Politechnika Wrocławska

ACS2

QIPLSIGML

Kraków, April 26 — 28, 2018

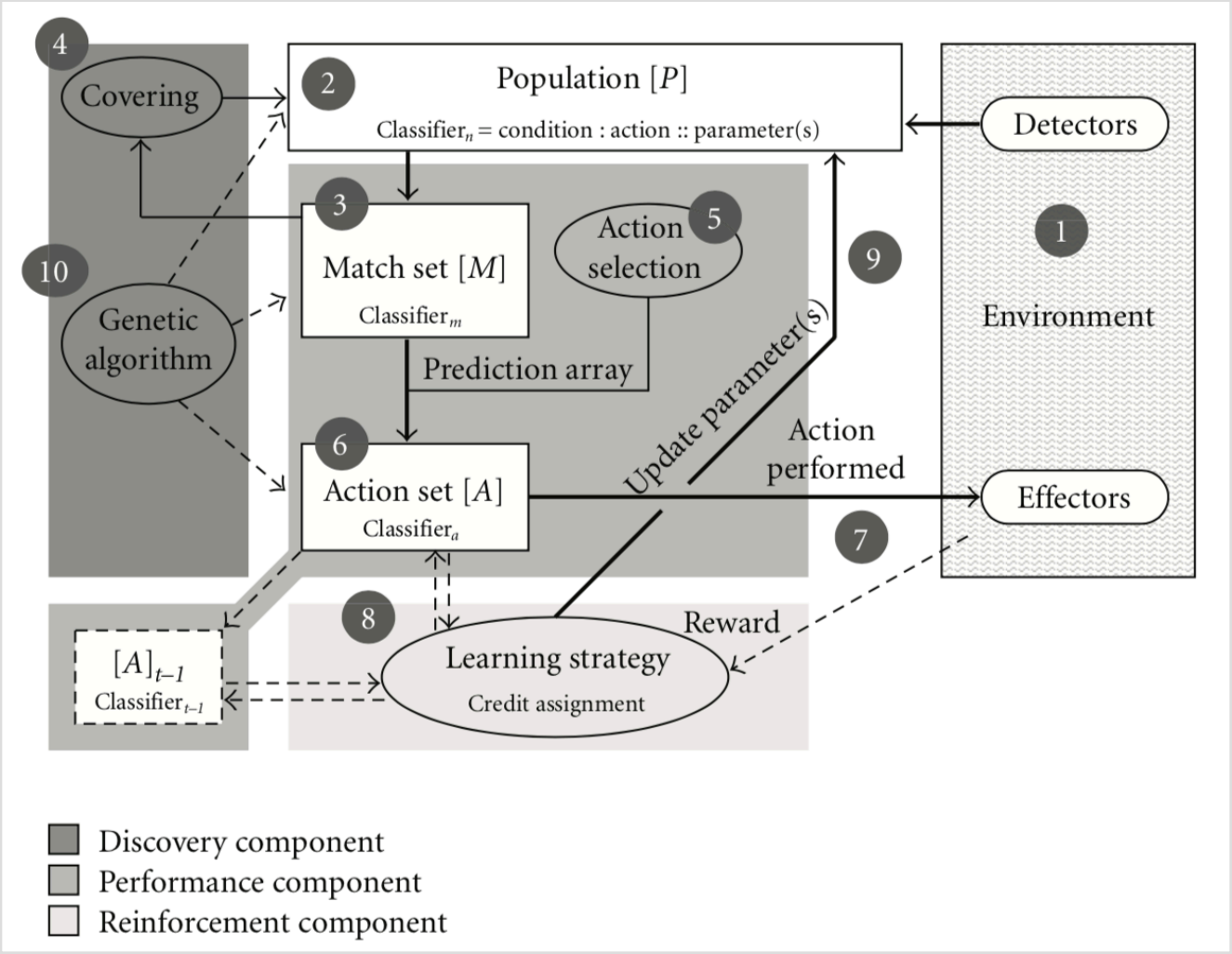
Machine Learning meets Quantum Computation

