Learning and Reusing Goal-Specific Policies 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. In certain adversarial environments, reinforcement learning (RL) techniques require a prohibitively large number of episodes to learn a high-performing strategy for action selection. For example, Q-learning is particularly slow to learn a policy to win complex strategy games. We propose GRL, the first GDA system capable of learning and reusing goal-specific policies. GRL is a case-based goal-driven autonomy (GDA) agent embedded in the RL cycle. GRL acquires and reuses cases that capture episodic knowledge about an agent's (1) expectations, (2) goals to pursue when these expectations are not met, and (3) actions for achieving these goals in given states. Our hypothesis is that, unlike RL, GRL can rapidly fine-tune strategies by exploiting the episodic knowledge captured in its cases. We report performance gains versus a state-of-the-art GDA agent and an RL agent for challenging tasks in two real-time video game domains.

Keywords: Case-based learning, reinforcement learning, goal-driven autonomy

1 Introduction

Reinforcement learning (RL) algorithms attempt to optimize an agent's behavior when interacting in an environment. The agent's behavior is defined by a policy π, which maps for every state s and action a the value $\pi(s,a)$ of selecting a in s. The optimization function is frequently defined as the summation of future rewards, which in our context of adversarial games can be defined as the difference in score between the agent and its opponents. Thus, an RL agent seeks to maximize this difference.

A difficulty arises in situations where the adversaries frequently change their strategies or, equivalently, when each adversary has a fixed and unique strategy and adversaries are frequently changed without divulging their identity to the RL agent. In such situations, the RL agent will learn a "maximum common denominator" strategy that is optimal regardless of the opponent. Unfortunately, as evidenced in our empirical study (Section 6), either (1) this strategy performs poorly or (2) learning it requires a prohibitively large number of episodes.

To address this difficulty, we propose to use case-based reasoning (CBR) techniques to augment RL's cycle by acquiring and reusing cases that capture episodic knowledge about an agent's (1) expectations, (2) goals to pursue when discrepancies exist (i.e., where these expectations are not met), and (3) actions for achieving these goals in given states. Agents that reason with expectations, discrepancies, and goals are the focus of *goal reasoning*.

In this paper, we introduce GRL (Goal Reasoning Learner), a case-based Goal-Driven Autonomy agent. 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 (Molineaux *et al.*, 2010).Unlike previous work integrating RL and CBR and work on GDA (see Section 2), GRL embeds GDA in an RL cycle by learning and reusing the three kinds of cases mentioned above. GRL is the first GDA agent capable of learning and reusing goal-specific policies. Our hypothesis is that, as a result of this capability, GRL can fine-tune strategies by exploiting the episodic knowledge captured in its cases. We report performance gains versus a state-of-the-art GDA agent and an RL agent for challenging tasks in two real-time gaming domains.

Section 2 discusses related work. Sections 3 and 4 then describe GRL's GDA instantiation and its detailed algorithm. Section 5 presents an example of GRL's behavior. We present our empirical studies in Section 6 and conclude in Section 7.

2 Related Work

GDA agents use a four-step strategy to respond competently to unexpected situations in their environment: (1) detect any discrepancy between the observed state and the expected state(s), (2) explain this discrepancy, (3) formulate a goal to resolve it (if needed), and (4) manage this new goal along with its pending goals (Molineaux *et al.*, 2010; Muñoz-Avila *et al.*, 2010). In step 3, these agents use a variety of models to formulate new goals. For example, Intro (Cox, 2007) uses explanation patterns represented as *cause* $\rightarrow$ *effect* rules such that, if a state is judged to be a discrepancy and it maps to the effects of a rule, then Intro will select the negation of that rule's cause as its new goal. ARTUE (Molineaux *et al.*, 2010) uses rule-based reasoning for goal formulation and ranking (i.e., pending goals are maintained in a priority list). Its rules encode expert knowledge in a manner similar to Intro's rules, but ARTUE adds a more robust process by encoding planning dependencies in a truth-maintenance system. EISBot (Weber *et al.*, 2010b) instead uses a case-based model to formulate goals, where a case c_i=($c_{i,1}$,…,$c_{i,n}$) is an expert-provided sequence of states for accomplishing a task, and states are represented as a vector of numeric values. Given current state c, EISBot retrieves a most similar state $c_{i,j}$ in its case base along with $c_{i,j+w}$, where w is the length of its planning window. It computes the difference $c_{i,j+w}$-$c_{i,j}$ and adds this to c to define its new goal. In contrast to these GDA agents, GRL *learns* its goal formulation knowledge.

T-ARTUE (Powell *et al.*, 2011) is an extension of ARTUE that interactively learns goal formulation knowledge; it can query the user to ask for new goals or confirm their formulation, and the user can provide feedback on these decisions. In contrast, GRL *automatically* learns goal formulation knowledge and new goals.

