Integrated Learning for Goal-Driven Autonomy

Ulit Jaidee[1], Héctor Muñoz-Avila[1], David W. Aha[2]

[1]Department of Computer Science & Engineering; Lehigh University; Bethlehem, PA 18015
[2]Navy Center for Applied Research in AI; Naval Research Laboratory (Code 5514); Washington, DC 20375
ulj208@lehigh.edu | munoz@cse.lehigh.edu | david.aha@nrl.navy.mil

Abstract

Goal-driven autonomy (GDA) is a reflective model of goal reasoning that controls the focus of an agent's planning activities by dynamically resolving unexpected discrepancies in the world state, which frequently arise when solving tasks in complex environments. GDA agents have performed well on such tasks by integrating methods for discrepancy recognition, explanation, goal formulation, and goal management. However, they require substantial domain knowledge, including what constitutes a discrepancy and how to resolve it. We introduce LGDA, a learning algorithm for acquiring this knowledge, modeled as cases, that and integrates case-based reasoning and reinforcement learning methods. We assess its utility on tasks from a complex video game environment. We claim that, for these tasks, LGDA can significantly outperform its ablations. Our evaluation provides evidence to support this claim. LGDA exemplifies a feasible design methodology for deployable GDA agents.

1 Introduction

Agents that perform goal reasoning explicitly model and reason about the goals they try to achieve (Aha et al. 2010). For example, *goal-driven autonomy* (GDA) is a goal reasoning model in which agents continuously monitor the current plan's execution and assess whether the encountered states match expectations (Molineaux et al. 2010). When a GDA agent detects a state discrepancy (i.e., when the expected and actual states mismatch), it will consider whether to formulate new goals that, if achieved, would fulfill its overarching objectives such as maximizing a long-term reward.

There have been several recent contributions on goal reasoning, including research on goal management in cognitive architectures (Choi 2010), goal generation (Hanheide et al. 2010), and meta-reasoning (Cox, 2007). Applications have included simulated robots (Meneguzzi and Luck 2007), real-time strategy games (Weber et al. 2010), first-person shooters (Muñoz-Avila et al. 2010), and Navy training simulators (Molineaux et al. 2010).

To perform well, comprehensive GDA agents require substantial domain knowledge (e.g., to determine expected states, identify and explain discrepancies, formulate new goals, and manage pending goals). This requires, for example, experts to anticipate what discrepancies can occur, identify what goals can be formulated, and define their relative priority. However, few techniques have been investigated for learning this knowledge, and those that do learn only goal formulation knowledge (Weber et al. 2010; Powell et al. 2011). This can be problematic; while these agents may perform well in simple environments, in others a domain expert might not know the (state) expectations for executing every action in every state, nor which goal should be pursued to resolve every possible discrepancy, or even the space of all possible discrepancies.

We introduce Learning GDA (LGDA), a GDA algorithm that learns two types of cases from observed discrepancy resolution episodes: (1) *expectation* cases, which map state-action pairs to a distribution over expected states, and (2) *goal formulation* cases, which map goal-discrepancy pairs to a distribution of expected values over discrepancy-resolution goals. LGDA learns these through an integration of case-based reasoning (CBR) and reinforcement learning (RL) methods. It models goal formulation as an RL problem in which a goal's value is estimated based on the expected future reward for achieving it.

We claim that this integration can learn to perform as well as a non-learning GDA agent that employs expert knowledge, and can outperform GDA agents that use only CBR or RL. We report LGDA's comparative evaluation on a task involving the control of a team in a domination video game (DOM). The results show that LGDA outperforms most built-in hand-coded opponents (i.e., *adversaries*) and significantly outperforms its ablated versions. Finally, LGDA learns to perform almost as well as a non-learning GDA variant, whose case knowledge was hand-crafted by a domain expert such that it also significantly outperforms these adversaries (Muñoz-Avila et al. 2010).

2 Background

We briefly introduce the two processes that we will integrate to acquire knowledge for a GDA agent.

