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

ROCCA is essentially a reboot of [11] using a modernized version of the OpenCog pattern miner, as well as a simplified version of OpenPsi [3, 4] leaving aside various built-in urges and modulators from MicroPsi [1] and using an action selection policy based on Thompson Sampling [16].

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

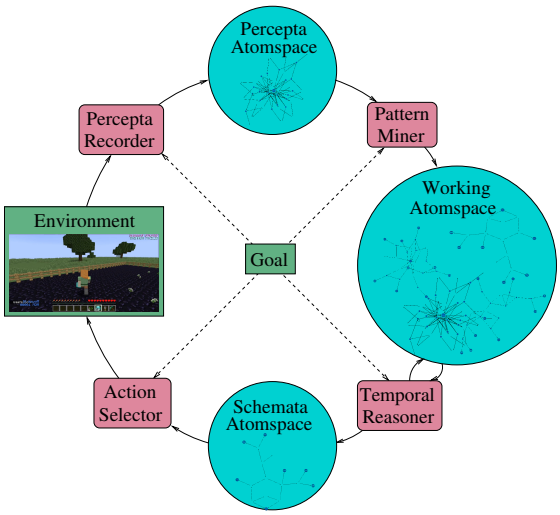

Fig. 1. Rational OpenCog Controlled Agent control and learning cycles merged into a single loop

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 [15] or OpenAI Gym [2] and OpenCog [14]. 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) [13] but uses PLN [8] as its core reasoning logic rather than NAL [17].

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 1 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,

[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.

(b) timestamp and store the selected action in the Percepta AtomSpace,
(c) run the selected action and by that update the environment,

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. Achieving this may entail incorporating other mechanisms such as filtering and forgetting cognitive schematics. 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 of *Attention Allocation* can be found in the Chapter 4 of [10].

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 [9, 12]. 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 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 would be used, 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 [5].

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

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. 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

$$C_1 \wedge A_1 \rightsquigarrow^T G \stackrel{\text{m}}{=} <0.6, 0.9>$$
$$C_2 \wedge A_2 \rightsquigarrow^T G \stackrel{\text{m}}{=} <0.7, 0.1>$$

Using only the strengths as proxy for probabilities, the choice is clear. Action A_2 should be selected, because its probability of success, which is 0.7, is greater than that of A_1, which is 0.6. However once introducing second order probabilities, that choice becomes less clear because the truth value of success of A_2 has a low confidence of 0.1. In that case, first order probabilities are first sampled from their corresponding second order distributions, and then these probabilities are compared. The action with the maximum probability wins. That is the essence of Thompson Sampling. More informally stated, 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 2 shows the second order distributions of the probabilities of success of A_1, in blue, and A_2, in red, for these truth values. As one may notice, the area under the red curve situated at the left of the blue curve is non-negligible. Meaning that the probability of being in a world where A_1 has a higher probability of success than A_2 is non-negligible as well. Because these strengths and confidences are periodically updated during the lifetime of the agent, one can see how Thompson Sampling is a great alternative compared to ε-greedy, offering a parameter-free mechanism to balance exploration and exploitation that dynamically adapts with the knowledge of the agent.

Note that in this example above only two actions among two cognitive schematics are considered, but in practice there usually is a handful of actions among a potentially very large number of cognitive schematics, with overlapping, possibly conflicting, contexts and goals. The resulting distributions of success of each actions in those cases are typically multi-modal and do not reduce to beta distributions. How to deal with such multitude of cognitive schematics is treated in [5].

