Informed Expectations to Guide GDA Agents in Partially Observable Environments

Dustin Dannenhauer and **Hector Munoz-Avila**
Dept. of Computer Science and Engineering
Lehigh University, Bethlehem, PA USA
dtd212,hem4@lehigh.edu

Michael T. Cox
Wright State Research Institute
Wright State University, Dayton, OH
michael.cox@wright.edu

Abstract

Goal Driven Autonomy (GDA) is an agent model for reasoning about goals while acting in a dynamic environment. Since anomalous events may cause an agent's current goal to become invalid, GDA agents monitor the environment for such anomalies. When domains are both partially observable and dynamic, agents must reason about sensing and planning actions. Previous GDA work evaluated agents in domains that were partially observable, but does not address sensing actions with associated costs. Furthermore, partial observability still enabled generation of a grounded plan to reach the goal. We study agents where observability is more limited: the agent cannot generate a grounded plan because it does not know which future actions will be available until it explores more of the environment. We present a formalism of the problem that includes sensing costs, a GDA algorithm using this formalism, an examination of four methods of expectations under this formalism, and an implementation of the algorithm and empirical study.

1 Introduction

Goal-driven autonomy (GDA) is an agency model where an agent revises its goals by reasoning over discrepancies. Discrepancies arise when the agent's own expectations do not match the agent's observations. Such discrepancies arise when acting in dynamic environments (i.e., changes occur independently from the agent's actions). When discrepancies occur, the GDA agent will suggest alternative goals. An example, adapted from [Molineaux *et al.*, 2010], involves an agent performing navy operations. A naval convoy is in route to deliver some equipment and along the way an escort vessel identifies an unknown contact. At this point the agent could pursue one of multiple alternative goals including (1) abort the mission and route back the vessels to the departing port, (2) hold the convoy and send escort vessels to identify the contact.

In partially observable and dynamic environments it may not be obvious what the agent needs to know about its current state to achieve its goals and what is irrelevant. This becomes especially true for GDA agents which may change the goals they pursue over time. In previous GDA work, partial observability still enabled generation of a grounded plan to reach the goal. Furthermore, previous work on GDA agents [Molineaux *et al.*, 2010; Weber *et al.*, 2012; Jaidee *et al.*, 2011; Shivashankar *et al.*, 2013] does not address sensing actions with associated costs. In contrast, researchers have long observed that acquiring knowledge about the state of the world can be expensive both in terms of running time to complete the tasks and in resource consumption [Knoblock, 1995]. For example, virtual agents might require to plan the information gathering tasks including which information sources to access and what information to acquire, which in turn will enable the agent to identify other information sources [Knoblock, 1996]; physical agents may use sensors which require power, time, and potentially other resources [Mei *et al.*, 2005].

This paper integrates ideas of information-gathering agents into the GDA framework. In particular, we propose a new family of GDA agents that adopt the convention of distinguishing between planning actions and sensing actions, the latter of which have associated costs. We investigate how different kinds of expectations affect the performance of GDA agents in environments that are partially observable and dynamic. We evaluate agents where observability is limited in such a way that the agent cannot generate a grounded plan because it does not know which future actions will be available until it explores more of the environment. Our contributions are as follows: (1) A goal-reasoning formalism, tailored towards GDA agents, that models sensing costs. (2) A re-examination of four different forms of expectations for GDA agents operating under this formalism. (3) A GDA algorithm for acting under this formalism. (4) An implementation of the algorithm and empirical study on two domains making comparative studies between the four forms of expectations.

2 Related Work

Deterministic (STRIPS) planning assumes that actions have a pre-determined outcome [Fikes and Nilsson, 1971]. The result of planning is a sequence of actions that enable the agent to achieve its goals. A Markov Decision Process (MDP) is a frequently studied planning paradigm whereby actions have multiple outcomes [Bonet and Geffner, 2006]. In MDPs, solutions are found by iterating over the possible outcomes until a policy is generated which indicates for every state that the agent might encounter, what action to take that will enable

the agent to achieve its goals. A Partial Observable Markov Decision Process (POMDP) is an extension of MDP for planning when the states are partially observable [Kaelbling *et al.*, 1998]. In POMDPs, solutions are found by iterating over the possible states that the agent believes itself to be in and the possible outcomes of the actions taken on those belief states until a policy is found over the belief states. In GDA agents, goals may change while the agent is acting in the environment. Previous GDA research has used both plans (i.e., sequences of actions) [Molineaux *et al.*, 2010] and policies [Jaidee *et al.*, 2012].

