An Integrated Architecture for Online Adaptation to Novelty in Open Worlds using Probabilistic Programming and Novelty-Aware Planning

Paper ID:

Bryan Loyall, Avi Pfeffer, James Niehaus, Tyler Mayer, Paola Rizzo, Alex Gee, Sanja Cvijic, William Manning, Mary Kate Skitka, McCoy Becker, Sevan Ficici, Peter Galvin, Glenn Takata, John Steigerwald

Charles River Analytics, Inc.

Abstract

AI systems today are often brittle to changes open world environments that were not previously anticipated by their designers. This brittleness can be a source of both reduced performance and (at times, catastrophic) failure. In this research, we seek to address this challenge with a general-purpose reasoning architecture that automatically adapts to open world environments that present novelty to the agent. The approach combines and extends probabilistic programming inference and planning under uncertainty to (1) recognize novelty when it occurs, (2) incrementally characterize the novelty as observations of the novelty accrue, and (3) continually adapt its task-based reasoning to the evolving understanding of the novelty in order to maximize task performance. Extending Bayesian and statistical inference, the system is designed to recognize and characterize novelty rapidly (e.g., with 10's of observations) rather than requiring massive data sets for training, facilitating online adaptation. We demonstrate the research approach with an instantiation in two game domains in which unanticipated novelty can be injected at random times during a tournament. Empirical evaluation of the approach over a range of novelty types shows the expected decrease in performance when novelty is first injected, with a rapid (typically within a fraction of a single game), online recovery of performance during task execution as novelty is detected and characterized, and with stable, improved task performance in the later stages of the tournament after the novelty has been fully or (nearly-fully) characterized.

Introduction

An important capability of an intelligent agent is to be able to handle novel situations that differ from what it has previously encountered. In the past, AI systems have generally struggled to exhibit this capability. Learning-based systems struggle with out-of-distribution inputs, while knowledge-based systems struggle with situations that differ from their precoded knowledge. The challenges and requirements for AI systems to handle novelty have been well captured by (Langley 2020).

As a simple illustrative example, imagine an agent playing a game of Monopoly with three friends. Unbeknownst to her, the friends have changed the rules. Perhaps they have changed the dice so that they are loaded towards lower numbers. Or perhaps all prices and rents associated with Boardwalk have doubled. Alternatively, perhaps the players start the game with $200 instead of $1,500.

These three examples illustrate the three challenges an agent that can handle novelty must overcome. In the case of the loaded dice, the agent must be able to *detect* the novel probability distribution over die rolls. In the case of prices and rents associated with Boardwalk, the agent can quickly detect the novelty when it sees a different purchase price, but it must be able to *characterize* the nature of the novelty precisely in order to play well, without having to see all the possible prices and rents. In the case of starting the game with $200, although detection and characterization are straightforward, this is a completely different strategic situation that would never be encountered in the standard game, and the agent must know *how to reason and plan in this new situation*. If the agent cannot overcome these challenges, it will fail miserably in novel situations. If the agent does overcome these challenges, it will have a significant advantage over agents that are unable to adapt.

Previous approaches to automatically handling novel environments include work in transfer learning (Senator 2011) and reinforcement learning techniques (Kaelbling et al. 1996), however these are both focused on adapting ahead of time to the new environment, and don't include the ability to adapt in an online fashion during execution. Meta reasoning (Cox 2005), analogical reasoning (Veloso 1992), case-based reasoning (Marling et al. 2002), and related technologies reason about inaccuracies in the agent's knowledge, but don't focus on novel changes in the mechanisms of the world. One of the closest research areas is in cybersecurity (Friedman et al. 2014; Schulte et al. 2015), where systems attempt to automatically generate new code to make their systems handle novel inputs or novel behavior caused by cyber attackers, however these systems are typically working with narrow changes to repair exploitable bugs in the application code.

In this work, we seek to develop AI agents that can handle novel situations in open worlds by combining learning-based and knowledge-based approaches in an integrated architecture. Our architecture, named Coltrane, combines hierarchical probabilistic programming, which enables the agent to detect novelty; a discovery process to characterize novelty; and a planning approach that uses fundamental principles to determine how to act in new situations. In particular, Coltrane makes four fundamental contributions that contribute to novelty handling:

1. A combination of Bayesian and statistical inference to weigh different hypotheses and detect novelty
2. A formalization of a scientific discovery process using probabilistic program synthesis to characterize novelty
3. A principled planning method based on Monte Carlo tree search and adaptive heuristics for strategizing in novel situations
4. An architecture that combines all the above elements, including a knowledge compilation process that ensures that new knowledge is compiled into a form that can be readily exploited by the first principles planner

Coltrane is a general framework that is not tied to any particular domain. We demonstrate our framework in a novel Monopoly domain and are currently also developing its applications to a Minecraft variant; application to Angry Birds is also planned. In Monopoly, we have demonstrated the ability to adapt to a range of different kinds of novelty. Specifically, we study class novelty, in which new kinds of entities are introduced into the environment; attribute novelty, in which the attributes of entities are changed; and representational novelty, in which the representations of entities in the environment are changed. These constitute levels 1-3 in Senator's novelty hierarchy (Senator 2019).

