Rational OpenCog Controlled Agent

Nil Geisweiller and Hedra Yusuf

SingularityNET Foundation, The Netherlands
{nil,hedra}@singularitynet.io

Abstract. TODO

Keywords: Symbolic Reinforcement Learning · Procedural Reasoning · OpenCog · Minecraft

1 Introduction

The goal of this project is to make an agent as rational as possible, not necessarily as efficient as possible. This stems from the concern that in order to autonomously gain efficiency the agent must first be able to make the best possible decisions, starting first in the outer world, and then in the inner world.

The paper presents

The agent starts in a completely unknown environment

The idea is that reasoning is used at all levels, discovering patterns from raw observations, building plans and making decisions.

It is a work in progress.

Neural networks are excellent at interpolation, but are rather poor at extrapolation, what we need for true intelligence is a system that thinks critically.

Rarely do causes and effects take place over arbitrary temporal scales. For instance it is unlikely to find a cause that may produce the same effect, or an effect at all, after 1ms, 1 century or any time in between. For that reason we focus on a real time temporal logic.

1.1 Related Work

1.2 Contributions

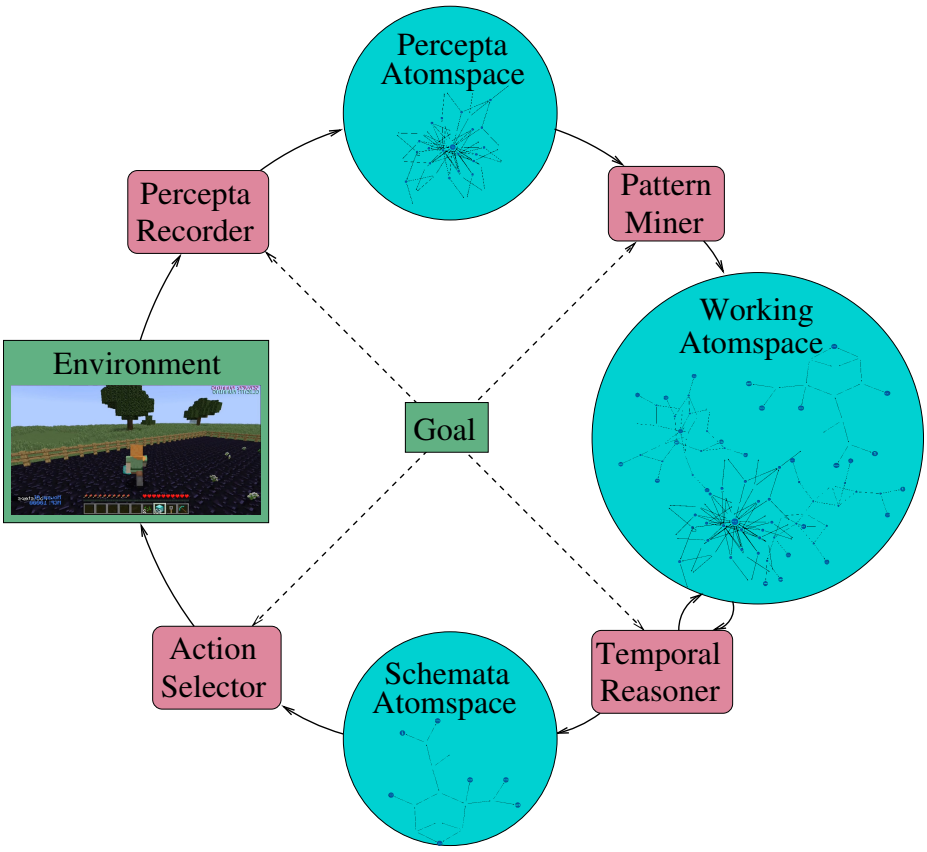
The contributions of that paper are:

