Higher Order Probabilistic Analysis Of Network Trajectories Of Intelligent Agents In Thespian

Georgios Drakopoulos
Ionian University
c16drak@ionio.gr
0000-0002-0975-1877

Phivos Mylonas
University of West Attica
mylonasf@uniwa.gr
0000-0002-6916-3129

***Abstract*—Intelligent agents (IAs) are autonomous pieces of software designed to be deployed to and operate on infrastructure, whether physical or digital, and perform various tasks on them ranging from integrity check and structure discovery to functionality monitoring and information collection. Their task performing capability has been considerably increased with the recent advent of machine learning. An important parameter of IAs operating on networks such as the Web is their trajectory, which in many engineering scenarios depends heavily on random outcomes taking place at each vertex visited by the IA. In order to study the probabilistic properties of the trajectory length, said outcomes instead of being modeled or simulated are computed as the result as a game taking place between the vertex and the IA, developed in the Thespian framework for Python, and the vertex. The latter selects a random but fixed strategy, whereas the IA can adapt to learn this strategy either by observing the entropy of the choices of its opponent. If IA loses, then it backtracks, otherwise it chooses its next destination with a preferential attachment scheme. The mean, variance, skewness, and kurtosis of the trajectories of IAs operating on three scale-free graphs generated by NetworkX. Emphasis was placed on proper Pythonic code as many major Python modules such as threads, for generating game instances, Counter instances from collections to keep track of player choices, and functools for map/reduce functionality. Results indicate that IAs learning the opponent strategy have longer and richer network trajectories in terms of vertices, indicating the importance of learning.**

***Index Terms*—intelligent agents, random walks, network trajectory, Thespian, NetworkX, strategy estimation, entropy, higher order data, collections, threading, functools, argparse, globals**

I. Introduction

Intelligent agents (IAs) are autonomous pieces software designed to perform a broad spectrum of tasks including but not limited to structural integrity, link prediction, and even negotiation with other IAs. This functionality increases the digital awareness of their operators by collecting information about the environment they operate in. Interestingly, as IAs keep functioning therein, they generate information of their own, in addition to that they collect regularly, which can be mined. A major such piece of information for IAs operating in networks is the length of their trajectory expressed in the number of edges crossed. By probabilistically mining it, knowledge about the underlying network and about how easily the IA can cross it can be obtained.

The IAs presented and examined here traverse the network in order to visit each vertex. However, an opponent at each vertex the IA is currently visiting challenges the latter to a rock/paper/scissors (RPS) game whose outcome determines the next IA move. Specifically, if the IA emerges as a winner, then it chooses the next vertex to visit based on a preferential attachment policy, otherwise the IA backtracks to the vertex it came from. The RPS game was selected as it is straightforward to collect its results. Moreover, the IA moves based on the outcomes of actually played games instead of computing or simulating the probability of IA victory in each game.

The twofold primary research objective of this conference paper is to collect and perform a higher order probabilistic analysis of the lengths of the IA trajectories including the computation of skewness and kurtosis as well as to analyze the difference in performance of IAs relying on strategy estimation in each game from IAs which do not. As a secondary objective, heavy emphasis was placed on developing Pythonic code in order to achieve the desired functionality as well as to translate probability theoretic concepts to programming ones as determined by PEP8[1] complete with docstrings as stated in PEP 257[2]. Additionally, the IA was implemented in the Thespian[3] framework and the three synthetic benchmark datasets were obtained from the NetworkX[4] graph library. The methodology presented here can be applied to the study of network propagation phenomena such as meme diffusion is social media, SIR models, and the spread of biological virus.

The remainder of this work is structured as follows. In section II the recent scientific literature regarding IAs, Python modules, and parameter estimation is briefly reviewed. In section III the IA components and mechanics and the probablistic strategy estimation are explained, whereas in section IV the RPS source code is described. Results is explained in section V. Possible future research directions are given in section VI. Capital calligraphic letters symbolize random variables (rvs) and normal lowercase scalars. Technical acronyms are explained the first time they are encountered in text. Finally, the notation of this work is summarized in table I.

[1]https://peps.python.org/pep-0008
[2]https://peps.python.org/pep-0257
[3]https://github.com/thespianpy/Thespian
[4]https://networkx.org

TABLE I
NOTATION SYNOPSIS.