Working in conjunction with an independent evaluator (identity redacted for blind review), we evaluate Coltrane's ability to handle levels 1-3 of novelty. We show that our agent is indeed able to adapt to unanticipated novel situations. In particular, in the presence of novelty, our agent is able to detect the novelty, characterize the novelty, and achieve greater than pre-novelty performance after novelty is introduced.

Research Approach – Integrated Symbolic and Probabilistic Architecture for General Purpose Reasoning in the Presence of Unanticipated Novelty

Our research approach tightly integrates probabilistic program inference, probabilistic program synthesis, and symbolic reasoning and planning inside a single integrated agent architecture to enable online adaptation to unanticipated novelty.

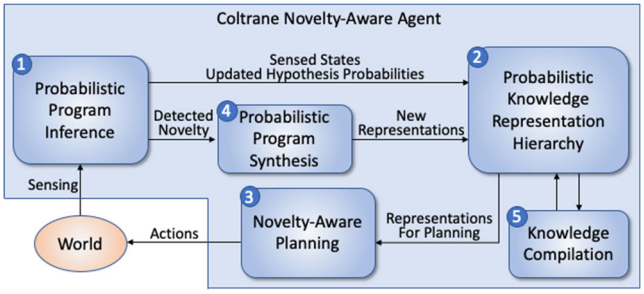

Figure 1: The Coltrane architecture tightly integrates probabilistic program inference and synthesis with planning to allow novel representations to be created and reasoned about in a traditional sense-decide-act cycle.

The Coltrane system architecture is shown in Figure 1. The core of Coltrane's execution is a sense-decide-act cycle using modules (1), (2), and (3) in a loop that interacts with the world. Unlike typical sense-decide-act architectures, the *sense* portion uses probabilistic program inference (PPI) to include Coltrane's confidence for each sensed instance.

Coltrane extends this traditional execution when novelty is detected. When surprising observations are sensed by PPI (1) that do not match the existing model of the world in the knowledge representation, PPI triggers the probabilistic program synthesis (PPS) module (4) to generate multiple candidate representational novelty hypotheses. Those new hypothesis representations (along with Coltrane's confidence in each) extend the knowledge representation hierarchy (2), and can then be used for planning, along with all of the previous knowledge.

Each time through the sense-decide-act loop, the confidences in each hypothesis are updated by PPI (1) as new observations accrue, refining and selecting among the candidate hypotheses, and allowing Coltrane to progressively converge on an accurate characterization of the novelty.

Throughout this process, the sense-decide-act loop continues, with Coltrane continuing to reason and act in the world using its best characterization at each time point. Depending on the confidence in the new knowledge, this can involve working around the new uncertain elements in the world; using them in a way that is robust to their different possible characterizations; or a combination of the two.

In a traditional agent system, the knowledge representations are optimized for reasoning. With novelty introducing new representations, this optimization must be performed online in response to newly introduced foundational elements of the world that are characterized by PPS. This is the role of the Knowledge Compilation (5) module in Coltrane.

Hierarchical Probabilistic Programming for detection of novelty

Coltrane organizes its knowledge about the domain using hierarchical probabilistic programming. This knowledge includes both specific information about the current situation (like the current assets of the players and their location on the board), as well as more general information that govern how the situation develops (like the die roll distributions and prices). Coltrane's hierarchical probabilistic programming is implemented in Scruff (Pfeffer & Lynn 2018), a framework for building AI systems that sense, reason, and improve over time based on the cognitive theory of predictive coding (Clark 2013; Friston 2008; Rao & Ballard 1999). Predictive coding is based on the principle that the brain does not simply process sensory data bottom-up; rather, it makes predictions about what it expects to see and processes the errors from its predictions to produce beliefs. In predictive coding, knowledge is organized hierarchically, with higher layers representing more general or abstract concepts. Each layer generates predictions for the layer below and receives error signals from the layer below. Scruff is a computational implementation of predictive coding using a Bayesian formulation and message passing inference based on belief propagation. In Scruff, each node of the hierarchy is represented as a probabilistic program that generates a node at the layer below.