Most GDA agents (e.g., CB-gda (Jaidee *et al.*, 2010)) are given knowledge about state expectations, discrepancies, goals to achieve, and the means to achieve the goals (e.g., the plans or the policies). While some prior work has focused on learning some of these, GRL is the first GDA agent to learn ***all*** of them simultaneously. Finally, in contrast to most prior GDA work, GRL learns and reasons with *stochastic* expectations, which we define in Section 3.

Agents can compute state expectations using action models (i.e., their preconditions and effects) and the current state. Bouguerra *et al*. (2008) use description logics to model and infer expectations after executing a plan, which is particularly useful for partially observable environments. For example, an agent might observe John entering a vehicle at a location *A* and the vehicle later arriving at location *B*, where its occupants departed. Given this, it could infer that John arrived at *B*. GDA agents vary in how they compute expectations, including using a model of abstract explanation patterns (Cox, 2007), or by defining discrepancy detectors to trigger when state expectations fail (Weber *et al.*, 2010a). Unlike these (and most other) GDA agents, GRL *learns* its action models for computing state expectations. The only related agent is LGDA (Jaidee *et al.*, 2011a), which learns action models it uses to compute expectations but it assumes that the policies and goals are given as input. In contrast, ***GRL identifies new goals,*** and ***it learns and reuses goal-specific policies***.

There is substantial interest in integrating CBR and RL, as exemplified by Derek Bridge's ICCBR-05 invited talk on potential synergies between CBR and RL (Bridge, 2005), the SINS system that solves problems in continuous environments (Ram & Santamaria, 1997), and CBRetaliate, which stores and retrieves Q-tables (Auslander *et al*., 2008). Most previous contributions focused on improving the performance of an agent by exploiting synergies among CBR and RL or by enhancing the CBR process by using RL (e.g., to improve similarity metrics). More recently, researchers have studied ways in which CBR can improve reinforcement learning. This includes reducing the memory requirements of RL (Dilts & Munoz-Avila, 2010), using cases as a heuristic to speed up the RL process (Bianchi *et al.*, 2009) and using cases to approximate state value functions in continuous spaces (Gabel & Riedmiller, 2005; 2007). GRL falls in this latter category; it uses CBR to fine-tune strategies by exploiting the episodic knowledge captured in the cases while embedded in the RL cycle. In this context, GRL's novelty is that it automatically identifies goals, learns policies specific to those goals, learn expectations about the action's outcomes, and reasons when a discrepancy occurs.

3 Case-Based Goal-Driven Autonomy with GRL

Like other GDA agents, GRL conducts a meta-process for online planning (Nau, 2007). Figure 1 illustrates GDA's information flow, which naturally embeds the standard RL model (Sutton & Barto 1998), where the objective is to maximize the expected return. The return is a function of the rewards obtained. For example, the return can be defined as the summation of the future rewards. Like RL, GRL executes an action a in the environment and observes from the environment the next state s' and a reward r. Unlike RL, which selects the next action based on the current policy (i.e., a mapping $S \rightarrow 2^{A\times[0,1]}$ from states to a probability distribution over the actions), GRL selects an action based on the current goal g, the policy π for that goal, and the current state s. A crucial challenge is that, in many environments, there is no optimal policy for all situations. For example, in an adversarial game, a policy might be optimal versus one opponent but not others. This in part motivates why GDA agents reason with expectations, discrepancies, and goals.

GRL learns and reasons with (1) an ***Expectation Case Base*** (ECB), which is a mapping $S \times A \rightarrow 2^{S \times [0,1]}$ from (state, action) pairs to a probability distribution over the expectations, (2) a ***Goal Formulation Case Base*** (GFCB), which is a mapping $G \times D \rightarrow 2^{G \times [0,1]}$ from (goal, discrepancy) pairs to a distribution over the expected values for formulated goals, and (3) a ***Policies Case Base*** (Π), which is a set of goal-policy pairs (g, π). We discuss the relation between g and π below.

GRL's ***Discrepancy Detector*** compares state observations s with expectations X. If a discrepancy (i.e., an unexpected observation) $d \in D$ is found, then it is passed to the Goal Formulator. The discrepancy may warrant a change in the current goal. The ***Goal Formulator*** generates a goal $g \in G$ given a discrepancy d and next state s'. The ***Goal Manager*** checks for opportunities to learn new goals that are not in G. In Section 4 we clarify how and when new goals are learned. Finally, ***Policy Learner*** learns policies for new goals and refines the policies of existing goals.

