Case-Based Learning of Applicability Conditions for Stochastic Explanations

Giulio Finestrali, Héctor Muñoz-Avila

Department of Computer Science & Engineering; Lehigh University; Bethlehem, PA 18015
gif311@lehigh.edu | munoz@cse.lehigh.edu

Abstract. This paper studies the problem of explaining events in stochastic environments. We explore three ideas to address this problem: (1) Using the notion of *Stochastic Explanation*, which associates with any event a probability distribution over possible plausible explanations for the event. (2) Retaining as cases *(event, stochastic explanation)* pairs when unprecedented events occur. (3) Learning the probability distribution in the stochastic explanation as cases are reused. We claim that a system using stochastic explanations reacts faster to abrupt changes in the environment than a system using deterministic explanations. We demonstrate this claim in a CBR system, incorporating the 3 ideas above, while playing a real-time strategy game. We observe how the CBR system when using *stochastic* explanations reacts faster to abrupt changes in the environment than when using *deterministic* explanations.

Keywords: case-based learning, explanations, stochastic explanations

1 Introduction

A good way to evaluate an agent's intelligence is by measuring its ability to react to unprecedented events. An agent interacts with the environment by performing actions, and expecting these actions to have a certain effect. In a stochastic domain (i.e., agent's actions can have multiple outcomes and events happen independently of the agent's actions), though, unexpected and often unprecedented events can always happen. Whenever the results of an agent's action on the environment are different than what the agent expected them to be, this situation is called a *discrepancy* (Cox, 2007). Whenever an unexpected event occurs, we have two choices: relying on the previously collected knowledge to *react* to the event, or trying to *explain* the event and then reacting accordingly to the explanation. The first method can be effective but if the unexpected event is completely new, the knowledge acquired in the past might not be sufficient to react effectively to it.

Explaining unexpected events is a much more complicated task, but offers a higher degree of flexibility than merely relying on previously acquired knowledge to react to discrepancies. The process of explaining an event can be divided in two tasks: given an event, provide a set of possible explanations for it; after that, pick the best explanation among the possible ones. In this paper, we present a novel method to perform the second task efficiently using Case-Based Reasoning (CBR).

At the center of our approach are three ideas: (1) Using the notion of *Stochastic Explanations*, which associates with any event a probability distribution over possible

plausible explanations for the event. Stochastic explanations are necessary when operating in a stochastic domain, because in such domains it is implausible to have a set of fixed cause-effect rules for unexpected events. (2) Retaining as cases *(event, stochastic explanation)* pairs when such unexpected events occur. (3) Learning the probability distribution in the stochastic explanation as the cases are retrieved in subsequent episodes.

In stochastic environments, abrupt changes can make explanations that were previously true suddenly become false. We claim that a system using stochastic explanations reacts faster to such changes in the environment than a system using deterministic explanations. We demonstrate this claim with a CBR system, implementing the three ideas above, while playing the real-time strategy Wargus game. This game is an example of a stochastic domain. We observe how the CBR system when using *stochastic* explanations reacts faster to abrupt changes in the environment than when using *deterministic* explanations.

2 Related Work

There have been many studies about explanations. A thorough discussion on the types of reasoning failure can be found in Cox (1996). Using Cox's categorization, in our work we deal with *contradiction*: a failure condition characterized by an agent's expectations not matching the resulting state; we refer to this as *unexpected failure*. Whenever an agent performs an action, it expects the environment to be modified in a certain way as a result of the action performed. When the actual state of the environment does not match these expectations, we have a contradiction.

Cox (2007) provides an interesting insight on the difference between Learning Goals and Achievement Goals. Whenever an anomaly happens, we try to generate an explanation for it; this process involves finding the culprit of the anomaly. An intelligent agent must be able to blame itself and its own cognitive process in order to improve it: this can be done by generating a Learning Goal. Achievement Goals are the regular goals the agent pursues to accomplish its task. In our work we use stochastic explanations to determine the learning goal.

Existing work on generating explanations to reasoning failures use deterministic explanations (e.g. (Molineaux *et al*., 2011; 2012)). We label the kind of explanations generated by these systems as *deterministic* because when provided with the same discrepancy, the system will output the same explanation. DiscoverHistory relies on a deterministic Hierarchical Task Network planner. The planner cannot account for every plausible event in the domain; it makes assumptions that the conditions of the environment are static to generate its plan. The system then uses the DiscoverHistory algorithm whenever one of these assumptions fails. The way DiscoverHistory works is by modifying the generated plan, introducing new elements that solve discrepancies, or by modifying the initial assumptions. Often, this process introduces new contradictions, which are then recursively solved until the whole plan is consistent, or until a maximum number of modifications have been made. This process is deterministic: if we provide to the algorithm the same anomaly twice, given the same state of knowledge, the system will produce the same explanation.

More recently, Klenk et al. (2012) presented ARTUE, which uses a direct application of explanations in a strategy simulation. ARTUE is based on a modified

version of the SHOP2 planner, and it tries to explain discrepancies similarly to DiscoverHistory. When ARTUE cannot create an explanation, it discards the discrepancy entirely. ARTUE's approach is deterministic and does not deal with anomalies that should trigger the generation of Learning Goals. This means that ARTUE is not able to blame itself to explain the discrepancies that arise.