Research in GDA has resulted in techniques to learn GDA knowledge automatically; this includes research to learn goals and goal formulation knowledge [Jaidee *et al.*, 2011] and learn explanations [Weber, 2012]. Researchers have also explored applying GDA for playing computer games [Weber *et al.*, 2010; Dannenhauer and Muñoz-Avila, 2013], for conducting naval operations, and for controlling robots [Roberts *et al.*, 2014] among others. Thus far, GDA work has not considered explicit models of the cost of sensing actions; examining the state is assumed to have no cost for the agent. Our work is the first to use GDA with an explicit model of partial observability that accounts for the cost of sensing actions. Furthermore, current GDA research assumes that enough information in the state is observable to plan ahead a sequence of grounded actions to achieve the goals. In our work we drop this assumption presenting a model that accounts for situations when such planning is not always possible (while at the same time not precluding this possibility).

There is a long-standing tradition of interleaving planning and execution [Goldman *et al.*, 1996]. Briefly, Sage was an early system interleaving planning and execution to enable the system to further plan when possible while, in parallel, sensing actions are executed [Knoblock, 1995]. This enables the agent to cope with information gathering tasks including query execution for data integration [Ives *et al.*, 1999]. Interleaving planning and execution has also been used for multi-agent systems architectures [Paolucci *et al.*, 1999]. More recently, there has been a renewed call about the importance of interleaving planning and execution [Ghallab *et al.*, 2014]. In those works, the agents gather information in order to achieve predefined goals. In contrast, GDA agents might change its goals.

3 Problem Definition

We define the problem of sensing in a partially observable and dynamic environment whereby a variety of goals can be pursued. First, if Q is the collection of all states in the world, then a collection of atoms s is a *partial state* if there is a state $q \in Q$ such that $s \subseteq q$ holds. The 6-tuple input to the guiding sensing problem is defined as $(S, \Sigma, s_0, G, \phi_g, c)$ and is composed of the following elements:

$\mathbf{S}$: The set of all partial states; if Q is finite, then S is finite too. $\mathbf{s_0}$: An initial partial state. $\mathbf{G}$: A collection of goals.

$\mathbf{\Sigma}$: Actions in the domain. $\Sigma = \Sigma_{plan} \cup \Sigma_{sense}$, where Σ_{plan} are planned actions in the domain (e.g., move the agent to a neighboring location from its current location). An action $a \in \Sigma_{plan}$ consists of the usual triple, $(prec(a), a^+, a^-)$ indicating the preconditions, positive effects and negative effects of a. Σ_{sense} consists of sensing actions; one, γ_τ, for every condition τ that may be satisfied in the environment. γ_τ returns $\{true, false\}$ depending if τ is a valid condition in the environment (e.g., checks if a specific beacon is activated).

$\phi_\mathbf{g}$: A heuristic function, one for each goal $g \in G$. It maps for every state, the next action to take, $\phi_g : S \rightarrow \Sigma_{plan}$. The heuristic function ϕ_g can be seen as encoding a strategy about how to achieve goal g. In our model the heuristic functions do not need to provide "hints" of actions to take that somehow circumvent the partial observability in the domain. However, our model does not preclude that possibility: if a plan π to achieve goal g is known, then the heuristic ϕ_g would simply pick the next action in π from the current state. It also does not preclude the case where we have a policy (i.e., a mapping from states to actions) indicating which action to take when visiting an state. In such a case when the goal changes, the plan/policy must change. In our empirical evaluation, we do not assume that we have knowledge about such plans/policies.

$\mathbf{c}$: Sensing Cost Function, $c : \Sigma_{sense} \rightarrow \mathbb{R}_{\geq 0}$. Returns a non-negative number for each sensing action.

The guiding sensing problem is defined as follows: given a guiding sensing problem $(S, \Sigma, s_0, G, \phi_g, c)$, generate a sequence of actions $\pi = (a_1 ... a_m)$, each $a_k \in \Sigma$, and a sequence of partial states $(s_0 ... s_m)$ such that:

1. If $\pi_{plan} = (a_{k1} ... a_{kn})$ denotes the subsequence of all planning actions in π (i.e., each $a_{kj} \in \Sigma_{plan}$), then the preconditions of each a_{kj} were valid in the environment at the moment when a_{kj} was executed.
2. One or more goals $g \in G$ hold in s_m.
3. If $\pi_{sense} = (a_{k1} ... a_{kz})$ denotes the subsequence of all sensing actions in π (i.e., each action in π_{sense} is of the form $\gamma_\tau()$ for some condition τ), then the total sensing cost $C(\pi) = \sum_{i=1}^{n} c(a_{ki})$ is minimal.