1. Design an architecture for controlling an agent based on that temporal reasoning extension.

1.3 Outline

1. ROCCA
2. Minecraft experiment

2 Rational OpenCog Controlled Agent



To experiment with temporal and procedural reasoning in the context of embodied virtual agents in unknown environments we have implemented a project called ROCCA, which stands for *Rational OpenCog Controlled Agent*. ROCCA essentially acts as an interface between virtual environments such as Malmo [**?**] or OpenAI Gym [**?**] and OpenCog. It provides an Observation-Planning-Action control loop as well as various launchers to run OpenCog processes such as PLN reasoning, pattern mining, etc. Provided a top goal, such as maximizing a reward, ROCCA orchestrates the necessary learning and the planning to fulfill that goal. One may possibly see ROCCA as a reinforcement learning agent with the particularity that learning and planning are, at least in principle, entire done via reasoning. In that respect it is similar in spirit to OpenNARS for Applications (ONA) [**?**] but uses PLN as its core reasoning logic rather than NAL [**?**].

2.1 Memory

The memory of the agent is split into three AtomSpaces:

1. The *Percepta*[1] AtomSpace holds timestamped observations as they come into the system.
2. The *Working* AtomSpace holds any kind of data, ranging from timestamped observations to predictive implications. Most knowledge used and inferred during the course of learning are usually dumped into this AtomSpace.
3. The *Schemata* AtomSpace holds predictive implications representing cognitive schematics.

The reasons for such splitting are:

1. Increased *efficiency*, both the Percepta and Schemata AtomSpaces are specialized to hold only what is required for rapid IO processing.
2. Increased *clarity*, troubleshooting and repairing is easier that way.

2.2 Processes

Figure **??** shows a large cycle moving through all phases sequentially. In theory that is valid, but in practice we want to split processes, TODO.

ROCCA is composed of two main processes:

1. *Control* for real-time reactive agent control;
2. *Learning* for non-reactive background learning.

2.3 Control

The control process is composed of control cycles, each decomposed into Observation, Planning and Action phases, described below:

1. *Observation*:
   (a) receive and timestamp observations, reward included, from the environment,
   (b) store the timestamped observations and reward in the Percepta AtomSpace.
2. *Planning*:
   (a) select the goal for that iteration,
   (b) find plans fulfilling that goal from the Schemata AtomSpace,
   (c) build a mixture distribution from these plans.
3. *Action*:
   (a) select the next action via Thompson Sampling according to that mixture distribution,
   (b) timestamp and store the selected action in the Percepta AtomSpace,
   (c) run the selected action and by that update the environment,

[1] Percepta means percepts in Latin. It is the plurial form of perceptum. Latin was chosen over English so that the difference between singular and plurial does not reduce to a single letter, s, which can be prone to error when reading and writing code.

None of these steps are computationally expensive and involve algorithms that are at most linear with the size of the Percepta and Schemata AtomSpaces. For real-time responsiveness however such control cycle must be temporally bound by a constant. This may entail incorporating other mechanisms such as filtering and forgetting. As important as this problem is however, it has been let aside for now and is left for future research. Given the small environments ROCCA has been tested with, it has not been a practical problem at this point but will soon be. More on the subject can be found in some of our previous publications [**?**].

Let us now provide a more detail account of these three phases.

Observation During the observation phase, data coming from the environment are timestamped and stored in the Percepta AtomSpace. The format used in this paper to represent that is `datum@timestamp`. For instance if at cycle 10 the agent observes `outside(house)` and `hold(key)`, then `outside(house)@10` and `hold(key)@10` are inserted to the Percepta AtomSpace.

Planning The first step of the planning phase is to select a goal G to fulfill. That is an important step that has been elaborated in publications such as [**?**, **?**]. In the current version of ROCCA though it merely returns a constant goal which is to gain a reward within a forward window. Once a goal has been selected, the agent queries the Schemata AtomSpace with the following pattern matcher query

$$\$C \wedge \$A \rightsquigarrow^{T} G$$

where

- $\$C$ is a variable representing the context,
- $\$A$ is a variable representing the action or action plan [2],
- T is a time delta selected within that forward window,
- G is a selected goal.

All returned candidates, predictive implications, are then filtered according to their contexts, only retaining the ones with contexts being evaluated to true at the current time. Ideally a second order probability of a context being current true should be evaluated, but in the current version of ROCCA a crisp evaluation is used for the sake of simplicity.