Explaining discrepancies is a hard problem, and research efforts to solve it are a recurrent research topic. DiscoverHistory and ARTUE generate explanations by blaming the environment for any given discrepancy, and by then finding a set of modifications to the previous plan that resolve the inconsistencies. This solution is greedy and deterministic, and does not change over time. Stochastic explanations represent a novel approach to the problem: instead of blaming the environment, we blame the agent itself. When the culprit of the discrepancy is the environment, blaming the agent will make the agent learn that it is operating in an environment where this kind of discrepancies can happen, and should account for them in the future. Not only does our system reacts to discrepancies by selecting an explanation, but the explanation can change over time, adapting to changes in the environment. Once an explanation is selected, the agent follows the results of the reaction, increasing the likelihood of selecting the same explanation again in the future in case of success, and decreasing it otherwise. Obviously, a stochastic approach requires many iterations to perform well, because it learns by failure. Nonetheless, we believe that the flexibility and dynamicity of this approach justify the learning curve.

3 Stochastic Explanation

We begin by defining explanations in general, and then we will focus on stochastic explanations and how they differ from deterministic explanations. We will show a motivating example for stochastic explanations in the RTS game Wargus.

3.1 Definitions

The task of generating an explanation for a given discrepancy can be divided in two parts: (1) generating a set of possible explanations and (2) choosing among the explanations in the set the one we think is correct. In this work we are tackling the second part.

In our representation, an explanation is merely a label, which can be more or less descriptive of the discrepancy it is trying to explain. More importantly, each explanation is associated with a *reaction*: a goal that the agent will try to pursue after the explanation is picked in attempt to solve a discrepancy. Learning the reaction associated with each explanation is still an open problem.

Let us now discuss what a stochastic explanation is and how it differs from deterministic explanations. Intuitively, stochastic explanations have a probability value associated to them, which expresses the likelihood of an explanation to be correct. Formally, let $\boldsymbol{E}$ be the set of all possible explanations. For any set A, the notation $\boldsymbol{2^A}$ is the power set of A; the collection of all subsets of A. A stochastic explanation is an element $\boldsymbol{\varepsilon}$ in $2^{E\times\{0,1\}}$ such that ε is a probability distribution. That is, $\boldsymbol{\varepsilon} = \{(e_1,p_1),\ldots,(e_n,p_n)\}$ such that each $e_k \in E$ and $\Sigma_k\, p_k = 1$ hold.

A ***discrepancy*** occurs whenever the state *s'* resulting after taking an action in a state *s* doesn't match an expectation *X*. Frequently, the expectation X is defined as the expected state. In this situation a discrepancy occurs if $X \neq s'$ holds (e.g., (Jaidee et al., 2012)). In our work a discrepancy happens when the attempt to accomplish a goal fails. This is because the system keeps a positive outlook on the goal it generates. Since we generate only one kind of goal for now, we also have one possible kind of discrepancy. There are many possible explanations for the same discrepancy. Whenever a discrepancy happens, the system classifies it (see Section 4), and then provides a stochastic explanation ε. The probabilities in ε are constantly updated during the execution of the system. The system will then pick an explanation from ε, according to the probability distribution. If every time we picked the most likely explanation, the one having the highest probability, our system would behave like existing explanation-generating systems. For a given stochastic explanation ε, we refer to the explanation in ε that has the highest probability as the ***greedy explanation*** and denote it by greedy(ε): $\boldsymbol{greedy(\varepsilon) = max\ arg_{\ p}\ \{(e_1,p_1),\ldots,(e_n,p_n)\}}$.

The probability distribution of a stochastic explanation changes during execution. More concretely, the system learns these probabilities from experience by using reinforcement learning (RL). Once the system picks an explanation for a discrepancy, the reaction associated to the explanation is executed, and has two possible outcomes: success or failure. In case of success, we consider the explanation to be correct, and we increase its probability. In case of failure, we consider the explanation as incorrect and we decrease its probability. After several iterations, RL guarantees that the correct probability distribution of each stochastic explanation is learned.

3.2 Motivating Example

To better understand the shortcomings of greedy explanations, we will now consider a motivating example. First, let us describe in more detail the Wargus domain. We focus on combat tasks in our experiments. Any scenario starts with a predetermined number of units both for the player and the adversary; these units battle and as soon as one player has no units left, the scenario is concluded. The units are balanced with a typical rock-paper-scissors mechanism: every unit has other units that it's strong against, weak against, and fairly balanced against. On top of this, some units can be "upgraded" to become more powerful units. For example, an *archer* can be upgraded to a *ranger*, improving its health points and other stats. Finally, the units are divided in three categories: land units, air units, and sea units. Units usually have constraints on the kinds of units they can attack; for example, knights cannot attack flying units. Our agent starts its execution having none of these notions: it does not know which units every unit can attack, or which units are strong against a specific type of unit. The only information the agent knows at the beginning of the system's execution is how to perform the basic policy *attack unit,* needed to achieve the only goal generated by the system *kill unit A with unit B*, where *A* and *B* are picked randomly between the enemy units and our units respectively.

