# Community Detection with Reinforcement Learning

STAT-8289 Final

William Arliss
Department of Statistics, George Washington University

December 19, 2025

**Abstract**

Inferring community structure from relational data is an important task in network sciences. This paper proposes a Reinforcement Learning (RL) approach to community detection. An RL environment is described that allows agents to iteratively update the community assignments of graph nodes in order to maximize a modularity-based reward signal. The framework is evaluated on real-world citation networks, offering an alternative approach to modularity optimization.

*Keywords:* Community Detection, Modularity, Deep Q-Network

# 1 Introduction

Community detection refers to the task of dividing the nodes of a graph into groups in order to reveal some underlying structure. Communities are typically defined so that more graph edges occur within groups than across groups. In the example of citation networks, where communities may represent research interests, authors are considered to be part of a community if they frequently co-author with members of their own community and infrequently co-author with members of other communities.

Identifying a partition which adequately represents the community structure of a graph is a difficult problem, as there are many ways to measure the quality of a proposed partition and many ways to search for a quality-maximizing partition. Different approaches to community detection have been developed with various objectives in mind. For example, Stochastic Block Models (Lee & Wilkinson 2019) seek to identify communities with sound Statistical properties while Minimum Cut algorithms (Wang et al. 2015) take motivation from Information Theory. In this paper, a modularity maximization approach is taken.

Modularity (Newman 2006) measures the degree to which nodes within the same community are connected. It is higher for partitions where more edges occur within communities than between communities. Optimizing modularity can be computationally expensive, as calculating the metric for every possible partition is prohibitively expensive. Popular approximations for this problem include the Louvain algorithm (Blondel et al. 2008) and the Leiden algorithm (Traag et al. 2019).

This paper proposes to use Reinforcement Learning (RL) to address the combinatorial challenge of estimating the maximum-modularity partition of a graph. RL is a paradigm of machine learning in which a model interacts with an environment in order to maximize a reward signal. Casting community detection as a RL problem requires defining an

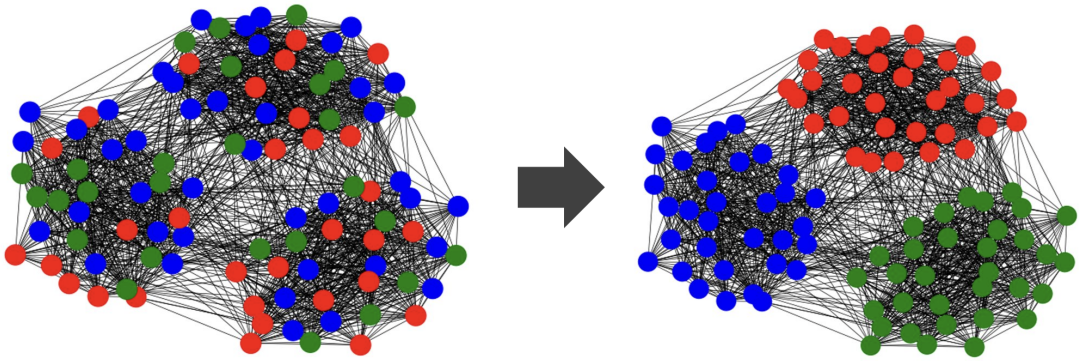

Figure 1: Community detection starts with a partition with low modularity and iteratively improves to a high-modularity partition.

environment around a graph and designing an agent that can effectively interact with that environment to infer a partition.

The community detection environment must have valid transition and reward dynamics. The agent must be able to observe environment states and make decisions that affect its reward. In the remainder of this paper, the agent and environment are defined and their ability to satisfy these requirements is described. Then the framework is evaluated on both synthetic graphs and real citation networks.

## 2 Methods

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. Let $\mathbf{A}$ be the $n \times n$ adjacency matrix, where $\mathbf{A}_{uv} = 1$ if $(u, v) \in \mathcal{E}$ and 0 otherwise. Let $\mathbf{d} = \mathbf{A}\mathbf{1}$ be the degree vector of $\mathcal{G}$. Each node may also be attributed by a vector $\mathbf{X}_u$ of node features.

Assume that the nodes are partitioned into $k$ communities according to the vector $\mathbf{z}$. That is, $\mathbf{z}_u = j$ if node $u \in \mathcal{V}$ is a member of community $j$. The quality of a given partition $\mathbf{z}$ is

measured by the modularity metric. Modularity is defined

$$M(\mathbf{z}; \mathbf{A}) = \frac{1}{2m} \sum_{uv} \left[ \mathbf{A}_{uv} - \frac{\mathbf{d}_u \mathbf{d}_v}{2m} \right] \delta(\mathbf{z}_u, \mathbf{z}_v) \tag{1}$$

where $\delta(x, y) = 1$ if $x = y$ and 0 otherwise. Modularity was presented in Newman & Girvan (2004) and Newman & Girvan (2004) as an objective function for community detection. Thus in the context of this paper, we refer to the optimal partition as the one which maximizes $M$.