mgr inż. Norbert Kozłowski

Road map

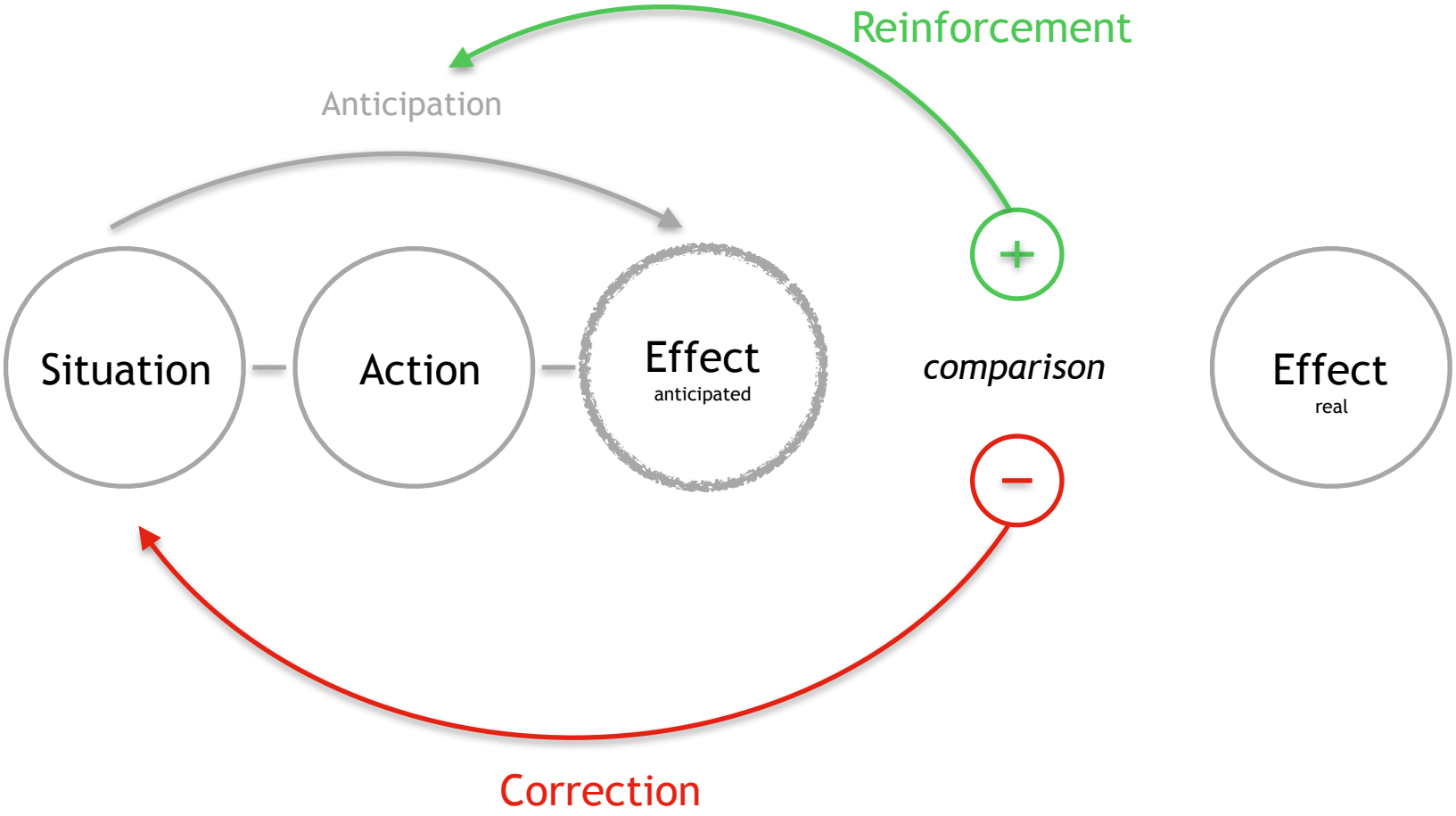
- Learning Classifier Systems (LCS)
- OpenAI Gym
- PyALCS framework
- Experiments

Learning Classifier Systems



Anticipatory Behavioral Control

Joachim Hoffmann 1993



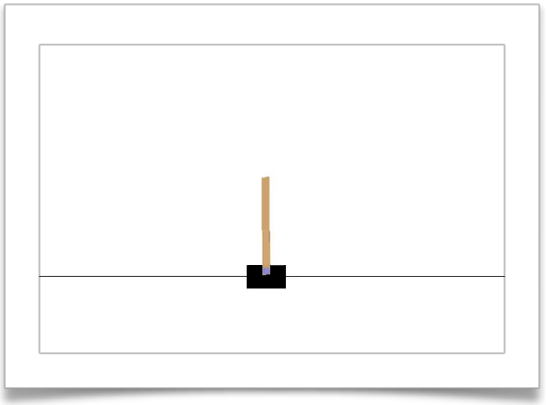
Classifier Structure

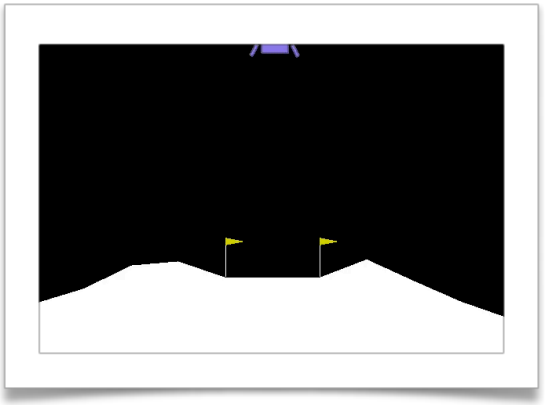
CONDITION - ACTION - EFFECT

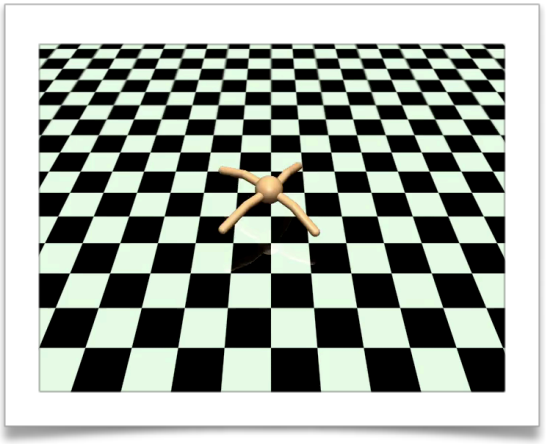
QUALITY, REWARD, INTERMEDIATE REWARD, MARK, NUMEROSITY, EXPERIENCE, T_{ALP}, T_{GA}, T_{AV}