The agent always keeps an optimistic outlook on the generated goal. In case the attack fails, we have a discrepancy. This triggers the *Explanation Generator*, assigning it the task of explaining the discrepancy. As previously explained, the set of possible explanation is predetermined. In our example, we have three possible explanations to why the attack failed: ***(e_1)*** *unit A was not upgraded;* ***(e_2)*** *unit A is a bad choice for attacking unit B;* ***(e_3)*** *unit B has an environmental advantage*. The

latter explanation needs some clarification. We consider an environmental advantage an obstacle on the map that prevents unit A to attack unit B. For example, unit B could be surrounded by trees, preventing close-ranged land units to attack it. The reactions for these explanations are as follows: (1) *attack unit B with the upgraded version of unit A;* (2) *attack unit B with a better unit;* (3) *attack with a ranged or a flying unit*. As we said before, the agent has no knowledge of which unit is strong against unit B, therefore to perform Reaction 2, the system keeps track of success rates for every attack it performs, learning with time the rock-paper-scissors mechanism of the game.

Our motivating example is composed of two scenarios. We play the first scenario for 50 iterations, and then the second scenario for 50 iterations. The knowledge acquired during the execution of the first scenario, is kept when switching to the second scenario. The agent has no perception about the change of scenario: we treat this as an unexpected event, very similar to the kinds of unpredictable events that can always happen in a stochastic domain. In the first scenario (See Figure 1), the agent has several *knight* units, some *paladin* units (which is the upgraded version of a knight), and also several *griffin* units, which are powerful flying units that can attack land units. The enemy AI, which is a passive script that only attacks back when attacked, only controls one *paladin*, surrounded by trees (environmental advantage). During the execution of the first scenario, whenever the agent attacks the *paladin* with a land unit, it fails. With time, it learns that the right explanation for this fact is that the enemy has an environmental advantage, and therefore the right thing to do is attack with a *griffin*. After many iterations the system will learn a stochastic explanation such as $\varepsilon_1 = \{(e_1,p_1), (e_2,p2), (e_3,p_3)\}$ where $p_3 > p_1$ and $p_3 > p_2$. Hence $greedy(\varepsilon_1) = e_3$ holds.

In the second scenario (see Figure 2), we have basically the same situation, but our agent does not have any *griffin* units, and there are no more trees surrounding the enemy *paladin;* this can happen in a regular RTS game, because the *peasant* units can chop trees down. Now, when attacking the *paladin* with a *knight*, the system fails but the correct explanation changed: the *knight* is not upgraded (i.e., explanation e_1), and we should have attacked with a *paladin*. At first, when we switch to the second scenario, when we fail to kill the enemy *paladin*, the system will have a high probability associated with the explanation *environmental advantage* (i.e., explanation e_3). This means that the greedy explanation will *always* pick *environmental advantage*, as long as the probability is higher than the others. Therefore, a greedy approach would have to wait until the probability of *environmental advantage* falls below the probability of the other explanations, making a lot of mistakes until then. Stochastic explanations, instead, will pick a different explanation sooner, and it will actually realize much faster that the correct explanation changed.

This is the reason that motivates the experiments we present in Section 5. If an intelligent agent operates in a stochastic domain, then greedy explanations will have a hard time reacting effectively and in a timely manner to the sudden changes that happen in such domain. If the agent still had griffin units in the second scenario, then both explanations e_3 and e_1 are plausible, since both their reactions would provide a positive outcome. In this case, stochastic explanations would learn a probability distribution that represents these conditions. Greedy would keep using explanation e_3 instead, since the outcome received from it remained positive. While the performance of the two approaches would be similar in this case, the stochastic explanation's representation is more representative of the situation. If, eventually, the agent won't

be able to use griffins anymore, an agent using stochastic explanations will then be able to recover rapidly, while an agent Greedy Explanations will suffer greatly, taking a much longer time to re-establish good performance.

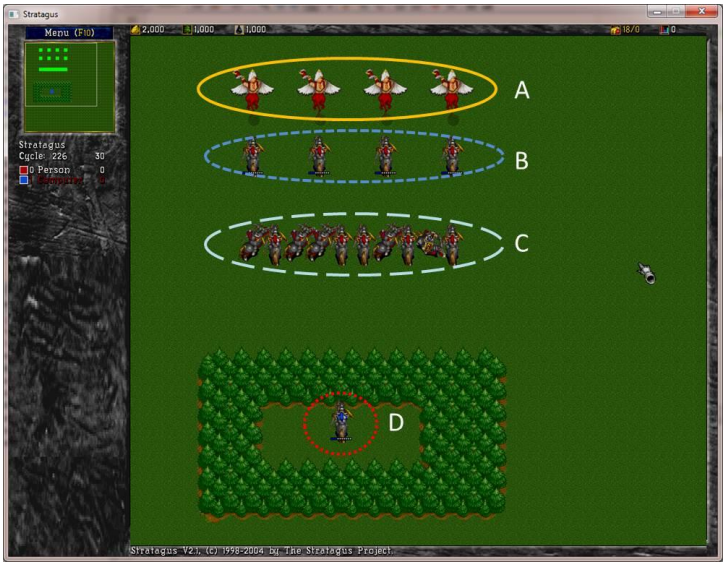

Figure 1: Scenario 1. A are the griffins, B are the paladins and C are the knights. D is the enemy paladin surrounded by trees and hence inaccessible for land units.