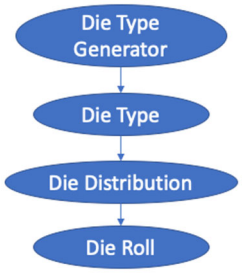

Figure 3: Simple hierarchy for die rolls

For example, consider how an agent might model the roll of a single die. The agent's knowledge includes both the specific results of die rolls as well as knowledge about how die rolls are generated. *Figure 3* shows a simple hierarchy for die rolls. At the bottom layer are the individual Die Rolls. At the layer above is a Die Distribution that generates individual die rolls. In the baseline game, this is a uniform distribution over the numbers 1-6; in our example above, it is a non-uniform distribution over the same numbers; in other examples, it might include other numbers as well. The die distribution is generated by the Die Type at the layer above. For example, the starting Die Type might generate distributions over small positive integers, with uniform distributions being preferred. At the top of this hierarchy is a Die Type Generator, which can generate other kinds of dice, such as

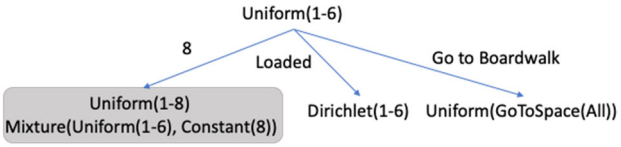

Figure 2: Hypergraph fragment for hypotheses about die rolls

dice with faces that tell the player to move their piece to a specific location on the board.

While interacting with the environment, Coltrane maintains beliefs about each of the nodes in the hierarchy using Bayesian inference. However, the total set of possible hypotheses for nodes might be infinite; for example, there are an infinite number of sets of integers that could be the faces of a die. Therefore, rather than attempting to maintain a full probability distribution over all hypotheses, Coltrane uses a lazy representation of hypothesis space. It does this by explicitly maintaining a small number of hypotheses for each node and leaving open the possibility that the true value of the node is none of the current hypotheses. Coltrane uses statistical tests to identify when the current hypotheses are insufficient to explain the data. Statistical tests are used because there are no explicit alternatives to compare to using Bayes rule. When no current hypothesis explains the data, novelty is detected.

For example, if a die roll of 8 is observed, any hypothesis that says the maximum possible roll is 6 is invalidated. If a die roll results in "Go to Boardwalk", the baseline Die Type hypothesis is invalidated. For the loaded die, the process is slower; over time, statistics are accumulated about die rolls, and when it becomes exceedingly unlikely that the rolls are generated by a uniform distribution, the baseline Die Distribution hypothesis is invalidated.

Probabilistic program synthesis for characterization of novelty

Once novelty is detected, it must be characterized by synthesizing new hypotheses that better explain the data. In general, there may be ambiguity as to the correct new hypothesis, particularly with partial observability. For example, when we observe that the price of a property is double what it was before, we might not observe the rents associated with the property, so there are many different possible hypotheses about those rents. One hypothesis is that all rents associated with that property are doubled; a second is that the purchase price, but not the rents of all properties in the monopoly are doubled; and a third is that only that particular price is doubled. Therefore, Coltrane maintains multiple hypotheses about the novelty simultaneously. Over time, evidence will be accumulated that will strengthen or weaken each of these hypotheses, or perhaps invalidate them all, resulting in further hypotheses needing to be synthesized.

Because the number of possible hypotheses can be infinite, the synthesis process must be carefully controlled. We formalize the synthesis process using a hypergraph. *Figure 2* shows an example of a small fragment of a hypergraph for hypotheses about die rolls. Nodes in the hypergraph represent hypotheses. Hypernodes represent hypotheses that can coexist. Edges represent the synthesis of a new set of coexisting hypotheses from a previous hypothesis; edges are annotated by evidence that invalidated the previous hypothesis and led to the synthesis of the new hypothesis.

Our approach to probabilistic program synthesis contrasts with most previous work (e.g. (Lake et al. 2015; Saad et al. 2019). Normally, synthesis proceeds by searching over a domain specific language (DSL) of programs to find the best program that combines prior probability with likelihood of data. We also use a DSL of programs in the language of hypotheses for a node, but search is incremental, taking only small steps as required by observations, proceeding using an Occam's Razor principle to only prefer small changes. Our approach is based on the idea that when novelty is introduced in an existing domain (which is the focus of our research), it will usually be implemented by relatively small changes to the domain, rather than needing to learn entirely new concepts from scratch.

Architecture for Planning with Uncertainty from Novelty

Once novelty has been characterized, the Coltrane system must decide what actions to take to achieve its goals. The requirement to adapt to novelty online, as it occurs, means that extended periods of machine learning or re-encoding of knowledge are infeasible. For example, when the price of a Monopoly property is changed, the system must then make its next move within a reasonable amount of time, continuing to play the game in the face of potentially partially characterized novelty. In addition, the reasoning element should be generally applicable to a variety of reasoning domains, handling uncertainty, adversarial or multi-agent domains, and partial knowledge. To be relevant, it must scale to domains with significant reasoning complexity and approaching real-world applications.