Symbol	Meaning	First in
$\triangleq$	Definition or equality by definition	Eq. (1)
$\{s_1,\ldots,s_n\}$	Set with elements $s_1,\ldots,s_n$	Eq. (8)
$\langle s_k \rangle$	Sequence with elements s_k	Eq. (6)
$(t_1,\ldots,t_n)$	Tuple with elements $t_1,\ldots,t_n$	Eq. (12)
$\lvert\cdot\rvert$	Set or sequence cardinality functional	Eq. (6)
$\hat{y}$	Estimator of quantity y	Eq. (8)
$\mathrm{prob}\left\{\Omega\right\}$	Probability of event Ω occurring	Eq. (2)
$\mathrm{E}\left[\mathcal{X}\right]$	Expected value of random variable $\mathcal{X}$	Eq. (3)
$\mathrm{Var}\left[\mathcal{X}\right]$	Variance of random variable $\mathcal{X}$	Eq. (3)
$\deg\left(v\right)$	Degree of vertex v	Eq. (2)
$\Gamma\left(v\right)$	Neighborhood of vertex v	Eq. (2)

II. Previous Work

IAs are designed to operate autonomously in infrastructure [1], whether physical [2] or digital [3], to perform through machine learning (ML) a wide range of critical tasks such as link prediction [4], secure communications in unmanned aerial vehicles (UAVs) [5], and cooperation in autonomous vehicles [6]. In network environments such as social media [7], which can be represented by ordinary [8] or fuzzy graphs [9], or smart cities [10] IAs can be particularly effective as the modular and commuity-based graph structure [11] allows them to roam in order to collect information [12] about the network state such as expansion potential [13] and increase digital awareness [14] through state estimation [15] and swarm intelligence [16]. In such environments multi-objective optimization [17] and trajectory prediction in multi-agent engineering scenarios [18] are key problems. In settings like recommender systems [19] [20] the user acceptance of IAs is critical [21]. IAs can also be deployed in blockchains to negotiate smart contracts [22], collect state information [23], and verify consensus [24]. A survey for IA applications is [25].

Parameter estimation is an established field in signal processing [26] where a scalar parameter or a vector of parameters need to be inferred from a number of possibly corrupted measurements [27]. In classical estimation said parameters are considered fixed and they are estimated through probasilictic techniques like least minimum variance estimation [28] or model-based techniques like deterministic least mean squares [29] and maximum likelihood [30]. On the contrary, Bayesian estimation is also model-based but it treates the parameters as realizations of a random process [31]. Hence, a prior estimation is additionally required [32].

Python modules cover a broad spectrum of functionality such as behavioral analysis [33] and JSON parsing [34]. Functional programming [35] is an established programming paradigm where priority is placed on functions, considered first-class citizends, and their composition [36]. Besides purely functional languages such as Haskell, functional elements, mainly in the form of lambda expressions, can be found in Scala [37] and Java [38]. Functional data structures are described in [39] and functional aspects include map, filter, and reduce [40] [41].

III. Intelligent Agent Design

A. Random Trajectories

In order to successfully carry out their task, the IAs typically have an architecture consisting of the following components:

- Sensors in order to perceive its environment.
- Actuators to act on its environment.
- A decision making scheme fed by sensor information.
- A state vector where the IA status is stored.

The trajectory an IA makes while traversing a network can be mined through statistical analysis or ML for patterns in order to extract working knowledge about the underlying network structure in addition to the information IA collects. In the context of this work the IA objective is to visit every vertex, which is a typical assignment. However, in order for it to be allowed to move, the IA must win an RPS game. The bot opponent, for brevity termed *bot* or *opponent*, of the IA selects independently and randomly one of the strategies of table II. The IA can be configured to either estimate the bot strategy or to select a strategy at random from the same table.

A number of N runs, shown in table VI, was executed. In such run the IA was first configured to estimate the bot strategy and then to select a strategy at random and the trajectory length, denoted respectively as L_e and L_r, was recorded and the ratio L of equation (1) was formed.

$$L \triangleq \frac{L_r}{L_e} \tag{1}$$

When IA in a vertex u beats its opponent, then it can move to one of the neighbors of u. The probability of IA moving from vertex u to a neighboring v is computed by a preferential attachment policy shown in equation (2).

$$\mathrm{prob}\left\{u \rightarrow v\right\} \;\propto\; \frac{\deg\left(v\right)}{\sum_{s\in\Gamma(v)}\deg\left(s\right)} \tag{2}$$

The values of the trajectory length ratio of equation (1) can be considered as samples of an rv $\mathcal{L}$. Once its probabilistic mean and variance, respectively denoted as $\mathrm{E}\left[\mathcal{L}\right]$ and $\mathrm{Var}\left[\mathcal{L}\right]$, can be computed from the respective sample counterparts under mild ergodicity assumptions, then higer order probabilistic metrics can be computed. A common third order metric is the skewness κ_3 defined as in equation (3). On the condition that κ_3 comes from a unimodal distribution, when κ_3 is positive, then the tail of the distribution is to its right, while when it is positive, then its tail is to its left.

$$\kappa_3 \triangleq \frac{\mathrm{E}\left[\left(\mathcal{L}-\mathrm{E}\left[\mathcal{L}\right]\right)^3\right]}{\mathrm{Var}\left[\mathcal{L}\right]^{3/2}} \tag{3}$$

Another higher order probabilistic metric is kurtosis, which is a fourth order metric evaluating the concentration around the mean and defined as in equation (4).