3.1 Case-Based Reasoning

CBR is a four-step learning process for solving new problems by adapting solutions to similar problems:

1. Retrieval: Given problem p, retrieve cases C from a library L (i.e., those relevant to solving p), where each case $c \in C$ is a problem-solution pair (p_c, s_c).
2. Reuse: Adapt C's solutions to derive a proposed solution s.
3. Revision: Apply s to p in real world (or simulated) scenarios. If needed, revise accordingly. This yields s'.
4. Retention: If revision s' successfully solves p, store the resulting experience as a new case (p, s') in L.

LGDA (§5) uses CBR to predict expected states and to formulate goals in response to detected discrepancies.

3.2 Reinforcement Learning

RL concerns the topic of how agents should select actions in an environment so as to maximize cumulative reward. RL agents learn through trial-and-error by interacting with the environment. Many RL algorithms exist, including Q-learning (Sutton and Barto 1998), which learns a Q-table that associates each (s,a) pair with an estimate of $Q(s,a)$, where here s is a state and a is an action. It employs an ε-greedy action selection strategy: in s, with probability $1-\varepsilon$ it selects the action a with highest $Q(s,a)$ and with probability ε it selects randomly. This prevents RL agents from getting stuck in a local optimum. Q-learning searches for a policy π that maximizes the sum of the returned rewards, where π maps states to actions. Each state s has a utility value $U(s)$. We define the reward between s and its successor s' as $U(s')-U(s)$. LGDA (§5) uses RL to learn expected values of goals to formulate, given the current goal and discrepancy.

3 Related Work

Several groups have studied integrations of CBR and RL. Bridge (2005) noted that these typically attempt to use the advantages of one to improve the other. For example, RL has been used to help CBR solve problems in continuous environments (Ram and Santamaria 1997; Molineaux et al. 2008) and to improve case retrieval (Juell and Paulson 2003). Analogously, CBR has been used to speed up the RL process (Gabel and Riedmiller 2007; Auslander et al. 2008; Bianchi et al. 2009) and to reduce RL's memory footprint (Dilts and Muñoz-Avila 2010). We instead integrate them to automatically acquire and reuse GDA knowledge.

Our work relates to planning in dynamic environments, which spawned *contingency planning* methods (Dearden et al. 2003), in which agents plan for plausible contingencies that may occur during plan execution. *Conformant planning* methods (Goldman and Boddy 1996) instead generate plans that are guaranteed to succeed. These methods require the a priori identification of possible contingencies. *Plan repair* methods instead adapt a plan's remaining actions whenever the state conditions required to execute the plan's next action are not satisfied (Fox et al. 2006). These agents cannot change their goals, while GDA agents instead dynamically reason about which goals they should achieve.

Our work focuses on the meta-process of goal reasoning. Some goal reasoning planners relax the requirement that the plan must achieve all of its goals. For example, *over-subscription planners* attempt to satisfy only a maximal subset of the goals (Van den Briel et al. 2004).

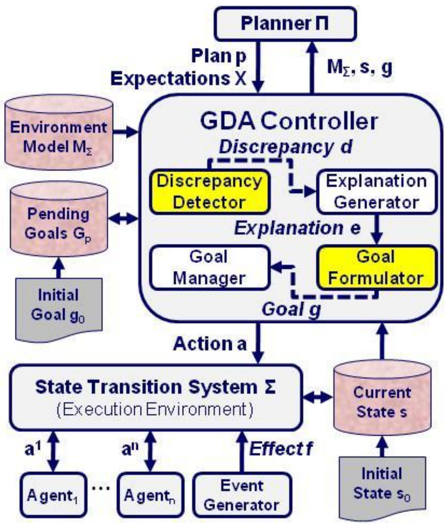

Figure 1: The GDA Conceptual Model

Cox's (2007) research on self-aware agents inspired the GDA model of goal reasoning. Most research on GDA assumes that experts provide domain knowledge on what to expect when an action is executed and which goal should be achieved next if a state discrepancy arises. The two exceptions are recent work on learning goal selection knowledge. First, Weber et al. (2010) uses CBR for this task, but doesn't learn about expectations. Their cases map discrepancies (between the current state and the goal the agent is trying to achieve) to new goals, which are represented as states, and their nearest neighbor algorithm compares the current state with recorded cases to perform goal selection. LGDA instead learns expectations, discrepancies, and goals. Furthermore, goals can be state abstractions (e.g., win the game) and LGDA could map a discrepancy to multiple goals. Second, Powell et al.'s (2011) active learner requires a user to indicate which goal to achieve next when discrepancies occur. In contrast our approach is fully automated.