Coltrane combines Monte Carlo tree search (MCTS) with structured knowledge heuristics and knowledge compilation to reason in response to novelty (Chaslot et al. 2008). MCTS, with enhancements, has been successful in addressing multiple game domains (Silver et al. 2016) as well as general gameplaying (Genesereth & Björnsson 2013). *Figure 4* depicts the approach. MCTS is an online policy learning search function that represents nodes as states and links as actions. MCTS iterates over four stages of reasoning: (1) Selection, select the next leaf node *L* to explore according to current rankings; (2) Expansion, add a child node *C* to the tree according to legal or preferred actions; (3) Simulation,

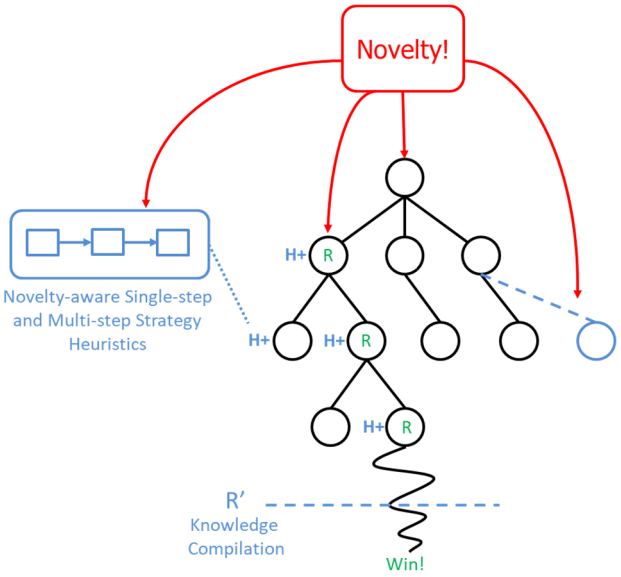

Figure 4: MCTS with heuristics for addressing novelty

playout the new child node according to a simple policy (e.g., random moves) to estimate the reward *R* of that new action; and (4) Backpropagation, use *R* to update the values from *C* to the root of the tree. We apply the UCT explore/exploit reasoning to enact these steps (Gelly & Wang 2006).

MCTS uses the newly characterized novelty knowledge representations: leveraging new and changed actions; working around obstacles and using the previously understood aspects of the domain; or combinations of the two to best meet the goals. For example, MCTS may take advantage of improved mortgage rates to gain more cash to build houses faster than opponents.

To enable scaling of reasoning to domains such as novel Monopoly, we enhance the basic algorithm with: (1) novelty-aware simple and complex strategy heuristics that are factored into each node's favorability along with the reward signal following (Baier & Winands 2014); and (2) knowledge compilation that computes ramifications of novelty in ways that are useful to the planning.

Coltrane's strategy heuristics are used to bias the search, and they can be single-step or multi-step. For example, one heuristic prefers to maintain a single-step cash reserve to avoid bankruptcy at all times, and another prefers to make multi-step successive purchases and trades to obtain valuable monopolies. The heuristics are built using the same domain model primitives that are characterized by novelty, allowing them to automatically adapt and be invalidated when the parts of the world they rely on are modified due to novelty. For example, when novel prices and rents are introduced, the heuristic to obtain high value monopolies adjusts to the newly advantageous monopolies.

Knowledge compilation in the Coltrane architecture is designed to create optimized representations of the

ramifications of inferred novelty for use by reasoning. It is inspired by the intuitions humans have when a new novel information is understood, e.g. when a person discovers that the rents on all of the railroads in Monopoly are 10x their previous value, they intuitively know that these properties are strategically valuable. Coltrane's knowledge compilation uses dynamic programming and other compilation techniques to provide declarative versions of these ramifications. These are computed when novelty is incrementally characterized, and made available directly to MCTS, which can use them for early cut-offs of projections, partial evaluation functions, or as elements of heuristics, providing improved reasoning and scalability that is novelty aware.

Demonstration by Applying to Game Domain with Unanticipated Novelty Injection

We have demonstrated our architecture using the games of Monopoly and Polycraft (a Minecraft video game variant). We worked with independent evaluators. One developed a Monopoly simulator called GNOME that includes the ability to introduce novelty into the game. GNOME also provides standard Monopoly-playing agents to provide opponents for Coltrane. One developed a pogo stick construction task within the Polycraft environment. To create the pogo stick, the agent must find trees in the environment, break them to get logs, craft the logs into sticks and planks, create a tree tap to get rubber from the tree sap, and finally combine the ingredients at a crafting table to create the pogo stick. Each action incurs a cost, and the task is to construct the pogo stick while minimizing cost.