$$\kappa_4 \triangleq \frac{\mathrm{E}\left[\left(\mathcal{L}-\mathrm{E}\left[\mathcal{L}\right]\right)^4\right]}{\mathrm{Var}\left[\mathcal{L}\right]^{2}} \tag{4}$$

B. Strategy Estimation

In this game the bot player is configured such that at each round of each duel it makes a move which is independent of the past ones and according to a preselected stationary policy. In other words, the distribution of the next move made by the bot player is given in general by equation (5). Observe that said distribution generates categorical data. Moreover, decision independence precludes Markov chain type analysis.

$$s_k \triangleq \begin{cases} \textbf{rock}, & \text{prob}\left\{s_k = \textbf{rock}\right\} \;=\; p_r \\ \textbf{paper}, & \text{prob}\left\{s_k = \textbf{paper}\right\} \;=\; p_p \\ \textbf{scissors}, & \text{prob}\left\{s_k = \textbf{scissors}\right\} \;=\; p_s \end{cases} \tag{5}$$

A move sequence $\langle s_k \rangle$ of length n follows the trinomial distribution shown in equation (6):

$$\text{prob}\left\{s\right\} \triangleq \binom{n}{n_r}\binom{n-n_r}{n_p} p_r^{n_r} p_p^{n_p} p_s^{n_s}, \quad |s| \;=\; n \tag{6}$$

The distribution of (6) holds under the constrtaint pair of (7). This ensures that the distribution adds up to one.

$$\begin{aligned} n_r + n_p + n_s \;&=\; n \\ p_r + p_p + p_s \;&=\; 1 \end{aligned} \tag{7}$$

An estimation $\hat{p}_x$ of each of the three probabilities p_x of equation (6) can be constructed as a function $g_x(\cdot)$ of the frequencies of appearance, or equivalently the number of appearances, of rock, paper, and scissors denoted respectively as n_r, n_p, and n_s as shown in equation (8).

$$\hat{p}_x \triangleq g_x(n_r, n_p, n_s), \qquad x \in \{r, p, s\} \tag{8}$$

One estimator originating directly from equation (6) but applied to a more generic context is equation (9), which coincides with the frequentist approach coined by Laplace.

$$\hat{p}_x \triangleq \frac{n_x}{n_r + n_p + n_s}, \qquad x \in \{r, p, s\} \tag{9}$$

Based on the probability estimation the entropy of the opponent can be computed. In table II the entropy for the bot player strategies is shown. Equation (10) was used. Therein the base of logarithm b determines the base in which information is measured. Given that each logarithm is of the same order of magnitude, the selection of b does not critically influence the value of H. In this case b equals two and information is represented in bits.

$$H \triangleq \sum_k p_k \log_b \frac{1}{p_k} \;=\; -\sum_k p_k \log_b p_k \tag{10}$$

Quite often reduce is combined with **lambda**. For instance, in the following code segment the value of entropy is updated if and only if the value of argument p is not zero as otherwise a numerical error would occur.

```
import functools as f
import numpy as np

vals = d.values()
H = f.reduce(lambda H,p: \
     H-p*np.log2(p) if p else H, vals , 0)
```

TABLE II
STRATEGIES AND THEIR ENTROPY.

Strategy	Distribution	Entropy (bits)
uniform	all moves 1/3	$\log_2 3$ (max)
last	play opponent's last move	between 0 and $\log_2 3$
rockx3	60% rock, 20% paper, scissors	1.3709
rockx2	50% rock, 25% paper, scissors	1.5
rockx0	50% paper, 50% scissors	1
rockx1	100% rock	0 (min)

Lambda expressions, functional elements, or even functional programming as a paradigm is supported in prominent programming languages such as Java[5] and C++[6]. Moreover, in newer languages such as Julia[7] the lambda expressions, therein called *anonymous functions*, are an integral part of the original language specifications[8]. Finally, in purely functional languages such as Haskell[9] and Scala[10], lambada functions not only are available, but they are *first class citizens*, namely the primary means of computation.

IV. RPS Code Overview

A. Overview

The source code comprises of the files shown in table III. Therein is also shown a brief description of the respective file and the subsection which describes their functionality. The game relies heavily on the players exchanging messages in order to monitor RPS progress and keep track of moves.

TABLE III
SOURCE CODE FILES.

Name	Description	Shown in
bg_client_app.py	Client application	Sec. IV-B
bg_client.py	Client configuration	Sec. IV-C
bg_client_player.py	Basic player class	Sec. IV-D
bg_client_custom_players.py	Opponent player class	Sec. IV-E
bg_client_my_player.py	IA player classes	Sec. IV-F
bg_server_app.py	Server application	Sec. IV-G
bg_server.py	Main server file	Sec. IV-H
bg_server_worker.py	RPS game logic	Sec. IV-I
bg_messages.py	Server/client messages	Sec. IV-J

The event driven nature of the game means a non-linear source code execution. Moreover, the IA and the bot need not be on the same computer, in which case the network overhead should be taken into consideration.