4 Goal-Driven Autonomy

GDA (Figure 1) is a goal reasoning model that permits autonomous agents to dynamically self-select their goals during plan execution (Molineaux et al. 2010). This model extends the conceptual model of online classical planning (Nau 2007), whose components include a Planner, a Controller, and a State Transition System $\Sigma = (S,A,V,\gamma)$ with states S, actions A, exogenous events V, and state transition function γ: $S\times(A\cup V)\rightarrow 2^S$. In the GDA model, the Controller is centric: it receives as input a planning problem (M_Σ, s_t, g_t), where M_Σ is a model of Σ, s_t is the current state (e.g., initially s_0), and $g_t \in G$ is a goal that can be satisfied by some set of states $S_g \subseteq S$. It gives this problem to the Planner Π, which generates a sequence of actions $A_t = [a_t, \ldots, a_{t+n}]$ and a corresponding sequence of *expectations* $X_t = [x_t, \ldots x_{t+n}]$, where each $x_i \in X_t$ is a set of constraints that are predicted to hold in states $[s_{t+1}, \ldots, s_{t+n+1}]$ when executing A_t in s_t using M_Σ. The Controller sends a_t to Σ for execution and retrieves resulting state s_{t+1}, at which time it performs four knowledge-intensive tasks:

1. *Discrepancy detection*: This compares observations s_{t+1} with expectation x_t. If a discrepancy (i.e., unexpected observation) $d_t \in D$ is found, then explanation generation is performed to explain it.
2. *Explanation generation*: This explains a detected discrepancy d_t. Given also state s_t, it hypothesizes one or more explanations $e_t \in E$ of the cause.
3. *Goal formulation*: Resolving a discrepancy may warrant a change in the current goal(s). This task generates goal $g_t \in G$ given a discrepancy d_t, its explanation e_t, and current state s_t.
4. *Goal management*: New goals are added to the set of pending goals $GP \subseteq G$, which may also warrant other edits to GP. The Goal Manager will select the next goal $g_{t+1} \in GP$ to be given to the Planner. (It is possible that $g_t = g_{t+1}$.)

We focus on discrepancy detection and goal formulation here and will address the others steps in future work.

5 Learning Algorithm

5.1 Definitions

Let S be the set of states that an agent can visit, G the goals that an agent can pursue, and A the actions that can be executed. We define an ***expectation case base*** (ECB) as a mapping $S \times A \rightarrow 2^{S \times [0,1]}$ from the current state and selected action to a probability distribution over expected next states (i.e., actions can be non-deterministic). LGDA clusters ECB cases that involve the same action and have similar states, and learns a state probability distribution for each cluster.

In LGDA (see §5.2), Get(ECB,s,a) returns, as the expected state, the one with maximal probability among the ECB cluster $\chi_{s,a}$ whose state is most similar to s and has the same action a. If no such cluster exists, then Update(ECB,s,a,s') will create one. Otherwise, it will update the probability distribution for $\chi_{s,a}$.

A ***goal formulation case base*** (GFCB) instead maps the current goal g_t and discrepancy d_t into a distribution over the expected values, $G \times D \rightarrow 2^{G \times [0,1]}$, for formulated goals. That is, multiple goals g' may be formulated to resolve d when pursuing g. Cases with the same goal and similar discrepancies are clustered together in GFCB. LGDA uses Q-learning to track the expected value for each g'. In LGDA (§5.2), Get(GFCB,g,d,g') returns the expected value q from cluster $\chi_{g,d}$ when g' is formulated. If no such cluster exists, it returns 0 and initializes $\chi_{g,d}$. Function call Update(GFCB,g,d,g',q') updates the value of q for g' in $\chi_{g,d}$.

LGDA receives as input the policies Π: $G \times S \rightarrow A$ that are implemented by hard-coded *adversaries*. The call $\Pi(g)$ returns the policy π for G, while π(s) returns the action that is pursued in state s (if multiple such actions exist, one is randomly selected). These input policies are static, not learned. LGDA instead learns the case bases (1) ECB and (2) GFCB. This learned knowledge is dynamic; their application varies based on the environment's state in which the actions are executed.