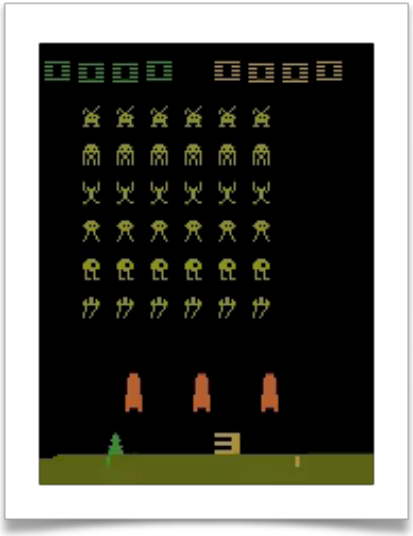
OpenAI

OpenAI Gym



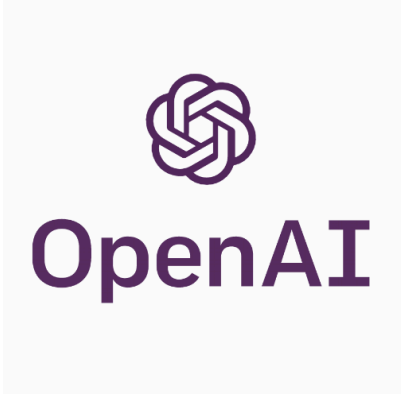






OpenAI Gym Interface

- Initialization (constructor)
- Methods `step`, `reset`
- Optionally `render`



PyALCS

1. Why?
2. `ACS2`, `ACS2Configuration` classes
3. Working modes (explore/exploit)
4. Perception standarization
5. Action mapper standarization

C++ vs Python two-point crossover

```
void CharPosList::twoPointCrossover(CharPosList *
cpl2, int len) {
  //determine two distinct crossing points and
bring them in order
  int p1, p2;
  p1 = (int)(frand() * (double)(len + 1));
  while ((p2 = (int)(frand() * (double)(len + 1)))
== p1); //Just make sure that we get two distinct
positions
  if (p1 > p2) {
    int help = p1;
    p1 = p2;
    p2 = help;
  }

  //set the pointers of the two lists to the first
crossing site
  CharPosItem * cplp, * cplpl = 0;
  CharPosItem * cplp2, * cplpl2 = 0;
  int s1 = 0, s2 = 0;
  for (cplp = first; cplp != 0; cplp = cplp - >
next) {
    if (cplp - > p >= p1)
      break;
    s1++;
    cplpl = cplp;
  }
```

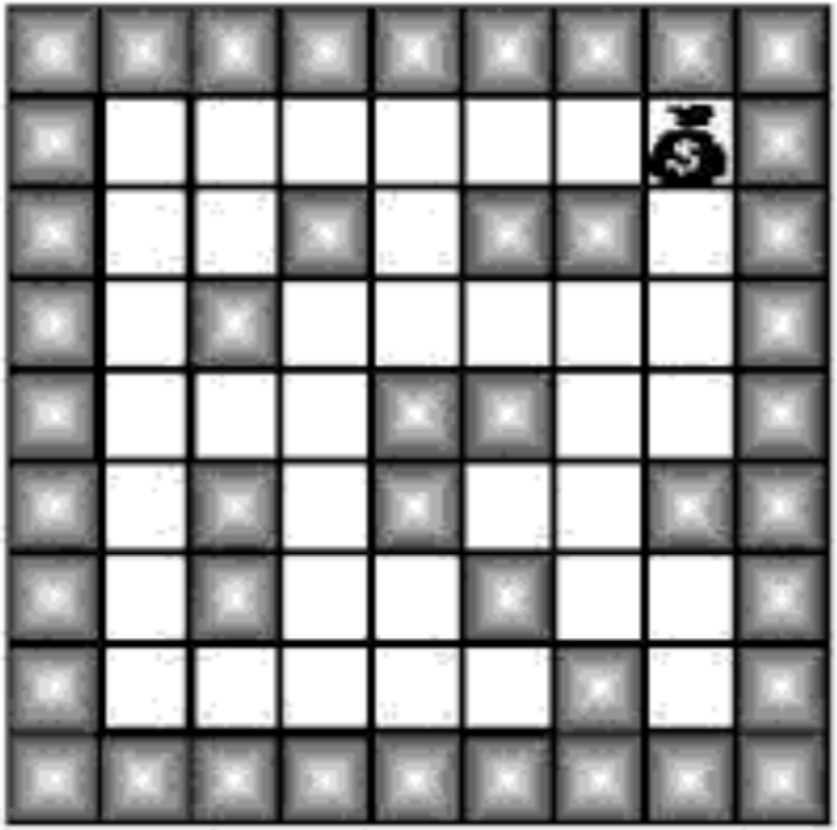
…

C++
383 lines

```
def two_point_crossover(self, other,
samplefunc = sample):
    left, right = samplefunc(range(0,
self.cfg.classifier_length + 1), 2)

    if left > right:
      left, right = right, left

    # Extract chromosomes
    chromosome1 = self[left:right]
    chromosome2 = other[left:right]

    # Flip them
    for idx, el in enumerate(range(left,
right)):
      self[el] = chromosome2[idx]
      other[el] = chromosome1[idx]
```