Condition 1 guarantees that the actions taken while the agent was acting in the environment were sound. Condition 2 guarantees that at least one of the goals is achieved. This condition is compatible with the special case of over-subscription planning [Smith, 2004], where the agent tries to achieve the maximum number of goals. It is also consistent with GDA where the agent chooses the goals to achieve as a result of situations encountered while acting on the environment. Condition 3 represents an ideal condition where the agent minimizes the cost of sensing while achieving its goals. In our work, we explore a variety of criteria (see next section) that affects the overall costs of the action trace pursued by the agent but our algorithms will provide no guarantees that Condition 3 is met.

4 GDA and Expectations

In Goal-driven autonomy (GDA) the agent repeatedly performs the following four steps (for further details please see an overview of GDA - e.g. [Aha, 2011]): (1) **Discrepancy detection:** after executing an action a, it compares the *observed state* o after sensing and the agent's expectation x

(i.e., it tests whether any constraints are violated, corresponding to unexpected observations). For example, in the marsworld domain an expectation is the atom *activated(beacon5)* and a discrepancy is the observed contradicting atom *deactivated(beacon5)*. If a discrepancy D is found (e.g., $D = x \setminus o$ is not empty), then the agent executes the following step. (2) **Explanation generation:** Given an observed state o and a discrepancy D, this step hypothesizes an explanation e causing D. (3) **Goal formulation:** This step generates a goal $g \in G$ in response to D, e, and o. (4) **Goal management:** Given a set of existing/pending goals $\hat{G} \subseteq G$ (one of which may be the focus of the current plan execution) and a new goal $g \in G$, this step may update $\hat{G}$ to create $\hat{G}'$ (e.g., $\hat{G}' = \hat{G} \cup \{g\}$) and will select the next goal $g' \in \hat{G}'$ to pursue.

The GDA cycle is triggered when discrepancies occur. In turn, discrepancies hinge on the notion of an agent's expectations. We explore 5 kinds of expectations from the literature [Muñoz-Avila *et al.*, 2010a; 2010b; Dannenhauer and Muñoz-Avila, 2015; Mitchell *et al.*, 1986], adapted to consider the guiding sensing problem. We use the following conventions: $\pi_{prefix} = (a_1 ... a_n)$ is the sequence of actions executed so far, $(s_0 ... s_n)$ are the sequence of partial states the agent believes to have visited so far, and $a_{n+1} \in \Sigma_{plan}$ is the next action to be executed.

1. *No expectations:* The agent performs sensing actions that are only related to the preconditions of a_{n+1} where $a_{n+1} \in \Sigma_{plan}$. For every precondition $\tau \in a_{n+1}$, the agent performs the sensing action of $\gamma_\tau()$.
2. *Immediate expectations:* The agent performs sensing actions for both the preconditions and effects of a_{n+1} where $a_{n+1} \in \Sigma_{plan}$.
3. *Eager expectations:* The agent checks if the belief state s_n is consistent with the current observed state and if s_{n+1} is consistent with the state observed after executing $a_{n+1} \in \Sigma_{plan}$.
4. *Informed expectations:* $Inf(\pi_{prefix}, s_0)$ move forward all valid conditions computed so far in π_{prefix}. Informed expectations are formally defined as follows: *Inf*$(\pi_{prefix}, s_0) = Inf(\pi_{prefix}, s_0, \emptyset)$
   - $Inf((), s, cc) = cc$.
   - $Inf((a), s, cc) = cc'$ if (1) $a \in \Sigma_{plan}$ and is applicable in s, then $cc' = (cc \setminus a^-) \cup a^+$, where a^- are the negative effects from a and a^+ are the positive effects from a.
   - $Inf((a_k a_{k+1} ... a_n), s_k, cc) = Inf((a_{k+1} ... a_n), s_{k+1}, Inf((a_k), s_k, cc))$.
5. *Goal-regression expectations:* Given a plan suffix $\pi_{suffix} = (a_{k+1} ... a_m)$ achieving a collection of goals G from some state, goal regression expectations *Regress*(π_{suffix}, G) is formally defined as follows:
   - $Regress((), cc) = cc$.
   - $Regress((a), cc) = (cc \setminus a^+) \cup precs(a))$, where $precs(a)$ are the preconditions of a.
   - $Regress((a_{k+1} ... a_m), cc) = Regress((a_{k+1} ... a_{m-1}), Regress((a_m), cc))$.

Checking for no expectations is typical of systems performing deliberative planning, where the agent's actions are not executed in the environment and therefore cannot fail. In a dynamic environment, actions may become invalid and hence an agent using no expectations is prone to fail to achieve its goals. Immediate expectations is an improvement in that the agent checks if the conditions for the next action to be applied hold in the observed state. But if earlier conditions are no longer valid, then the agent will be unaware (e.g., a beacon needed to achieve a goal condition has become deactivated). Hence, immediate expectations is also prone to fail to fulfill its goals.