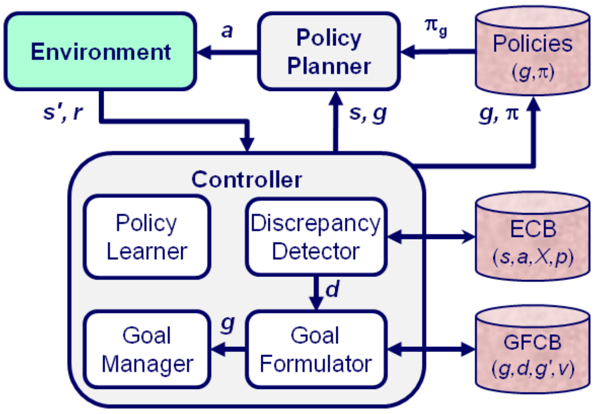


Figure 1: Information flow in GRL

We now provide formal definitions for GDA elements. The ***expectations*** of an action a when executed in state s is a collection of states $S_{a,s} \subseteq S$. Each expectation $x \in S_{a,s}$ will occur with a non-zero conditional probability $p(s_{t+1}=x \mid s_t=s, a_t=a)$ (assuming the Markov property). A ***discrepancy*** occurs whenever executing a yields state $s' \notin S_{a,s}$. We assume that the explanation for a discrepancy reflects GRL's incomplete domain knowledge. Hence, it revises its expectations knowledge whenever a discrepancy occurs, as explained in Section 4.

The representation of a discrepancy depends on the state representation. Here we assume a state is represented as a vector $s=(v_1,\ldots,v_n)$, where v_i is a value of a feature f_i. We represent a discrepancy as a vector of Boolean values $d=(b_1,\ldots,b_n)$, where b_i is true iff the values in position i of the actual and expected states are the same.

Any state can be a goal. We are interested in particular goals that we call ***trajectory goals***, which are defined as follows. First, when an agent follows a policy it generates a ***trajectory***, which is the sequence of states that it visits. A ***trajectory goal relative to***

a policy π is any state along a trajectory produced by π from the start state to a terminal state. All pairs $(g,\pi) \in \Pi$ are such that g is a trajectory goal relative to π. Our policies behave like weak solutions (Ghallab *et al.*, 2004) in that there is no guarantee that a particular goal will be reached because policies are learned incrementally (i.e., when executing the policy a trajectory might be generated that does not contain the goal). Hence, many iterations may be needed before strong solutions are obtained (which guarantee that certain goals are always reached).

4 The GRL Algorithm

We now present GRL, which incrementally learns expectations, goal formulation knowledge, and goal-specific policies. GRL uses Q-learning as its RL algorithm. Q-learning is frequently used as the prototypical RL algorithm due to its bootstrapping capabilities, which enables it to estimate state-action values based on other state-action values estimates. As a result, it tends to converge to optimal policies faster than other RL methods (Sutton & Barto, 1998).

GRL receives as input the start state s_0, a waiting time Δ, the Policy Case Base Π, the Expectation Case Base (ECB), the Goal Formulation Case Base (GFCB), the actions A, and some parameters. The parameters α and γ are the step-size and discount-rate parameters for Q-learning. Parameters ε_1 and ε_2 are for the ε-greedy selection of action and goals, respectively. Parameters c_a and c_b are used to learn new goals as will be explained later, and t is a threshold used to determine when two goals are similar to one another. GRL runs one episode of a game and returns updated values for Π, ECB, and GFCB.[1]

GRL executes an iterative decision making cycle with the following steps: (1) identify discrepancies when they arise, (2) decide which goals to achieve to resolve any such discrepancies, and (3) perform actions to accomplish these goals. Simultaneously, GRL learns knowledge about state expectations, discrepancies, goals to achieve, and the actions to achieve these goals (e.g., goal-specific policies).

GRL has three phases: In Phase 1, which occurs during an episode, GRL uses and updates ECB and GFCB. Phases 2 and 3 occur immediately after an episode ends. In Phase 2 new goals are identified and in Phase 3 goal-specific policies are updated.