Coltrane played tournaments consisting of 20-50 games with three standard opponents in Monopoly and alone in Polycraft. At some point during a tournament, novelty may be introduced. As much as possible, this novelty was designed to be important enough that failure to handle the novelty would cause a severe deterioration of performance, or alternatively cause a failure to exploit a significant new opportunity. In some tournaments, no novelty is introduced to check for false positives.

To ensure a challenging and interesting environment for studying novelty, we limited Coltrane to only be able to observe events that happen during the game. For example, we did not allow Coltrane to simply survey the board for new properties; it could only discover a new property if a player landed on it.

Description of Domains with Level 1-3 Novelty

GNOME implements Monopoly with near-standard rules and includes the ability to inject novelty at different levels. Polycraft uses the underlying Minecraft engine with a symbolic application programmer's interface (API) to observe and take actions in the world. We studied Coltrane's ability to handle levels 1-3 of Senator's novelty hierarchy (Senator 2019). Level 1 is class novelty, which corresponds to the appearance of new entities with unknown attributes in the environment. For example, there may be a new property on the board, or a new card in the Chance deck. Level 2 is attribute novelty, which corresponds to changes in the attributes of entities, such as changes in prices and rents, the amount paid for street repairs, or a change to the color of a property, which determines the sets of properties that constitute monopolies. Changes in probability distributions, such as over die rolls or cards, also fall under level 2 novelty. Level 3 is representational novelty. Representational novelty is harder to define precisely. In GNOME, it includes changes to the topology and structure of the game board, such as properties taking up multiple spaces on the board.

Instantiation of COLTRANE for Example Games

For detection, we used Scruff to build hierarchies of the sort shown in Figure 3. In Monopoly, we built hierarchies for many aspects of the game, including:

- **Dice** For this test instantiation, we model dice as sets of variable size, where each individual die is modeled in a manner similar to Figure 3. In our experiments, we have focused on lower levels of hierarchy, modeling a wide range of distributions over integers, but we have not yet modeled other kinds of faces. Changes to the number of dice or content of faces are rapid to detect, while changes to distributions take time and require statistical tests.
- **Cards** We modeled changes to the content of existing cards (e.g. amount paid or Go To target) in the Chance and Community Chest decks, addition or removal of cards, and changes to the distributions over cards. Changes to the contents of known existing cards are obvious to detect, additions take time but are detected as soon as the additional card is turned up, while removals and changes to distributions require more time and tests.
- **Prices and rents** We modeled changes to a wide variety of prices and rents in the game. Although these do not present a challenge to detection, they do present an interesting opportunity for characterization (see below).
- **Colors** In Monopoly, colors of properties are significant, as they define the sets of properties that can support houses and hotels. We modeled the possibility that the colors of properties could change, which can be detected easily but provides interesting opportunities for characterization.
- **Board structure** We modeled the board as a graph of locations, which allows for a wide range of changes to the structure of the board. Since we do not allow Coltrane to see the full board, novelty is only detected when it lands at a different location from expected based on the die roll.

In Polycraft, we built hierarchies for individual actions, including **breaking blocks**, **moving between locations**,

crafting each recipe, and **applying tools**. Each action's preconditions and effects were modelled as lists of symbolic changes in the environment. The model elements of each action are the nouns in the precondition of each action, and these are arranged hierarchically according to the class hierarchies in Polycraft (e.g., a log is a type of block).

Characterization

For each of the above aspects of the game for which we modeled novelty, we used our incremental probabilistic program synthesis based on the Occam's Razor principle to characterize the novelty. For some aspects of the games, such as dice or cards, the characterization is relatively simple, but others open a rich set of possibilities.

When a new monetary amount, such as a price or rent is observed, novelty is immediately detected, but a question arises as to how general or specific that the change is. There are two dimensions of generality. One dimension is in the entity to which the new amount applies. For example, if the purchase price for Boardwalk is doubled, does that mean that the purchase price for Park Place, which has the same color, is doubled? What about other properties on that side of the board? What about all properties on the board? The second dimension is in the type of monetary amount. For example, if the base rent of a property is doubled, does that apply to all other rents of the property? What about mortgage price and price for building houses?

Our synthesis process considers all these possibilities. When a new monetary amount for a price on a property is observed, it generates the cross-product of entity clusters containing that property and price category clusters containing that price. For each of these possibilities, a new hypothesis is synthesized saying that for all properties in the entity cluster and all prices in the price category, the price is adjusted, relative to baseline, in a manner consistent with the observed monetary amount. Over time, new monetary amounts will be observed that confirm or invalidate these hypotheses. This mechanism enables Coltrane to quickly characterize significant changes to the game rules that have a major impact on game play, for example, that all rents with hotels on all properties on the board have doubled. With this rule change, a winning strategy is to race to build as many hotels as possible. Coltrane's characterization explores the infinite hypothesis space incrementally similar to scientific exploration, applying Occam's Razor.