Eager expectations check that all conditions in the agent's belief state are satisfied at every iteration. Hence, they are an improvement over the previous two kinds of expectations in that checking for eager expectations guarantees that if these conditions are valid, the plan is still valid (e.g., it will continue checking if a previously activated beacon, needed to achieve a goal, is still active). A drawback is that it may incur high sensing costs; it will also check for conditions that are not relevant for the current plan (e.g., any atom in the state, even if irrelevant to the current goal, will be checked and the agent will incur in the corresponding sensing costs). In contrast, goal regression expectations are an improvement in that they guarantee that the expectations computed, *Regress*(π_{suffix}, G), are minimal. That is, if any condition in *Regress*(π_{suffix}, G) is removed then some precondition in the suffix plan π_{suffix} is no longer applicable and therefore G cannot be fulfilled. The main drawback is that some of these conditions might become irrelevant if the agent needs to replan which is prone to occur in dynamic environments. Informed expectations addresses this limitation in that they move forward all conditions validated by the plan, π_{prefix}, executed so far. We state the following property implying advantages and disadvantages of informed expectations over goal regression expectations: *Let s_0 be a state, G a collection of goals and a plan $\pi = \pi_{prefix} \bullet \pi_{suffix}$ achieving G from s_0 (where $\bullet$ is the concatenation of the two plans). Under these conditions, if Inf$(\pi_{preffix}, s_0)$ are applicable in a state s_k, then Regress(π_{suffix}, G) is also applicable in s_k but not the other way around.*

This follows from the fact that *Regress*(π_{suffix}, G) computes the minimal conditions and our assumption that $\pi_{prefix} \bullet \pi_{suffix}$ achieves G from a state s_0. This result means that an agent checking for *Inf*(π_{prefix}, s_0) will check for unnecessary conditions assuming that there is no need to replan after executing π_{prefix}. On the other hand, if there is a need to replan to achieve the same goals G, then the informed expectations, $Inf(\pi_{prefix}, s_0)$, will compute the needed conditions regardless of how the plan is completed. In contrast, *Regress*(π_{suffix}, G) conditions might no longer be valid.

5 Goal Driven Autonomy Agent

Algorithm 1 shows the pseudo-code for our agent that is operating in a partially observable and dynamic environment. The main algorithm GDA starts on Line 18. The algorithm uses the following variables, initialized in Line 2: a plan π (initially empty), a collection of states (initially consisting of

Algorithm 1

```
1: Global Variables Ŝ, Σ, s_0, Ĝ, G, φ_g, X : S × Π → S
2: π ← (); Ŝ ← (s_0); g ← defaultGoal; Ĝ ← {g}
3: return GDA()
4:
5: procedure CHECK(x, s, π)
6:     D ← ∅; s' ← s                                   ▷ initialization
7:     for τ ∈ x do
8:         cond ← γ_τ()                                ▷ Sensing Action
9:         if (positive(τ) and not(cond)) or (negative(τ)
   and cond) then                                      ▷ Discrepancy!
10:            D ← D • (τ))
11:            π ← π • γ_τ()
12:            if positive(cond) then
13:                s' ← s' \ {τ}                       ▷ τ is not valid in s
14:            if negative(cond) then
15:                s' ← s' ∪ {τ}                       ▷ τ is valid in s
16:    return D
17:
18: procedure GDA()
19:    s ← lastState(Ŝ)
20:    if terminationCondition(Ĝ, s, Ŝ, π) then
21:        return (π, Ŝ)
22:    a ← φ_g(s)                                      ▷ selects applicable action
23:    D ← check(precs(a), s, π)
24:    if D == ∅ then
25:        execute(a)
26:        s ← apply(a, s)
27:        π ← π • (a)
28:        D ← check(X(s, π), s, π)
29:        if D == ∅ then
30:            Ŝ ← Ŝ • (s)
31:            return GDA()
32:        else Ŝ ← Ŝ • (s)
33:    else replace(Ŝ, s)
34:    E ← explain(D, s)                               ▷ There was a discrepancy
35:    Ĝ ← formulate_new_goals(G, Ĝ, D, E, s)
36:    g ← goal_selection(Ĝ, s)
37:    return GDA()
38:
39: procedure TERMINATIONCONDITION(Ĝ, s, Ŝ, π)
40:    g' ← goalsSatisfied(s, Ĝ)
41:    if g' ≠ ∅ then
42:        if check(g', s, π) then
43:            return true
44:        else return false
45:    else return false
```

the starting partial state s_0), a default goal, g, and a collection of goals that the agent is currently pursuing $\hat{G}$. It also uses a global variable, G with all potential goals that the agent might pursue. The algorithm also uses an expectation function, $X(s, \pi)$, as defined in the previous section (excluding goal regression expectations).