## 2.1 Environment

We now define a reinforcement learning environment around $\mathcal{G}$. This environment allows a reinforcement learning agent to infer the optimal node partition through iterative interactions. An agent that solves this environment will be useful for the task of community detection.

A single episode in this environment starts with a randomly initialized partition $\mathbf{z}^{[0]}$. At each step in the episode, the agent observes the current partition $s_t = \mathbf{z}^{[t]}$ and predicts a new partition, $a_t = \hat{\mathbf{z}}^{[t+1]}$. The intermediate reward at each step is the change in modularity given the predicted partition

$$r_t = \Delta(\hat{\mathbf{z}}^{[t+1]}; \mathbf{z}^{[t]}) = \frac{M(\hat{\mathbf{z}}^{[t+1]}) - M(\mathbf{z}^{[t]})}{|M(\mathbf{z}^{[t]})|}. \tag{2}$$

The current partition is then transitioned to $s_{t+1} = \mathbf{z}^{[t+1]}$ according to the prediction $\hat{\mathbf{z}}^{[t+1]}$. The episode continues until a convergence, which occurs when the absolute change in modularity is less than $\epsilon > 0$ for a given number of steps. If convergence is reached at step $T$, the terminal reward is $r_T = 5 + 10 \cdot M(\hat{\mathbf{z}}^{[T]})$. If convergence cannot be reached in a reasonable number of steps, the episode is truncated. The truncated reward after $\tau$ steps is $r_\tau = 10 \cdot M(\hat{\mathbf{z}}^{[\tau]})$.

The procedure for transitioning from $s_t = \mathbf{z}^{[t]}$ to $s_{t+1} = \mathbf{z}^{[t+1]}$ is similar to a single step of label propagation applied to $a_t = \hat{\mathbf{z}}^{[t+1]}$. Label propagation (Zhu & Ghahramani 2002) is

a procedure for inferring unobserved node labels by transferring label information across network edges. The intuition for community detection is that a node's community label should be consistent with the labels of its neighbors.

The label propagation step is adopted for our state-transition procedure as follows. First, define "one-hot encoding" as a mapping operation $\mathbf{z} \mapsto \mathbf{Z}$ where $\mathbf{Z}$ is a $n \times k$ matrix with $\mathbf{Z}_{uj} = 1$ if $\mathbf{z}_u = j$ and 0 otherwise. The one-hot encoding of the predicted partition $\hat{\mathbf{Z}}^{[t+1]}$ is used to create a probability distribution $\hat{\mathbf{P}}_u^{[t+1]}$ for the next partition. This is done by first propagating the predicted partition across edges, then by clamping the probabilities back to $\mathbf{Z}^{[t]}$ if $\mathbf{Z}^{[t]}$ and $\hat{\mathbf{z}}^{[t+1]}$ agree. That is,

$$\text{Propagate:} \qquad \hat{\mathbf{P}}^{[t+1]} = \mathbf{D}^{-1}\mathbf{A}\hat{\mathbf{Z}}^{[t+1]} \tag{3}$$

$$\text{Clamp:} \qquad \hat{\mathbf{P}}_u^{[t+1]} = \begin{cases} \mathbf{Z}_u^{[t]} & \text{if } \mathbf{z}_u^{[t]} = \hat{\mathbf{z}}_u^{[t+1]} \\ \hat{\mathbf{P}}_u^{[t+1]} & \text{o.w.} \end{cases} \tag{4}$$

where $\mathbf{D} = \text{diag}(\mathbf{d})$ is the degree matrix. At this point, the current partition may be updated either stochastically by sampling from this distribution (i.e., $\mathbf{z}_u^{[t+1]} \sim \text{Multinomial}(1, \hat{\mathbf{P}}_u^{[t+1]})$) or deterministically by taking the row-wise arg max (i.e., $\mathbf{z}_u^{[t+1]} = \arg\max_j \hat{\mathbf{P}}_{uj}^{[t+1]}$). The focus of this paper is restricted to the deterministic case.