Python
13 lines

Maze



Maze5

Perception
N, NE, E, SE, S, SW, W, NW

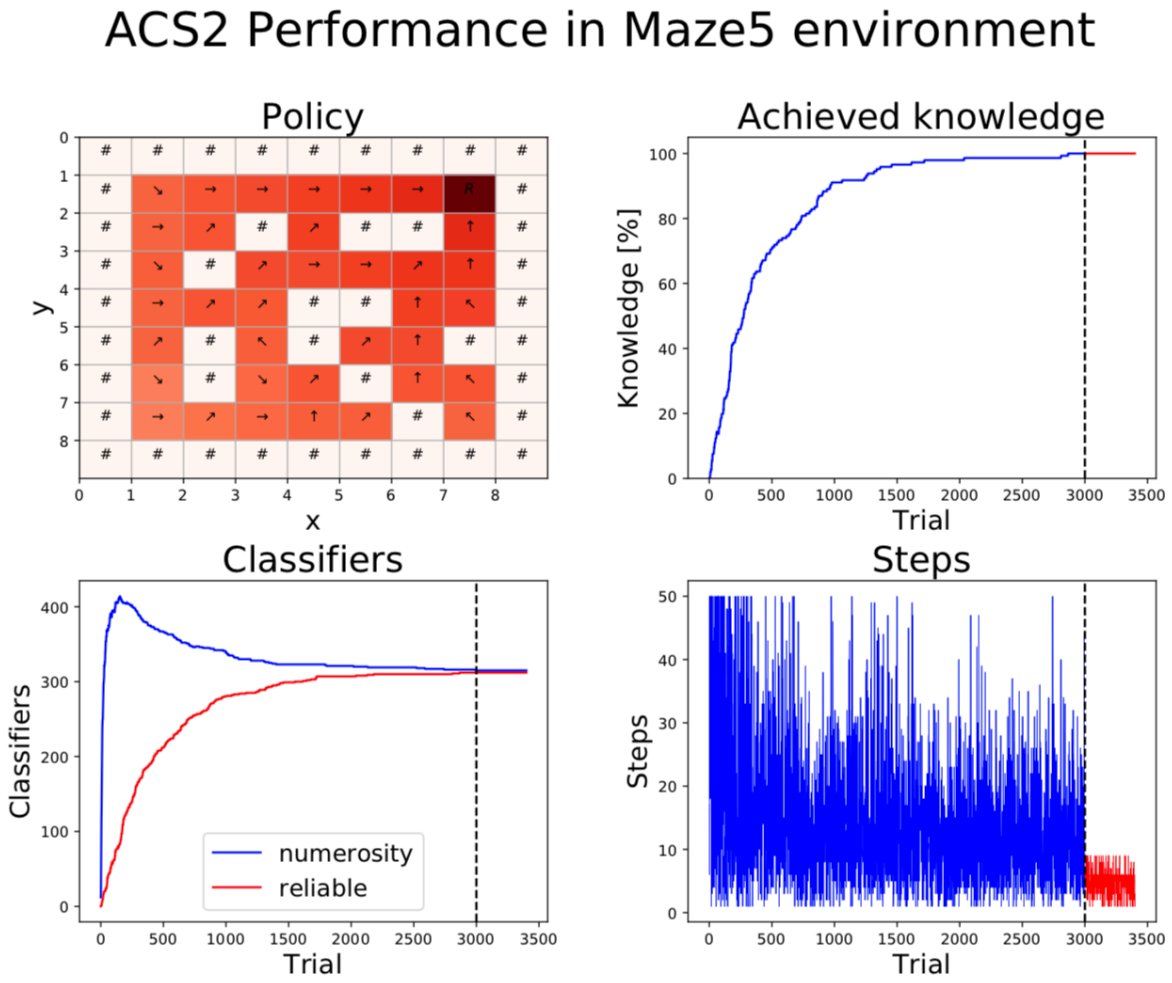
0 - path
1 - wall
9 - reward

Actions
N(0), NE(1), E(2), SE(3),
S(4), SW(5), W(6), NW(7)