For that we use a variation of Solomonoff induction described in [5] which is especially suited for plans described by conditional second order distributions, in other words predictive implications. More specifically plans are predictive implication links of the form

$$C \wedge A \rightsquigarrow^T G \stackrel{\text{m}}{=} TV$$

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*

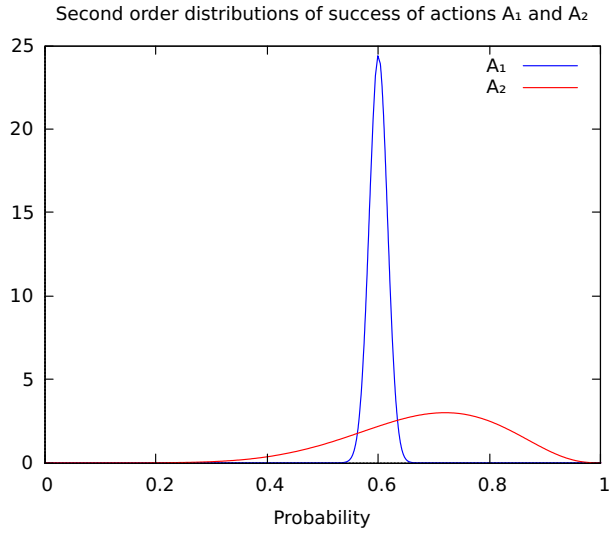

Fig. 2. Second order distributions of the probability of success of actions A_1 and A_2, using as parameters of the beta distribution $\alpha(s,c) = \alpha_0 + \frac{s.c.K}{1-c}$ and $\beta(s,c) = \beta_0 + \frac{(1-s).c.K}{1-c}$ where K, the *lookahead*, is set to 100, and α_0 and β_0 are set to Jeffreys prior.

be fulfilled, with probability described by TV". 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, akin to behavior trees. In other words, Cognitive Schematics should be expressive enough for general decision making. The difficulty then comes down to discovering cognitive schematics that are as predictive and widely applicable as possible, which translates into predictive implications which broad contexts, high strength and high confidence. An example of an ideal cognitive schematic would be

$$\mathbb{T}\mathtt{rue} \wedge A \rightsquigarrow^T G \stackrel{\mathrm{m}}{=} <1,1>$$

that has a strength and confidence of one, and is universally applicable[3]. For real environments and goals however, such ideal is beyond reach. More 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, in a manner that approximates a Solomonoff distribution, as described in [5]. Action selection is then performed based on that mixture using Thompson Sampling [16].

[3] $\mathbb{T}\mathtt{rue}$ represents the predicate that is always true

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 the pattern miner can discover temporal relationships between predicates, such as

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

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

meaning that if the agent goes to the right (resp. left), at the next time unit, it will be located on the right (resp. left) square. But it can also discover abstractions such as

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

The pattern mining algorithm used in ROCCA is actually a specialized form of reasoning, that is easily interlaced with more general forms of reasoning [6].

As it stands, pattern mining can be a relatively inexpensive way to generate mono-action cognitive schematics. Mining poly-action cognitive schematics is possible but 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 observation takes a certain amount time, 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. Also, in some cases, a particular action sequence may even never be observed at all, yet we still would like to have a way to estimate the likelihood of its success. In order to address these limitations, and to build complex cognitive schematics with good probability estimates and confidences, we need reasoning.

Temporal and Procedural Reasoning ROCCA uses a temporal extension of PLN, described in detail in [7], to infer new cognitive schematics, not only from direct observations, or also based on other cognitive schematics and background knowledge. An example would be to combine mono-action plans to infer poly-action plans. Another example would be to specialize a context or a goal. As for now three temporal rules are integrated into ROCCA:

1. Predictive Implication Direct Introduction: TODO.
2. Temporal Conditional Conjunction Introduction: TODO.
3. Temporal Deduction: TODO.

The precise semantics of these rules is detailed in [7]. An example of how these rules are used in ROCCA is detailed in Section 3.

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. 3. 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

We have introduced an agent called ROCCA that leverages the OpenCog framework for controlling an agent in unknown and uncertain environments. ROCCA is meant to be as rationally as possible, at the expense of efficiency when needed. For now it is only able to operate is very simple environments involving a handful of actions and observations. The type of learning used is however meant be open-ended, even though for now it is limited to syntactic hypergraph pattern mining and other limited form of temporal reasoning. As of now it is able to build mine small plan from direct observations and larger plans from smaller plans using probabilistic reasoning.