We did not implement goal regression in our algorithm because in the domains we tested our agent cannot generate a complete grounded plan from the outset due to partial observability.

The algorithm returns the pair $(\pi, \hat{S})$, where π is the trace of all planning and sensing actions executed and $\hat{S}$ all states the agent believes it visited (depending on the kind of expectations used, the agent may not have checked if every condition in the state is valid in the environment). The procedure begins by taking the last partial state, s, believed to be visited (Line 19). It first checks if the termination conditions are met in s. The *terminationCondition* procedure is detailed on Line 39 and is explained later. If the termination condition is not met, then the agent selects an applicable action a (based on the belief state s) to execute using the heuristic for the current goal (Line 22). It checks if the preconditions of a are valid in the environment by performing sensing actions (Line 23). The procedure *check* is detailed in Line 5. We will explain it later, but briefly, it will log in π any sensing action performed and it will modify s based on the discrepancies D it detected. If the preconditions are satisfied in the environment (Line 24), then the action is executed (Line 25), the belief state is moved forward by applying action a on s (Line 26). Action a is added into π (Line 27). Afterwards, it checks if the expectations (Line 28) are met in the environment (Line 29). If so, s is added into $\hat{S}$ and calls *GDA()* recursively (Lines 30-31). Otherwise, it adds s into $\hat{S}$ (Line 32; the procedure *check* updates s whenever there is a discrepancy).

If the preconditions are not satisfied, the last state in $\hat{S}$ is replaced with the updated state s (Line 33; calling the *check* function in Line 23 changes s based on the discrepancies detected). If there is a discrepancy either in the preconditions or in the expectations, Line 34 is reached. In it, the algorithm generates an explanation E for the discrepancy, formulates new goals $\hat{G}$ to achieve, selects a new goal g to pursue among those in $\hat{G}$ and calls GDA recursively (Lines 35-37).

We now discuss the auxiliary procedures *check* and *terminationCondition*. The *check* procedure (Line 5), receives as parameters the conditions x to be checked, the belief state s and the actions executed in the environment so far π. It checks for every atom τ in x if τ is sensed in the state (Line 8) while accounting for the fact if it is a positive or negative condition (Line 9). D maintains all discrepancies found (Lines 6 and 10). π is updated with any sensing actions performed (Line 11). The state is updated when there is a discrepancy (Lines 12-15). The procedure returns the discrepancies (Line 16). The auxiliary procedure *terminationCondition* (Lines 39-45) checks if the current goals g' (with $g' \subseteq \hat{G}$) are (1) satisfied in the belief state s (lines 40-41) and (2) satisfied in the environment (Line 42).

6 Experiments

To compare how the different kinds of expectations affects the agents' performance, we implemented two simulated environments, marsworld and blockscraft; both are partially observable and dynamic. Our hypothesis is that *informed* expectations will achieve all goals while having less sensing costs

than other expectations.

Figure 1: 1000 Scenarios per Agent in Marsworld

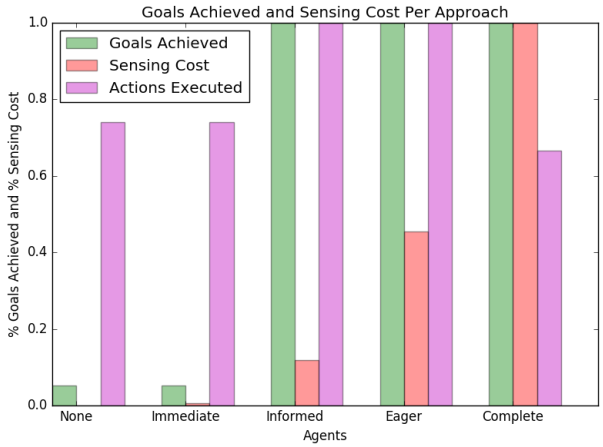
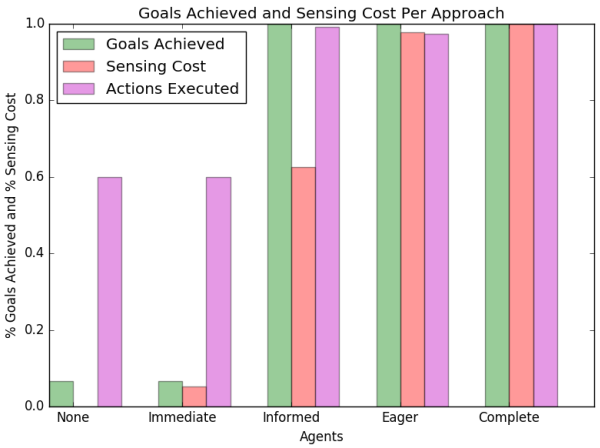