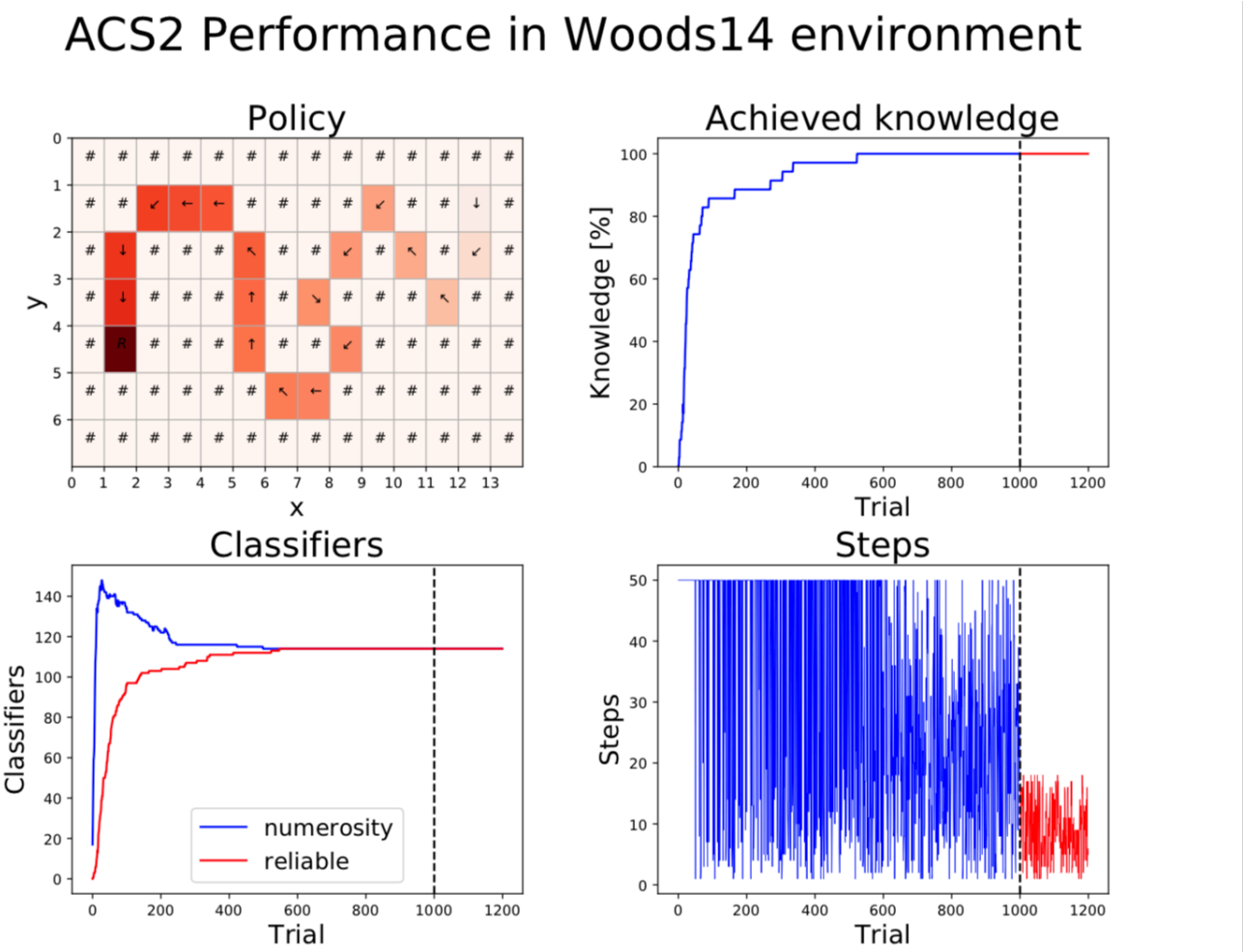
Maze

```
##901###-2-##110### @ 0x112f1a748      q: 0.95      r: 877.97      ir: 877.91
1#0110##-2-##90#1## @ 0x11317bb00      q: 0.99      r: 569.26      ir: 1.48
110110##-2-##90#1## @ 0x112f03b38      q: 0.98      r: 569.26      ir: 1.48
11011001-2-##90#1## @ 0x112ed9860      q: 0.96      r: 569.05      ir: 1.33
9####010-0-1####101 @ 0x113154a90      q: 0.82      r: 624.97      ir: 622.65
00100#10-1-11011#01 @ 0x1131c7f60      q: 0.99      r: 301.77      ir: 0.00
00100010-1-11011#01 @ 0x1131c7f28      q: 0.96      r: 301.94      ir: 0.00
11010101-2-####10## @ 0x112f03550      q: 0.99      r: 206.36      ir: 1.27
1#0101##-2-####10## @ 0x113194f98      q: 0.99      r: 206.20      ir: 2.01
#1##0101-2-####10## @ 0x1131b26a0      q: 0.95      r: 205.86      ir: 1.64
```