A similar process applies to property colors. When we encounter a new color for a property, Coltrane synthesizes multiple new hypothesizes, including the hypothesis that only that property has changed color, the hypothesis that all properties in the set have changed color, and the hypothesis that that set has traded colors with another set.

For board structure, as we discussed, novelty is detected when a player lands on a different location from that which was expected based on die rolls. When this happens, characterization is quite challenging because there are so many possible explanations. Coltrane's synthesis process searches through a space of board mutilations, such as addition of locations, removal of locations, and swapping of locations. Because of the large number of possibilities, Coltrane sometimes maintains thousands of concurrent hypotheses about the board structure before narrowing them down. Coltrane's representation allows for arbitrarily large boards, so the set of possible hypotheses is infinite, requiring our lazy hypergraph search using the Occam's Razor principle.

For Polycraft actions, Novelty is detected when actions do not have their predicted effects, and the characterization of that novelty is estimated by transformations that change existing actions into new actions. These include an effect change transformation (e.g., the cost of an action is halved) that applies at various levels of the hierarchy and a specialization transformation (e.g., a new breaking action is created for a when you hold an axe).

Planning

For planning in Monopoly, we implemented the MCTS-UCT algorithm to conduct adversarial tree search (Browne et al. 2012) using a domain model of the non-novel GNOME Monopoly simulation. This search evaluates the nodes in the tree according to the player making the decision, and the favorability values are propagated accordingly. For random actions such as dice rolls or card draws, the node favorability value is backpropagated by the expected value, i.e., the weighted probabilities of the outcomes for each player. For planning in Polycraft, we implemented a hierarchical task network (HTN) representation of the domain and used the MCTS-UCT algorithm to search the plan space.

The Monopoly instantiation includes heuristics that are common in high-level play, creating trade offers for property groups that showed the highest return-on-investment (ROI), calculated as the ratio between rents and prices. We also created heuristics to help the agent maintain a cash reserve to avoid overspending and therefore going bankrupt too often, which we discovered early in our testing as a result of the value focused heuristics and node evaluations. The Polycraft instantiation uses HTN recipes to constrain the plan space and uses the cost of actions as the main heuristic. Because all of the representations in the system are modifiable by novelty characterization, each of these heuristics automatically adapt to novelty when it is discovered and characterized.

The Monopoly instantiation includes compiled knowledge in the form of a cash-flow heuristic, which computes the probability weighted expectation of cash income minus cash payout from the current board per player. This is computed for all starting board positions and combinatoric variations so that the relevant values are accessed via a look up table. This compiled knowledge was applied after a simulation cut-off of 100 game actions (which is about 10-

20 turns of Monopoly). This compiled knowledge showed a significant increase in win rate during initial testing, partially because it avoids the signal attenuation of long Monopoly games with a random simulation policy while providing a strong indication of overall favorability in the base game. The expected cash-flow is recompiled when the board state changes (e.g., a property is purchased) and when novelty is characterized.

Evaluation Results

We present two sets of experiments for each domain. The first set of experiments test Coltrane's ability to detect, characterize, and respond to specific forms of novelty known precisely by our team. Experiments were conducted on an Intel i7 Dell laptop 32GB of ram; a 100 game trial took between 1 and 2 hours. Table 1 shows the individual novelties chosen for this test. These novelties were chosen to cover the space of novelties that Coltrane is able to detect and reason about, some may have a limited impact on play strategies (such as N1) and some may have a great impact on play strategies (such as N10). Table 2 shows the internal monopoly test detection and characterization F1 scores for these novelties, running in 10 trials of 10 games each. Coltrane can accurately detect and characterize most of these novelties in the span of a few games, and card and dice novelties take longer due to partial information and infrequent events. Table 3 shows the novelty reaction performance for these novelties in tournaments of 100 games. Coltrane can recover from novelty introduction to increase its win rate for a majority of these novelties, with results significant at p=0.01 in N9, N10, and in total, calculated by Fisher's exact test.

The second set of experiments includes studies performed by the independent evaluator, where they introduced unanticipated novelties into the game and tested the performance of Coltrane against those novelties. These novelties were unknown to our team; even now, only a small subset of these novelties have been revealed to us, to allow for future experimentation. These experiments represent a true test of Coltrane's ability to handle unanticipated novelty.