5.2 LGDA Algorithm

Expectations can be learned by recording the occurrence frequency of *(s,a,x)* triples. The interpretation of the most frequent triple *(s,a,x)* among those in (s, a) in the ECB is that, when s is the current state, it is most likely that x will be the next state when a is executed. We use s, s', and x for the previous, current, and expected states, respectively

Let g, g', and g'' be the previous, current, and next goals, respectively. LGDA uses Q-learning to learn the values of goals in each GFCB cluster. The following pseudocode provides details and is documented below.

```
LGDA($s_0$,$g_0$,$d_0$,$\Delta$,ECB,GFCB,$\alpha$,$\gamma$, $\epsilon$, $G$,$\Pi$) =
  1: $s \leftarrow s_0$; $x \leftarrow s_0$; $a \leftarrow \emptyset$; $g \leftarrow g_0$; $g' \leftarrow g_0$; $d \leftarrow d_0$
  2: While the game-playing episode continues
  3:    wait($\Delta$); $s' \leftarrow$ GETSTATE()
  4:    ECB $\leftarrow$ UPDATE(ECB, $s, a, s'$)
  5:    $d \leftarrow$ CALCULATEDISCREPANCY($s'$, $x$)
  6:    $q \leftarrow$ GET(GFCB,$g$, $d$, $g'$)
  7:    $r \leftarrow U(s') - U(s)$
  8:    $q' \leftarrow q + \alpha\,(r + \gamma\,$ARGMAX$_{gl}$(GET(GFCB,$g$,$d$,$gl$)) - $q$)
  9:    GFCB $\leftarrow$ UPDATE(GFCB,$g$, $d$, $g'$, $q'$)
 10:    if $r < 0$
 11:       if RANDOM(1) $\geq \epsilon$
 12:             $g'' \leftarrow$ ARGMAX$_{gl}$(GET(GFCB,$g$,$d$,$gl$))
 13:       else  $g'' \leftarrow$ RANDOM($|G|$)
 14:       $g \leftarrow g'$; $g' \leftarrow g''$
 15:    $\pi \leftarrow \Pi(g')$;  $a' \leftarrow \pi(s')$
 16:    $x \leftarrow$ GET(ECB, $s'$, $a'$)
 17:    EXECUTEACTION($a'$)
 18:    $s \leftarrow s'$; $a \leftarrow a'$
 19: return ECB, GFCB
```

LGDA initializes previous state $\boldsymbol{s}$ to the initial state $\boldsymbol{s_0}$, action $\boldsymbol{a}$ to the null action, previous goal $\boldsymbol{g}$ and current goal $\boldsymbol{g'}$ to the dummy goal $\boldsymbol{g_0}$, and discrepancy $\boldsymbol{d}$ to dummy value $\boldsymbol{d_0}$ (Line 1). Entries with these dummy values are not added to the case base, and will be assigned to non-dummy values after the algorithm's first iteration. During a game-playing episode (Line 2), LGDA periodically waits and observes the current state $\boldsymbol{s'}$ (Line 3), which it uses to update the distribution of expected states when taking action $\boldsymbol{a}$ in $\boldsymbol{s}$ (Line 4). It then calculates the discrepancy $\boldsymbol{d}$ between the current and expected states (Line 5) and uses it to retrieve GFCB's estimated $\boldsymbol{q}$ value for formulating $\boldsymbol{g'}$ (Line 6).[1] It then computes the reward (Line 7), updates the $\boldsymbol{q}$ value using the Q-learning formula (Line 8), and records it in the GFCB (Line 9). If the agent is performing poorly (Line 10), LGDA retrieves a new goal g'' from GFCB using ε-greedy exploration (Lines 11-13), and updates its previous

[1] Although not shown, if no discrepancy exists, the goal does not change.