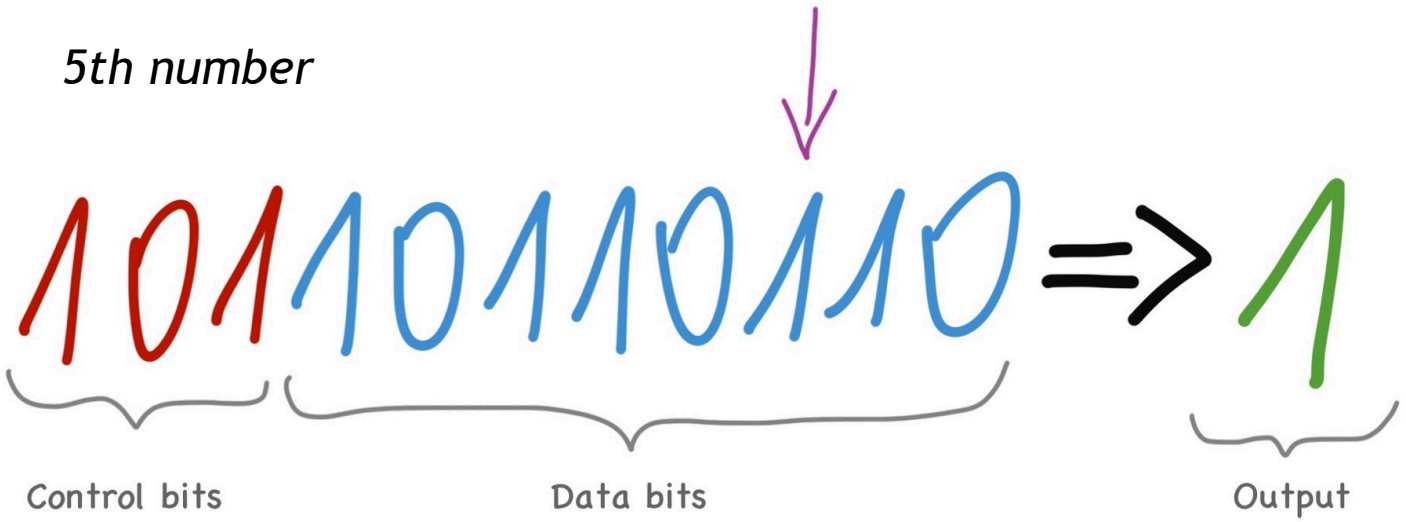
Maze


ACS2 Performance in Maze5 environment
Policy
x
y
Achieved knowledge
Knowledge [%]
Trial
Classifiers
Classifiers
numerosity
reliable
Trial
Steps
Steps
Trial

Maze

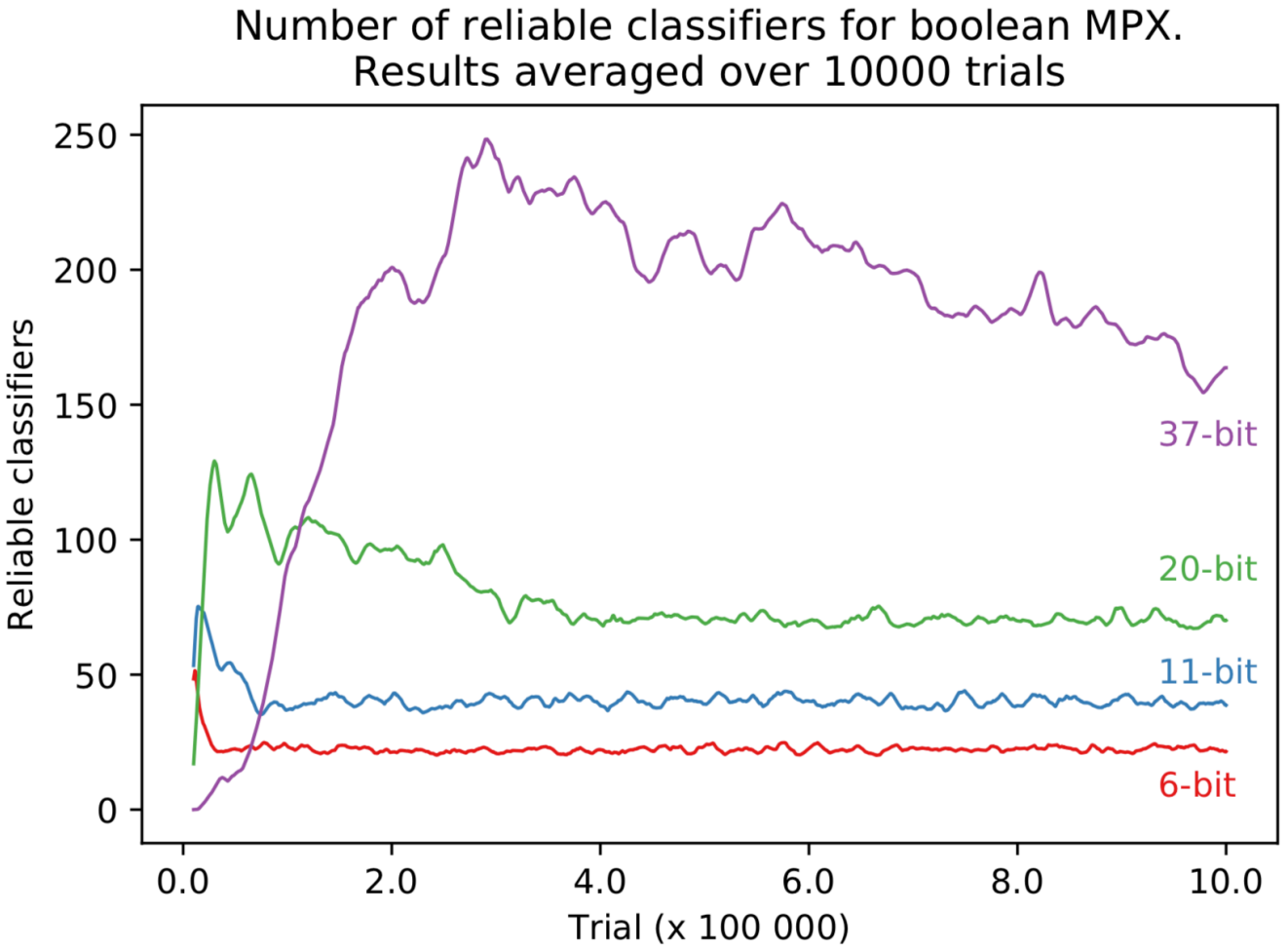

ACS2 Performance in Woods14 environment
Policy
y
x
Achieved knowledge
Knowledge [%]
Trial
Classifiers
Classifiers
numerosity
reliable
Trial
Steps
Steps
Trial

Boolean Multiplexer (MPX)



11bit MPX

Boolean Multiplexer (MPX)