The transition probabilities are

$$\text{P}(s_{t+1} = \mathbf{z}^{[t+1]} | s_t = \mathbf{z}^{[t]}, a_t = \hat{\mathbf{z}}^{[t+1]}) = \text{clamp}(\text{propagate}(\mathbf{A}, \hat{\mathbf{Z}}^{[t+1]}), \mathbf{Z}^{[t]}) \tag{5}$$

which is largely a function of $\mathbf{A}$ and $\hat{\mathbf{Z}}^{[t+1]}$. Importantly though, it is also a function of $\mathbf{Z}^{[t]}$ through the clamping operation. This supports the validity of this environment as a Markov Decision Process, as the next state is only dependent on the current state and not on previous states. Additionally, the matrix $\mathbf{A}$ in the propagation operation is fixed, the transition dynamics are stationary.

## 2.2 Agent

Let $Q(s_t, a_t)$ be a matrix-valued state-value function. It is defined such that the $u^{\text{th}}$ row is the expected change in modularity from reassigning node $u \in \mathcal{V}$ from community $\mathbf{z}_u^{[t]}$ to community $\hat{\mathbf{z}}_u^{[t+1]}$. The "partial" state-value function for node $u$ is defined $Q_u(s_t, a_t) = [Q(s_t, a_t)]_u$.

Note that the state and action spaces of the environment defined here are $\mathcal{S} = \mathcal{A} = \{1, 2, ..., k\}^n$. The size of this space is extremely large ($k^n$) even for moderately sized graphs, which makes complete enumeration of each state-action pair impossible. Thus, a functional approximation for $Q$ is needed.

In Deep Q-learning (Mnih et al. 2013), a neural network approximator is used estimate the $Q$ function when the state and action spaces are large. The approximator used in this paper is a Graph Convolutional Network (GCN) (Kipf & Welling 2017). The GCN operation is defined on a graph adjacency matrix $\mathbf{A}$ and an associated node feature matrix $\mathbf{H}$ as

$$\text{GCN}(\mathbf{H}, \mathbf{A}; \mathbf{W}) = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}\, \mathbf{W} \tag{6}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the augmented adjacency matrix, $\tilde{\mathbf{D}} = \text{diag}(\tilde{\mathbf{A}}\mathbf{1})$ is the augmented degree matrix, and $\mathbf{W}$ is a matrix of learnable weights. In this setting, the feature matrix is defined as $\mathbf{H}^{[t]} = [\mathbf{X}||\mathbf{Z}^{[t]}]$, which is the concatenation of the node attribute matrix $\mathbf{X}$ and the one-hot encoded current partition $\mathbf{Z}^{[t]}$. If no node attributes exist for a given graph, it is simply $\mathbf{H}^{[t]} = \mathbf{Z}^{[t]}$. The estimated state-value function is

$$Q(s_t, a_t; \mathbf{W}) = \hat{\mathbf{Z}}^{[t+1]} = \text{GCN}(\mathbf{H}^{[t]}, \mathbf{A}; \mathbf{W}) \tag{7}$$

and is referred to as the DQN.

Off-policy algorithms, like Q-learning, use the Q-approximator to select the next action during an episode. In the environment used in this paper, the agent uses the DQN to

predict the next partition given the current partition through row-wise arg-max: $\hat{\mathbf{z}}_u^{[t]} = \arg\max_i \hat{\mathbf{Z}}_{ui}^{[t+1]}$. In training, the agent is configured to select random actions at first and then gradually select actions using its DQN with increasing frequency as training continues. Selecting random actions allows the agent to explore the action space early on to build a buffer of "experience" on which to draw for updating its parameters. Once a sufficient buffer is accumulated, the agent exploits its experience by selecting actions according to its learned parameters. This progression from exploration to exploitation is controlled by an exponential decay rate.

The full training evolution consists of the agent repeatedly interacting with the environment then updating the parameters of its DQN by sampling from its replay buffer. Specifically, as the agent interacts with the environment and at each step in a given episode it updates its replay buffer with its gained experiences ($s_t$, $a_t$, $r_t$, $s_{t+1}$, $d_t$). At the end of an episode, the agent samples a batch of experiences from its buffer and computes the following loss:

$$\text{loss} = \sum_{u \in \mathcal{V}} \left[ \left( r_t + \gamma \, (1 - d_t) \max_a Q_u(s_{t+1}, a; \tilde{\mathbf{W}}) \right) - Q_u(s_t, a_t; \mathbf{W}) \right]^2 \tag{8}$$

where $d_t = 1$ if $s_t$ is a terminal state and 0 otherwise. This loss is averaged over every sample in the batch, then the gradient is propagated back through the DQN. Also, $\gamma$ is a factor that discounts the value of future rewards. A lower discount factor will allow the agent to prioritize the change in modularity at intermediate steps while a higher discount factor allows the agent to prioritize the final modularity.