[1] We assume episodic tasks in which the instances eventually end. This is an effect of the underlying Q-learning. If the task did not terminate, then we would need to use mechanisms such as eligibility traces (Sutton & Barto, 1998).

```
     GRL($s_0$, $\Delta$, $\Pi$, ECB, GFCB, $A$, $\alpha$ , $\gamma$, $\varepsilon_1$, $\varepsilon_2$, $c_a$, $c_b$, $t$) =
     // **Phase 1: Online execution and updating**
1:     $s \leftarrow x \leftarrow s_0$; $a \leftarrow l_a \leftarrow l_b \leftarrow \varnothing$;  $g \leftarrow g' \leftarrow g_0$;  $d \leftarrow d_0$;  $G \leftarrow$GetGoals($\Pi$)
2:     **while** episode continues
3:         wait($\Delta$)
4:         $s' \leftarrow$ GetState()                    // **Periodically observe the state**
5:         $r \leftarrow U(s') - U(s)$                  // **Compute the reward**
6:         $l_a \leftarrow$ concat($l_a$ ,<$s'$>); $l_b \leftarrow$ concat($l_b$ ,<$s$,$a$,$s'$,$r$>)
7:         ECB $\leftarrow$ Update(ECB, $s$, $a$, $s'$)   // **Update ECB's distribution**
8:         $a' \leftarrow$ Random($|A|$)                 // **Random current action**
9:         **if** $\Pi \neq \varnothing$
10:            $q \leftarrow$ Get(GFCB,$g$,$d$,$g'$)       // **Fetch/update Q value**
11:            $q' \leftarrow q + \alpha(r + \gamma$ argmax$g_{i \in G}$(Get(GFCB,$g$,$d$,$g_i$))$-q$)
12:            GFCB $\leftarrow$ Update(GFCB,$g$,$d$,$g'$,$q'$)
13:            **if** $r < 0$                       // **Performing poorly?**
14:                $d \leftarrow$ CalculateDiscrepancy($s'$, $x$)
15:                **if** Random(1) $\geq \varepsilon$       // **Formulate next goal**
16:                    $g'' \leftarrow$Argmax$g_{i \in G}$(Get(GFCB,$g$,$d$,$g_i$))
17:                **else** $g'' \leftarrow$Random($|G|$)
18:                $g \leftarrow g'$ ; $g' \leftarrow g''$
19:                $\pi \leftarrow \Pi(g')$                  // **Retrieve a new policy**
20:            **if** Random(1) $\geq \varepsilon_2$
21:                $a' \leftarrow$ Argmax$a_{i \in A}$(Get($\Pi$, $\pi$, $s'$, $a_i$))
22:        $x \leftarrow$ Argmax $x_{i \in X}$(Get(ECB,$s'$,$a'$,$x_i$))
23:        Execute($a'$)                        // **Execute current action**
24:        $a \leftarrow a'$; $s \leftarrow s'$
     // **Phase 2: Goal extraction**
25:    $G' \leftarrow$ TopFrequency($l_a$,$c_a$,$c_b$) ; $G \leftarrow \varnothing$
26:    **for-each** $g' \in G$                     // **Iterate over the most frequent goals**
27:        $H \leftarrow \varnothing$
28:        **for-each** ($g$, $\pi$) $\in \Pi$            // **Attempt to group $g'$ with an existing goal**
29:            **if** Similarity($g$, $g'$) $\geq t$ **then** $H \leftarrow H \cup \{g\}$
30:        **if** $H = \varnothing$ **then** $H \leftarrow \{g'\}$        // **g′ is a new goal**
31:        $G \leftarrow G \cup H$
     // **Phase 3: Policies revision**
32:    **for-each** $g \in G$
33:        **if** $\Pi \neq \varnothing$ **then** $\pi \leftarrow \Pi(g)$ **else** $\pi \leftarrow$ nil
34:        **if** $\pi$ = nil **then** $\pi \leftarrow$ New($g$); $\Pi \leftarrow \Pi \cup \{(g, \pi)\}$
35:        **for-each** <$s$,$a$,$s'$,$r$> $\in l_b$
36:            $q \leftarrow$ Get($\Pi$,$\pi$,$s$,$a$)
37:            $q' \leftarrow q + \alpha\,(r + \gamma$argmax$a_{i \in A}$(Get($\Pi$,$\pi$,$s'$,$a_i$)) $- q$)
38:            $\pi \leftarrow$ Update($\pi$, $s$, $a$, $q'$)
39:        $\Pi \leftarrow$ Update($\Pi$, $g$, $\pi$)
40:    **return** $\Pi$, ECB, GFCB
```

In Phase 1 (Lines 1-24), GRL applies and updates ECB and GFCB. It first initializes s and x to the initial state s_0, action a to the null action, lists l_a and l_b to empty, g and current goal g' to the dummy goal g_0, discrepancy d to dummy value d_0, and G to the set of goals that can be accomplished by Π (Line 1) (i.e., policies are annotated with the goals they accomplish). During an episode (Lines 2-24), GRL periodically waits (Line 3) and then observes the current state s' (Line 4), calculates the reward r (line 5), and concatenates $<s'>$ to l_a and $<s,a,s',r>$ to l_b (Line 6) for use after the game episodes concludes. It then updates the distribution of expected states when taking action a in s (Line 7) and generates the current action a' randomly (Line 8). This guarantees that, if Π is empty, then GRL still has an action to perform. Otherwise (Line 9), it retrieves GFCB's estimated q value for formulating goal g' given (g,d) (Line 10), updates the new q' value using Q-learning (Line 11), and records it in the GFCB (Line 12). If the agent is performing poorly (Line 13), it then calculates the discrepancy between the current and expected states (Line 14) and retrieves a new goal g'' from GFCB using ε-greedy exploration (Lines 15-17), and updates its previous and current goal (Line 18). GRL then retrieves a new policy from Π using goal g' as the index (Lines 19-21). It retrieves from the ECB the expected state x from executing a', executes a', and updates the previous action a, current action a', and previous state s (Lines 22-24).