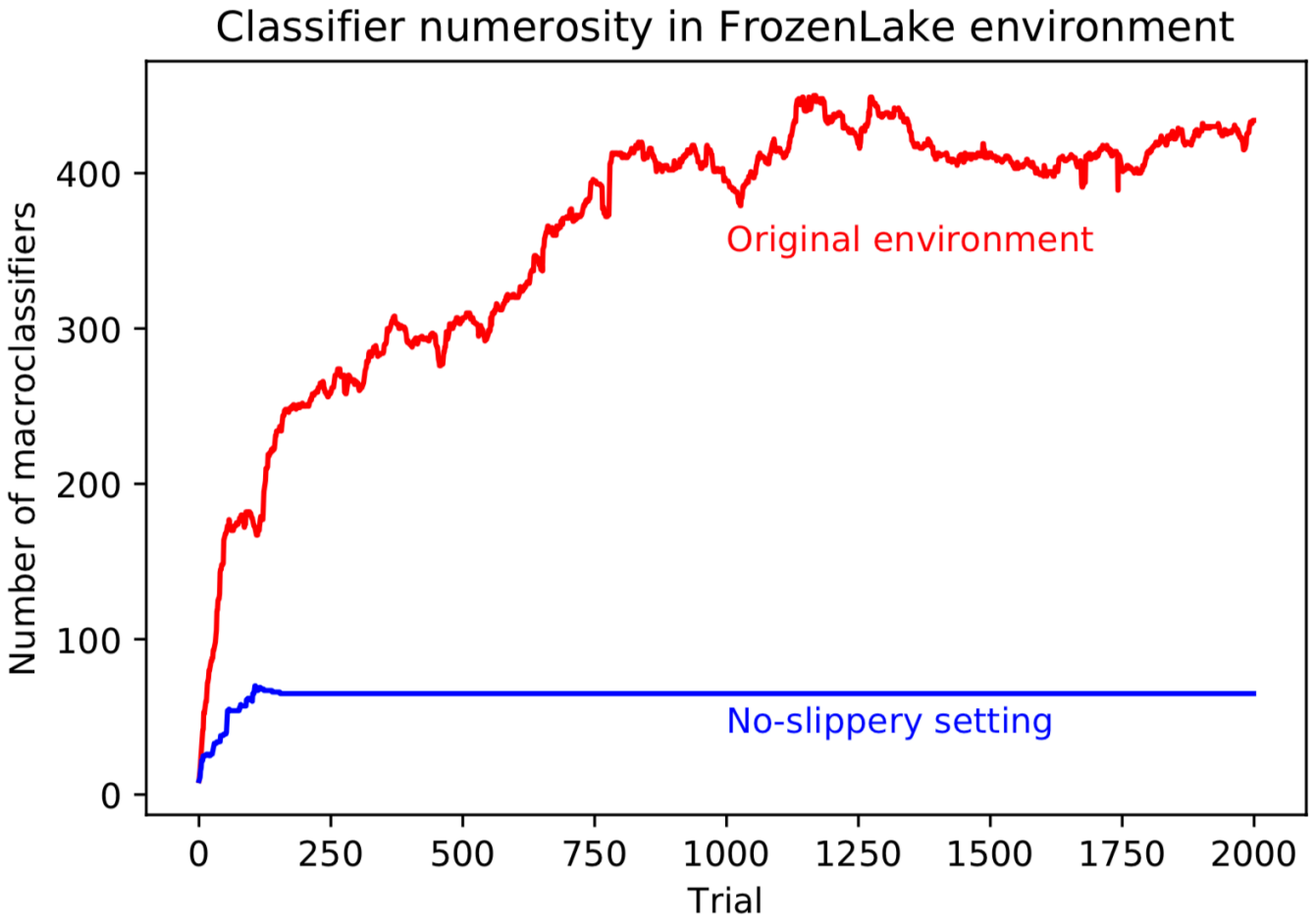


FrozenLake

SFFF
FHFH
FFFH
HFFG

S: starting point
F: frozen surface
H: hole
G: goal

FrozenLake


Classifier numerosity in FrozenLake environment
Number of macroclassifiers
0
100
200
300
400
0
250
500
750
1000
1250
1500
1750
2000
Trial
Original environment
No-slippery setting

Go

Next steps

1. Handle non-deterministic environments
2. Handle continuous input (rACS)
3. *Mental Acting* and *Action Planning*
4. *Concept drift* research
5. Better default settings