4 Case-based learning of applicability conditions for explanations

We use case-based learning techniques to acquire knowledge when generating and testing an explanation for a given discrepancy and use case retrieval to dynamically recognize the applicability of explanations captured in a previous episode.

Whenever a discrepancy happens, the system takes a snapshot of the state of the environment. This snapshot contains a set of feature-value pairs that are domain-dependent. For our experiments in Wargus we use five features: the Euclidean distance between the attacker and the target, the difference in health points between the attacker in the retained case and the query and the same difference for the target in the retained case and the query, and the types of both attacker and target.

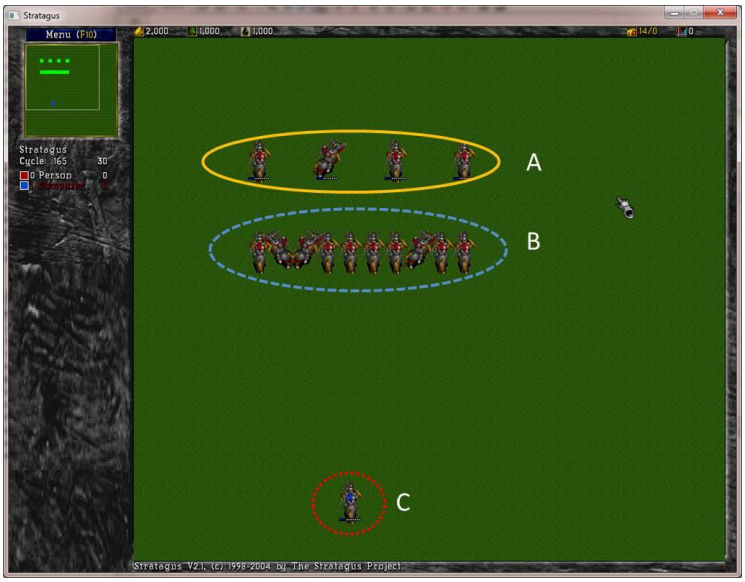

Figure 2: Scenario 2. A are the knights, B are the paladins. C is the enemy paladin, now accessible from land units.

The system then checks if a similar discrepancy ever happened in the past by measuring the similarity of the current discrepancy with the previous ones that are stored in the system's case base. To do so, the system has to compute the local similarity $sim_i(x_i,y_i)$ between pairs of features x_i, and y_i (Ricci and Avesani, 1995). The global similarity between two discrepancies is then computed with an aggregated similarity metric SIM_{agg} is defined as: $\boldsymbol{SIM_{agg}(X,Y) = \Sigma_{i=1,n}\ \alpha_i \cdot sim_i(x_i,y_i)}$. This similarity and its weights α_i are also domain-dependent. If the similarity of the nearest neighbor for the current situation is above a user-defined threshold, then the system considers the unexpected event as "already seen" in the past, and will utilize the learned explanations for it. Otherwise, the current situation is considered as "unprecedented", and the case ***(features, equiprobable stochastic explanation)*** is retained in the case-base. Since we don't know anything about unprecedented situations, we associate the equiprobable stochastic explanation (i.e., every known explanation has the same probability). These probabilities are refined over time.

The following is a pseudo-code presenting how the system handles the task of retrieving and generating explanations for any given discrepancy. The algorithm requires five elements in input: the case-base CB which contains the discrepancies previously encountered (CB will be empty at the first iteration of the system), the policies Π to execute the goals as defined by their reactions (e.g., pre-coded instructions of what to do next), the pre-determined set E of (explanations, reaction) pairs <*e*,*r*>, the similarity threshold σ, and the maximum number of goals θ.

```
EXP_GEN(CB, Π, E, σ, θ)
1:   GoalsInProgress ← ∅;  ExplanationsInProgress ← ∅, FailedGoals ← ∅
2:   while true
3:
4:       s ← GetState() // Get the current state from the environment
5:       if ScenarioOver
6:           return CB
7:
8:       for-each g ∈ FailedGoals  // Generate Explanations for discrepancies
9:           c  ←  NewCase(g,s)  //Create query case
10:          c' ← GetNearestNeighbor(CB, c)//Retrieve NN(c)
11:           if (Similarity(c',c) ≥ σ)
12:                // Consider the query case as the same case of c'
13:                e ← PickStochasticExplanation(c'.Explanations)
14:          else
15:               // Consider case c as unprecedented
16:               CB.retain(c)
17:               for-each <e,r> in E
18:                   //initialize every explanation as equiprobable
19:                   c.addExplanation() ←  (<e,r>, 1/|E|)
20:               e ← PickStochasticExplanation (c.Explanations)
21:          ReactToExplanation(e, Π_e)
22:          ExplanationsInProgress.add(e)
23:
24:      for-each e ∈ ExplanationsInProgress
25:          UpdateExplanationState(e)
26:          if (e.state == success)
27:               RaiseProbability(e.p)
28:          if (e.state == failed)
29:               LowerProbability(e.p)
30:          if (e.state != in_progress)
31:               ExplanationsInProgress = ExplanationsInProgress – {e}
32:
33:      if GoalsInProgress < θ & ExplanationsInProgress =∅
34:           g ← GenerateGoal()
35:          PerformGoal(g, Π_g)
36:          GoalsInProgress.add(g)
37:
38:      for-each g ∈ GoalsInProgress
39:          UpdateGoalState(g)
40:          if (g.state == failed)
41:               FailedGoals.add(g)
42:          if (g.state != in_progress)
43:               GoalsInProgress = GoalsInProgress – {g}
44:  //end while
```