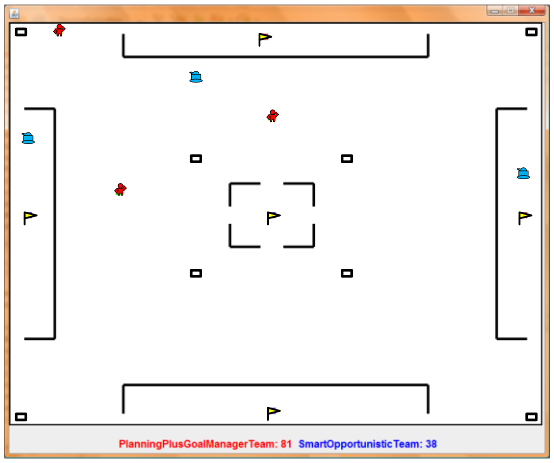

Figure 2: A DOM game map with 5 domination locations (yellow flags), where rectangles identify the agents' respawning locations, and the remaining icons denote each player's agents.

and current goal (Line 14). LGDA then retrieves the next action $\boldsymbol{a'}$ using policy $\boldsymbol{\pi}$, where π is the policy in $\boldsymbol{\Pi}$ for goal g′. (Line 15), retrieves an expected state $\boldsymbol{x}$ from the ECB (Line 16), executes $\boldsymbol{a'}$ (Line 17), and updates previous state $\boldsymbol{s}$ and action $\boldsymbol{a}$ (Line 18). Finally, when the game-playing episode ends, it returns the revised case bases.

6 Implementation and Example

An LGDA agent must determine how to cluster cases using a similarity metric and re-cluster when necessary. Our implementation was inspired by the design of Retaliate (Smith et al. 2007), an RL agent that we will use for benchmarking. For LGDA, we ensure that the states S and actions A represented in ECB and GFCB, as well as the state utility U, are the same as those in Retaliate. Theoretically, this permits a fair comparison between LGDA and Retaliate.

Retaliate was applied to control one team's actions in a domination game called *DOM* (see Figure 2), in which two competing teams attempt to capture specified *domination locations* on a two-dimensional map. Teams are composed of k *bots*. The teams' actions are k-tuples $(l_1,..l_k)$ indicating the domination location l_i to which each bot b_i is assigned. Domination locations are captured by a team when one of its bots moves into its location and no bots from the other team arrive within five ticks. A team receives a point for every five seconds that it owns a domination location. The first team to earn a predefined number of points wins. Each bot starts with a max number of health points, which can be lost in combat, which occurs when two or more opposing bots are within a certain range of each other. Combat is solved using a biased random function that determines the health points lost by each competing bot (it favors bots on the team that has more bots within range). When a bot's health is zero, it is removed, *respawns* after a few ticks in a (randomly-selected) respawning location, and continues with its initial max health points.

Retaliate selects the actions for the friendly team's bots. Its state representation includes only information on domination location ownership. The **state** is a vector $(l_1,l_2...,l_d)$, where d is the number of domination locations and l_k indicates the team which owns location k. For a 2-team game and b bots per team, this reduces the number of states to d^3 and the space of actions to $(2b)^d$. The *utility* U of state s is defined by the function $U(s) = F(s)–E(s)$, where $F(s)$ is the friendly team's score and $E(s)$ is the enemy's score. The **discrepancy** between states s and s' is a d-dimensional vector $(v_1,v_2...,v_d)$, where v_i is true if s and s' have the same value in coordinate i and false otherwise.

Given this representation, LGDA's cases implement a simple similarity metric: two states are deemed similar if they have the same feature values for domination location ownership. Analogously, two discrepancies are similar if they mismatch on the same features. Thus, after a case is assigned to a cluster, it will never be reassigned.

Example. Suppose the domination locations in the current game are $(l_1,\ l_2,$ and $l_3)$ and there are three bots per team (b_1, b_2, b_3). Each location l_i can be in one of the three states: un-owned (U), owned by the friendly team (F), or owned by the enemy (E). Therefore, state (E,F,F) denotes that the first domination location is owned by the enemy and the others are owned by the friendly team. Suppose the previous state s is (U, E, U), the current state s' is (F, E, U), the expected state x is (F, F, U), and the friendly team's previous actions were $(b_1 \rightarrow l_1, b_2 \rightarrow l_2, b_3 \rightarrow l_1)$. After updating the relevant ECB distribution (Line 4), LGDA will calculate the discrepancy d between the current and expected states (Line 5). Here, d is (true, false, true), where *true* means they match. After calculating the q value and updating the $GFCB$ (Lines 6-9), suppose the reward is negative, and that LGDA will retrieve/formulate a new goal. Suppose the current goal g' is *to control the first half plus one of the domination locations*, and the next goal g'' that was retrieved from the GFCB is *to control all domination locations*. Then the action a' that will be retrieved from policy π will be $(b_1 \rightarrow l_1, b_2 \rightarrow l_2, b_3 \rightarrow l_3)$. The expectation x that it retrieves from GFCB for executing action a is (F, F, F) (Line 16). LGDA will then execute a' and record the new values for the previous state s and action a (Line 18).