Figure 2: 1000 Scenarios per Agent in Blockscraft



Marsworld is a modified version of the domain described in [Dannenhauer and Muñoz-Avila, 2015]. Modifications were needed to make the domain partially observable; in that work the authors only deal with dynamic but not partially observable domains. It consists of a square grid of a 100 tiles, with randomly generated objects: beacons and piles of wood. The agent also begins with flares in its inventory. The agent has one overarching goal: signal a nearby agent for assistance. The agent creates a signal in one of three ways: activate a specified number of beacons, light a specified number of fires using wood, or drop and light a specified number of flares. When the agent is in the same tile as a beacon or a pile of wood, it can activate the beacon or create a fire. If the agent is in an empty location, it can drop and light a flare. Each of these may fail: beacons deactivate and fires and flares can be permanently extinguished. Only after the target number of activated beacons, fires, or flares have been reached can the agent signal for assistance. The agent is endowed with 7 plan actions: moveup, movedown, moveright, moveleft, activatebeacon, makefire, dropflare. The agent can sense anything in its current tile and any adjacent tiles (N,S,E,W) with a cost of 0. When an object is no longer within view, the agent can check on the object with a sensing action at a cost of 1. Hence, an agent can perform enough sensing to know everything it has seen, but at a high cost (i.e., the cost for each sensing action required to view everything in its belief state).

The **blockscraft** domain is inspired by the popular sandbox game Minecraft. In this domain the agent's goal is to build a 10-block tower by picking up and stacking blocks. Blocks can only be stacked on the ground or on top of blocks of the same type. The agent does not know what blocks will be available to it over the course of its execution. In the game Minecraft, often the player will dig for blocks and uncover different types of blocks, only known after acquiring them. The agent always has 3 blocks in a nearby quarry to choose from, and only after it uses a block will a new one become available. In our experiments there are three different types of blocks and each new block has a randomly selected type. Blockscraft is dynamic due to external agents, unseen to our agent, that may remove blocks from a tower as well as building their own towers. This is akin to the online multiplayer aspect of Minecraft, where many players can modify a shared world. Blockscraft is partially observable in that our agent only has a top-down view of the blocks. The top-down view enables sensing actions of cost 0 for the top two blocks of each tower it has built. Any other block (those under the top two) can be sensed with a cost of 1.

In each domain we generated 1000 random scenarios and ran five GDA agents (Figures 1 and 2). Four of the GDA agents implemented the four expectations described previously. The fifth GDA agent acts as an upper bound on our experiments to show what the behavior would be if the agent could sense the whole environment. In this way, we're able to observe what would happen if the agent can gain full observability of the environment with the same cost model as the other agents (we call this the *complete expectations* agent).

7 Results

In Figures 1-2, the first bar of each agent (green) is the percentage of goals achieved. The second bar (red) is the percentage sensing cost out of the maximum sensing cost. The maximum sensing cost is the sensing cost of all atoms in the state (as shown by our upper bound: complete expectations). The third bar (purple) shows the normalized total of actions executed by each agent. In Figure 1 the chance of failure per action executed was 20% for beacons, fires, and flares each. In Figure 2, the chance that a block would be removed was 10% and the chance that a block would be added was 30%. Chances for discrepancy to occur are computed per every planning action executed by the agent.

Looking at Figure 1, we see that agents using *none* and *immediate* expectations were unable to achieve most of their goals. Agents using *informed* and *eager* were able to achieve all of their goals. However, *informed* expectations incurred in significantly less sensing costs than *eager*, and less than the upper bound shown by *complete*. The *none* and *immediate* expectations agents do not become aware of failures outside their limited view and thus fail to switch goals, reaching

their (falsely believed) goal with less actions (compared to *informed* and *eager*). In these experiments we turned off the sensing checking of the goals (Line 42 of the Algorithm) as the *none* and *immediate* expectation agents were taking too long to complete their goals while the *informed*, *eager*, and *complete* agents are guaranteed to satisfy the goals they believe they achieved.

Figure 3: Sensing Cost per Failure Rate in Marsworld

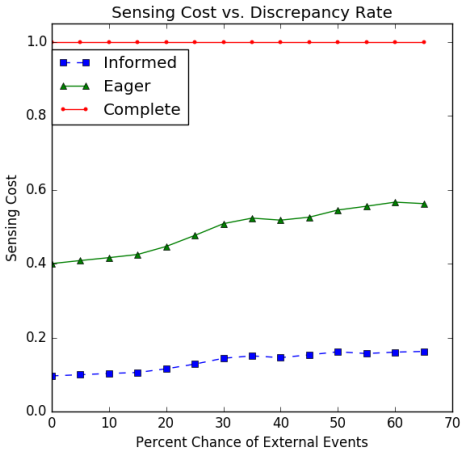


Figure 4: Sensing Cost per Failure Rate in Blockscraft

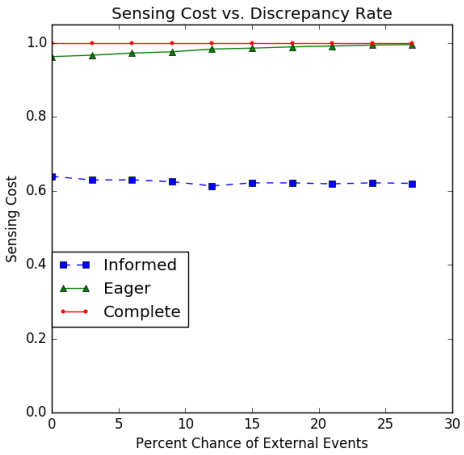


Figure 2 shows results from the blockscraft domain. We see similar results to those in marsworld. The none and immediate fail to achieve goals most of the time because even a change in a single block will go unnoticed. We see a cost gap in the *informed* and *eager* expectations as a result of the *eager* expectations agent sensing everything it has ever seen (including in this case the other towers under construction by other agents), while *informed* only keeps track of those atoms in the state that are related to its previous actions. Thus it performs sensing only on the blocks the agent itself has stacked.

Figures 3 and 4 show the total sensing (relative to complete expectations) varied by the chance of failure in each domain. Each data point is the total sensing cost performed out of the maximum possible sensing cost over 100 runs. In Figure 3 the chance for each of beacons, fire, and flares to fail varied from 0% to 65% in increments of 5%. Figure 4 shows the results of blockscraft where the chance that blocks were removed had a failure rate that varied from 0% to 27% in increments of 3%. In Figure 4, only the chance that blocks were removed was varied, the chance for blocks being added was held at 30%. We did not test with values close to 100% failure rate in both domains because at some point the environment changes so frequently it is not possible to achieve any goals, even with perfect sensing. By stopping at 65% and 27% respectively, we are able to see what is happening while still enabling agent's to achieve all of their goals. In both of these figures we see that informed expectations is performing substantially less overall sensing than eager and complete expectations. In blockscraft the difference between eager and complete is negligible because both agents basically see the same blocks regardless if used by the own agent or by the external agents.

8 Final Remarks

We present a formulation of the guiding sensing problem where sensing actions have associated costs for environments that are partially observable. Our formulation is amenable to, both, environments where GDA agents can plan ahead (e.g., as a sequence of grounded actions) and environments where this is not possible. We analyze trade-offs between five forms of expectations (i.e., none, immediate, eager, informed, and goal regression) used by GDA agents when dealing with dynamic environments. We presented an algorithm for a GDA agent operating in both partially observable and dynamic environments approximating a solution to the guiding sensing problem when planning ahead as a sequence of grounded actions is not possible. We evaluated our algorithm in two simulated environments. From this evaluation, we see that informed expectations performs the best among the four expectations (i.e., none, immediate, eager, informed) and using complete expectations (i.e., when full observability is enabled); informed expectations has less sensing costs compared to other expectations that achieve all goals.

For future work, we would like to explore solutions in the context of GDA agents, that either meet Condition 3 of the guiding sensing problem or provide a better approximation than informed expectations. We would also like to explore agents that can decide whether or not to perform sensing between each plan action and at the time of believed goal achievement. Essentially, these agents can vary the frequency of sensing (e.g. instead of sensing per each plan action, perform sensing every n actions). As such, Informed and Eager expectations would no longer guarantee that a believed to be true goal is actually true. Since goals must then be confirmed by additional sensing at the time of presumed goal completion, future work is needed to identify for which, if any, sensing frequencies there exists an improvement in minimizing overall sensing cost.

Acknowledgements: This work was supported in part by

grants NSF 1217888 and ONR N00014-15-1-2080

References

[Aha, 2011] Molineaux M. Klenk M Aha, D.W. Goal-Driven Autonomy. Technical report, NRL Review, 2011.

[Bonet and Geffner, 2006] Blai Bonet and Hector Geffner. Learning depth-first search: A unified approach to heuristic search in deterministic and non-deterministic settings, and its application to mdps. In *ICAPS*, volume 6, pages 142–151, 2006.

[Dannenhauer and Muñoz-Avila, 2013] D. Dannenhauer and H. Muñoz-Avila. LUIGi: A Goal-Driven Autonomy Agent Reasoning with Ontologies. In *Advances in Cognitive Systems (ACS-13)*, 2013.