After resetting Goals and Explanations in progress in Line 1, the algorithm enters a loop, which will be terminated when the scenario ends (Line 5). At Line 4, we update the agent's state of the environment representation; if the scenario is over, the algorithm returns the updated case base.

At Line 8, the algorithm loops through the failed goals (if any), generating a new case for each one. Then, at Line 10, the algorithm looks for the most similar case to *c* in the case base. If the similarity of *c'* and *c* is greater than σ (Line 11), the system picks the explanation from *c'* at Line 13, otherwise the case c is considered unprecedented. It is then added to the case base (Line 16), its explanations are initialized with the given input set E all having the same probability 1/|E|, and an explanation is then picked at Line 20. The function PickStochasticExplanation takes a stochastic explanations *c'.Explanations* as input. It picks an explanation <e,p> according to the probability distribution in *c'.Explanations*

At Line 21 we call the function ReactToExplanation, which takes in input the explanation *e* and the reaction associated to it in order to perform the reaction. Then, at Line 22 we add this explanation to the *ExplanationInProgress* collection. Lines 24-31 handle the update of the state of each explanation in progress. In case the reaction succeeds, which means the explanation was correct, the probability of the explanation *e* is raised. If the executing the reaction fails to achieve the goal, the probability of *e* is reduced. Finally, unless the reaction associated with explanation *e* is still in progress, we remove it from the *ExplanationInProgress* collection at Line 31.

The probability value of an explanation is the sum of the ratio of successes over the total number of trials, and an additive term called *booster* value.

$$P(explanation) = \frac{Times\ Succeeded}{Times\ Tested + 1} + multiplier \cdot boosterValue$$

There are two kinds of booster values: a *reward booster* value and a *punishment* one. The former is positive; the latter is zero or negative. If in the future the explanation is tested again, and the outcome is again positive, the multiplier will be now 2 and therefore the probability will be raised more. The multiplier will increase by one for each consecutive success/failure and will be reset to 0 (no booster) whenever a success-failure or failure-success pattern occurs. The two booster values are user-defined parameters and they can impact on the reaction time of the EXP_GEN agent. In our experiments, we use a value of .12 for rewarding correct explanations and a value of .01 to punish incorrect ones. When applying the formula above, values fall off the [0,1] range: in this case we re-scale them so they will fall into [0,1].

Goal generation in EXP_GEN. We generate a new goal only if there are no explanations in progress (Line 33) and only if there are no more than θ goals already in progress; in our experiments, we set $\theta = 1$. This makes the system pursue one goal at a time, eliminating possible noise from the performance analysis we will present in Section 5. The GenerateGoal function generates only one kind of goal: *kill unit A with unit B*, where *A* and *B* are randomly chosen (so there are many goals that can be pursued, one for every pair *(A,B)*. This goal then gets started by the PerformGoal function, which takes the goal and its associated reaction as input (Line 35). The goal is then added to the *GoalsInProgress* collection (Line 36). At Line 38 we handle the update of the state of the current goal. For each goal *g*, if *g* failed then we add it to the *GoalsFailed* collection. And finally, unless the goal is still in progress, we remove it from the *GoalsInProgress* collection (Line 43).

GENERATE_GOAL+. The EXP_GEN pseudo-code learns over time the applicability conditions and probability distribution of the stochastic explanations. We can exploit this learned knowledge to prevent discrepancies from happening in the future. The GENERATE_GOAL+ function uses the case base before selecting the next goal to pursue. In case the goal that was picked originally generated discrepancies in the past (we can compute this with a relaxed similarity metric over the case base), then it has a high chance of failure and the goal should be changed. This will not prevent discrepancies from happening, because we cannot know the future events that might occur in a stochastic domain, but we expect that it will nonetheless dramatically decrease the number of times that our agent will find its expectations inconsistent with the state of the world. The GENERATE_GOAL+ function is represented in the following pseudo-code.