7 Empirical Study

7.1 Experimental Setup

We used the task of winning DOM games to investigate two hypotheses: (**H1**) LGDA can learn to perform as well as a non-learning GDA agent that employs expert knowledge, and (**H2**) LGDA can significantly outperform ablated agents that use only RL or only CBR, respectively.

We also used six hand-coded **adversaries** as baselines. Munoz-Avila et al. (2010) describe each except *Priority*, which prefers to send bots to those locations that are owned by its opponent. Briefly, these adversaries pursue a unique goal g_i to play DOM. Their behavior is *approximately* modeled using a policy π_i. That is, while the first three adversaries are easy to defeat, the latter three cannot be perfectly represented as policies based on our models for S

and *A* because they reason about the proximity of bots to locations. Proximal information is not represented by any of the four agents we tested. Thus, the latter three adversaries pose difficult challenges for the agents.

We compared LGDA versus the following **agents**: Retaliate, which performs Q-learning, the ablation Random GDA (RGDA), which replaces LGDA's ε-greedy goal selection procedure with a random selection procedure, and CB-gda, a non-learning CBR agent whose case bases were manually crafted by a domain expert (Muñoz-Avila et al. 2010). It includes two case bases, whose mappings are:

$$\text{PCB: } G \times S \times A \rightarrow \text{S, and MCB: } G \times D \rightarrow G$$

PCB records the expected state for each (goal, state, action) tuple, and MCB records the preferred goal to formulate for each (goal, discrepancy) pair. All agents (CB-gda, LGDA, Retaliate, and RGDA) use the same model for *S* and *A*. The learning agents (the latter three) use the same utility function *U*. The definitions for *S*, *A*, and *U* are given in §6.

Games are won by the first team to reach 2000 points on Figure 2's map. There were 8 bots per team, which is typical (i.e., there are usually more team members than domination locations). Scores were averaged over 10 games. In our first study, we indirectly compared the agents by testing them against the six hard-coded adversaries using a LOOCV method: we trained each learning agent versus five adversaries, using four repetitions per starting state, and tested it against the remaining adversary. The second study addresses our hypotheses: it directly compares LGDA versus the other agents. We trained each learning agent versus the six adversaries. LGDA received as input the policies for the six adversaries but not the policies for the other agents. We recorded results before and after each training repetition of LGDA versus each of the three agents, continuing until their relative performance stabilized. Knowledge learned during testing was flushed between games. Our metric is state utility, as defined in §6.

7.2 Results

Experiment 1 (Table 1): CB-gda recorded the best performance among the agents; it outperformed all of the adversaries, although barely so versus EachBotToOneDom, which is the strongest of the hard-coded adversaries. This adversary maintains at least one bot in each location. LGDA outperformed five of the opponents, losing only to EachBotToOneDom. In contrast, Retaliate and RGDA performed poorly versus all three of the difficult adversaries. This provides initial evidence that LGDA performs comparatively well compared to its ablations but it is outperformed by CB-gda. Our next experiment provides strong support for these observations.

Experiment 2 (Figure 3): LGDA is outperformed by CB-gda. The mean of the underlying distribution for their relative utility values after training was -14.6 ± 2.6 at the 95% confidence level. Thus, H1 is ***not*** supported, though LGDA's final performance is fairly close. This is not too surprising, given that CB-gda's case bases were manually encoded by a domain expert.