[Dannenhauer and Muñoz-Avila, 2015] D. Dannenhauer and H. Muñoz-Avila. Raising Expectations in GDA Agents Acting in Dynamic Environments. In *International Joint Conference on Artificial Intelligence (IJCAI-15)*, 2015.

[Fikes and Nilsson, 1971] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[Ghallab *et al.*, 2014] Malik Ghallab, Dana Nau, and Paolo Traverso. The actors view of automated planning and acting: A position paper. *Artificial Intelligence*, 208:1–17, 2014.

[Goldman *et al.*, 1996] R Goldman, M Boddy, and Louise Pryor. Planning with observations and knowledge. In *AAAI-97 workshop on theories of action, planning and control*, 1996.

[Ives *et al.*, 1999] Zachary G Ives, Daniela Florescu, Marc Friedman, Alon Levy, and Daniel S Weld. An adaptive query execution system for data integration. In *ACM SIGMOD Record*, volume 28, pages 299–310. ACM, 1999.

[Jaidee *et al.*, 2011] Ulit Jaidee, Héctor Muñoz-Avila, and David W Aha. Integrated Learning for Goal-Driven Autonomy. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pages 2450–2455. AAAI Press, 2011.

[Jaidee *et al.*, 2012] Ulit Jaidee, Héctor Muñoz-Avila, and David W Aha. Learning and Reusing Goal-Specific Policies for Goal-Driven Autonomy. In *Case-Based Reasoning Research and Development*, pages 182–195. Springer, 2012.

[Kaelbling *et al.*, 1998] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

[Knoblock, 1995] Craig A Knoblock. Planning, executing, sensing, and replanning for information gathering. In *In Proceedings Of The Fourteenth International Joint Conference On Artificial Intelligence*, 1995.

[Knoblock, 1996] Craig A Knoblock. Building a planner for information gathering: A report from the trenches. In *In AIPS-96*. Citeseer, 1996.

[Mei *et al.*, 2005] Yongguo Mei, Yung-Hsiang Lu, Y Charlie Hu, and CS George Lee. A case study of mobile robot's energy consumption and conservation techniques. In *Advanced Robotics, 2005. ICAR'05. Proceedings., 12th International Conference on*, pages 492–497. IEEE, 2005.

[Mitchell *et al.*, 1986] Tom M Mitchell, Richard M Keller, and Smadar T Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine learning*, 1(1):47–80, 1986.

[Molineaux *et al.*, 2010] Matthew Molineaux, Matthew Klenk, and David W Aha. Goal-Driven Autonomy in a Navy Strategy Simulation. In *AAAI*, 2010.

[Muñoz-Avila *et al.*, 2010a] Héctor Muñoz-Avila, David W Aha, Ulit Jaidee, Matthew Klenk, and Matthew Molineaux. Applying Goal Driven Autonomy to a Team Shooter Game. In *FLAIRS Conference*, 2010.

[Muñoz-Avila *et al.*, 2010b] Héctor Muñoz-Avila, Ulit Jaidee, David W Aha, and Elizabeth Carter. Goal-Driven Autonomy with Case-Based Reasoning. In *Case-Based Reasoning. Research and Development*, pages 228–241. Springer, 2010.

[Paolucci *et al.*, 1999] Massimo Paolucci, Onn Shehory, Katia Sycara, Dirk Kalp, and Anandeep Pannu. A planning component for retsina agents. In *Intelligent Agents VI. Agent Theories, Architectures, and Languages*, pages 147–161. Springer, 1999.

[Roberts *et al.*, 2014] Mark Roberts, Swaroop Vattam, Ronald Alford, Bryan Auslander, Justin Karneeb, Matthew Molineaux, Tom Apker, Mark Wilson, James McMahon, and David W Aha. Iterative goal refinement for robotics. In *ICAPS Workshop on Planning and Robotics*, 2014.

[Shivashankar *et al.*, 2013] Vikas Shivashankar, UMD EDU, Ron Alford, Ugur Kuter, and Dana Nau. Hierarchical goal networks and goal-driven autonomy: Going where ai planning meets goal reasoning. In *Goal Reasoning: Papers from the ACS Workshop*, page 95, 2013.

[Smith, 2004] David E Smith. Choosing objectives in over-subscription planning. In *ICAPS*, volume 4, page 393, 2004.

[Weber *et al.*, 2010] Ben George Weber, Michael Mateas, and Arnav Jhala. Applying Goal-Driven Autonomy to StarCraft. In *AIIDE*, 2010.

[Weber *et al.*, 2012] Ben George Weber, Michael Mateas, and Arnav Jhala. Learning from demonstration for goal-driven autonomy. In *AAAI*, 2012.

[Weber, 2012] Ben Weber. *Integrating Learning in a Multi-Scale Agent*. PhD thesis, University of California, Santa Cruz, June 2012.