Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Ablation Study of AC2CD Hyperparameters

Rafael Henrique Nogalha de Lima

Artigo apresentado como requisito parcial para
conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof. Dra. Célia Ghedini Ralha

Brasília
2023

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Ablation Study of AC2CD Hyperparameters

Rafael Henrique Nogalha de Lima

Artigo apresentado como requisito parcial para
conclusão do Bacharelado em Ciência da Computação

Prof. Dra. Célia Ghedini Ralha (Orientadora)
CIC/UnB

Prof. Dr. Thiago Paulo Faleiros
CIC/UnB

MSc. Aurélio Ribeiro Costa
STI/STF

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 14 de julho de 2023

Ablation Study of AC2CD Hyperparameters

Rafael Henrique Nogalha de Lima

Department of Computer Science, University of Brasília
Campus Darcy Ribeiro, Brasília, Federal District, Brazil
rafael.nogalha@aluno.unb.br

Abstract. This article presents an ablation study for Deep Reinforcement Learning (DRL) with a case of the Actor-Critic Architecture for Community Detection (AC2CD), developed upon DRL and Graph Attention Networks (GAT). The ablation study method is sustained by the explainable artificial intelligence approach, including execution time, memory, and GPU usage to assess the AC2CD performance. The datasets used in the experiments include two real-world data. The first is an email network between members of a European research institution (Email-Eu) with $1,005$ nodes, $25,571$ edges, and 42 communities, available on the Stanford Snap Project. The second is a High School contact and friendship network in Marseilles, France, in December 2013 with 329 nodes, $45,047$ edges, and nine communities, available on Socio Patterns Website. The three hyperparameters used to analyze the architecture execution are the learn_rate, batch_size, and n_games, varying from 10%, 30%, 50%, and 70%. With the achieved experimental results, we find a set of hyperparameters with optimal balance contributing to analysis that might interest the DRL and GAT communities for each analyzed dataset.

Keywords: Ablation study · AC2CD · Hyperparameters.

1 Introduction

Undoubtedly, Machine Learning (ML) is no longer far from the reality of the Artificial Intelligence (AI) community and the current society. Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) refer to learning how to make decisions sequentially while being influenced by the environment, becoming mature in the past years [30]. In short, the RL goal is to map situations to actions that maximize a numerical reward signal indicating how well the agent performs tasks. Agents learn through trial and error, adjusting actions to achieve the highest possible reward. And DRL integrates deep learning into RL techniques to train an agent.

Given the advancement in research and the diverse applications of ML, specifically RL, in various fields, including scientific and commercial domains, it becomes imperative to understand the impact of selecting specific components and parameters for developing an ML system, given its complexity. Therefore, a compelling approach to address this issue is to conduct an Ablation Study (AS) [22].

As presented in [29], the AS technique is a scientific examination of ML systems to gain insight into the effects of code blocks on performance. And because of that, the method of AS has gained significant attention in the field of ML in recent years.

The AS findings inform future research on optimizing actor-critic architectures and potentially lead to the development of automated hyperparameters tuning techniques. The actor-critic is a Temporal Difference (TD) version of policy gradient with two networks: actor and critic. The actor decides which action to take, and the critic informs the actor how good the action was and how it should adjust. Investigating the impact of learning strategies and other components on the optimization process contributes to the broader understanding of DRL models and their transformation towards based Explainable Artificial Intelligence (XAI) [14].

Therefore, the main objective of this work is to present an AS approach using an actor-critic architecture developed upon DRL, and Graph Attention Networks (GAT), called the Actor-Critic for Community Detection (AC2CD) [4]. The AS aims to analyze and identify the components that significantly influence the AC2CD algorithm when executing a real-world dataset of an email network between members of a European research institution (Email-Eu) and a High School contact and friendship network in Marseilles, France, in December 2013 presented in Section 3.6. We evaluate the AC2CD performance by empirically modifying specific hyperparameters within the algorithm.

This work contribution is to present an AS with the AC2CD architecture, shedding light on the influential factors that contribute to the algorithm's performance. The insights gained from this study contribute to the ongoing efforts to enhance the efficiency and effectiveness of DRL algorithms in real-world applications in the XAI direction. The rest of the manuscript presents in Section 2 related work, Section 3 materials and methods, Section 4 the results with discussion, and in Section 5 conclusion and future work.

2 Related Work

This section presents related work focusing on AS and DRL using an actor-critic architecture with publications from 2018 to 2023. Table 1 presents an outline of the related work, including AS, DRL, actor-critic, and GAT aspects.

The authors in [8] present a new approach using DRL and actor-critic for a multi-agent system that analyzes and simulates an environment with multiple intelligent agents across various domains. Additionally, the authors conduct an AS to assess the effectiveness of the innovative components in the proposed method. The results show that each actor-critic algorithm component is indispensable for good interception performance, including success rate, good reward, and interception steps.

In [21], the authors explore the utilization of DRL for congestion control in cellular network settings. Congestion control uses algorithms responsible for regulating the data transmission rate in a network to prevent congestion. Using the

Table 1. Related work outline.

Reference	AS	DRL	Actor-critic	GAT
Fan et al. (2023) [8]	✓	✓	✓	
Naqvi & Anggorojati (2022) [21]	✓	✓		
Ye et al. (2022) [32]	✓	✓	✓	
da Silva Filho et al. (2022) [5]	✓	✓		
Hessel et al. (2018) [15]	✓	✓		
This study	✓	✓	✓	✓

policy gradient method, the author employs an AS to identify the component that influences the algorithm. With the AS, the authors remove or modify parameters to analyze the impact of the changes on the algorithm's performance. In conclusion, a higher reward for the method presented is only sometimes related to better networking performance.

In [32], the authors focus on the popularity of multi-agent DRL demand for large-scale real-world tasks, which hamper the models' low sample efficiency and the high data collection cost. Thus, AS is used to investigate, validate and understand the contribution of each component in multi-agent actor-critic methods. The authors propose PEDMA, a plugin unit for multi-agent DRL that consists of three techniques: (i) parallel environments to accelerate the data acquisition; (ii) experience augmentation that utilizes the permutation invariance property of the multi-agent system to reduce the cost of acquiring data; and (iii) delayed updated policies to improve the data utilization efficiency of the multi-agent DRL model. Experiments on three multi-agent benchmark tasks show that the multi-agent actor-critic model trained with PEDMA outperforms the baselines and state-of-the-art algorithms.

The authors in [5] discuss the learning-to-optimize method for automatically optimizing algorithms from data instead of using traditional hyperparameters tuning. The focus is on learning global optimization by DRL. The authors advocate that learning to optimize could be a better-explored theme. It provides a direct framework to understand an optimizer able to deal with the exploration-exploitation dilemma and that the applied techniques improved stability and generalization. Thus, the authors conducted AS to investigate the significance of learning strategies and components concerning this optimization.

In [15], the authors perform an AS to understand the contribution of the components and parameters in the Deep Q-Network (DQN) algorithm that utilizes DRL to address the challenge of learning in complex and high-dimensional environments. In each ablation phase, an algorithm component is removed or changed. Subsequently, the algorithm's performance is analyzed. The authors propose Rainbow to combine improvements in DRL. In experiments, the authors examine six extensions to the DQN algorithm and empirically study their combination. The results show that the combination provides state-of-the-art

performance on the Atari 2600 benchmark considering data efficiency and final performance.

Note this work is the only one that includes GAT as a novel convolution-style neural network architecture that operates on graph-structured data. GAT is one of the most popular types of graph neural networks applied to the community detection problem. But although GAT presents a significant direction for ML research, it has received comparatively low levels of attention, motivating this AS to assess its effectiveness through the AC2CD case study.

3 Material and Methods

3.1 AI Overview

According to [30], the AI field is vast, encompassing various domains of knowledge such as engineering, pharmacy, biology, medicine, and many others. Currently, AI has branched out and formed subfields such as ML. Since 1959, [27] has defined ML as a field of study that allows computers to learn without being explicitly programmed. ML aims to emulate human intelligence through learning based on the parameters of the environment and context in which the machine is embedded [7]. Traditionally, ML is categorized into three types: supervised learning, unsupervised learning, and RL [30].

Supervised learning is a type of learning that utilizes a training set of labeled examples provided by an experienced external supervisor. Unsupervised learning is a type of ML that seeks to find hidden structures in unlabeled data. RL operates through rewards given by the model to the learner for each correct learning instance. The objective is to learn mapping situations to actions to maximize the accumulated reward. There are two essential characteristics: trial-and-error search refers to the learner's attempts throughout the algorithm involving trial and error to receive a reward upon successful execution. The delayed reward relates to the consequences of the agent's learning, which determines the immediate reward and the next state of the environment and future rewards [16].

For [9], DRL is an RL approach combined with deep learning employed when decisions become too complex for RL alone. A neural network estimates states instead of mapping all possible solutions, allowing for a more manageable solution space in the decision-making process.

3.2 Community Detection

Community detection is one of the fundamental problems in network analysis, belonging to the field of complex network studies. According to [10], the community detection technique is characterized by having a community structure, where the nodes in the network are grouped into sets such that each set of nodes is densely connected. For [2], community detection is the process of identifying relevant communities in a network that evolves as in a dynamic network. Community detection is vital to understanding the structure of complex networks.

Community detection techniques are helpful for social media algorithms to discover people with similar opinions, functions, purposes, and shared interests significant to scientific inquiry and data analytics. There are classic methods of community detection using spectral clustering [23] and statistical inference [17]. However, such methods drop out, as deep learning techniques demonstrate an increased capacity to handle high-dimensional graph data with impressive performance.

3.3 Actor-critic

Actor-critic is a TD learning method representing the policy function independent of the value function. The policy function returns a probability distribution over possible agents' actions based on the provided state or the agents' strategy to achieve a goal. On the other hand, the value function determines the expected return for an agent starting in a particular state and continually acting under a specific policy [11, 30].

In the actor-critic learning method, the actor decides which action to take. The critic provides feedback to the actor on the quality of the action and how it can be adjusted to achieve the goal [18]. In short, the actor-critic is a hybrid architecture combining value-based and policy-based methods that help to stabilize the training by reducing the variance. It provides a solution to reducing the RL algorithm variance, training agents faster and better.

3.4 AC2CD Overview

The AC2CD is employed to find an optimal community structure in a dynamic social network while also serving as a learning component to select actions and improve the value function [4]. It explores the power of GAT, proposing a community detection model using an actor-critic DRL-based architecture to maximize the local modularity density of a community structure in the context of dynamic online networks. AC2CD uses the message-passing feature of GAT as an element to propagate the label for each community, thus improving the modularity density of the community structure.

The implementation of actor-critic uses Proximal Policy Optimization (PPO) in the clipped version and generalized advantage estimation to compute the surrogate function of the policy gradient. According to [6], PPO performs the best in terms of profit and loss, training time, and data needed compared to Q-learning and deep Q-learning. It is worth noting that the proposed architecture can accommodate other implementations of Graph Neural Networks (GNN) as Graph Convolution Networks (GCN). The source code is in Python language and available to the research community.[1]

Figure 1 presents the AC2CD architecture overview highlighting the actor-critic components in gray. The GNN components include a *Dropout* regularization layer and the first attention layer *GATConv1*. The *ReLU* activation and

[1] `https://gitlab.com/InfoKnow/SocialNetwork/ac2cd`

Dropout layers, and the second attention layer, *GATConv2*, with the output *Softmax* activation layer. The input data corresponds to network snapshots taken at each network change and embedded by the encoder in the data manipulation as a discrete-time dynamic graph. The Edge list.txt is a file in the coordinate format (or ijv format), where each line corresponds to an edge, i.e., a pair of node ids and possibly edges attributes (timestamp, weight, and other features).

The *Data manipulation* module in AC2CD utilizes *Edge list.txt* and *Ground truth.txt* files for generating embeddings. The latter contains node-community assignments, while the former stores network information in various formats, including temporal and non-temporal networks. Figure 1 shows the interaction among the agent, environment, and internal entities, with nodes represented as 256-dimensional vectors using the Node2Vector method [12].

Figure 1 shows the *Environment* component presenting its data structures, including the edge index of the network and a coordinate format list representing the network's connectivity. The environment manages the action and observation spaces, which are initialized at the beginning of the execution, i.e., at the time step t_0. At each time step t, the environment receives an action a_t generated by the agent and computes the effects produced a_t, developing a new state s_{t+1}, a reward r_{t+1}, and indicating if the state is terminal (line 9 of the Algorithm 1).

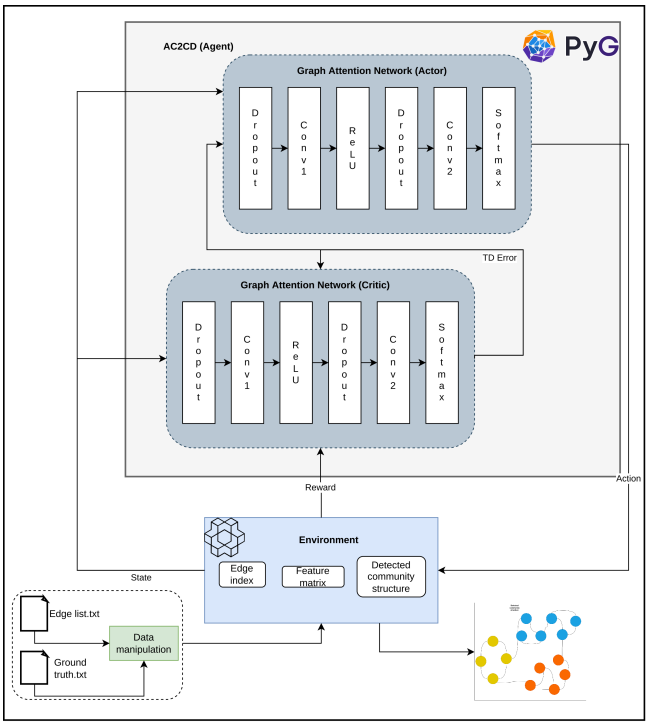

Fig. 1. The AC2CD architecture from [3].

Algorithm 1 presents the AC2CD for community detection, including the input (i.e., Input:) with *Dataset* and the hyperparameters (Hp). The manipulation of *Dataset* is made by *DataManip(Dataset)*. The method *Agent(Hp)* create the agent according to *Hp* and consequently makes the actor, the Critic networks, and the experience memory store the last episodes of execution. Inside the loop, *line 5* to *line 12*, while not in a terminal state, the agent chooses an action based on the observations and executes a step returning a new observation, the reward, and a flag indicating that the new state is terminal. After the learning process, the model file is ready to infer the community assignment for nodes in a network, and the output is the score stored in the *score_history* variable.

The Ac2CD experimental work indicates that AC2CD copes well with dynamic real-world social networks. Nevertheless, the performance of such complex architecture motivates AS approach to enhance performance to evaluate the growing size of dynamic social networks.

Algorithm 1: Community Detection in AC2CD

```
   Input: Dataset
   Input: Hp
 1 node_emb, edge_index ← DataManip(Dataset);
 2 env ← GATEnv(node_emb, edge_index);
 3 agent ← Agent(Hp);
 4 score_history ← [];
 5 while n < Hp.max_iter do
 6     obs ← env.reset();
 7     while not done do
 8         action, prob, val ← agent.choose_action(obs);
 9         new_obs,reward,done ← env.step(action);
10         agent.remember(obs, action, reward,prob,val);
11         if n % Hp.train_interval == 0 then
12             agent.learn();
```

3.5 Ablation Study

With the growth of ML approaches, various domains of knowledge have benefited. However, systems with AI have become complex and hard to understand and explain. As a result, a new approach to AI-based systems has emerged to provide explainability to human users, highlighting the strengths and weaknesses of the algorithm and conveying an understanding of how it will behave in the future. According to [13], XAI enables greater transparency and interpretability in complex AI systems allowing users' trust and permitting humans to make informed decisions while effectively cooperating with such systems. XAI bridges the gap between the black-box nature of traditional AI and the human need for

comprehensibility by providing explanations for algorithmic decisions. That enhances the usability and ethical considerations of AI applications across various domains.

The first idea of AS comes from speech recognition studies [25]. Although not a new idea, it is a relatively young AI research theme [14]. AS is defined by [20] as a scientific method that involves highlighting or removing individual or blocks of components from a system to prove and understand which aspects of a system are vital through statistical analysis. Using statistics and analyzing the results obtained from AS, it is possible to gain insights into the relative importance of the parameters of architecture or model. With these insights, improving systems' design, optimization, and interpretability is possible. AS is a valuable tool for discovering the component's influence in ML systems. Through statistical analysis, it is possible to enhance the interpretability of ML approaches.

There are different ways to conduct an AS. One can remove architectural elements or use the Hp. They are used to configure an ML model and specify the algorithm to minimize the loss function, for example, [24]. The use of AS in XAI systems becomes interesting, considering its complementarity to understanding AI systems. The AS aims to understand the importance of parameters and code blocks in an architecture or model [14]. This study enables identifying and quantifying the influence of various components on an algorithm, model, or architecture, leading to a better understanding of the underlying mechanisms. This understanding is crucial for building trust and ensuring transparency in AI systems.

3.6 Ablation Study Method

Figure 2 presents the experimental method with three steps. The first step includes the datasets and baseline definitions. The first dataset is called Email-Eu-Core (EC), available on the Snap Project website[2] and used as input to AC2CD. The second dataset is the High School Contact and Friendship Network (HS), available on the Socio Patterns website[3] and also used as input to AC2CD. The second step includes the hyperparameter variations during the AC2CD executions (*tunning*). Finally, in the third step, we use the AS to observe the importance of the selected hyperparameters. The effect analysis focuses on the execution time, GPU, and memory usage.

The EC is a directed network representing an email network between members of a European research institution (Email-Eu). According to [31], the network is formed by an edge (u, v), where u represents the person who sends at least one email to v. The communities in the dataset represent the departments in the organization. There are $1,005$ nodes, $25,571$ edges, and 42 communities, with the longest path being seven, and the average clustering coefficient is 0.3994.

[2] https://snap.stanford.edu/data/email-Eu-core.html
[3] http://www.sociopatterns.org/datasets/high-school-contact-and-friendship-networks/

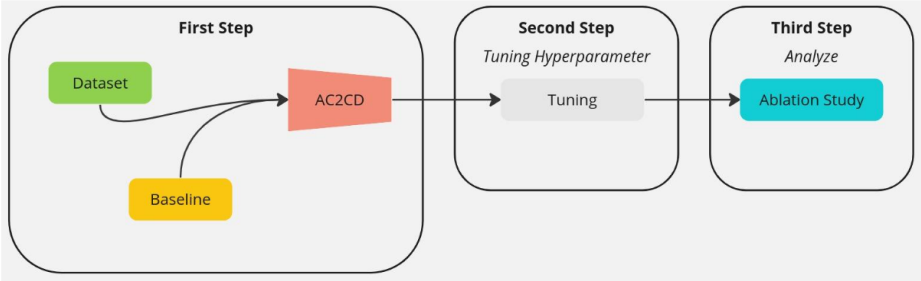

Fig. 2. Experimental method diagram.

The HS comprises a directed network of contacts and friendship relations between students in a high school in Marseilles, France, in December 2013. According to [19], the network is formed by lines with the form (ij), meaning that the student i reported a friendship with student j. And is formed by a metadata file where each line has the form $(IDiCiGi)$, meaning that class Ci and gender Gi of the student has IDi. There are 329 nodes, 45047 edges and 9 communities.

3.7 Experimental Setup

The AC2CD includes 16 Hp.[4] We use in the AS three empirically defined as the most influential ones with the baseline values of *learn_rate=40* (LR), *batch_size=40* (BS), *n_games=100* (NG). The AS setup uses I equal to the percentage variations of 10%, 30%, 50%, and 70%. Tests provided the comprehensiveness coverage of these percentages. The LR determines how much an agent learns from each sample in the environment [26]. The BS represents the number of samples propagated during the training session [28]. Lastly, NG defines the number of episodes the agent will process.

The AS method uses the Scalene profiler.[5] The AS execution starts, and the matplotlib[6] tool to generate graphics is used for GPU and memory analysis. This experiment aims to determine a set of Hp in H that minimizes GPU and memory consumption for the AC2CD while achieving maximum speed. As shown in Figure 3, Scalene is chosen due to its superior performance compared to other well-known profilers. The results indicate its effectiveness in slowing down the program, profiling memory and GPU usage, and providing system time analysis. Additionally, a *.json* file generated by Scalene allows for line-level and function-level profiling, offering information about specific functions and lines of code, as documented in [1].

The executions followed 125 turns for each dataset with six scenarios, including the baseline for a GPU consumption comparison, memory usage, and execution time. The selection of the six scenarios considered the lowest GPU

[4] `https://gitlab.com/InfoKnow/SocialNetwork/ac2cd`
[5] https://github.com/plasma-umass/scalene
[6] https://matplotlib.org/

Profiler	Slowdown	Lines or Functions	Unmodified Code	Threads	Multi-processing	Python vs. C Time	System Time	Profiles Memory	Python vs. C Memory	GPU	Memory Trends	Copy Volume	Detects Leaks
CPU-only profilers													
pprofile (stat.)	1.0×	lines	✓	✓	-	-	-	-	-	-	-	-	-
py-spy	1.0×	lines	✓	✓	✓	-	-	-	-	-	-	-	-
pyinstrument	1.7×	functions	✓	-	-	-	-	-	-	-	-	-	-
cProfile	1.7×	functions	✓	-	-	-	-	-	-	-	-	-	-
yappi wallclock	3.2×	functions	✓	✓	-	-	-	-	-	-	-	-	-
yappi CPU	3.6×	functions	✓	✓	-	-	-	-	-	-	-	-	-
line_profiler	2.2×	lines	-	-	-	-	-	-	-	-	-	-	-
Profile	15.1×	functions	✓	-	-	-	-	-	-	-	-	-	-
pprofile (det.)	36.8×	lines	✓	✓	-	-	-	-	-	-	-	-	-
memory-only profilers													
fil	2.7×	lines	-	-	-	-	-	*peak only*	-	-	-	-	-
memory_profiler	≥37.1×	lines	-	-	-	-	-	*RSS*	-	-	-	-	-
memray	4.0×	lines	-	✓	-	-	-	*peak only*	✓	-	-	-	-
CPU+memory profilers													
Austin (CPU+mem)	1.0×	lines	✓	✓	✓	-	-	*RSS*	-	-	-	-	-
Scalene (CPU+GPU)	1.0×	**both**	✓	✓	✓	✓	✓	-	-	✓	-	-	-
Scalene (all)	1.3×	**both**	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Fig. 3. Comparison of profiler tools.

consumption, memory usage, and execution time. It is essential to highlight that the set of Hp does not operate individually for GPU consumption, memory, and execution time. In other words, if we choose a set of Hp, it will be chosen for GPU consumption, memory, and execution time collectively rather than individually for each of them.

The experiments use a computer with a CPU Intel® Xeon Gold 5220R with 48 cores, 187GB of RAM, and two GPU NVIDIA® V100S. The operating system used is Ubuntu, with external libraries provided by the Conda project.[7]

4 Results and Discussion

In this section, the GPU and memory usage, and run-time execution results are present for the AC2CD architecture after executing it with I variations of 10%, 30%, 50%, 70% using EC and HS datasets. The results of the baseline execution for EC were 12.092 GiB for the memory GPU usage, 1.124 GiB for memory consumption, and $1h40m24s$ for run-time execution. And HS was 17.339 GiB for the memory GPU usage, 1.006 GiB for the memory consumption, and $10m26s$ for run-time execution (Figures 4 and 7).

4.1 GPU Consumption

Figure 4 presents the best results for GPU usage with EC and HS datasets. Note that GPU consumption with EC (i.e., blue bars) is higher for the baseline (black dotted line with 12.092 GiB), BS(10%) (i.e., 12.025 GiB), and NG(10%) (i.e., 11.999 GiB). For the LR(70%) (i.e., 10.911 GiB), and LR(30%), BS(30%), NG(10%) (i.e., 10.223 GiB), both sets of hyperparameters are adequate, but the last is the best result in terms of GPU consumption. This figure also presents the

[7] Conda Project available at `https://docs.conda.io/en/latest/`

best results for GPU usage with HS (i.e., green bars). The baseline (red dotted line) is the highest value with 17.339 GiB of GPU consumption, and for this dataset, the best result is LR(70%) and BS(70%) (i.e., 14.123 GiB).

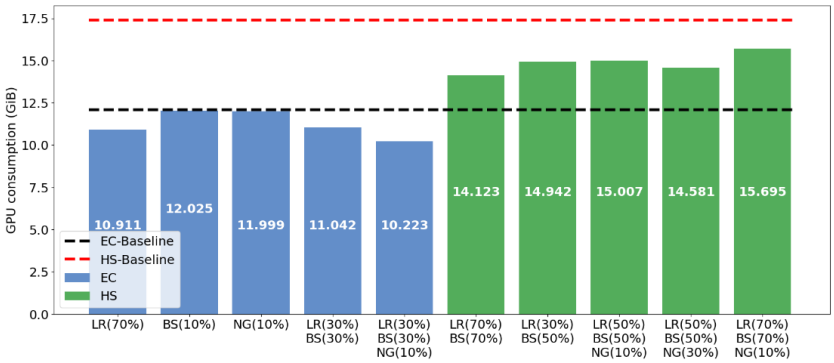

Fig. 4. Best results in terms of GPU consumption with EC and HS datasets.

Furthermore, Figure 5 presents the Scalene interface for LR(30%), BS(10%), and NG(10%) with the EC dataset. The percentage of GPU consumption (i.e., 31.2% corresponds to 10.223 GiB) is displayed using a pie graph in the GPU utilization column (*GPU util.*). The *GPU memory* column presents the code line memory consumption. Finally, the function that consumes the GPU referring to the previously shown code line appears below the *FUNCTION PROFILE* column. Note that the code segment related to agent learning (*Agent.learn*) for each node in the EC is the one that consumes the most GPU in this configuration. In general, this is the code block that consumes most of the AC2CD algorithm. In addition, Figure 6 presents the Scalene interface for the GPU consumption (i.e., 43.1% corresponds to 14.123 GiB) with LR(70%), BS(70%) with the HS dataset, where the *Agent.learn* consumes more GPU memory.

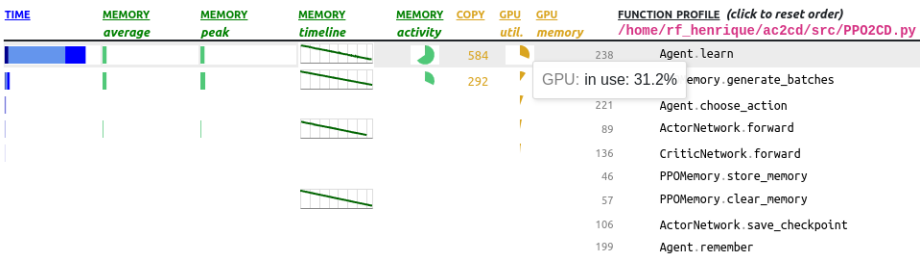

Fig. 5. Scalene interface for LR(30%), BS(10%), NG(10%) with the EC dataset.

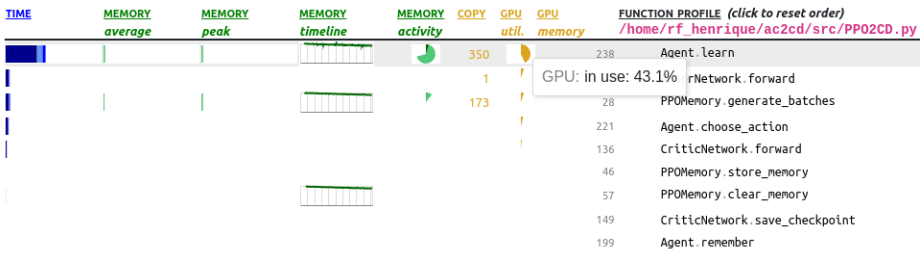

Fig. 6. Scalene interface for LR(70%), BS(70%) with the HS dataset.

What is interesting in the execution of HS is that, despite having fewer communities and being smaller than the EC dataset, it consumes more GPU. That might occur due to the different transformations in the dataset file for EC, where the *torch.LongTensor* generates the index, transposing data by the generated index, as seen in Listing 1. In contrast, for the HS dataset, both the index and data are derived using the $from\_networkx$ function, which converts a graph to a *torch_geometric.data.Data* instance[8] as seen in Listing 2. Another possibility for the high GPU consumption is the use of metadata in HS, which makes the execution of this dataset more complex, requiring more computational power.

```
index = torch.LongTensor(
    np.vstack((adj.row, adj.col))
)
data = Data(edge_index=torch.transpose(index,0,1))
enconder = embedding.Encoder(data,index,device=self.device)
```

Listing 1. Code execution fragment with the EC dataset.

```
data = from_networkx(graph)
encoder = embedding.Encoder(data,
    data.edge_index,
    device=self.device
)
```

Listing 2. Code execution fragment with the HS dataset.

[8] https://pytorch-geometric.readthedocs.io/en/latest/

Memory Consumption

As observed in Figure 7, the memory consumption does not exhibit as much variation comparing the GPU consumption of the EC dataset (Figure 4). We can attribute this behavior to the fact that most memory allocation in the AC2CD is related to tensors GPU stored consumption. As possible to observe, BS(10%) is the best result of memory consumption with 1.123 GiB.

For the HS dataset, the baseline consumes 1.006 GiB, and for LR(70%), BS(70%) consumes 1.021 GiB, and LR(70%), BS(70%), NG(10%) consumes 1.022 GiB. In relation to LR(50%), BS(50%), NG(30%) the consumption is 1.007 GiB. Finally, LR(30%), BS(50%), and LR(50%) BS(50%), NG(10%) represent the best results for memory consumption, with a consumption of 1.003 GiB.

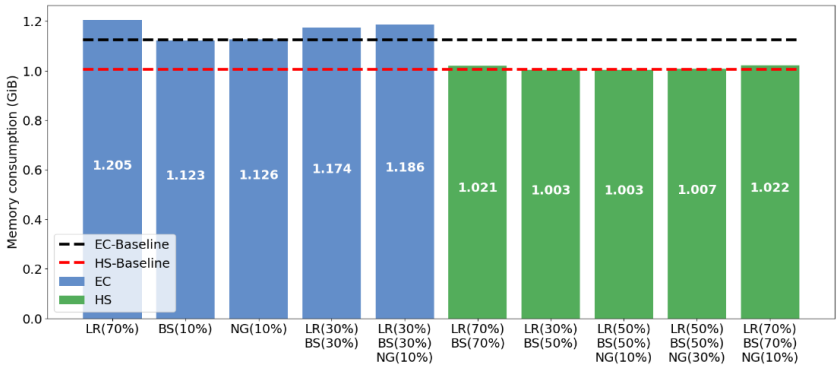

Fig. 7. Best results regarding memory consumption with EC and HS datasets.

Run time Execution

We can observe in Table 2 that in the EC dataset, the LR(70%) is the faster execution with $1h19min23s$. On the other hand, the execution with LR(30%), BS(30%), and NG(10%) is the slowest compared to the other five percentage variation executions taking $2h7min35s$. It happens because the NG hyperparameter determines the number of episodes the agent will process during the execution of the architecture, affecting the run time execution.

Regarding the HS, the execution time is shorter than the EC dataset due to having fewer nodes, edges, and communities. The shortest time is achieved with LR(70%), BS(70%) with $3min25s$, and LR(70%), BS(70%), NG(10%) with $4min08s$ also proved to be a good option. On the other hand, the baseline showed the worst time in the execution comparison with $10min26s$.

Table 2. Run-time results for EC and HS datasets.

EC hyperparameter	EC Run-time execution	HS hyperparameter	HS Run-time execution
baseline	1h40m24s	baseline	10m26s
LR(70%)	**1h19m23s**	LR(70%) BS(70%)	**3m25s**
BS(10%)	1h25m02s	LR(30%) BS(50%)	4m53s
NG(10%)	1h34m10s	LR(50%) BS(50%) NG(10%)	7m08s
LR(30%) BS(30%)	1h29m23s	LR(50%) BS(50%) NG(30%)	8m05s
LR(30%) BS(30%) NG(10%)	2h7m35s	LR(70%) BS(70%) NG(10%)	4m08s

5 Conclusion

This work contribution is to present an AS to AC2CD architecture, shedding light on the influential factors that contribute to the algorithm's performance. The insights gained contribute to the ongoing efforts to enhance the efficiency and effectiveness of DRL algorithms in real-world applications in the XAI direction.

The AS application in AC2CD reveals that it is vital to carefully consider the values of LR, BS, and NG hyperparameters to optimize GPU and memory consumption and run-time execution. Based on the results obtained from our experiments and analyses for the EC and HS datasets, it is evident that LR and BS influence most of the GPU and memory usage, and NG has the most significant impact on run-time execution.

Therefore, for the EC dataset, the LR(30%), BS(30%), NG(10%), and BS(10%) proved adequate options for the hyperparameters configuration of AC2CD architecture considering GPU memory usage and memory consumption, respectively. And for run-time execution, LR(70%) proved to be adequate. Regarding the HS dataset, the LR(70%) and BS(70%) indicate a good option for GPU consumption and execution time. Additionally, it was a good option for memory consumption. Although LR(30%), BS(50%), and LR(50%), BS(50%), NG(10%) perform better in terms of memory, the GPU consumption and execution time are not favorable.

Finally, the AS results indicate that DRL architectural structures and hyperparameters impact GPU, memory, and run-time execution results. In addition, the dataset manipulation by the architecture also influences the execution results. We suggest the DRL developers give special attention to simplifying the architecture, considering tensors, encoders, and learning algorithms to accelerate execution, achieving the best infrastructure results.

In future work, we will consider the implementation of AS for the remaining hyperparameters of the AC2CD architecture with other datasets exploring the potential for automatic hyperparameter tuning. Additionally, investigating the impact of an AS on the agent's learning stage within the AC2CD architecture presents a promising research area for advancing the study of DRL models with a focus on XAI transformations. This path of research holds the potential for significant advancements in understanding and interpreting the decision-making processes of DRL models.

Acknowledgment

I would be glad to express my gratitude to my supervisor Prof. Célia G. Ralha, for her assistance during my final project study and to Aurélio R. Costa for providing technical support throughout the project.

References

1. Emery Berger, Sam Stern, and Juan Pizzorno. Triangulating python performance issues with scalene, 2022. arXiv:2212.07597 [cs.PL].
2. Rémy Cazabet and Frédéric Amblard. *Dynamic Community Detection*, pages 404–414. Springer New York, New York, NY, 2014.
3. Aurélio Ribeiro Costa. *Adaptive Model to Community Detection in Dynamic Social Networks*. PhD thesis, Computer Science Department, University of Brasília, Brasília, Brazil, 2023.
4. Aurélio Ribeiro Costa and Célia Ghedini Ralha. AC2CD: An actor–critic architecture for community detection in dynamic social networks. *Knowledge-Based Systems*, 261:110202, 2023.
5. Moésio Wenceslau da Silva Filho, Gabriel A. Barbosa, and Péricles B. C. Miranda. Learning global optimization by deep reinforcement learning. In *Proc. of 11^{th} Brazilian Conference on Intelligent Systems (BRACIS)*, page 417–433, 2022.
6. Jiayi Du, Muyang Jin, Petter N Kolm, Gordon Ritter, Yixuan Wang, and Bofei Zhang. Deep reinforcement learning for option replication and hedging. *The Journal of Financial Data Science*, 2(4):44–57, 2020.
7. Issam El Naqa and Martin J. Murphy. *What Is Machine Learning?*, pages 3–11. Springer International Publishing, Cham, 2015.
8. Dongyu Fan, Haikuo Shen, and Lijing Dong. Switching-aware multi-agent deep reinforcement learning for target interception. *Applied Intelligence*, 53(7):7876–7891, 2023.
9. Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, et al. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3-4):219–354, 2018.
10. Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.
11. Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.

12. Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proc. of the 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 855–864, 2016.
13. David Gunning and David Aha. Darpa's explainable artificial intelligence (xai) program. *AI magazine*, 40(2):44–58, 2019.
14. Isha Hameed, Samuel Sharpe, Daniel Barcklow, Justin Au-Yeung, Sahil Verma, Jocelyn Huang, Brian Barr, and C. Bayan Bruss. Based-xai: Breaking ablation studies down for explainable artificial intelligence, 2022. arXiv:2207.05566 [cs.LG].
15. Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, et al. Rainbow: Combining improvements in deep reinforcement learning. In *Proc. of AAAI Conf. on Artificial Intelligence*, volume 32, 2018.
16. Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *J. Artif. Int. Res.*, 4(1):237–285, may 1996.
17. Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1), 2011.
18. Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in Neural Inf. Proc. Sys.*, 12, 1999.
19. Rossana Mastrandrea, Julie Fournet, and Alain Barrat. Contact patterns in a high school: a comparison between data collected using wearable sensors, contact diaries and friendship surveys. *PloS one*, 10(9):e0136497, 2015.
20. Richard Meyes, Melanie Lu, Constantin Waubert de Puiseau, and Tobias Meisen. Ablation studies in artificial neural networks, 2019. arXiv:1901.08644 [cs.NE].
21. Haidlir Naqvi and Bayu Anggorojati. Ablation study of deep reinforcement learning congestion control in cellular network settings. In *Proc. of 25th Int. Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 80–85. IEEE, 2022.
22. Allen Newel. A tutorial on speech understanding systems. In D. R. Reddy, editor, *Speech Recognition Invited Papers Presented at the 1974 IEEE Symposium*, pages 3–54. Academic Press, CMU, USA, 1975.
23. Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in Neural Inf. Proc. Sys.*, 14, 2001.
24. Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *The Journal of Machine Learning Research*, 20(1):1934–1965, 2019.
25. Dabbala Rajagopal Reddy. *Speech recognition: invited papers presented at the 1974 IEEE symposium*. Elsevier, 1975.
26. Joshua Romoff. *Decomposing the Bellman Equation in Reinforcement Learning*. PhD thesis, School of Computer Science, McGill University, Montreal, Canada, 2021.
27. A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.
28. Brennan Shacklett, Erik Wijmans, Aleksei Petrenko, Manolis Savva, Dhruv Batra, et al. Large batch simulation for deep reinforcement learning, 2021. arXiv:2103.07013 [cs.LG].
29. Sina Sheikholeslami. Ablation programming for machine learning. Master's thesis, KTH Royal Institute of Technology, School of Electrical Eng. and Comp. Science (EECS), SE-100 44 Stockholm, Sweden, 2019.
30. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

31. Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proc. of 15th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, page 817–826, 2009.
32. Zhenhui Ye, Yining Chen, Xiaohong Jiang, Guanghua Song, Bowei Yang, and Sheng Fan. Improving sample efficiency in multi-agent actor-critic methods. *Applied Intelligence*, pages 1–14, 2022.