Computing Agent's Expectations in Nondeterministic Domains

Noah Reifsnyder and Héctor Muñoz-Avila
Lehigh University
Bethlehem, PA 18015
{ndr217, hem4}@lehigh.edu

Abstract

In this paper we summarize our ideas for computing policy expectations. We present a taxonomy of 4 types of expectations and briefly indicate how they can be computed.

1 Introduction

Over the past years, there has been an increasing interest on goal reasoning, the study of agency that is capable of explicitly reasoning about their goals and changing them when the need arises [1]. Part of the interest in goal reasoning is its potential applications in fields such as controlling robots [2], including underwater unmanned vehicles [10], and air combat [7]. One of the goal reasoning mechanisms is Goal-driven autonomy (GDA). GDA agents perform a 4-step cycle in which (1) actions are executed in the environment from a solution (e.g., a plan) generated for achieving previously formulated goals; (2) expectations on the agent's actions are computed and checked against the actual outcomes of the actions; (3) when there is a discrepancy between the outcome and the expectations, the agent formulates an explanation for this discrepancy; (4) as a result of this explanation, new goals to pursue are formulated.

Researchers have observed that the notion of expectations plays a key role in the overall goal reasoning of GDA agents [5; 4]. If the expectations are too narrow, agents might fail to detect discrepancies when they are needed leading to agents continuing to pursue a goal that will lead to a failure. If the expectations are too general, the agent might trigger a procedure to formulate new goals in situations when it is not required to do so. A common characteristic of these works is that they compute the expectations over sequences of actions, typical of deterministic domains.

In this paper, we summarize our re-examination of the notion of expectations when the solutions achieving the goals are policies, which are mappings from states to actions. Policies are needed in nondeterministic domains, where actions may have multiple possible outcomes. A policy accounts for all foreseen states that the agent might encounter based on the action definitions and the starting state. However, agents might encounter situations it has not planned for. This can happen in particular when the agent is operating autonomously for extended periods of time. Reasons for encountering such unforeseen situations include: (1) exogenous events, that is, conditions in the state that were not planned for and (2) changes in the actions. That is, the actions' effects. An example of the latter is a failure in a mechanical motion device that causes it to operate in a different way from what it is modeled in the action definitions. [3] present a taxonomy of failure reasons that is attributable to factors such as discrepancies in the state, discrepancies in the action model, discrepancies in the goals and discrepancies in the environment.

In this paper we summarize our work on expectations for ND domains:

- A formalization of four notions of policy expectations.
- Embedding these into the GDA framework.

While our work is centered around goal reasoning, our definitions for expectations are agnostic on the particular planning paradigm used to generate the policies and on the paradigm used for correction of the course of action. In this paper we are using goal reasoning to ground these definitions but nothing in the specifics of these definitions precludes using other forms of correction such as replanning or simply stopping the agent's execution (e.g., in [8] a robotic arm deactivates itself when encountering a discrepancy).

2 Policy Immediate Expectations

Agents that use Policy Immediate Expectations make sure the effects of the previously executed action that lead to the current state, and the next action's preconditions hold in the current observed state. This is a simple set of light-weight expectations to allow an agent to know it isn't *stuck* and that it is able to continue the execution of its current policy. An example of a violation to an immediate expectation is one where the agent finds itself in an state s that according to the plan transition model is not reachable when executing action a in state s' but this is precisely what happened.

3 Policy Informed Expectations

Agents that use Policy Informed Expectations make sure the accumulated effects from all previously executed actions so far still hold in the current state. Succinctly, this is accomplished by taking the transforming the policy into a graph by viewing states and actions as states and state-action and action-state transitions as edges. Then performing a depth-first search (DFS) on the graph from the starting state s_0, which labels all the edges as Tree-, Cross-, or Back-edges. Then create a so-called DFS tree consisting of all the nodes in the graph and only the tree edges. We compute informed expectations on every path from the s_0 node to a terminal node as an independent path. Using Policy Informed Expectations allows the agent to make sure that all intended changes to the

environment made by the agent persist throughout the execution of the Policy. This is important to track, as we assume to achieve it's goals, the agent must alter the environment. Thus we want to make sure those changes persist.