Table 1: Internal Monopoly testing novelties

Novelty	Category	Description
N0	Prices	Increase street and general repairs card costs
N1	Spatial	Swap Boardwalk and Park Place positions
N2	Dice	Change probability distributions on dice
N3	Prices	Increase tax penalty
N4	Prices	Increase Virginia Ave. rent 10x
N5	Prices	Increase price of railroads 4.5x
N6	Dice	Change die to 1d6
N7	Color	Change colors for Park Place, Boardwalk, Baltic Ave.
N8	Prices	Increase rents for Indiana Ave.
N9	Prices	All Red properties to 0.25x price, 10x rent
N10	Spatial	Increase Boardwalk to 10 spaces

Table 2: Internal Monopoly test detection and characterization

Novelty	Novelty Detection F1 score	Novelty type Characterization F1 score	Games until Characterized
N0	0.000	0.000	N/A
N1	1.000	1.000	2
N2	0.667	0.889	25
N3	0.889	1.000	4
N4	1.000	1.000	2
N5	1.000	1.000	4
N6	0.667	0.667	100
N7	1.000	1.000	3
N8	1.000	1.000	2
N9	0.947	0.9474	2
N10	1.000	1.000	4

Table 3: Internal Monopoly novelty reaction performance results

Novelty	Detection off wins / 100 games	Detection on wins / 100 games	Win percent change	p value
N0	45	38	-15.6%	0.399
N1	45	40	-11.1%	0.740
N2	52	50	-3.8%	0.888
N3	40	41	2.5%	1
N4	45	48	6.7%	0.777
N5	50	54	8.0%	0.671
N6	52	59	13.5%	0.393
N7	39	45	15.4%	0.474
N8	34	45	32.4%	0.148
N9	43	64	48.8%	**0.005**
N10	39	61	56.4%	**0.003**
Total	**484 / 1100**	**545 / 1100**	**12.6%**	**0.010**

Table 4 shows the results of the Monopoly independent evaluation, and Table 7 shows the result for Polycraft. About 120 trials with 20 games per trial were run in each domain. Most of the trials contained a single novelty that was introduced partway through the trial, and some small number of trials contained no novelty. The M1 metric shows the number of novel examples needed for detection, i.e., the number of games before Coltrane detected novelty.

In Monopoly, on average the Coltrane system detected the novelty after 6.79 games out of the 20 game trial. This number dropped to .32 games for the trials in which Coltrane detected novelty at all. When novelty was detected, it was often within a single game. The M2 metric shows the novelty detection performance, i.e., is the percentage of games in which novelty was present and Coltrane reported a novelty detection. The accuracy of the novelty detection was 66.11%, indicating that most of the unanticipated novelties were detected by Coltrane's inference. The M3 metric shows the novelty reaction performance, which is the percentage of wins post-novelty introduction in a 4 player agent game. Coltrane's win rate went from 50.01% pre-novelty to 51.81% post-novelty when playing against rule-based agents that were specifically developed to handle the novelty. This win rate dropped and recovered quickly as novelty was characterized. and is three times the win rate of the other

players in the game, indicating strong recovery and play post-novelty.

Table 4: Independent Monopoly evaluation results

	Mean	Std. Error
M1: # of Novel Examples Needed	6.79	0.49
M2: Novelty Detection Performance	0.6611	0.0104
M3: Novelty Reaction Performance	0.5181	0.036

Table 5 shows the internal Polycraft testing novelties, and Table 6 shows the detection and performance results on these novelties. These Polycraft novelties were relatively straightforward to detect, as they each include noticeable changes to the items and blocks in the world or the outcomes of actions. Coltrane addressed the Item novelties, some of the Blocks novelties, and the Recipe novelty at pre-novelty performance levels. It was unable to accommodate the Obstacle novelty (the movement API requested teleportation), the distractor novelty, and some Blocks novelties. The distractor and blocks novelties seemed to cause the Coltrane planner to time out with a large number of objects to be hypothesized about and planned over.

Table 5: Internal Polycraft testing novelties

Novelty	Category	Description
NP0	Item	Wooden Axes are introduced that decrease the cost for chopping trees. Axes are in inventory
NP1	Item	Axes are on the ground
NP2	Item	Axe recipe is given
NP3	Obstacle	Wooden fences are introduced that surround trees
NP4	Blocks	Some of the Oak Trees are changed to Jungle Trees, which decrease the cost of collecting rubber
NP5	Blocks	The number of logs produced when chopping down a tree increases by a variable amount
NP6	Distractor	The outer wall blocks surrounding the arena are changed from bedrock to a different type of block
NP7	Recipes	Recipes are incorrectly altered, such that they would be rotated in a 3x3 crafting table

Table 6: Internal Polycraft test detection and performance

Novelty	Novelty Detection F1 score	Novelty Performance Score on Last 10 games (173K mean on base game)
NP0	1.000	178K
NP1	1.000	171K
NP2	1.000	163K
NP3	1.000	0
NP4	1.000	62K
NP5	1.000	173K
NP6	1.000	0
NP7	1.000	169K

In Polycraft, on average the Coltrane system detected the novelty after 2.75 games, and, in practice, Coltrane either detected novelty in the first game introduced or not at all. The M2 novelty detection performance captured 64% of novelties introduced. The M3 novelty reaction performance showed 34% of novelty games completed with a pogo stick.