This is a preparatory work of a greater goal, which is to explore deeper forms of meta-learning and introspective reasoning. Towards that end the remaining milestones are:

1. Support temporal intervals and scales and add more temporal, as well as spatial inference rules.
2. Integrate ECAN [**?**] for *Attention Allocation*, as well as record attention spreading to learn Hebbian links and improve attention allocation.
3. Carry out concept creation and schematization, also called crystallized attention allocation, to speed up repetitive information processing even further. It should be noted that this done well should also provide a solution for the problem of creating abstract observations and actions.
4. Record more internal processes, not just attention spreading, as internal percepta to enable deeper forms of introspection.
5. Plan internal actions, not just external, to enable self-growth.

References

1. Bach, J.: Micropsi 2: The next generation of the micropsi framework. In: Bach, J., Goertzel, B., Iklé, M. (eds.) Artificial General Intelligence. pp. 11–20. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
2. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym (2016)
3. Cai, Z., Goertzel, B., Geisweiller, N.: Openpsi: Realizing dörner's "psi" cognitive model in the opencog integrative agi architecture. In: Schmidhuber, J., Thórisson, K.R., Looks, M. (eds.) Artificial General Intelligence. pp. 212–221. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
4. Cai, Z., Goertzel, B., Zhou, C., Huang, D., Ke, S., Yu, G., Jiang, M.: Openpsi: A novel computational affective model and its application in video games. Engineering Applications of Artificial Intelligence **26**, 1–12 (01 2013). https://doi.org/10.1016/j.engappai.2012.07.013
5. 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)
6. Geisweiller, N., Goertzel, B.: An inferential approach to mining surprising patterns in hypergraphs. In: Hammer, P., Agrawal, P., Goertzel, B., Iklé, M. (eds.) Artificial General Intelligence. pp. 59–69. Springer International Publishing, Cham (2019)
7. Geisweiller, N., Yusuf, H.: Temporal probabilistic logic networks (being reviewed for agi-23). In: Iklé, M., Franz, A., Rzepka, R., Goertzel, B. (eds.) Artificial General Intelligence. Springer International Publishing (2023)
8. Goertzel, B., Ikle, M., Goertzel, I.F., Heljakka, A.: Probabilistic Logic Networks. Springer US (2009)
9. Goertzel, B., Pennachin, C., Geisweiller, N.: Engineering General Intelligence, Part 1: A Path to Advanced Agi Via Embodied Learning and Cognitive Synergy. Atlantis Press (2014)
10. Goertzel, B., Pennachin, C., Geisweiller, N.: Engineering General Intelligence, Part 2: The CogPrime Architecture for Integrative, Embodied AGI. Atlantis Press (2014)
11. Goertzel, B., Pitt, J., Wigmore, J., Geisweiller, N., Cai, Z., Lian, R., Huang, D., Yu, G.: Cognitive synergy between procedural and declarative learning in the control of animated and robotic agents using the opencogprime agi architecture. Proceedings of the AAAI Conference on Artificial Intelligence (2011)

12. Hahm, C., Xu, B., Wang, P.: Goal generation and management in nars. Springer-Verlag, Berlin, Heidelberg (2021)
13. Hammer, P., Lofthouse, T.: 'opennars for applications': Architecture and control. Springer-Verlag, Berlin, Heidelberg (2020)
14. Hart, D., Goertzel, B.: Opencog: A software framework for integrative artificial general intelligence. In: Proceedings of the 2008 Conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference. p. 468–472. IOS Press, NLD (2008)
15. Johnson, M., Hofmann, K., Hutton, T., Bignell, D.: The malmo platform for artificial intelligence experimentation. In: International Joint Conference on Artificial Intelligence (2016)
16. 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)
17. Wang, P., Awan, S.: Reasoning in non-axiomatic logic: A case study in medical diagnosis. In: Schmidhuber, J., Thórisson, K.R., Looks, M. (eds.) Artificial General Intelligence. pp. 297–302. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)