GENERATE_GOAL+(*CB*, τ)
1: $g \leftarrow$ CreateGoal()
2: **if** FindDiscrepancies(g, *CB*) > τ
3: PickBestGoal(g)
4: **return** g

At Line 1, the function CreateGoal will generate a domain-specific goal. As explained before, the goal generated will always be *KillUnit(A,B)* where A and B can be any unit. At Line 2 the function FindDiscrepancies will scan the case base CB, looking for previous failures that involved the same kinds of unit that were picked for g. This is a retrieval operation from the case base that involves a similar, but more relaxed, similarity metric. Here we consider only the unit types as features to compute the similarity.

If FindDiscrepancies can find more than τ previous failures, where τ is a user-defined threshold, then the algorithm calls PickBestGoal at Line 3. This function will look for the best unit to attack the target among the agent's units. To do this, we rely on knowledge acquired during the execution of the system. Recall that the system starts its execution knowing nothing about the rock-paper-scissors mechanism of the game, but improves its knowledge during its execution. Therefore, this algorithm will improve its performance over time, effectively learning from past mistakes.

5 Empirical Study

We claim that an agent using Stochastic Explanations will better react to sudden changes in the environment than an agent using Deterministic Explanations. Also, we will show how an agent that learns from past mistakes greatly outperforms an agent that does not perform this process.

5.1 Experimental setup

In our scenarios, we used a similarity threshold $\sigma = 0.7$, and the explanation for each discrepancy is chosen from a predetermined set of three possible explanations:

Unit not upgraded, Wrong Unit, and *Environmental advantage.* The first represents the situation where the target should be attacked with an upgraded version of the unit that was used originally (e.g. *ranger* instead of *archer*). The second is probably the most intuitive explanation, and means that the attacker should have been a completely different unit (e.g. *knight* instead of *peasant*). Finally, the latter explanation, Environmental Advantage, represents the situation where the enemy has some kind of obstacle protecting it from being attacked by closed-range land units and should therefore be attacked with flying or ranged units.

In the description below we refer to ***team A*** to the one controlled by our AI and ***team B*** to the opponent.

As in other works on explanations (Molineaux et al. 2012), our scenarios are hand-crafted to reduce noise and present more accurately the effectiveness of our theories. We will present the results from three scenarios, each composed by two parts. In the first part, the system acquires knowledge about the environment, the discrepancies that can happen within it, and their corresponding explanations. In the second part the scenario is slightly modified, so that the previously acquired knowledge is now flawed and not effective for the new conditions. We will show how Stochastic Explanations react to these events in comparison with Greedy Explanations and Random Explanations.

In the first scenario, the first part presents team A consisting of *archers* and *rangers* attacking team B which consisted of *rangers* only. When archers are selected to accomplish the goal, they will fail because rangers are stronger. The correct explanation for this kind of discrepancy is *Upgraded Unit*. In the second part, we also have *griffins* available, and the enemy *rangers* are now surrounded by trees. When the rangers attack they will sometimes fail. The correct explanation becomes *Environmental Advantage*; it is best to attack with the griffins.

The second scenario is similar to the first one, except that now Team A have *knights* and *paladins* but neither *archers* nor *rangers*; team B controls only *paladins*. The main difference between the first scenario and the second one is that *paladins* cannot attack *griffins*, while *rangers* can. Therefore in the first scenario the *Environmental Advantage* explanation has a small chance to fail, while in the second it can only fail in case of a *timeout*, which happens when the *griffin* does not kill the *paladin* in a reasonable amount of time.

Finally, the third scenario is the same as the second one, except that we switch the order of the first and second part.

Stochastic Explanations behave as explained in Section 3 and 4; Greedy Explanations utilize the same mechanics, except for the explanation decision part: here, the system will always pick the explanation having the highest probability. Random Explanations, intuitively, will just pick a random explanation among the three possible ones for each discrepancy that occurs during the system's execution. We expect Greedy to be a very effective heuristic during the first part of each experiment because we repeatedly try the same scenario, but then to perform much worse than Stochastic in the second part, having a much harder time to adapt to changes in the environment. We also expect Random not to present behavioral differences between part one and two, and to be worse than both Stochastic and Greedy Explanations.

In the charts that we will show, the x-axis will represent the number of iterations: for our experiments, we ran part one of each experiment for 50 iterations before switching to part two, which was then ran for 50 iterations, totaling 100 iterations per-experiment. Each experiment was also run 5 times, and the results were then averaged. The y-axis shows the number of failed explanations – that is, explanations generated by the system that failed when tested, providing a negative feedback.