Notice that in order to keep **from** ... **import** commands simple, all source files have been moved to the same directory. In this way some initial problems stemming from directory dependencies were automatically resolved.

Before a print operation takes place, a lock is obtained to ensure that only one thread displays a message. Moreover, all messages are sent to the standard stream sys.stderr,

[5]https://www.infoworld.com/article/2078836/love-and-hate-for-java-8.htm
[6]https://en.cppreference.com/w/cpp/language/lambda
[7]www.julialang.org
[8]https://docs.julialang.org/en/v1/manual/functions
[9]https://wiki.haskell.org/Anonymous_function
[10]https://docs.scala-lang.org/scala3/book/fun-anonymnous-functions.html

which is the proper stream for diagnostic and error messages. Moreover, for each such operation the flush option is enabled, meaning that each diagnostic message is guaranteed to appear on the screen the time the programmer intends to. Otherwise, given the buffered nature of I/O operations, there is a non-negligible probability that a program may crash before a message appears on screen, giving thus the wrong idea about the code crashing point. When flushing is enabed, however, all outbound messages have been properly displayed before the exception stack is unwound. Therefore, the developer has the correct perception about where code execution has reached and can look for the right causes. In order to ensure this, the partial method creates a restricted version of the generic print which sends its input to the desired stream with flushing. Recall that in order for this to work, the target function must support a keyword-pair structure of arguments.

```
import functools as f

ffprint = f.partial(print,\
    file=sys.stderr, flush=True)
```

B. File *bg_client_app.by*

This is the main client file, namely the file IA has to start in order for its player class to be initiated and for the connection to be set up. This file creates a client configuration dictionary and passes it to the main function of the *bg_client.py*. Note that the fields silent and verbose are set to **None** since they will be filled by the values of the server configuration dictionary.

C. File *bg_client.py*

This is the module containing the main client function, which is responsible for creating the various client players according to the appropriate command line arguments.

The globals() function is used early in this file in order to return a dictionary with every global variable[11] and then instantiate a player class, either an IA or a bot one depending on who is calling the rps_client_main() function. The latter is determined by the class name stored in the field class of the client configuration dictionary. As the bot player initialization function __init__() accepts different arguments than its counterparts of the IA classes, care is taken before invoking it. The variable klass() is an alias of the right initiator.

One the client is connetect to the server network socket, an initial message from the client is sent. Notice that in order to do so, a lock is obtained so that other threads may not print at the same time a message as well. The pair of functions acquire() and release() form a safe context manager, similar to the well-known **with** ... **as** one.

Note that a general technique after receiving over the network and deserializing a message using the pickle library is to check whether it is of the expected type. This is accomplished with as **assert** condition.

Since the bot player class does not have methods for keeping track of who the opponent is and for monitoring the opponent moves, a check is made to determine whether the player class currently active has an attribute of the right name through the hasattr function[12] followed by a second check through the callable method to determine whether this attribute appears to be callable. The latter means that even if callable returns **True**, a function call with that attribute may still fail –a case not happening here by design, whereas if it returns **False**, then a function call with this attribute is bound to fail[13]. Because of the short-circuit evaluation[14] of logical **and** in Python, if the first check fails, then the second one is not even executed.

D. File *bg_client_player.py*

The rudimentary player class is stored in this short source file. This class is used to build the standard bot and IA classes through simple inheritance, even though Python is one of the few modern programming languages supporting multiple inheritance[15] like C++.

Another highlight is that since this is an elementary class without a fully fledged functionality, the method game_result() is just a placeholder. To emphasize this, a NotImplementedError is exception raised if this method is called instead of just putting a **pass** instruction in its definition.

E. File *bg_client_custom_players.py*

In this source file the standard bot player class can be found. If the __init__() is called with an argument of **None**, then a strategy out of the possible ones is picked at random. Otherwise, the prespecified strategy is selected and played out throughout the entire game.

Bias can be inserted in the method next_move() of the bot class by removing from or inserting to the initial list the *rock* option one or two times. Additionally, the bot can be instructed to play only *rock* as a move. Of course there is nothing special with this move, as taking the same steps with any of the other two moves would create similar distributions. Notice that in the selection of the next move the copy() method is needed, since the append() and remove() functions operate *in-place* and, hence, their return value as a result is **None**.

F. File *bg_client_my_player.py*

In this module the IA player class is defined. The latter inherits the majority of its functionality from the former, with the single exception of the function determining its next move. In this way, code is reused and the differences between the two classes become clearly visible. In table IV the methods of the IA class are explained.

At the heart of the IA player class lies a composite data structure where:

- A dictionary with keys the opponent ids as returned by the game server.
- Each corresponding value is a list with two elements:

[11]https://docs.python.org/3/faq/programming.html

[12]https://docs.python.org/3/library/functions.html#hasattr

[13]https://docs.python.org/3/library/typing.html

[14]https://www.geeksforgeeks.org/short-circuiting-techniques-python

[15]https://realpython.com/inheritance-composition-python

TABLE IV
IA PLAYER METHODS.

Method	Description
__init__	Initializer
__str__	String conversion
__repr__	Object representation
update_moves	Keep track of opponent moves
update_opponent	Prepare for a new opponent
update_winner	Keep track of duel results
get_encounters	Return duel result history
rec_freq	Recommendation based on moves
rec_strategy	Recommendation based on entropy
next_move	Get next move

- A dictionary with each possible move as keys and an integer counter as value.
- A bool flag indicating whether the player is the IA.

Recall that int and str types are hashable and they can be dictionary keys. When this structure is initialized or when a new opponent in encountered, then a new entry is created using the dictionary update() method, which works on both existing and new structures. Should this measure had not been taken, a KeyError exception would have been raised.

G. File bg_server_app.py

This module must be started in order to set up the server controlling the game and spawing any bot players necessary. Since this file controls a considerable amount of functions, it also accepts a number of command line arguments through the argparse module in order to configure this functionality.

H. File bg_server.py

This is the main server file where the connections to players, whether IAs or vertex opponents, are established and the game is executed. Moreover, the main server function is an infinite **while** loop which can be only terminated when an exception occurs. In the exception hadling code care is taken to close the socket before exiting the program.

I. File bg_server_worker.py

This file contains the code for the actual execution of the game once the establishment of network sockets, the initial communication, and the game setup are done. Moreover, the game logic is implemented here.

J. File bg_messages.py

All possible messages sent from the server to client and vice versa are defined in file bg_messages.py. Messages with originally three arguments or more were converted to accept a dictionary with the individuals arguments as fields. Moreover, in order to secure message integrity, every initiator has an **assert** clause. Table V contains the messages as well as basic information about them.

V. RESULTS

The configuration parameters of the RPS game as well as the synopsis of the three synthetic benchmark datasets are shown in VI. Implementation was done in Python 3.12, the latest version currently available. The Thespian framework implements the Actor model in Python and relies heavily on message passing between the entities. The NetworkX library offers a wide array of functionality pertaining to graphs, including the generation of scale-free graphs with a predetermined number of vertices and density. NetworkX defines the density ρ for a graph with n_v vertices and n_e edges as in (11)[16].

$$\rho \triangleq \frac{2n_e}{n_v(n_v-1)} \tag{11}$$

Therefore ρ is defined as the ratio of the number of edges of the graph to the number of edges of a complete graph with the same number of vertices. For directed graphs ρ is defined as half of that in (11). The synthetic benchmark graphs used here are undirected. Also please note that this definition of ρ differs from the one frequently used in the scientific literature.

From a graph structure perspective, the graph density ρ essentially determines on the average the number of choices the IA has when it wins an RPS game. Along the same line of reasoning, the graph diameter roughly determines how difficult is to reach the outermost vertices, which in turn may lead to longer trajectories if it is sufficiently high.

In table VII the results of the code execution with D duels of R rounds each per vertex are shown. The total number of games equals the number of vertices visited by the IA. Each table entry has the structure of equation (12). Since the game results are totally independent from the underlying graph topology, the IA and bot victory probabilities have been averaged over all three benchmarks. Also, the tie probability is not shown, as it is redundant information. If a tie occurs, then a new game starts on the same vertex.

$$(\text{prob}\left\{\text{IA wins}\right\}, \text{prob}\left\{\text{bot wins}\right\}) \tag{12}$$

Table VII should be interpreted as follows: The last row has the probabilities of victory, defeat, and tie for IA when it is configured to estimate bot strategy, whereas the remaining of the table has the same statistics when both the IA and the bot each randomly select a strategy. Each column corresponds to a bot strategy, while each row to an IA strategy. From the entries of table VII the following conclusions can be drawn:

- Each strategy against itself practically lead to tie.
- There was a symmetry between strategies played by bots, namely that the streategy pair (S_1, S_2) had roughtly the same results as that of (S_2, S_1).
- The most effective strategy is *uniform*, which is the most unpredictable, followed by *rockx2*. This is expected given the entropies of these strategies.
- The worst one is *rockx1*, which is totally predictable. Moreover, it can be easily countered by an opponent constantly playing paper.

[16]https://networkx.org/documentation/stable/reference/generated/networkx.classes.function.density.html

TABLE V
MESSAGE TABLE.

Class	Sender	Meaning	Parameters
ServerMsgHello	Server	Configuration message	Silent and verbose flags, rounds, duels
ServerMsgDuelStart	Server	Duel notification	Player ids, player and IA flags
ServerMsgRoundStart	Server	Round notification	Round id
ServerMsgRoundWinner	Server	Round winner	Player ids, player moves, winner id
ServerMsgDuelWinner	Server	Duel winner	Winner id
ClientMsgHello	Client	Connection message	Name, version, IA flag
ClientMsgOk	Client	General acknowledgement	Message
CientMsgRoundMove	Client	Move in round	Move, player id

TABLE VI
CONFIGURATION PARAMETERS.

Parameter	Value
Number of runs N	100000
Number of RPS duels D	101
Number of RPS rounds R	201
Number of vertices n_v	10000 / 10000 / 10000
Number of edges n_e	90000 / 70000 / 60000
Density ρ	0.00009 / 0.00007 / 0.00006
Diameter δ	11 / 12 / 14

- The *last* strategy is a peculiar case as the player employing it is totally dependent on the other. Still, it leads to a lot of ties, especially when the other plays *rockx1*.

In table VIII the probabilistic metrics of the trajectory length ratio $\mathcal{L}$ of (1) are shown. From the entries of this table the following conclusions can be drawn:

- The mean of $\mathcal{L}$ shows that on average the IA has to cross considerably more times the number of vertices when the bot strategy estimation is disabled.
- The relatively large value of the variance means that $\mathcal{L}$ is not very concentrated around its mean.
- The positive skewness indicates that the majority of the values, namely the tail of the distribution, of $\mathcal{L}$ is to the right of its mean.
- The kurtosis also seems to support the case that $\mathcal{L}$ flustuates. This could be possibly attributed to the random nature of the RPS game.
- The sparser and the more difficult to traverse the graph is, the higher the mean value of $\mathcal{L}$ and its fluctuations are, indicating more variability.

Entries from table VIII tend to support the hypothesis that enabling the IA to estimate the strategy of the bot in each vertex through the entropy of the distribution of the decisions of the latter reduces considerably the number of hops the IA must do in order to visit every vertex of the underlying graph.

VI. CONCLUSIONS AND FUTURE WORK

The focus of this conference paper is to perform higher order probabilistic analysis of the random trajectories of an intelligent agent (IA) operating on network infrastructures. At each vertex a bot opponent challenges the IA to a game of rock/paper/scissors (RPS). Depending on the outcome, the IA selects a neighboring vertex based on a preferential attachment rule if it is victorious or the IA backtracks to the previous vertex. Two scenarios were tested, depending on whether the IA could estimate the entropy of the decision distribution of its opponent or not. Results indicate that when that was true, the IA trajectory length was considerably shorter. The IA was implemented in the Python Thespian framework and the RPS game in Python utilizing modules like argparse, collections, threads, and functools. The RPS game was chosen such that the probability of the IA successfully advancing from one vertex to one of its neighboring ones to be determined by an actual game instead of being computed or simulated. This can be beneficial in the analysis of network propagation phenomena such as meme diffusion or virus propagation.

This work can be expanded in a number of ways. First and foremost, strategy estimators such as *chi square test* and *run test* can be implemented, both of which rely on the analysis of long sequences. Additionally, the proposed technique can be applied to larger benchmark networks to test its scalability.

ACKNOWLEDGMENT

This research was funded by the European Union and Greece (Partnership Agreement for the Development Framework 2014-2020) under the Regional Operational Programme Ionian Islands 2014-2020, project title: "Indirect costs for project "TRaditional corfU Music PresErvation through digiTal innovation", project number: 5030952.

REFERENCES

[1] S. Moussawi, M. Koufaris, and R. Benbunan-Fich, "The role of user perceptions of intelligence, anthropomorphism, and self-extension on continuance of use of personal intelligent agents," *European Journal of Information Systems*, vol. 32, no. 3, pp. 601–622, 2023.

[2] U.-H. Kim, J.-M. Park, T.-J. Song, and J.-H. Kim, "3-D scene graph: A sparse and semantic representation of physical environments for intelligent agents," *IEEE Transactions on cybernetics*, vol. 50, no. 12, pp. 4921–4933, 2019.

[3] J. Maurio *et al.*, "Agile services and analysis framework for autonomous and autonomic critical infrastructure," *Innovations in Systems and Software Engineering*, vol. 19, no. 2, pp. 145–156, 2023.

[4] M. Nikolaou, G. Drakopoulos, P. Mylonas, and S. Sioutas, "Intelligent agents with graph mining for link prediction over Neo4j," in *WEBIST*. SCITEPRESS, 2023.

[5] V. F. Sangeetha Francelin, J. Daniel, and S. Velliangiri, "Intelligent agent and optimization-based deep residual network to secure communication in UAV network," *International Journal of Intelligent Systems*, vol. 37, no. 9, pp. 5508–5529, 2022.

[6] H. Yang, Z. Wei, Z. Feng, X. Chen, Y. Li, and P. Zhang, "Intelligent computation offloading for MEC-based cooperative vehicle infrastructure system: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 7, pp. 7665–7679, 2022.

TABLE VII
IA VS BOT RESULTS (AVERAGE OVER THE THREE BENCHMARKS).

bot $\rightarrow$	**uniform**	**last**	**rockx3**	**rockx2**	**rockx0**	**rockx1**
uniform	$(0.3184, 0.3781)$	$(0.2189, 0)$	$(0.5621, 0.2238)$	$(0.4676, 0.2388)$	$(0.4527, 0.1293)$	$(0.955, 0)$
last	$(0.2686, 0.3681)$	$(0.199, 0.0099)$	$(0.6019, 0.1741)$	$(0.4975, 0.2587)$	$(0.4726, 0.4577)$	$(0.995, 0)$
rockx3	$(0.3532, 0.3034)$	$(0.2039, 0)$	$(0.592, 0.1691)$	$(0.4726, 0.2388)$	$(0.4676, 0.0746)$	$(0.998, 0)$
rockx2	$(0.3482, 0.3034)$	$(0.2039, 0)$	$(0.6218, 0.1641)$	$(0.5074, 0.2537)$	$(0.4427, 0.1839)$	$(0.995, 0)$
rockx0	$(0.2736, 0.3731)$	$(0.2039, 0)$	$(0.5971, 0.1939)$	$(0.4726, 0.2487)$	$(0.5771, 0)$	$(1, 0)$
rockx1	$(0.3333, 0.2338)$	$(0, 0.0049)$	$(0.5823, 0.1942)$	$(0.5472, 0.2089)$	$(0.4477, 0.3233)$	$(0, 0)$
entropy	$(0.4079, 0.194)$	$(0.4477, 0.1044)$	$(0.4378, 0.1144)$	$(0.4179, 0.099)$	$(0.4527, 0.1243)$	$(0.995, 0)$

TABLE VIII
TRAJECTORY LENGTH RATIO STATISTICS.

Metric	$\mathrm{E}[\mathcal{L}]$	$\mathrm{Var}[\mathcal{L}]$	κ_3	κ_4
Benchmark 1	3.7819	3.2154	1.4471	3.1333
Benchmark 2	4.1373	3.7998	1.6147	3.7711
Benchmark 3	4.6333	4.0011	1.8533	4.9347

[7] G. Drakopoulos, E. Kafeza, P. Mylonas, and L. Iliadis, "Transform-based graph topology similarity metrics," *NCAA*, vol. 33, no. 23, pp. 16 363–16 375, 2021.

[8] G. Drakopoulos, I. Giannoukou, P. Mylonas, and S. Sioutas, "A graph neural network for assessing the affective coherence of Twitter graphs," in *IEEE Big Data*. IEEE, 2020, pp. 3618–3627.

[9] G. Drakopoulos, E. Kafeza, P. Mylonas, and S. Sioutas, "A graph neural network for fuzzy Twitter graphs," in *CIKM companion volume*, G. Cong and M. Ramanath, Eds., vol. 3052. CEUR-WS.org, 2021.

[10] Z. Tong, F. Ye, M. Yan, H. Liu, and S. Basodi, "A survey on algorithms for intelligent computing and smart city applications," *Big Data Mining and Analytics*, vol. 4, no. 3, pp. 155–172, 2021.

[11] G. Drakopoulos and P. Mylonas, "A genetic algorithm for Boolean semiring matrix factorization with applications to graph mining," in *Big Data*. IEEE, 2022.

[12] H. Li, Q. Zhang, and D. Zhao, "Deep reinforcement learning-based automatic exploration for navigation in unknown environment," *IEEE Transactions on neural networks and learning systems*, vol. 31, no. 6, pp. 2064–2076, 2019.

[13] G. Drakopoulos, E. Kafeza, P. Mylonas, and S. Sioutas, "Approximate high dimensional graph mining with matrix polar factorization: A Twitter application," in *IEEE Big Data*. IEEE, 2021, pp. 4441–4449.

[14] B. Shneiderman, "Design lessons from AI's two grand goals: Human emulation and useful applications," *IEEE Transactions on Technology and Society*, vol. 1, no. 2, pp. 73–82, 2020.

[15] A. Anand, E. Racah, S. Ozair, Y. Bengio, M.-A. Côté, and R. D. Hjelm, "Unsupervised state representation learning in atari," *Advances in neural information processing systems*, vol. 32, 2019.

[16] M. Schranz, G. A. Di Caro, T. Schmickl, W. Elmenreich, F. Arvin, A. Şekercioğlu, and M. Sende, "Swarm intelligence and cyber-physical systems: Concepts, challenges and future trends," *Swarm and Evolutionary Computation*, vol. 60, 2021.

[17] N. Yang, L. Han, R. Liu, Z. Wei, H. Liu, and C. Xiang, "Multi-objective intelligent energy management for hybrid electric vehicles based on multi-agent reinforcement learning," *IEEE Transactions on Transportation Electrification*, 2023.

[18] X. Mo, Z. Huang, Y. Xing, and C. Lv, "Multi-agent trajectory prediction with heterogeneous edge-enhanced graph attention network," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 9554–9567, 2022.

[19] D. Roy and M. Dutta, "A systematic review and research perspective on recommender systems," *Journal of Big Data*, vol. 9, no. 1, p. 59, 2022.

[20] G. Drakopoulos, I. Giannoukou, S. Sioutas, and P. Mylonas, "Self organizing maps for cultural content delivery," *NCAA*, vol. 31, no. 7, 2022.

[21] E. Elshan, N. Zierau, C. Engel, A. Janson, and J. M. Leimeister, "Understanding the design elements affecting user acceptance of intelligent agents: Past, present and future," *Information Systems Frontiers*, vol. 24, no. 3, pp. 699–730, 2022.

[22] D. Kirli *et al.*, "Smart contracts in energy systems: A systematic review of fundamental approaches and implementations," *Renewable and Sustainable Energy Reviews*, vol. 158, 2022.

[23] S. Swain and M. R. Patra, "A distributed agent-oriented framework for blockchain-enabled supply chain management," in *ICBDS*. IEEE, 2022, pp. 1–7.

[24] G. Drakopoulos, E. Kafeza, and H. Al Katheeri, "Proof systems in blockchains: A survey," in *SEEDA-CECNSM*. IEEE, 2019.

[25] F. De la Prieta, S. Rodríguez-González, P. Chamoso, J. M. Corchado, and J. Bajo, "Survey of agent-based cloud computing applications," *Future generation computer systems*, vol. 100, pp. 223–236, 2019.

[26] X. Yi and C. Zhong, "Deep learning for joint channel estimation and signal detection in OFDM systems," *IEEE Communications Letters*, vol. 24, no. 12, pp. 2780–2784, 2020.

[27] K. Abratkiewicz, P. Samczyński, and K. Czarnecki, "Radar signal parameters estimation using phase accelerogram in the time-frequency domain," *IEEE Sensors Journal*, vol. 19, no. 13, pp. 5078–5085, 2019.

[28] T. L. Jensen and E. De Carvalho, "An optimal channel estimation scheme for intelligent reflecting surfaces based on a minimum variance unbiased estimator," in *ICASSP*. IEEE, 2020, pp. 5000–5004.

[29] S. Naz, R. Sultan, K. Zaman, A. M. Aldakhil, A. A. Nassani, and M. M. Q. Abro, "Moderating and mediating role of renewable energy consumption, FDI inflows, and economic growth on carbon dioxide emissions: Evidence from robust least square estimator," *Environmental Science and Pollution Research*, vol. 26, pp. 2806–2819, 2019.

[30] M. Li and X. Liu, "Maximum likelihood least squares based iterative estimation for a class of bilinear systems using the data filtering technique," *International Journal of Control, Automation and Systems*, vol. 18, no. 6, pp. 1581–1592, 2020.

[31] K. R. Mestav, J. Luengo-Rozas, and L. Tong, "Bayesian state estimation for unobservable distribution systems via deep learning," *IEEE Transactions on Power Systems*, vol. 34, no. 6, pp. 4910–4920, 2019.

[32] J. E. Bernhard, J. S. Moreland, and S. A. Bass, "Bayesian estimation of the specific shear and bulk viscosity of quark–gluon plasma," *Nature Physics*, vol. 15, no. 11, pp. 1113–1117, 2019.

[33] G. Drakopoulos, Y. Voutos, P. Mylonas, and S. Sioutas, "Motivating item annotations in cultural portals with UI/UX based on behavioral economics," in *IISA*. IEEE, 2021.

[34] G. Drakopoulos, E. Spyrou, Y. Voutos, and P. Mylonas, "A semantically annotated JSON metadata structure for open linked cultural data in Neo4j," in *PCI*. ACM, 2019.

[35] S. Chellappan and D. Ganesan, "Scala: Functional programming aspects," in *Practical Apache Spark*. Springer, 2018, pp. 1–37.

[36] P. Hudak, "Conception, evolution, and application of functional programming languages," *ACM CSUR*, vol. 21, no. 3, pp. 359–411, 1989.

[37] D. Pollak, V. Layka, and A. Sacco, "Functional programming," in *Beginning Scala 3*. Springer, 2022, pp. 79–109.

[38] M. Hartmann and R. Shevchenko, *Professional Scala: Combine object-oriented and functional programming to build high-performance applications*. Packt Publishing Ltd, 2018.

[39] C. Okasaki, *Purely functional data structures*. Cambridge University Press, 1999.

[40] S. F. Lott, *Functional Python programming: Discover the power of functional programming, generator functions, lazy evaluation, the built-in itertools library, and monads*. Packt Publishing Ltd, 2018.

[41] R. Dyer and J. Chauhan, "An exploratory study on the predominant programming paradigms in Python code," in *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2022, pp. 684–695.