From this set of predictive implications with true contexts, a second order mixture distribution is built approximating very roughly a Solomonoff distribution as explained in [**?**].

Action The Action phase consists of the following steps:

1. Select the next action via Thompson Sampling according to the mixture distribution built during the planning phase.

[2] Variables are actually typed so that the pattern matcher cannot confuse what is context and action.

2. Timestamp and store the selected action in the Percepta AtomSpace.
3. Run the selected action and updates the environment.

The trickiest step here is selecting the next action via Thompson Sampling. The novelty is that the second order probabilities can be leveraged by Thompson Sampling. For example, assume we have two actions to choose from, A_1 and A_2, among two predictive implications

$$C_1 \wedge A_1 \rightsquigarrow^T G \stackrel{\text{m}}{=} <0.7, 0.9>$$
$$C_2 \wedge A_2 \rightsquigarrow^T G \stackrel{\text{m}}{=} <0.8, 0.1>$$

Using only first order probabilities, the choice is clear, action A_2 should be selected, because its probability of success, 0.8, is greater than that of A_1, 0.7. However once introducing second order probabilities, that choice becomes less clear because the probability of success of A_2 has also a low confidence. In that case what Thomspon Sampling does is to sample first order probabilities from the second order distributions. Stated informally, the idea is to consider the possibilities that the agent might be living in a world where A_2 has a lower probability of success than A_1. Figure **??** shows a graphical representation of the second order distributions of the probabilities of success of A_1 and A_2.

In this example above only two actions among two cognitive schematics are considered, but in practice we usually have a handful of actions among a potentially very large number of cognitive schematics. How to deal with a multitude of cognitive schematics is treated in [1] as mentioned above.

For that we use a variation of Solomonoff induction described in [1] which is especially suited for plans described by conditional second order distributions, in other words predictive implication links. More specifically plans are predictive implication links of the form

$$C \wedge A \rightsquigarrow^T G$$

called *Cognitive Schematics* or *Schemata*. Which can be read as *"in some context C, if some action A is executed, then after T time units, the goal G is likely to be fulfilled"*. Note that A does not need to be limited to elementary actions but can be composite as well, potential describing entire plans composed of action sequences, conditionals and such [NEXT: mention behavior tree]. In other words, Cognitive Schematics should be expressive enough for general decision making. The likelihood of goal fulfillment is specified by the truth value of the predictive implication link, which is not indicated in that notational format but is present in the extended Atomese format. The difficulty then comes down to discovering cognitive schematics that are as predictive and applicable as possible, which translates into predictive implication links which broad contexts, high strength and high confidence. An example of an ideal cognitive schematic would be

$$\mathbb{True} \wedge A \rightsquigarrow^T G$$

that has a strength and confidence of one, and is universally applicable[3]. For real environments and goals however, such ideal is almost never reached. More

[3] $\mathbb{True}$ represents the predicate that is always true

often than not we will have instead a large collection of cognitive schematics with narrow contexts and poor predictiveness. To make the best of it, a mixture of second order distributions is built over all applicable cognitive schematics as described in [1] and action selection is performed using Thompson Sampling, which has been proven to be asymptotically optimal in general environments [2], and excellent at balancing exploitation and exploration in principle.

2.4 Learning

As hinted above, the ultimate goal of the learning phase is to discover maximally useful cognitive schematics, and by useful it is specifically meant that they are as predictive and cover as many cases as possible.

In principle these two processes, Control and Learning, could happen in parallel. In practice though, purely for technical simplicity, they alternate in series. Basically, the agent starts in a control phase, a number of control cycles occur as the agent interacts with its environment, gathers observations and takes (initially random) actions, which follows then by a long learning phase when the agent discover regularities in its environments and build plans, and finally resumes the control phase to test how the agent performs after learning.

Pattern Mining The advantage of using Pattern Mining is that already at this stage, abstract rules can be induced. For instance instead of having

$$go(right) \rightsquigarrow^1 square(right)$$
$$go(left) \rightsquigarrow^1 square(left)$$

we can have

$$go(x) \rightsquigarrow^1 square(x)$$

Pattern mining is a relatively inexpensive way to generate mono-action cognitive schematics. Mining poly-action cognitive schematics is possible but it has two drawbacks

1. In the worse case, the computational cost of mining goes up exponentially with the size of the action sequence to mine.
2. The number of observations to accumulate in order to generate cognitive schematics with decent confidences goes up exponentially as well.

The latter is really the most limiting factor because we cannot buy our way out of it. If each percept takes a certain amount time to observe, determined by the control cycle period in the case of primary observations, then we have to do through them, we cannot speed time up. This is even more true for abstract percepts that can only be observed at periods which are multiples of control cycle periods. In order to address these limitations, and to build poly-action cognitive schematics with realistic probability estimates and confidences, we need reasoning.

Temporal and Procedural Reasoning The most important

3 Experiment with Simple Minecraft Environment

In this experiment we built a minecraft environment using Malmo, which is a platform for Artificial Intelligence experimentation and research built on top of Minecraft. The demo environment consists of a small house with a locked door, diamonds inside and a key to get into the house. The agent, initially located outside of the house, can perform different actions like getting a key, opening a door of the house and collecting the diamonds in order to achieve a reward.

The aim of this experiment is to make the ROCCA agent learn from the actions and perceptions in the minecraft environment and do efficient planning so as to be able to collect as many diamonds as possible and accumulate reward. The ROCCA agent will be able to perform a series of possible actions with a goal of achieving a reward and learns from them by applying PLN (Probabilistic Logic Networks) and Pattern Miner, which can be seen as a specialized form of PLN reasoning. The Planning, the discovery of cognitive schematics, is also handled by PLN and its temporal reasoning rule base.

Fig. 1. Simple Minecraft demo with a house and a key.

There are lists of allowed actions provided by minecraft that an agent can perform like moving, turning, picking etc.. but due to the limited processing capacity we have to handle the observations from each action and to reduce complexity, we proposed to have an abstract action and perception where unnecessary details have been omitted. With that we generate three abstract actions namely go-to-key, go-to-house, go-to-diamonds where each of them contains a series of actions and returns an abstract perception about where the agent is (inside house, outside house, next to closed door etc..), about its inventory (has key, diamond pickaxe etc..) and the reward of completing a given action.

We perform various experiments tuning different parameters. A typical experiment has two iterations of the learning-training process with a duration of fifty iterations for each training. In the first cycle the agent will not have prior

knowledge hence no learning will take place. The agent pursues the environment and builds its knowledge base by trying a combination of fifty randomly weighted actions. At the end of the first cycle the agent will have enough background knowledge to apply Pattern miner and PLN temporal reasoning. Hence, during the second cycle, the agent will be able to learn and plan the desired cognitive schematics which leads to a positive goal of getting a reward.
The ROCCA agent is able to learn the following cognitive schematics with higher strength.

$$outside(self, house) \wedge \widehat{go\_to(key)} \rightsquigarrow^{1} hold(self, key)$$

$$hold(self, key) \wedge \widehat{go\_to(house)} \rightsquigarrow^{1} inside(self, house)$$

$$inside(self, house) \wedge \widehat{go\_to(diamond)} \rightsquigarrow^{1} reward(1)$$

In this experiment, we measure the agent's performance by the cognitive schematics learned and accumulated rewards achieved. The ROCAA agent is successful in learning the required cognitive schematics which leads the agent to collect more rewards in the second cycle. However, these findings with a simple minecraft environment with only few actions might not tell the overall performance of ROCCA. As a future work, further extensive experiments are needed to conclude the performance achieved.

4 Conclusion

1. Forgetting
2. Functional Pattern Mining
3. Attention Allocation
4. Inner actions
5. Port to OpenCog Hyperon
6. Second order evaluation of context being currently true

References

1. Geisweiller, N.: Partial operator induction with beta distributions. In: Iklé, M., Franz, A., Rzepka, R., Goertzel, B. (eds.) Artificial General Intelligence. pp. 52–61. Springer International Publishing, Cham (2018)
2. Leike, J., Lattimore, T., Orseau, L., Hutter, M.: Thompson sampling is asymptotically optimal in general environments. In: Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence. p. 417–426. UAI'16, AUAI Press, Arlington, Virginia, USA (2016)