After an episode completes, GRL's Phase 2 extracts a set of goals to update their policies. It first identifies the set $\boldsymbol{G'}$ of most frequent states that appear in the most recent c_a% of visited states, where the frequency of these states must be at least a threshold value ($c_a \times c_b \times |l_a|$). For example, assume that $l_a = \{s_a,s_b,s_c,s_a,s_a, s_b,s_a,s_a,s_a,s_b\}$, $|l_a|=10$, $c_a = 50\%$, and $c_b = 0.25$. Then the most recent 50% of l_a is $\{s_b,s_a,s_a,s_a,s_b\}$, and state s_a is the most frequent state among these (with frequency 3). The threshold value equals 1.75 (i.e., 0.5×0.25×10), which means GRL will also include state s_b in G' because its frequency is 2. However, if $c_b = 0.1$, then $G'=\{s_a\}$ (Line 25). GRL then adds new goals from Π that are at least as similar to goals in G' as the threshold t (Line 29). Similarity between goals is computed using a linear combination of local similarity metrics, one for each of the state's features (Lopez de Mántaras *et al.*, 2005). More precisely, we assume cases to be vectors of n-dimensional features $X=\{x_1,...,x_n\}$. For computing similarity, we define a collection of local similarity metrics $\mathrm{sim_i}()$, one per feature i, and a collection of weights α_i, which sum to 1. The aggregated similarity metric $\mathrm{SIM_{agg}}$ is defined as:

$$\mathrm{SIM_{agg}}(X,Y) = \Sigma_{i=1,n}\ \alpha_i \cdot \mathrm{sim_i}(x_i,y_i)$$

GRL groups goals by similarity to reduce the size of the Policies Case Base Π. However, if no similar goals exist, then GRL will interpret g' as a new goal (line 30).

In Phase 3, GRL refines or adds new policies. For each goal g in G (Line 32), if Π is not empty, then GRL will retrieve policy $\pi\in\Pi$ for this goal (Line 33). If either Π is empty or the policy associated with g is nil, a new policy π for g is created and π is added to Π (Line 34). It will then apply Q-learning to update π using the recent state transitions and rewards (Line 35-38) and update the (goal, revised policy) in Π (Line 39). Finally, GRL returns all the revised case bases (Line 40).

5 Example

Suppose in the real-time strategy (RTS) game Wargus a GRL-controlled agent is competing against one opponent (Wargus, 2012). Wargus is a combat game where each player controls a variety of units. One of the most challenging aspects in RTS games is that there is no “best” unit type. For example, archers can quickly kill footmen but are particularly vulnerable to knights. In our experiments each player controlled mages, archers, knights, ballistae, and footmen. The objective of this game is to be the first to reach a predefined number of points, which are earned by killing the opponent’s units. Some units award more points than others (e.g., killing a knight earns more points than a footman).

Assume each team begins with two footmen and two archers, and that the agent has already played many games. Thus, the case bases Π, ECB, and GFCB have recorded some results. For the Wargus state representation we use $s' = (u_1,u_2,\ldots,u_n, e_1,e_2,\ldots,e_m)$, where $u_i \in \mathbb{Z}$ and $e_j \in \mathbb{Z}$ denote the number of remaining units of type i on our team and e_j denotes the same of type j for the opponent. Usually, n and m are equal (e.g., if the current state equals (2,1,0,2), then our team has 2 footmen and one archer remaining while the opponent has only two archers). Actions in Wargus, denoted as $a' = (b_1,b_2,\ldots,b_k)$, where each b_i is a unit type of the opponent such as {R=archers, F=footmen}, means that units of type i on GRL’s team attacks opponent units of type b_i. For example, the action (R,F,F,$\varnothing$) means that a unit with id 1 attacks an opponent archer, units with id 2 and 3 attack opponent footmen, and units with id 4 do nothing.

Suppose the current state s' is s_{21} = (1,0,0,1) (i.e., GRL’s team has only one footman left and the opponent has only one archer) and r = -2 (Lines 4-5). In Phase 1, GRL adds $\{s_{21}\}$ to l_a and $\{(s_{20}, a_{20}, s_{21}, -2)\}$ to l_b (Line 6). After updating the appropriate ECB distribution (Line 7), GRL will generate the random action a' and then calculate and update the q value of GFCB (Lines 10-12). Because the reward is negative, GRL will change to a new goal (Line 13). After finding the discrepancy d_{21} = (T,T,T,F) between current state s_{21} and expectation x_{21} = (1,0,0,0), it will choose a new goal g'' in an ε-greedy fashion (Lines 14-18). Using this new goal to retrieve a policy $\pi \in \Pi$, suppose it retrieves (by chance) greedy action a'_{21} = ($\varnothing$,R,$\varnothing$,$\varnothing$) (i.e., send the remaining footman to attack an enemy archer) from policy π (Lines 19-21). GRL then updates the previous state and action, computes expectation x, and executes action a' (Lines 22-24). Suppose that this action eliminates the opponent’s units, which ends the game.

Phases 2-3 update Π. First, GRL uses TopFrequency to compute a set of new goals G', and then searches for goals from Π that are similar to any members in G' to create a set G (Lines 25-31). Suppose G'={(100,150,0,0)} (e.g., we have 100 footmen and 150 archers while the enemy has 0 footmen and 0 archers), meaning that GRL won the episode because it destroyed all enemy units. Assume Π = {((96,96,0,0),π_1), ((0,0,10,20),π_2), ((150,100,0,0),π_3), ((105,104,0,0),π_4)}. Then $G=G'$ assuming there are no (sufficiently) similar cases in Π. Lines 39-48 will learn a policy π_5, and {((100,150,0,0),π_5)} will be added to Π. On the other hand, if G'={(100,100,0,0)} and assuming it would be (above-threshold) similar to the first and fourth goals in Π, then G={(96,96,0,0), (105,104,0,0)} and Lines 39-48 will update the policies π_1 and π_4.

6 Empirical Study

We examined the task of winning two adversarial games to investigate the following hypothesis: GRL can significantly outperform a standard RL agent that learns ***only*** policies (i.e., Retaliate (Smith *et al*., 2007), which uses Q-learning) and an ablated GDA agent that does ***not*** learn policies (i.e., LGDA (Jaidee *et al*., 2011a), which is given policies representing an opponent's strategies and their goals, and learns only expectations and goal formulation knowledge). In our study, all three learning agents use the same models for states, actions, and rewards.

6.1 Domains and Scenarios

The adversarial games we use are Wargus and DOM. Both are two-player real-time video games: players make asynchronous moves. They exhibit the characteristics that we want to explore in this paper: there doesn't seem to be a universally good strategy for these games. Instead, they exhibit the "rock-paper-scissors" behavior whereby any strategy can be countered. LGDA have demonstrated good performance in DOM and Wargus (Jaidee *et al*., 2011a; 2011b) while Retaliate has demonstrated good performance in DOM (Smith *et al*., 2007), so they are good baselines for testing GRL.

We used two maps in our Wargus experiments. The first is a medium-sized map with 64×64 cells and 8 units per player, while the second uses the largest feasible map (128×128 cells) and 32 units per player. We set the games' score limits to be 200 and 1000 points, respectively. In our experiments, we used five hand-coded opponents that order all units of the same type to attack a single type of the agent's units. For example, they might assign knights to attack archers. These opponents differ in their attack order. In testing, no single opponent outperformed all the others. We used these built-in opponents to train the three agents (i.e., Retaliate, LGDA and GRL).

The second domain, DOM, is a domination game in which two opponent players try to capture specified domination locations on a 2-D map. Teams are composed of k bots. The player's actions are k-tuples $(l_1,..l_k)$ indicating the domination location l_i to which each bot b_i is assigned. A player captures a location by simply moving a bot to it. A team receives one point for every five consecutive ticks it "owns" a 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. When a bot's health is zero, it respawns after a few ticks in a (randomly-selected) respawning location with max health points. Combat losses are determined using a biased random function that computes the health points lost by each competing bot (it favors bots on the team that has more bots within a certain range). In our experiment, we use a map with five domination locations and eight bots per team.

We used the same six hand-coded opponents in DOM we previously used in (Jaidee *et al*., 2011a), where we used a variety of fixed strategies such as the "half plus one adversary", which attempts to control a majority of locations by sending bots to them whenever they are owned by the competing agent. Another strategy, called "smart opportunistic", sends a different bot to each domination location the team does not own. Among these six adversaries, there are two that are better than all the others, two that are middling performers, and the last two are defeated by all the others.

6.2 Protocol and Results

Agents played *N* episodes, where *N*=20 for Wargus and *N*=340 and 2000 for DOM. The difference in the number N of runs between Wargus and Dom is due to the fact that running DOM games is much quicker. During each training episode, each agent played each of the *M* built-in opponents once (*M*=5 for Wargus and *M*=6 for DOM). During training, the agents GRL, LGDA and Retaliate are learning. We tested GRL against Retaliate and LGDA after each training episode. Because both DOM and Wargus are highly stochastic, games during testing were repeated 10 times. Any knowledge learned during a game in the testing phase was removed after the game ends. Thus, the only knowledge affecting the performance of the agents when competing versus one another was learned during training and any knowledge learned online within that particular game episode.

Figures 2 and 3 summarize the average results. The x-axis plots the number of training episodes, while the y-axis plots the average utility (i.e., score difference of GRL versus another agent).

Experiment 1 (Wargus): In most Wargus episodes (Figure 2), GRL clearly outperformed the other agents, although LGDA sometimes defeated GRL in the medium-size map (Figure 2b). Nevertheless in all cases the differences are statistically significant (p<0.001), as determined by a two-tailed Student's t-Test on the utility scores of GRL versus the scores of another agent (i.e., Retaliate or LGDA). Hence our hypothesis is supported for Wargus, and we can draw three conclusions:

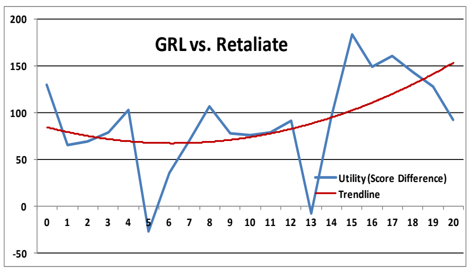
1. There is either no universally good strategy for these games or none can be found by Q-learning even after a large number of episodes.
2. GRL outperformed the Q-learning agent. This highlights the importance for using case-based approaches to learn and reason about expectations, goal formulation knowledge, and goal-specific policies in domains where no universally-best strategy can be elicited by RL.
3. GRL outperformed the LGDA agent. This highlights the importance of identifying new goals and using CBR to learn and reuse goal-specific cases.

We were surprised that GRL outperformed LGDA after only a few episodes because GRL begins with no goals and no policies. In contrast, LGDA begins with policies representing the built-in opponents' strategies and goals for these policies. Upon inspection we found that the opponents' strategies cause their units to form choke points while trying to reach the units they intended to attack. As a result, few units, mostly ranged attack units, actually were effective. Without knowledge about expectations and goals, LGDA rotates among the various opponents' strategies. As mentioned, these end up being ineffectual because it frequently results in choke points. GRL instead initially performs random actions that, on average, cause more of their own units to damage opponent units, which explains the relative results of the first few episodes.

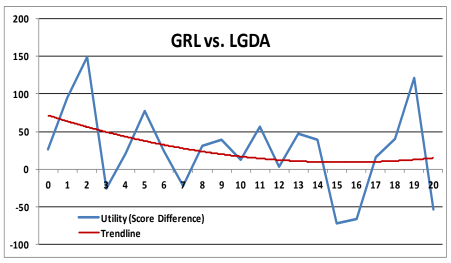
We also investigated why, despite its overall good performance, GRL will occasionally lose games to the opponents in the medium-sized map (e.g., in round 13 versus Retaliate (Figure 2a) and round 20 versus LGDA (Figure 2b)). We found that for this map the score limit was frequently reached even though both teams had several units left. That is, the maximum point threshold was set too low for the number and types of units in the scenario (i.e., killing a high-value unit such as a knight is worth many points, and the game ends sooner when any such unit is killed). This caused high variation in the results because, after a while, several units from both sides will have few health points. In this situation, after a few of these units die the game terminates because the point limit is reached. As a result, depending on the random factor that determines which unit attacks succeeded, units from either side die while others remain with few health points. However, points are only awarded for deaths, and not for low health points. This caused the variance in the results. This was not a factor in the large map because the number of points was set sufficiently high and, although there is fluctuation, GRL did not lose a game on average (Figures 2c and 2d).

Experiment 2 (DOM): Figure 3 summarizes the results with DOM games. In all cases GRL clearly outperformed the other agents, although initially both Retaliate and LGDA outperformed GRL. This is to be expected; GRL initially has no knowledge of which goals to pursue nor how to achieve them. Nevertheless in all cases the difference is statistically significant (p<0.001) across the entire curves, as determined by a two-tailed Student's t-Test for comparing the utility scores of GRL versus those of the other two learning agents). This also supports our hypothesis and allows us draw the same conclusions as mentioned above for Experiment 1.

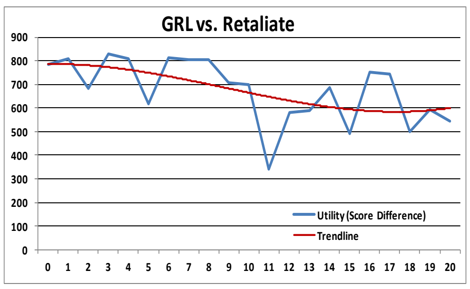
We investigated why it took so many episodes for GRL to start winning versus Retaliate and LGDA in the DOM game compared to Wargus. This occurred because the state model used by the agents forms a DAG for Wargus, meaning that a state is never visited more than once. As a result, for Wargus, we define the new goal to be the final state (whereas for DOM this is defined as the most frequently visited state). In contrast, the same state can be visited multiple times in DOM. Thus, multiple goals were frequently learned per DOM episode, resulting in many more goals being learned overall. Hence, Π grows faster in the DOM rather than in the Wargus experiments during the initial training episodes. This in turn increases the number of episodes needed to learn useful goal formulation knowledge and good policies. Thus, it takes longer for GRL to outperform the other agents in DOM scenarios.



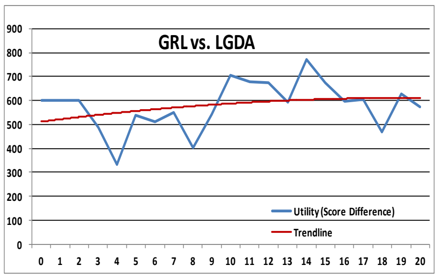
(a)



(b)

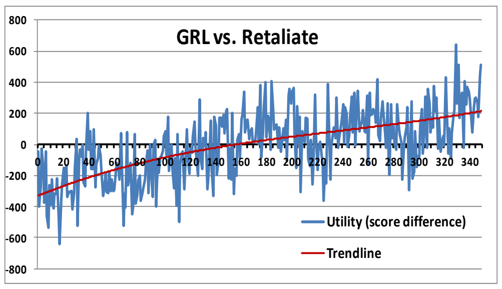


(c)