5.2 Results

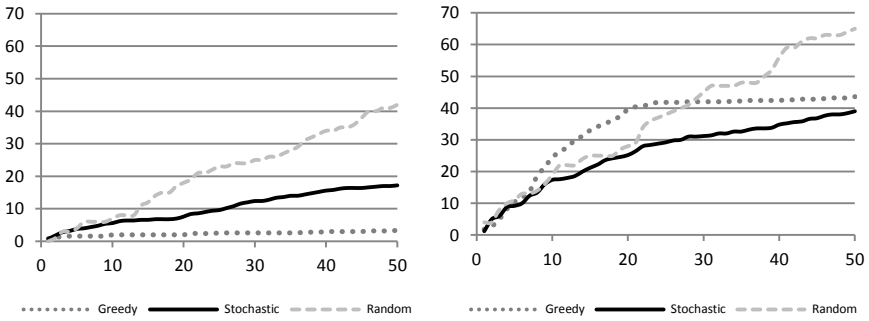

Figure 3: Results for the first scenario for parts 1 (left) and 2 (right)

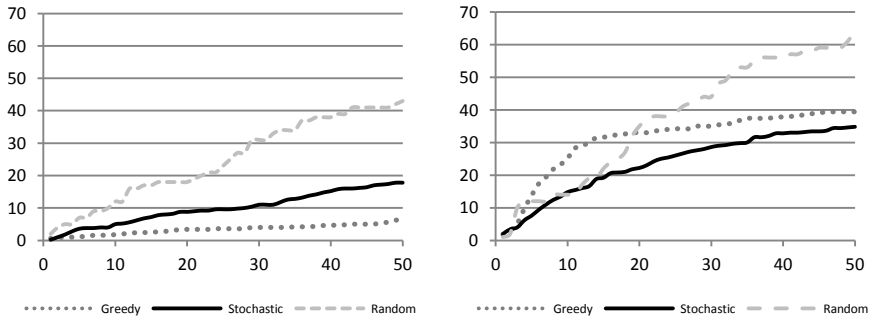

Figure 4: Results for the second scenario for parts 1 (left) and 2 (right)

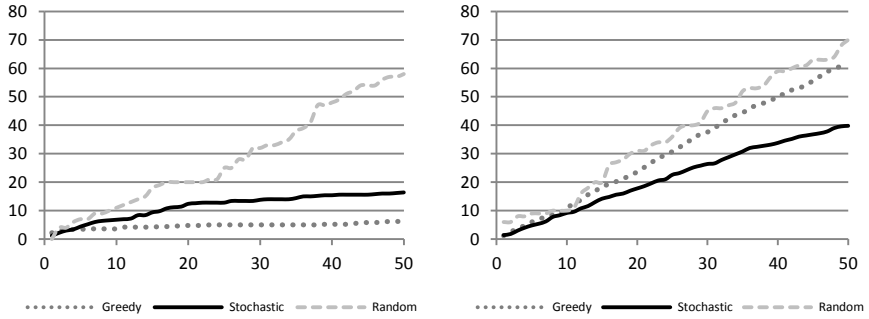

Figure 5: Results for the third scenario for parts 1 (left) and 2 (right)

Figures 3, 4, and 5 show the results for first, second and third scenario respectively. The results meet the expectations: while in part 1 Greedy Explanations outperform Stochastic Explanations, in part 2 Stochastic outperforms Greedy for all the three scenarios. Random Explanations, as expected, have a consistent behavior in all the scenarios, and are consistently outperformed both in part 1 and 2 and in all three scenarios by both Greedy and Stochastic Explanations.

The implications of these results are deeper and more subtle that one might think. A stochastic explanation-generating agent, as opposed to a deterministic one, will always question its own experience and expectations. While this line of thought will penalize its performance in deterministic scenarios (represented by the first part of each experiment), it will be extremely helpful in case of sudden environment changes (the second part of our experiments). A deterministic agent cannot react this quickly to unseen events, because for functioning correctly such an agent must entirely rely on its own previous experiences. A stochastic agent, instead, constantly keeps questioning itself and its past experiences, which might be influenced not only by the nature of the environment but also by the limitedness of the agent's perception of the environment. Therefore, depending on the nature of the domain we operate in, one or the other approach can be better. In the case of a stochastic domain, Stochastic Explanations is a more suitable choice. If anomalies are extremely rare in the domain we operate in, then using greedy explanations would be better; if, instead, anomalies are common, then using Stochastic Explanations is preferred.

We did a second experiment showing how GENERATE_GOAL+ (see Section 4) exploits knowledge learned from past mistakes. Learning from past mistakes reduces substantially the number of discrepancies occurring. For the same 3 scenarios, Figures 6, 7, and 8 show a substantial reduction in the number of discrepancies generated with ***error learning*** (i.e., using GENERATE_GOAL+) compared to ***normal*** (i.e., using the goal generation code in EXP_GEN). In these experiments, we use Stochastic Explanations in both parts and the settings are the same as the previous experiment.

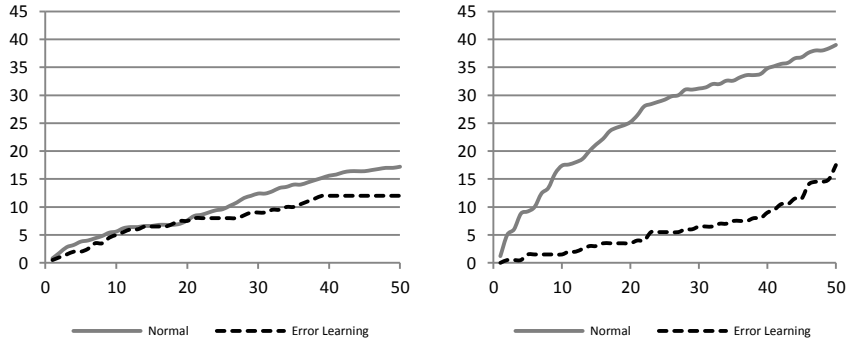

Figure 6: Results with learning for the first scenario for parts 1 (left) and 2 (right)

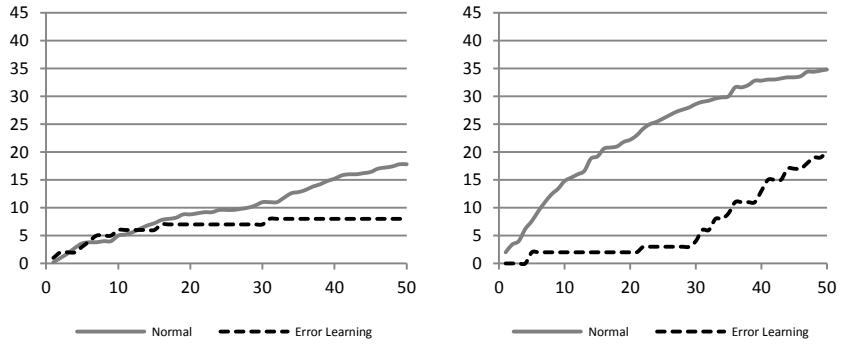

Figure 7: Results with learning for the second scenario for parts 1 (left) and 2 (right)

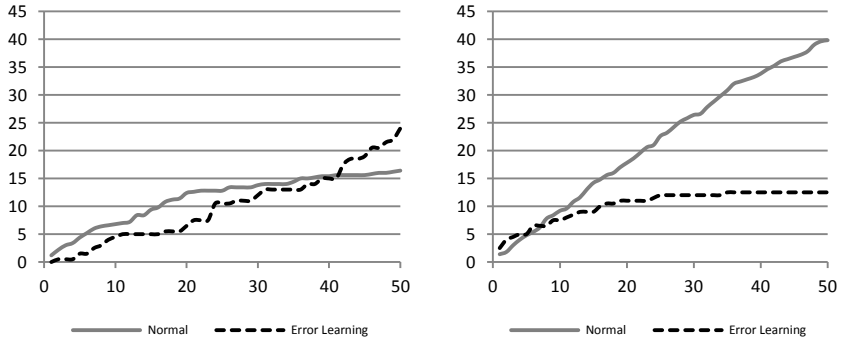

Figure 8: Results with learning for the third scenario for parts 1 (left) and 2 (right)

Using GENERATE_GOAL instead of the goal generation code in EXP_GEN shows the potential of taking advantage of the knowledge acquired by explaining discrepancies. This is even more pronounced in the second part of our scenarios,

where the agent is presented with an unexpected event; if an agent is able to reason on its own past mistakes and on the explanation that it gave to those mistakes, then it is able to greatly outperform an agent that does not perform this kind of analysis.

6 Conclusions and Future Work

We studied the problem of explaining events in stochastic environments. A challenge in such environments is that unpredicted changes can make explanations that were previously true suddenly become false. We explore three ideas to address this problem: (1) Using the notion of Stochastic Explanations, (2) Retaining as cases *(event, stochastic explanation)* pairs when unexpected events occur. (3) Learning the probability distribution in the stochastic explanation as the cases are retrieved. We conducted experiments with a CBR system, implementing the three ideas above, while playing Wargus. We observe how the CBR system when using *stochastic* explanations reacts faster to abrupt changes in the environment than when using *deterministic* explanations.

In future work, we will like to learn new explanations. This is a challenging problem because of the need for a vocabulary from which these explanations will be formulated. Furthermore, this will also involve learning the reaction for the explanation.

Acknowledgements. This work was supported in part by NSF grant 1217888.

References

Cox, M. T. (1996). *Introspective Multistrategy Learning: Constructing a Learning Strategy under Reasoning Failure.*Doctoral dissertation, Georgia Institute of Technology

Cox, M. T. (2007). Perpetual self-aware cognitive agents. *AI magazine, 28(1), 32.*

Jaidee, U., Munoz-Avila, H., & Aha, D.W. (2012). Learning and Reusing Goal-Specific Policies for Goal-Driven Autonomy. *In Proceedings of the International Conference on Case-Based Reasoning (ICCBR).* Springer.

Molineaux, M., Aha, D. W., & Kuter, U. (2011). Learning Event Models that Explain Anomalies. *In T. Roth-Berghofer, N. Tintarev, & D.B. Leake (Eds.) Explanation-Aware Computing: Papers from the IJCAI Workshop.* Barcelona, Spain

Molineaux, M., Kuter, U., & Klenk, M. (2012) *In Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems.*

Klenk, M., Molineaux, M., & Aha, D. W. (2012) Goal-Driven Autonomy For Responding To Unexpected Events In Strategy Simulations. *Computational Intelligence*.

Ricci, F. and Avesani, P. (1995) Learning a local similarity metric for case-based reasoning. *International Conference on Case-Based Reasoning (ICCBR-95)*, Springer: Sesimbra, Portugal.