Table 1: Average Utility Results from Experiment 1

Adversary ↓	**CB-gda**	**Retaliate**	**RGDA**	**LGDA**
Dom1Hugger	77.38	74.26	71.36	61.38
FirstHalf	75.47	58.88	74.23	64.91
SecondHalf	65.36	65.79	66.28	63.03
SmartOp	54.85	-10.62	-36.59	45.27
EachBotToOne	0.46	-47.13	-68.48	-50.11
Priority	45.14	-6.37	-45.28	23.08
Learning method	None	RL	CBR	CBR+RL

LGDA is initially outperformed by Retaliate because Retaliate quickly converges to an action that on average works well versus the adversaries. In contrast, LGDA needs to learn expectations and best goals to pursue when discrepancies occur. This results in a slower learning process in part because the interdependency between expectations and discrepancies. Over time we see that it pays off; LGDA eventually outperforms Retaliate. LGDA outperforms RGDA from the outset. Versus Retaliate, the same analysis reveals a mean of 34.3 ± 4.1 at the 95% confidence level, and the mean (at this level) versus RGDA was 62.6 ± 2.4. Thus, these results ***strongly support*** H2.

8 Conclusions

We introduced LGDA, a goal-driven autonomy agent that automatically acquires state expectation and goal selection knowledge. LGDA uses a case-based reasoning method to map (state,action) pairs to a distribution over expected states, and (goal,discrepancy) pairs to a value distribution over discrepancy-resolution goals. It also uses a reinforcement learning method to learn the goals' expected values. LGDA is the first GDA agent to automatically learn this knowledge, and the first to model these two distributions. Our ablation study demonstrated that, for a complex first-person shooter gaming task, LGDA significantly outperforms ablations that use only one of its two learning methods. LGDA also learns to play nearly as well as an expertly-engineered GDA agent.

We conjecture that the expertly-engineered GDA agent is outperforming LGDA because LGDA lacks capabilities to learn explanations of discrepancies and could benefit from a more mature goal management strategy. We will study this issue, related GDA extensions, and conduct more extensive evaluations in our future research.

Acknowledgements

This work was supported in part by NSF grant 0642882.

References

Aha, D.W., Klenk, M., Muñoz-Avila, H., Ram, A., & Shapiro, D. (Eds.) (2010). *Goal-Directed Autonomy: Notes from the AAAI Workshop* (W4). Atlanta, GA: [http://www.cse.lehigh.edu/~munoz/gda/]

Auslander, B., Lee-Urban, S., Hogg, C., and Muñoz-Avila, H. (2008). Recognizing the enemy: Combining reinforcement learning with strategy selection using case-based reasoning. *Proceedings of the Ninth European*

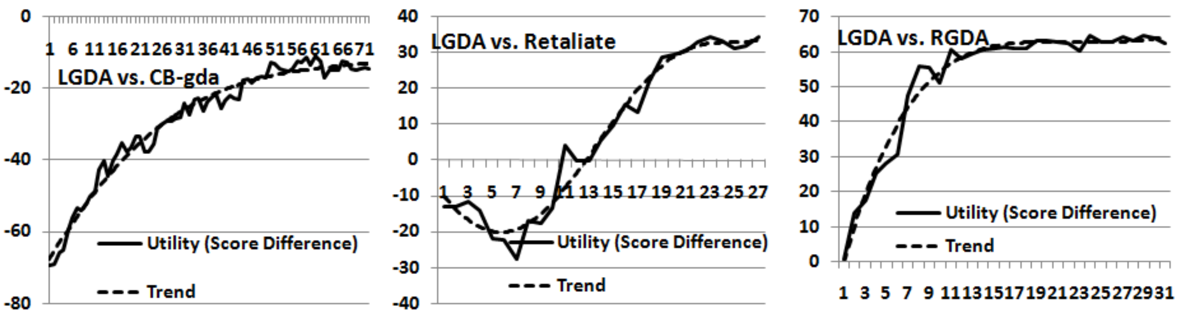

Figure 3: Results from Experiment 2: Average learning curves for comparing LGDA DOM performance vs. non-learning and ablated agents. The trend lines were generated using a polynomial fit to the raw curves.

Conference on Case-Based Reasoning (pp. 59-73). Trier, Germany: Springer.

Bianchi, R., Ros, R., & Lopez de Mantaras, R. (2009). Improving reinforcement learning by using case-based heuristics. *Proceedings of the Eighth International Conference on CBR* (pp. 75-89). Seattle, WA: Springer.