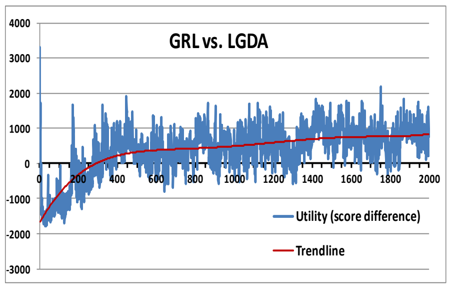


(d)

Figure 2: The results of the Wargus experiments: GRL vs. Retaliate (a) and vs. LGDA (b) on the medium map, and GRL vs. Retaliate (c) and vs. LGDA (d) on the large map. The x-axis plots the number of training episodes, while the y-axis plots the average utility (i.e., score difference of GRL versus another agent).



(a)



(b)

Figure 3: Results from the DOM experiments: (a) GRL vs. Retaliate and (b) GRL vs. LGDA. The x-axis plots the number of training episodes, while the y-axis plots the average utility (i.e., score difference of GRL versus another agent).

We also investigated why it took so many more episodes for GRL to outperform LGDA compared to Retaliate. Namely, it took around 150 episodes for Retaliate compared to almost 300 for LGDA. This was caused by the two strong hand-coded adversaries, which LGDA was able to leverage. This also explains why, in the first episode, GRL loses to Retaliate by approximately 350 points whereas it loses to LGDA by approximately 1600 points.

7 Conclusions and Future Work

We introduced a goal-driven autonomy (GDA) agent, named the Goal Reasoning Learner (GRL), which uses CBR processes to learn and reuse goal-specific action policies, state expectations, and goal selection knowledge. It is the first GDA agent that learns all of this information, and in particular the first to learn policies. GRL uses a reinforcement learning (RL) process to learn its policies and goal formulation knowledge. GDA agents are designed for complex environments in which unexpected situations can occur, as is the case for complex video game environments. In our empirical study with two such environments (Wargus and DOM), we found that GRL outperforms its RL-only ablation, even though GRL is embedded in the same RL process: it uses the same reward function, has knowledge of the same actions, and uses the same state-action transition model.

Our GDA agent does not perform discrepancy explanation (Molineaux *et al.,* 2010). GDA agents often generate an explanation for a discrepancy and formulate a new goal based on it. GRL bypasses this step by directly linking discrepancies with goals in the GFCB. Cox (2007) proposes a general taxonomy for plausible explanations of a discrepancy. This taxonomy classifies potential categories of explanations (called *meta-explanations*) including: incomplete domain knowledge, incorrect domain knowledge, and noise in the environment. Most research on explanations has been in the context of deterministic expectations, whereas GRL would require modeling stochastic explanations. We plan to study these issues in the future. As discussed in Section 6, learning too many goals caused GRL to require many episodes before it became competitive in the DOM domain. Hence, we also plan to study alternative criteria to learn goals.

Acknowledgements

This work was sponsored by NRL and NSF (#0642882). The views, opinions, and findings contained in this paper are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of NRL or the DoD.

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., & Muñoz-Avila, H. (2008). Recognizing the enemy: Combining reinforcement learning with strategy selection using case-based reasoning. *Proceedings of the Ninth European 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.

Bouguerra, A., Karlsson, L., & Saffiotti, A. (2008). Monitoring the execution of robot plans using semantic knowledge. *Robotics and Autonomous Systems*, **56**(11): 942-954.

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

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.

Gabel, T., & Riedmiller, M. (2005). CBR for state value function approximation in reinforcement learning. *Proceedings of the Sixth International Conference on Case-Based Reasoning* (pp. 206–220). Chicago, USA: Springer.

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.

Ghallab, M., Nau, D.S., & Traverso, P. (2004). *Automated planning: Theory and practice*. San Mateo, CA: Morgan Kaufmann.

Jaidee, U., Muñoz-Avila, H., & Aha, D.W. (2011a) Integrated learning for goal-driven autonomy. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence*. Barcelona, Spain: AAAI Press.

Jaidee, U., Muñoz-Avila, H., Aha, D.W. (2011b). Case-based learning in goal-driven autonomy agents for real-time strategy combat tasks. In M.W. Floyd & A.A. Sánchez-Ruiz (Eds.) *Case-Based Reasoning in Computer Games: Papers from the ICCBR Workshop*. U. Greenwich: London, UK.

Lopez de Mántaras, R., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M. L., Cox, M. T., Forbus, K., Keane, M., Aamodt, A., & Watson, I. (2005). Retrieval, reuse and retention in case-based reasoning. *Knowledge Engineering Review*, **20**(3), 215-240.

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. In *Proceedings of the Twenty-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.

Wargus (2012). Source code for Wargus. *http://wargus.sourceforge.net/*. Last checked: January, 2012.

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

Weber, B., Mateas, M., & Jhala, A. (2010b). Case-based goal formulation. In Aha et al. (2010).