Table 7: Independent Polycraft evaluation results

	Mean
M1: # of Novel Examples Needed	2.75
M2: Novelty Detection Performance	0.64
M3: Novelty Reaction Performance	0.34

Conclusions

This paper describes a new architecture and implementation of the Coltrane system that combines and extends probabilistic programming inference and planning under uncertainty to (1) recognize novelty when it occurs, (2) incrementally characterize the novelty as observations of the novelty accrue, and (3) continually adapt its task-based reasoning to the evolving understanding of the novelty in order to maximize task performance. We evaluated Coltrane on two domains with a set of unanticipated novelties and in a set of internal tests to show generality of performance. We found that Coltrane demonstrates this performance in the stochastic game of Monopoly and the deterministic game of Polycraft with novelties introduced.

Coltrane represents a new approach to creating resilient AI systems that can extend beyond their initial learning and knowledge encoding. Future work may: enhance Coltrane's Probabilistic Program Inference and Probabilistic Program Synthesis to detect and characterize new and wider ranges of novelty; enhance Coltrane's novelty aware planning to react to these novelties; apply Coltrane to new domains that have different inference, representation, and reasoning requirements; and extend these approaches to improve the performance of other AI systems under novelty.

Ethics Statement

We expect the work described in this paper to have significant benefit to society, and in particular to mitigate some of the ethical concerns with AI. Current AI systems' brittleness in the face of novelty is a serious ethical problem, because they are liable to persist with their behaviors even when it becomes unsafe. For example, a self-driving car that cannot recognize a novel situation where all the vehicles in front of it are exiting a lane may lead it to continue to drive in that lane, leading to a crash, as happened in a Tesla crash in China. One of the principles of our approach is to reason about representational uncertainty, so when faced with multiple hypotheses after detecting novelty, Coltrane could behave conservatively. This approach has the potential of making AI systems safer.

References

Baier, H., & Winands, M. H. (2014). *Monte-carlo tree search and minimax hybrids with heuristic evaluation functions*. 45–63.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(1), 1–43.

Chaslot, G., Bakkes, S., Szita, I., & Spronck, P. (2008). *Monte-Carlo Tree Search: A New Framework for Game AI.* AIIDE.

Clark, A. (2013). Whatever next? Predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*, *36*(3), 181–204.

Cox, M. T. (2005). Metacognition in computation: A selected research review. *Artificial Intelligence*, *169*(2), 104–141.

Friedman, S. E., Musliner, D. J., & Rye, J. M. (2014). Improving automated cybersecurity by generalizing faults and quantifying patch performance. *International Journal on Advances in Security*, *7*(3–4).

Friston, K. (2008). Hierarchical models in the brain. *PLoS Computational Biology*, *4*(11), e1000211.

Gelly, S., & Wang, Y. (2006). *Exploration exploitation in go: UCT for Monte-Carlo go*.

Genesereth, M., & Björnsson, Y. (2013). The international general game playing competition. *AI Magazine*, *34*(2), 107–107.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Lake, B. M., Salakhutdinov, R., & Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, *350*(6266), 1332–1338. https://doi.org/10.1126/science.aab3050

Langley, P. (2020). *Open-World Learning for Radically Autonomous Agents*. 13539–13543.

Marling, C., Sqalli, M., Rissland, E., Muñoz-Avila, H., & Aha, D. (2002). Case-based reasoning integrations. *AI Magazine*, *23*(1), 69–69.

Pfeffer, A., & Lynn, S. K. (2018, August). *Scruff: A deep probabilistic cognitive architecture for predictive processing*. Conference on Biologically Inspired Cognitive Architectures, Prague, Czech Republic.

Rao, R. P., & Ballard, D. H. (1999). Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, *2*(1), 79.

Saad, F. A., Cusumano-Towner, M. F., Schaechtle, U., Rinard, M. C., & Mansinghka, V. K. (2019). Bayesian synthesis of probabilistic programs for automatic data modeling. *Proceedings of the ACM Conference on Principles of Programming Languages (POPL)*, *3*, 37.

Schulte, E. M., Weimer, W., & Forrest, S. (2015). *Repairing COTS router firmware without access to source code or test suites: A case study in evolutionary software repair*. 847–854.

Senator, T. E. (2011). Transfer learning progress and potential. *AI Magazine*, *32*(1), 84–84.

Senator, T. E. (2019). *Science of AI and Learning for Open-world Novelty (SAIL-ON)*. DARPA Proposers' Day Meeting, Arlington, VA. https://www.darpa.mil/attachments/SAIL-ON Proposers Day Distro A no notes.pdf

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., & Lanctot, M. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Veloso, M. M. (1992). *Learning by analogical reasoning in general problem solving*. CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE.