesting good actions most of the time, but fails to do that on some portion of the states.

To use such mostly-correct machine learned polices efficiently, we propose to use discrepancy search with the machine learned policies. Discrepancy search is effective when the input heuristic function or the policy is mostly-correct and machine learned policies are good fit for discrepancy search technique. Typically *Policy rollout* (Bertsekas & Tsitsiklis 1996) is used to improve such mostly-correct policies. One-step look ahead policy can sometimes drastically improve the performance of the given policy. But, when the reward is only at a goal state like a planning problem, and the given policy commits multiple errors along the rollout trajectories, policy rollout may not be able to improve on the given policy. Multi-level policy rollout (Xiang Yan & Van Roy 2004) alleviates such weakness by looking ahead multiple states, but when the error distribution of the given policy is even across the whole rollout trajectories, rather than dense on the initial part of the trajectories, then multi-level policy rollout still may not be able to improve the performance of the given policy.

In our experiments, on benchmark planning domains, we empirically show that discrepancy search is a better fit in using machine learned policies than policy rollout technique as well as machine learned polices themselves. We also ex-

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

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

[1](McDermott 1996) reported that in some cases DS produced good results, but the result is not generalized to recent planning domains yet.

tend the use of reactive policy in search to heuristic search. In expanding a node or a state in the heuristic search, we add to the search queue the states that occur along the machine learned reactive policy trajectory from the state. Our experiments show that the performance of the heuristic search is much improved with the use of machine learned policies in expanding a node. Note that the heuristic search itself was the provider of the training data for the input policies.

Discrepancy Search

Discrepancy search (DS) (Harvey & Ginsberg 1995) gives depth or cost to the choices (discrepancies) that are against the defined heuristic function. In a search, heuristic function can be used in selecting a child node among the children of a node or ordering the children nodes. In limited discrepancy search (LDS), the focus is on discrepancies where a node is "not" favored by the heuristic function. The search process investigates all the nodes that can be searched with less than some predefined number of discrepancies, starting from the root node. So, if the heuristic function is wrong on n times along the successful traversal of the search tree, limit n discrepancy search will find a solution.

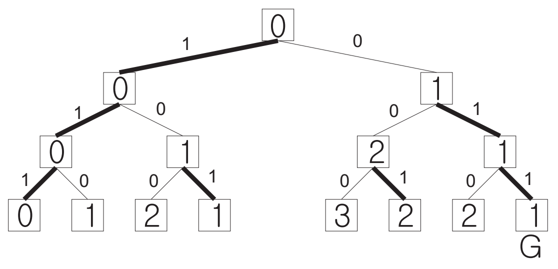

Figure 1: An Example of Discrepancy Search: Thin edges in the figure represent discrepancies. The goal is at depth 3 and is in discrepancy 1 from the root node with the current heuristic function. Greedy heuristic search needs many backtracks from the leaf nodes. Plain DFS or BFS search needs 14 nodes and Discrepancy limit 1 search only needs 9 nodes before they reach the goal node.

Figure 1 shows an example of discrepancy search (DS). The thick lines (paths) are choices favored by the heuristic function for the search tree and the thin lines are discrepancies. Each node is described as a rectangle box and the numbers inside the boxes are the number of discrepancies needed to reach those nodes. Consider the depth of the goal in the search tree and the number of discrepancies needed to reach the goal node from the root node. As can be seen in the figure 1, the goal node is in depth 3 and discrepancy 1 from the root node. Plain DFS or BFS search will need more time than limit 1 discrepancy search here. DFS and BFS need to visit 14 nodes before reaching the goal, while discrepancy search with limit 1 will find the goal after 9 nodes search. Greedy heuristic search on this tree needs to backtrack many times before it reaches the goal node. Note that these definitions of the discrepancies depend on the heuristic function.

In planning, DS has been attempted in (Bonet & Geffner 1999; McDermott 1996). (Bonet & Geffner 1999) reported that the result was not successful compared to best first search. While in search community, human provided mostly-correct heuristic can be used, in planning, search resorts to the automatically calculated heuristics. Although the recent development in automated heuristic calculation is good in many planning domains and led the performance improvement over past planning approaches, the automated heuristic is not suitable for DS, because it is not mostly-correct in many domains. As reported in (Hoffmann 2005), in most planning domains, and in many of the states, heuristics calculated from relaxed plans does not guide the search well, means that there are many faulty choices or no choice.

Learning to Improve Search in Planning

To improve the search performance in planning, researchers in classical planning have long studied techniques for applying machine learning to planning. For a collection and survey of works on "learning for planning domains" see (Minton 1993; Zimmerman & Kambhampati 2003). Two primary approaches are to learn domain-specific control rules for guiding search-based planners e.g., (Minton *et al.* 1989; Veloso *et al.* 1995; Estlin & Mooney 1996; Huang, Selman, & Kautz 2000; Ambite, Knoblock, & Minton 2000; Aler, Borrajo, & Isasi 2002), and, more closely related, to learn domain-specific reactive control policies (Khardon 1999; Martin & Geffner 2000; Yoon, Fern, & Givan 2002). The former ones try to prune out bad action choices restricting preconditions of actions beyond the specified definitions, while the latter ones map a state to a action choice directly. The latter ones, learning reactive control rules, have been successful, but they did not focus on improving the search but on improving the performance of the planning. As far as we know, the reactive control rules have rarely been used in search context and there was little effort to improve search using reactive control policy.

In this paper, we will try to improve search using reactive policies and at the same time, we attempt to enhance the performances of reactive policies using search.

Reactive Policy

Several research has shown that (Khardon 1999; Martin & Geffner 2000; Yoon, Fern, & Givan 2002; 2005), good reactive policies can be represented and learned for some AI planning domains. These approaches used decision list in representing policies and we will use Taxonomic (McAllester & Givan 1993) decision list in representing our policies.

Taxonomic Decision List Policy

In this paper we will use taxonomic decision list policy learning system. We represent reactive policy as a list of rules and each rule is a set of constraints on the arguments of the target action. The syntax for the decision list is the following.

$$\begin{aligned} DL &= \{\text{rule}_1, \ldots, \text{rule}_n\} \\ \text{rule}_i &= A_i : L_{i1}, \ldots, L_{im} \\ Lij &= x_j \in C_j \end{aligned}$$

A policy DL consists of ordered rules. The earliest rule that can be fired in the current state will be fired. Each rule $rule_i$ consists of an action type A_i and a set of constraints on each argument of the action type. Each constraint L_{ij} is specified by the concept expression C_j which constrains the jth argument of the action type. So, the rule can be fired if there is an applicable action, arguments of which are in the corresponding sets of the classes or concepts defined as C_j. The syntax for the concepts is the following.

$$\begin{aligned} C \quad = \quad & C_0 \mid \textbf{a-thing} \mid C \cap C \mid \neg C \mid x_i \\ & (R \;\; C_1 \ldots C_{i-1} \;\; * \;\; C_{i+1} \ldots C_{\text{n}(R)}) \end{aligned}$$

Please refer (Yoon, Fern, & Givan 2002; 2006) for the detailed specification of the syntax and semantics. Note that this syntax automatically derives from predicate and action symbols of the target planning domain. No human effort is needed in deriving the concept expressions.

Learning Reactive Policies

In this paper, we learn reactive policies from solved plans. The algorithm is the similar to the one we used in (Yoon, Fern, & Givan 2002). During the learning, we enumerate basic concepts and constraints, and greedily increase the constraints using the beam-search. The learning heuristic is designed to favor constraints that only cover selected actions in the solved plans and that do not cover other actions. As shown in (Yoon, Fern, & Givan 2005; 2002), it is good to have all the good actions labeled along the solved plans, to get good policies or good classifiers. Still, without all good actions labeled in a state, the learning algorithm found useful policies as we reported in (Yoon, Fern, & Givan 2005). In the experiments section of this paper, we have used the learned policies reported in (Yoon, Fern, & Givan 2005).

Using Reactive Policies

Suppose the learning system of (Yoon, Fern, & Givan 2005) found a reactive policy π for some planning domain $\mathbb{D}$ and the policy is mostly-good but commits some errors on some states. If we apply the policy π on the problems of $\mathbb{D}$ as it is, we might get some failure due to the error of π on some states.

One step look ahead policy rollout (Bertsekas & Tsitsiklis 1996) technique may improve the performance of the policy. Policy rollout sequentially selects actions that are the best according to the one-step lookahead policy evaluations of the given policy. When the domain is goal-directed and the given policy commits multiple errors during rollouts, the one-step lookahead will return all failures and will not help improving the performance of the policy. Multi-level policy rollout can be an alternative solution if the errors of the policy occurs only at the initial few steps of the rollouts but typically the error distribution is sporadic across rollout trajectories. To fix such a problem, we would like to search around the states along the trajectory produced by the policy.

Discrepancy Search with Reactive Policies

In this paper, we would like to use reactive policies in discrepancy search in place of heuristic function. Discrepancy search can efficiently exploit the structure of the heuristic function by assuming some faults of the heuristic function along the search tree and search the paths that are not favored by the heuristic function. Here we propose to do discrepancy search with mostly-correct reactive policies.

Consider a stochastic policy π. Define $p(\pi, s, n)$ to be the probability that the policy π moves from state s to n in a step. If the policy π is deterministic, $p(\pi, s, n) = 1$, if π selects the move or the action in s to n and $p(\pi, s, n') = 0$ for all the other neighbor states from s.

Then we assign weight on moving from state s to n with the policy π, as 1 - $p(\pi, s, n)$ + ϵ, where $\epsilon > 0$. See figure 2. When it is likely that π move from state s to n then ($p(\pi, s, n)$ close to 1), the assign-weight will assign low weight for that move, and if it is unlikely that π move from state s to n then ($p(\pi, s, n)$ close to 0), the assign-weight will assign high weight for that move. In any case the function will assign positive weight due to the parameter ϵ.

Let us then, define node expansion of the search as adding nodes that are within weight (or discrepancy) 1 from the node being expanded. Note that due to the weight consideration, this node expansion will consider states that are multiple steps or actions away from the the node being expanded, especially those favored by the given policy. See the figure 2 for the Neighbors function.

As a result of these definitions, when a search node is expanded, we not only add the neighbors of the node, but also many states that occur along the trajectory caused by the reactive policy. These states are multiple steps away from the node and this will make the search consider states that are deeper location in the search tree.

Suppose the reactive policy has n faulty choices compared to a good trajectory. Then with this search scheme and with limit n DS, the search will find a goal, even with such a faulty policy. Note that without such a policy, the general search depth needed to find a goal is the length of the successful trajectory l, which will be much costlier than DS with n limit, since usually $l >> n$. Also, note that using the policy as it is may not find a goal.

Our algorithm for Discrepancy Search with Reactive Policies is in figure 2. The algorithm searches all the states that are within the given discrepancy (or weight) limit and finishes when a goal is found or the state space is exhausted. The ordering of the states in the queue can follow the typical search approaches like BFS, DFS or heuristic search. The top level search is no different than usual search. The difference lies in the Neighbor function, which typically enumerates neighboring states but, in our algorithm, the neighbors are extended by the weighted path connection of the given policy. Note also that usually reactive policy is faster in calculation than heuristic function based calculation for the action choices. To accommodate stochastic policies, the

```
Discrepancy-Search ($\mathbb{S}, \pi, D$)
// problem $\mathbb{S}$, reactive policy $\pi$, discrepancy limit $D$.

$Q \leftarrow \{(\mathbb{S}, 0)\}$ : Search Queue
$s \leftarrow$ first($Q$)
repeat until solved(first($s$))
    Neighbors($s, \pi, 0$)
    if $Q ==$ NULL
        Return Fail
    $s \leftarrow$ first($Q$)
Return Plan($s$)
```

```
Neighbors ($s, \pi, d$)
if second($s$) $> D$ return
if $d > 1$ return
$N \leftarrow$ Next-States(first($s$))
for-each $n$ in $N$
    $w \leftarrow$ assign-weight($s, n, \pi$)
    $Q \leftarrow$ add($Q, (n, w +$ second($s$)))
    Neighbors(($n, w +$ second($n$)), $\pi, d + w$)
```

```
assign-weight ($s', n, \pi$) //example assign weight function
return $1 - p(\pi, s', n) + \epsilon$
```

Figure 2: Discrepancy Search with Reactive Policies: The algorithm searches with the given policy π on the problem $\mathbb{S}$. It uses a predefined cost ϵ to limit the search depth of the paths following the given policy.

assign-weight function is introduced and we just show an example function that can work as an assign-weight function. Investigating the use of logarithmic cost of the stochastic policy, instead of assign-weight function is in our agenda for future research.

Experiments

Discrepancy Search with Reactive Policies

To test the performance of discrepancy search with reactive policies, we conducted experiments on 2 benchmark planning domains, Blocksworld and Driverlog. We randomly selected a policy learned in the (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 3 contains the performance of 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. Column labeled DS(n) shows the SR of discrepancy search with the policy, where we limit the size of the discrepancy to n. We used 0.01 for ϵ and we aligned the search queue Q of discrepancy search in a DFS style.

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 3, the rollout policy does not improve the performance of the faulty reactive policy. The faulty selection of actions happens sporadically across the trajectory, and the rollout policy does not cure these faults. Rather the discrepancy search cures the faulty choices of the policy and improve the performance. In selecting actions from rollout results, we select actions randomly when there is a tie in rollout results. The results of PR can be similar to DS(1) if we let the rollout policy choose the action selected by the policy, when there is a tie in rollout results. When there is multiple faults in the trajectory incurred by the given policy, policy rollout may not be able to fix those faults. Comparing results of multilevel rollout PR(n) with DS(n) is in our future research agenda.

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 3: Using Reactive Policies: Compared to the performance of the reactive policy as it is (P) or policy rollout of the policy (PR), discrepancy search with the policy (DS(n)) performs better.

Reactive Policies in Heuristic Search

For the Driverlog domain in the previous experiments, the size of the problems is not as large as the problems in the third international planning competition. The randomly chosen policy used in the figure 3 selects many poor actions in solving larger problems in the competition and the DS with it could not find any solution with small discrepancy limit. Since the heuristic based forward search is used in the state of the art planners, in this second experiment, we used the policy in the heuristic search. In this experiment, we used the FF system (Hoffmann & Nebel 2001) and added the policy in its node expansion part. We defined node expansion as adding nodes in discrepancy (or weight) limit 1 from the node being expanded with respect to the given reactive policy, and we can directly use the Neighbor function in the figure 2. There can be two potential advantages in using a reactive policy in heuristic based search. First, if the reactive policy is consistent with the heuristic search then it will help reducing the heuristic calculation time and the planning process can finish faster. Note that typically a reactive policy calculation is faster than heuristic calculation of each neighbor of the state and selecting the best neighbor according to the heuristic. Second, when the heuristic search is stuck in a local minimum, and if the reactive policy is orthogonal to the heuristic or is helpful in escaping the local minimum, then the combination of the reactive policy with the heuristic search can guide the bailout process, and will result in better success ratio on testing problems.

Figure 4 shows the performance of reactive policy in heuristic search. Here we attempted to solve 20 problems of Driverlog domain in the third international planning competition and we also attempted to solve 20 randomly generated problems of 20 blocks in the Blocksworld domain. The CPU time limit was 30 minutes. We used a Linux box with a 2.8

Domains	FF	FF + P
Blocksworld	18	20
Driverlog	16	19

Figure 4: Using Reactive Policies in Heuristic Search: Compared to the heuristic search, heuristic search with the reactive policy performs better. The policy was learned from the heuristic search.

Intel Xeon Processor and 2 Gig RAM. The column named FF shows the number of problems solved by FF and the column named FF + P shows the number of problems solved by the heuristic search of FF with the learned reactive policy. In both of the domains, the usage of the policy helps. This result is interesting, considering the policy itself is not perfect. Though it is preliminary to conclude with these sets of experiments, it seems that using reactive policies in search helps in at least two forms, discrepancy search with reactive policies and reactive policies in heuristic search.

Conclusion and Future Works

We have shown empirically that at least for deterministic planning domains, discrepancy search is a better fit in using machine learned mostly-correct policies than policy rollout technique. We also have shown that heuristic search can be benefited by using machine learned policies in expanding the nodes during the search. We will continue the research on more various and recent planning domains and we are seeking expansion of the current work on stochastic planning domains.

Acknowledgement

I would like to thank Alan Fern for the helpful discussions on this paper.

References

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

Ambite, J. L.; Knoblock, C. A.; and Minton, S. 2000. Learning plan rewriting rules. In *Artificial Intelligence Planning Systems*, 3–12.

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

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *ECP*, 360–372.

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*.

Harvey, W. D., and Ginsberg, M. L. 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.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:263–302.

Hoffmann, J. 2005. Where ignoring delete lists works: Local search topology in planning benchmarks. *JAIR*.

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

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

Korf, R. E. 1996. Improved limited discrepancy search. In *AAAI/IAAI, Vol. 1*, 286–291.

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

McAllester, D., and Givan, R. 1993. Taxonomic syntax for first-order inference. *Journal of the ACM* 40:246–283.

McDermott, D. 1996. A heuristic estimator for means ends analysis in planning. In Drabble, B., ed., *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, 142–149. AAAI Press.

Minton, S.; Carbonell, J.; Knoblock, C. A.; Kuokka, D. R.; Etzioni, O.; and Gil, Y. 1989. Explanation-based learning: A problem solving perspective. *AIJ* 40:63–118.

Minton, S., ed. 1993. *Machine Learning Methods for Planning*. Morgan Kaufmann Publishers.

Veloso, M.; Carbonell, J.; Perez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI* 7(1).

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

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*.

Yoon, S.; Fern, A.; and Givan, R. 2006. Learning heuristic functions from relaxed plans. In *ICAPS*.

Zimmerman, T., and Kambhampati, S. 2003. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine* 24(2)(2):73–96.