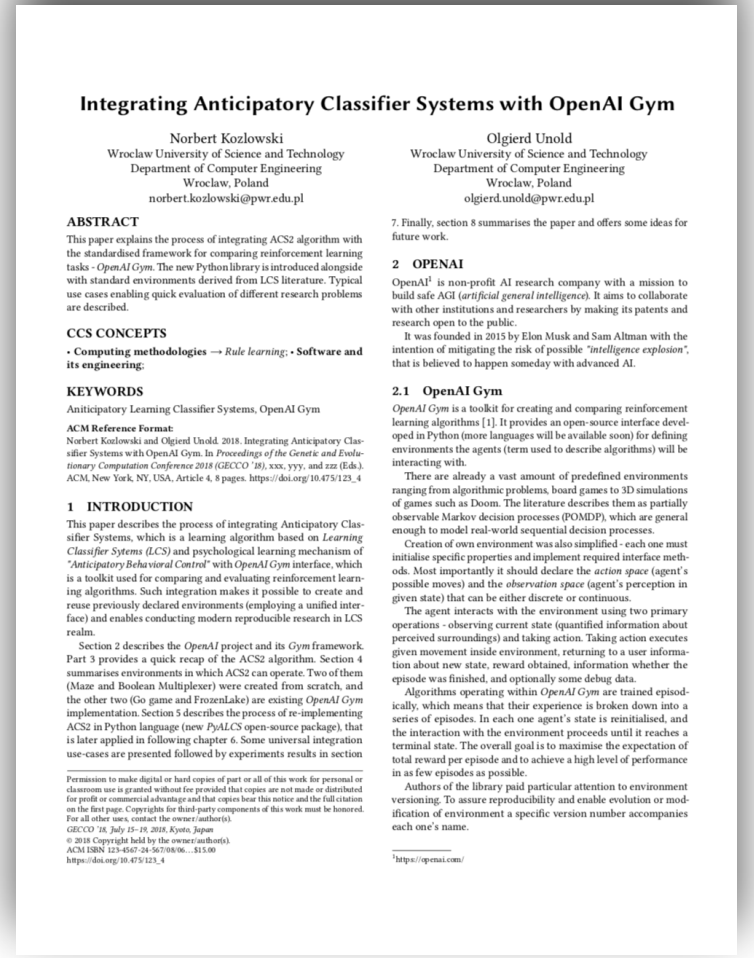
More info

Code

github.com/ParrotPrediction/pyalcs/
github.com/ParrotPrediction/openai-envs

Contact

Wroclaw University of Science and Technology
norbert.kozlowski@pwr.edu.pl



Integrating Anticipatory Classifier Systems with OpenAI Gym

Norbert Kozlowski
Wroclaw University of Science and Technology
Department of Computer Engineering
Wroclaw, Poland
norbert.kozlowski@pwr.edu.pl

Olgierd Unold
Wroclaw University of Science and Technology
Department of Computer Engineering
Wroclaw, Poland
olgierd.unold@pwr.edu.pl

ABSTRACT

This paper explains the process of integrating ACS2 algorithm with the standardised framework for comparing reinforcement learning tasks - *OpenAI Gym*. The new Python library is introduced alongside with standard environments derived from LCS literature. Typical use cases enabling quick evaluation of different research problems are described.

CCS CONCEPTS

• **Computing methodologies** → *Rule learning*; • **Software and its engineering**;

KEYWORDS

Anticipatory Learning Classifier Systems, OpenAI Gym

ACM Reference Format:
Norbert Kozlowski and Olgierd Unold. 2018. Integrating Anticipatory Classifier Systems with OpenAI Gym. In *Proceedings of the Genetic and Evolutionary Computation Conference 2018 (GECCO '18)*, xxx, yyy, and zzz (Eds.). ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

This paper describes the process of integrating Anticipatory Classifier Systems, which is a learning algorithm based on *Learning Classifier Sytems (LCS)* and psychological learning mechanism of *"Anticipatory Behavioral Control"* with *OpenAI Gym* interface, which is a toolkit used for comparing and evaluating reinforcement learning algorithms. Such integration makes it possible to create and reuse previously declared environments (employing a unified interface) and enables conducting modern reproducible research in LCS realm.

Section 2 describes the *OpenAI* project and its *Gym* framework. Part 3 provides a quick recap of the ACS2 algorithm. Section 4 summarises environments in which ACS2 can operate. Two of them (Maze and Boolean Multiplexer) were created from scratch, and the other two (Go game and FrozenLake) are existing *OpenAI Gym* implementation. Section 5 describes the process of re-implementing ACS2 in Python language (new *PyALCS* open-source package), that is later applied in following chapter 6. Some universal integration use-cases are presented followed by experiments results in section 7. Finally, section 8 summarises the paper and offers some ideas for future work.

GECCO '18, July 15–19, 2018, Kyoto, Japan

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

2 OPENAI

OpenAI[1] is non-profit AI research company with a mission to build safe AGI (*artificial general intelligence*). It aims to collaborate with other institutions and researchers by making its patents and research open to the public.

It was founded in 2015 by Elon Musk and Sam Altman with the intention of mitigating the risk of possible *"intelligence explosion"*, that is believed to happen someday with advanced AI.

2.1 OpenAI Gym

OpenAI Gym is a toolkit for creating and comparing reinforcement learning algorithms [1]. It provides an open-source interface developed in Python (more languages will be available soon) for defining environments the agents (term used to describe algorithms) will be interacting with.

There are already a vast amount of predefined environments ranging from algorithmic problems, board games to 3D simulations of games such as Doom. The literature describes them as partially observable Markov decision processes (POMDP), which are general enough to model real-world sequential decision processes.

Creation of own environment was also simplified - each one must initialise specific properties and implement required interface methods. Most importantly it should declare the *action space* (agent's possible moves) and the *observation space* (agent's perception in given state) that can be either discrete or continuous.

The agent interacts with the environment using two primary operations - observing current state (quantified information about perceived surroundings) and taking action. Taking action executes given movement inside environment, returning to a user information about new state, reward obtained, information whether the episode was finished, and optionally some debug data.

Algorithms operating within *OpenAI Gym* are trained episodically, which means that their experience is broken down into a series of episodes. In each one agent's state is reinitialised, and the interaction with the environment proceeds until it reaches a terminal state. The overall goal is to maximise the expectation of total reward per episode and to achieve a high level of performance in as few episodes as possible.

Authors of the library paid particular attention to environment versioning. To assure reproducibility and enable evolution or modification of environment a specific version number accompanies each one's name.

[1] https://openai.com/