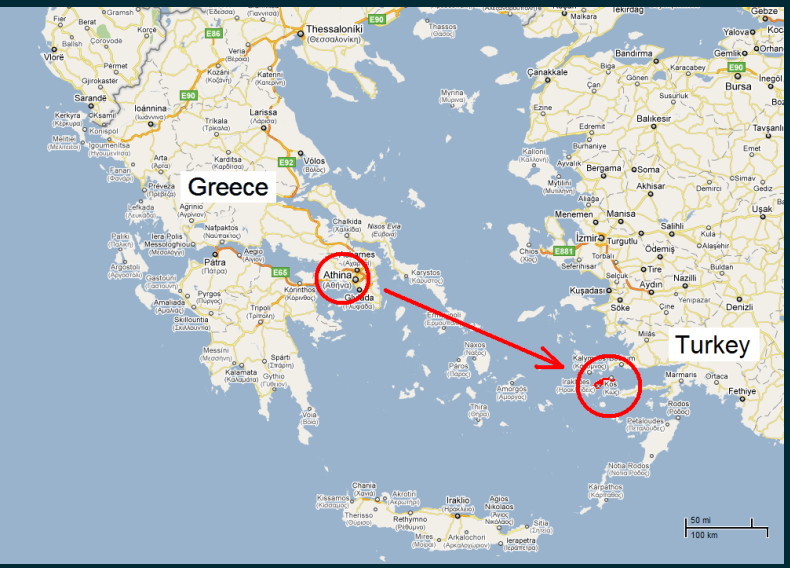
Nikoleta Glynatsi

@NikoletaGlyn

2016-02-11

Who am I?

Who am I?



CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

What do I do?

What do I do?

Research

- Game Theory
- Axelrod
- Machine Learning

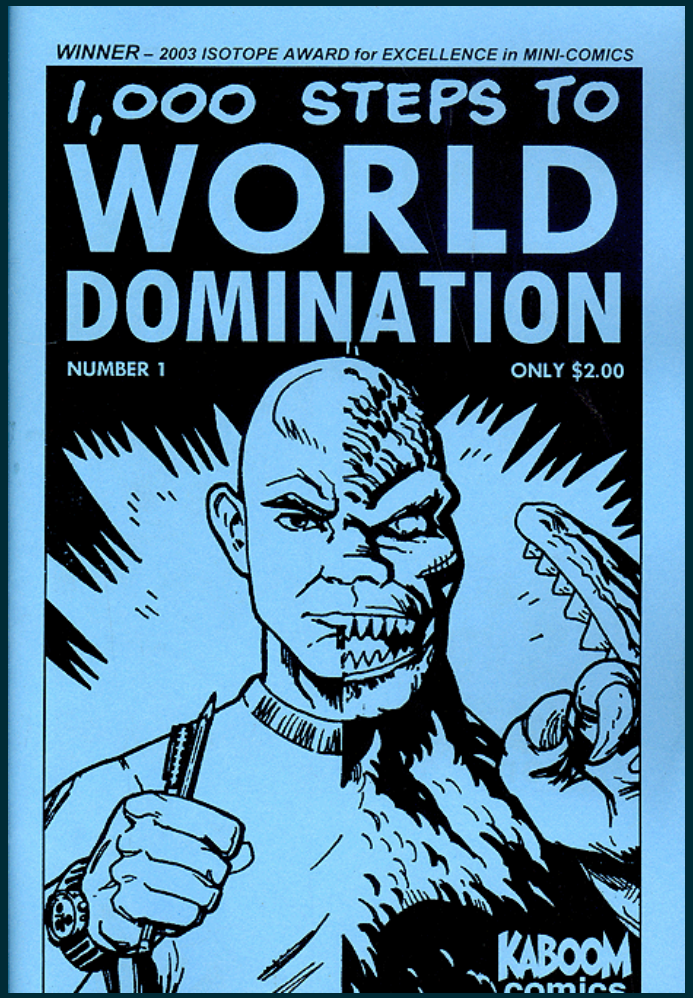
Groups

- PyDiff
- S.W.O.R.D.S.
- PyCon UK
- OR Club?!

Tutoring

- Computing
- Operational Research
- Statistics

My Plans



http://www.sequentialtart.com/reports.php?ID=1971&issue=2016-08-15

Fingerprinting: Visualization and Automatic Analysis of Prisoner's Dilemma Strategies

Daniel Ashlock and Eun-Youn Kim

***Abstract*—Fingerprinting is a technique for generating a representation-independent functional signature for a game playing agent. Fingerprints can be used to compare agents across representations in an automatic fashion. The theory of fingerprints is developed for software agents that play the iterated prisoner's dilemma. Examples of the technique for computing fingerprints are given. This paper summarizes past results and introduces the following new results. Fingerprints of prisoner's dilemma strategies that are represented as finite-state machines must be rational functions. An example of a strategy that does not have a finite-state representation and which does not have a rational fingerprint function is given: the majority strategy. It is shown that the $AllD$- and $AllC$-based fingerprints can be derived from the tit-for-tat fingerprint by a simple substitution. Fingerprints for four new probe strategies are introduced, generalizing pre-**

		$\mathcal{S}$	
		C	D
$\mathcal{P}$	C	3	5
	D	0	1

(1)

		$\mathcal{S}$	
		C	D
$\mathcal{P}$	C	C	T
	D	S	D

(2)

Fig. 1. (1) The payoff matrix for prisoner's dilemma used in this study—scores are earned by strategy $\mathcal{S}$ based on its actions and those of its opponent $\mathcal{P}$. (2) A payoff matrix of the general two player game—C, T, S, and D are scores given for the game as well.

Plans:

- Encourage SSI in my research community

Strategy

```
>>> import axelrod as axl
>>> me = axl.Human(name='me')
>>> players = [axl.TitForTat(), me]
>>> match = axl.Match(players, turns=3)
>>> match.play()

Starting new match
Turn 1 action [C or D] for me: C

Turn 1: me played C, opponent played C
Turn 2 action [C or D] for me: D

Turn 2: me played D, opponent played C
Turn 3 action [C or D] for me: C
[('C', 'C'), ('C', 'D'), ('D', 'C')]
```

Plans:

- Encourage SSI in my research community
- Workshops on Game Theory
- PyCon UK (including Django girls workshop)
- Encourage students
- PyCon Namibia

@NikoletaGlyn
https://github.com/Nikoleta-v3
https://github.com/Axelrod-Python/Axelrod