4 Policy Regression Expectations

Agents that use Policy Regression Expectations make sure all future preconditions are able to be met by the agent. To generate this set of expectations, we need to build a tree, called T_π, from the Policy graph. Informally, this can be done as follows: We perform a DFS on the graph, copying all nodes and edges into T_π. If the edge is a back-edge, we recursively call this algorithm on the policy graph, with that back-edge removed. When finished, T_π represents all possible paths an agent can take from the starting state to a terminal state. We don't need to worry about cases where the agent stays in an execution loop for a while, as the expectation set from the first iteration of the agent completing the loop is valid for all subsequent iterations of the execution loop. Once T_π is constructed, we begin aggregating all preconditions of actions, starting with the set of goals in the terminal states. Each expectation has an associated probability, starting at 1 for each expectation in the goal set for terminal nodes. Whenever we reach a branching point in T_π while working up the tree, we divide the probability for each expectation according to the probability distribution for how likely each child is to be visited from the parent, then aggregate probabilities from like expectations across the children (the parent being the vertex before the branch). Once we reach the starting state, since there can be multiple copies of each vertex from the policy graph in T_π, we do a topological sort on T_π, and take the expectations from the first instance of a vertex for use in the policy graph.

Policy Regression Expectations are useful in identifying when the agent is likely to fail somewhere in its future execution as early as possible. Since the expectation set is a probability distribution, you can set a threshold for what triggers a discrepancy. The probability that is attached to an expectation equates to how likely the plan is to fail if that expectation is invalidated.

Policy Regression Expectations are well-defined. The computation of Policy Regression Expectations is independent from the order in which DFS expands the Policy Graph. Meaning, there could be multiple ways the Regression Tree T_π is generated from the Policy Graph, however, the final Policy Regression Expectations for each vertex in the graph remains the same. The reason for this is as follows: the method for generating Policy Regression Expectations aggregates information on every state vertex, which is akin to taking every possible path from a state to a terminal state. It then combines all descendant expectations generating a probability distribution based on the possible paths to a terminal state. This will aggregate the same information no matter how T_π itself is generated from the Policy Graph.

5 Policy Goldilocks Expectations

Policy Goldilocks Expectations is used for scenarios in which the set of goals are unknown(i.e. the goals are tasks as in HTN planning) [6]. Another example are systems that combine plan and execution and plan ahead n steps, e.g., using heuristics, but without accomplishing the goals in intermediate iterations of the plan-execute steps (i.e., only in the final iteration the goals are achieved; [9; 5]). Policy Goldilocks Expectations aims to fill the holes both Informed and Regression have, without adding any excess information. For Policy Informed Expectations, if a future precondition is removed from the state, the agent won't know until it fails the action. For Policy Regression Expectations, if some goal that the agent fulfilled is nullified, the agent will terminate its execution without the full set of goals achieved.

To do this, we first calculate Policy Informed Expectations. Then when perform the same procedure as in Policy Regression Expectations, with the difference being we use the set of Policy Informed Expectations at the terminal vertices instead of the set of goals. What this is basically doing is using the final set of Policy Informed Expectations as the set of goals. This can intuitively make some sense. The agent needs to make some set of changes to the environment to satisfy its directive. Since Policy Informed Expectations is the set of all changes the agent made, some subset of them are the implied goals.

References

[1] David W. Aha. Goal reasoning: foundations emerging applications and prospects. *AI Magazine*, 2018.

[2] Michael T Cox, Dustin Dannenhauer, and Sravya Kondrakunta. Goal operations for cognitive systems. In *AAAI*, pages 4385–4391, 2017.

[3] Michael T Cox and Ashwin Ram. Introspective multistrategy learning: On the construction of learning strategies. *Artificial Intelligence*, 112(1-2):1–55, 1999.

[4] Dustin Dannenhauer and Hector Munoz-Avila. Raising expectations in gda agents acting in dynamic environments. In *IJCAI*, pages 2241–2247, 2015.

[5] Dustin Dannenhauer, Hector Munoz-Avila, and Michael T Cox. Informed expectations to guide gda agents in partially observable environments. In *IJCAI*, pages 2493–2499, 2016.

[6] Kutluhan Erol, James Hendler, and Dana S. Nau. Complexity results for hierarchical task-network planning. *Annals of Mathematics and Artificial Intelligence (AMAI)*, 18:69–93, 1996.

[7] Michael W Floyd, Justin Karneeb, Philip Moore, and David W Aha. A goal reasoning agent for controlling uavs in beyond-visual-range air combat. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 4714–4721. AAAI Press, 2017.

[8] Austin Gregg-Smith and Walterio W Mayol-Cuevas. The design and evaluation of a cooperative handheld robot. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1968–1975. IEEE, 2015.

[9] Dana S Nau, Malik Ghallab, and Paolo Traverso. Blended planning and acting: Preliminary approach, research challenges. In *AAAI*, pages 4047–4051, 2015.

[10] Mark A. Wilson, James McMahon, A. Wolek, and David W. Aha. Goal reasoning for autonomous underwater vehicles: responding to unexpected agents. *AI Communications*, 2018.