Bridge, D. (2005). The virtue of reward: Performance, reinforcement and discovery in case-based reasoning. *Proceedings of the Sixth International Conference on Case-Based Reasoning* (pp. 1). Chicago, IL: Springer.

van den Briel, M., Sanchez Nigenda, R., Do, M.B., & Kambhampati, S. (2004). Effective approaches for partial satisfaction (over-subscription) planning. *Proceedings of the Nineteenth National Conference on Artificial Intelligence* (pp. 562-569). San Jose, CA: AAAI Press.

Choi, D. (2010). Reactive goal management in a cognitive architecture. In (D.W. Aha et al. 2010).

Cox, M.T. (2007). Perpetual self-aware cognitive agents. *AI Magazine*, **28**(1), 23-45.

Dearden R., Meuleau N., Ramakrishnan S., Smith, D., & Washington R. (2003). Incremental contingency planning. *Planning Under Uncertainty and Incomplete Information: Papers from the ICAPS Workshop*.

Dilts, M., & Muñoz-Avila, H. (2010). Reducing the memory footprint of temporal difference learning over finitely many states by using case-based generalization. *Proceedings of the Eighteenth International Conference on Case-Based Reasoning* (pp. 81-95). Alessandria, Italy: Springer.

Fox, M., Gerevini, A., Long, D., & Serina, I. (2006). Plan stability: Replanning versus plan repair. *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling* (pp. 212-221). Cumbria, UK: AAAI Press.

Gabel, T., & Riedmiller, M. (2007). An analysis of case-based value function approximation by approximating state transition graphs. *Proceedings of the Seventh International Conference on Case-Based Reasoning* (pp. 344-358). Belfast, Northern Ireland: Springer.

Goldman, R.P., & Boddy, M.S. (1996). Expressive planning and explicit knowledge. *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems* (pp. 110-117). Edinburgh, Scotland: AAAI Press.

Hanheide, M., Hawes, N., Wyatt, J., Göbelbecker, M., Brenner, M., Sjöö, K., Aydemir, A., Jensfelt, P., Zender, H., and Kruijff, G-J. (2010). A Framework for Goal Generation and Management. In (D.W. Aha et al. 2010).

Juell, P., & Paulson, P. (2003). Using reinforcement learning for similarity assessment in case-based systems. *IEEE Intelligent Systems*, **18**(4), 60–67.

Meneguzzi, F.R., & Luck, M. (2007). Motivations as an abstraction of meta-level reasoning. *Proceedings of the Fifth International Central and Eastern European Conference on Multi-Agent Systems* (pp. 204-214). Leipzig, Germany: Springer.

Molineaux, M., Klenk, M., & Aha, D.W. (2010). Goal-driven autonomy in a Navy strategy simulation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. Atlanta, GA: AAAI Press.

Muñoz-Avila, H., Jaidee, U., Aha, D.W., & Carter, E. (2010). Goal directed autonomy with case-based reasoning. *Proceedings of the Eighteenth International Conference on Case-Based Reasoning* (pp. 228-241). Alessandria, Italy: Springer.

Nau, D.S. (2007). Current trends in automated planning. *AI Magazine*, **28**(4), 43-58.

Powell, J., Molineaux., M., & Aha, D.W. (2011). Active and interactive discovery of goal selection knowledge. To appear in *Proceedings of the Twenth-Fourth Conference of the Florida AI Research Society*. West Palm Beach, FL: AAAI Press.

Ram, A., & Santamaria, J.C., (1997). Continuous case-based reasoning. *Artificial Intelligence*, **90**(1-2), 25-77.

Smith, M., Lee-Urban, S., & Muñoz-Avila, H. (2007). RETALIATE: Learning winning policies in first-person shooter games. *Proceedings of the Nineteenth Innovative Applications of AI Conference* (pp. 1801-1806). Vancouver, British Columbia, Canada: AAAI Press.

Sutton, R.S., & Barto, A.G. (1998). *Reinforcement learning: An introduction*. MIT Press, Cambridge, MA.

Weber, B., Mateas, M., & Jhala, A. (2010). Applying goal-driven autonomy to StarCraft. In *Proceedings of the Sixth Conference on Artificial Intelligence and Interactive Digital Entertainment*. Stanford, CA: AAAI Press.