Notice in the loss function that $Q(\cdot, \cdot\,; \tilde{\mathbf{W}})$ is the *target* $Q$ function, a necessary component for the temporal difference minimization objective. The parameters $\tilde{\mathbf{W}}$ are initialized to $\mathbf{W}$ then periodically updated using Polyak averaging (Li et al. 2023): $\tilde{\mathbf{W}} = \tau \mathbf{W} + (1 - \tau)\tilde{\mathbf{W}}$ where $0 \leq \tau < 1$ is a small constant. This helps stabilize the temporal difference learning process.

To further improve stability, the Dueling DQN (Wang et al. 2016) architecture is implemented.

This is a modification of the original DQN algorithm which mitigates the problem of over-estimating the state-value function. The full neural network used in this paper consists of two GCN layers followed by a fully-connected prediction head.

The output dimension of the network represents the maximum number of communities that the agent will be able to find. The true number of communities in a graph is typically unknown in practice, so it is necessary to set the output dimension of the DQN to a reasonable overestimate of the true number. Experimentation suggests that the agent will learn to use as many communities as needed (up to the maximum number) in order to maximize its reward signal. If the inferred number of communities after training is equal to the output dimension, one can re-train with a higher output dimension. This is a more flexible approach than having to compare the results of multiple solutions, each with a different fixed number of communities.

## 2.3 Limitations

Although modularity is a commonly used objective for community detection (see Blondel et al. 2008, Traag et al. 2019, Tsitsulin et al. 2023), it is known to have issues. In Fortunato & Barthélemy (2007), a resolution-limit is identified for modularity maximization: an algorithm that optimizes modularity will fail to recover communities that contain fewer than $\sqrt{2m}$ nodes. In addition to this, Peixoto (2023) argues that modularity maximization has a tendency to overfit, cannot find communities of different sizes, cannot recover disassortative (non-modular) structures, and has a degenerate solution landscape.

Several alternatives exist which may be robust to these shortcomings. The Constant Potts Model (Traag et al. 2011) was proposed to address the resolution limit. The Cut Ratio, which is the objective function of the Minimum Cut problem (Dhillon et al. 2004), allows partitions to be identified spectrally. The Stochastic Block Model likelihood functions

(Lee & Wilkinson 2019, Karrer & Newman 2011) can be used to infer both assortative and disassortative community structures. Many SBM variants have been proposed to address different assumptions of the underlying generative model of a graph and to address challenges with distributional skew or statistical consistency.

Despite the limitations of modularity, it remains a popular choice for community detection. Thus it is chosen to be the backbone of the reward function in this RL environment. Optimizing alternative metrics of partition quality is left as a direction for future work.

# 3 Data

To evaluate this framework, three real-world citation networks are considered. The Cora McCallum et al. (2000), Citeseer Giles et al. (1998), and Pubmed Sen et al. (2008) datasets are all created with individual publications as nodes and co-authorship relations as edges. Each node is attributed with vector representations of text from the publication (e.g., keywords). These datasets also provide node labels according to the general category of the publication. Summary statistics for each graph are provided in table 1.

| Name | Num. Nodes | Num. Edges | Feat. Dim. | Num. Categories |
|---|---|---|---|---|
| Cora | 2,708 | 10,556 | 1,433 | 7 |
| Citeseer | 3,327 | 9,228 | 3,703 | 6 |
| Pubmed | 19,717 | 88,651 | 500 | 3 |
| SBM | 221 | 7250 | NA | 5 |

Table 1: Graph dataset summaries.

Due to limited computational resources, the node attributes are decomposed using Truncated Singular Value Decomposition (Halko et al. 2010). Truncated SVD is applied independently

to each dataset, reducing the dimensionality to 25. Another measure taken in the interest of computational limitations is to reduce the number of nodes in the Pubmed graph by removing (iteratively) any nodes with degree less than 4. The trimmed graph has 6,470 nodes (with degrees of at least 4) and 54,889 edges.

Also included in this evaluation is a synthetic graph generated by a Stochastic Block Model. The SBM is defined with a planted partition, where the probability of an edge within communities is 0.55 and the probability of an edge between communities is 0.05. The sizes of the 5 communities in this graph are sampled from a Poisson distribution. No node attributes are generated.

# 4 Results

| | Synthetic | | Cora | | Citeseer | | Pubmed | |
|---|---|---|---|---|---|---|---|---|
| | Avg. | Max. | Avg. | Max. | Avg. | Max. | Avg. | Max. |
| Leiden | 0.312 $\pm$0.0 | 0.312 | 0.427 $\pm$0.001 | 0.428 | 0.459 $\pm$0.001 | 0.46 | 0.389 $\pm$0.002 | 0.389 |
| Louvain | 0.312 $\pm$0.0 | 0.312 | 0.423 $\pm$0.001 | 0.424 | 0.458 $\pm$0.001 | 0.459 | 0.384 $\pm$0.004 | 0.388 |
| LPA | 0.309 $\pm$0.004 | 0.312 | 0.35 $\pm$0.003 | 0.357 | 0.39 $\pm$0.003 | 0.395 | 0.352 $\pm$0.01 | 0.363 |
| RL | 0.302 $\pm$0.005 | 0.309 | 0.338 $\pm$0.006 | 0.348 | 0.311 $\pm$0.005 | 0.321 | 0.352 $\pm$0.006 | 0.363 |

Table 2: Average (with standard deviations) and maximum modularity score for each dataset over ten evaluation runs of each algorithm.

Hyperparameter configurations for each dataset are provided in Appendix A. For benchmarking, the proposed framework is compared against the Leiden, Louvain, and Lapel Propagation (LPA) algorithms.

Table 2 shows the average and maximum modularity scores for each approach on each

| | Synthetic | Cora | Citeseer | Pubmed |
|---|---|---|---|---|
| Leiden | 5 ±0.0 | 106.1 ±0.994 | 470.5 ±1.716 | 17 ±1.491 |
| Louvain | 5 ±0.0 | 104.1 ±1.197 | 469.6 ±2.503 | 17.5 ±1.354 |
| LPA | 4.7 ±1.135 | 369.3 ±11.146 | 796.7 ±7.514 | 165.2 ±20.612 |
| RL | 4.3 ±0.458 | 5.6 ±0.663 | 7 ±0.0 | 7 ±0.0 |

Table 3: Average (with standard deviations) number of communities inferred for each dataset over ten evaluation runs of each algorithm.

dataset over ten evaluation episodes. The approach proposed in this paper is referred to as simply "RL". Notice that the RL approach performs decently on all datasets, roughly in line with LPA. The Leiden and Louvain algorithms significantly outperform RL and LPA on the Cora and Citeseer datasets.

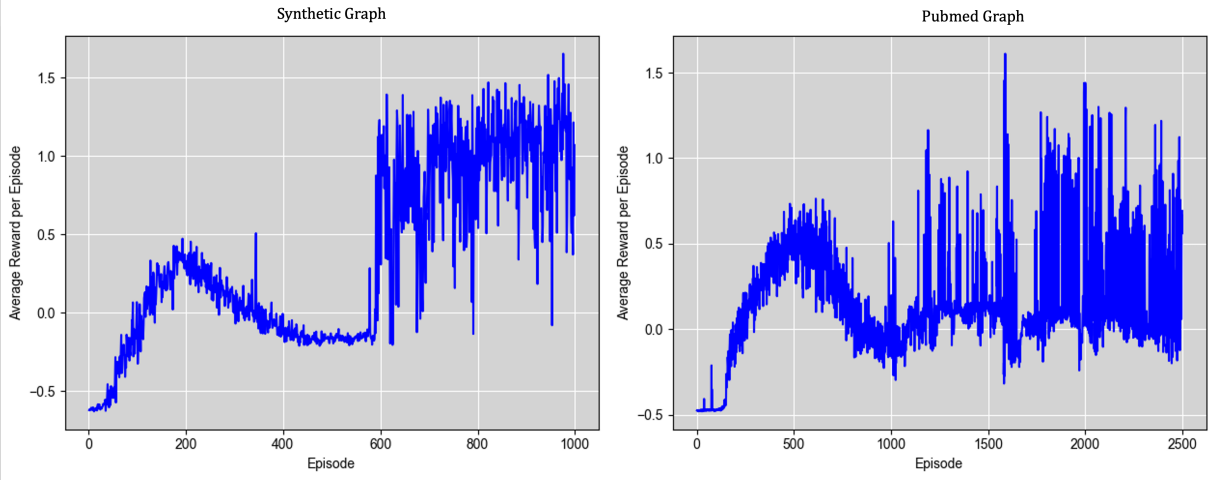


Figure 2: Average reward over each training episode for the synthetic dataset (left) and Pubmed dataset (right).

Table 3 shows the average number of communities found by each approach on each dataset. Notice that the number of communities found by the RL approach is significantly lower than the comparison approaches on real-world graphs. This could explain the difference

in performance in table 2. It might be that modularity is optimized at a larger number of communities. However, due to computational (and time) limitations in the implementation of the RL approach, the maximum number of communities possible for it to find (the output dimension of the DQN) has been fixed at 10.

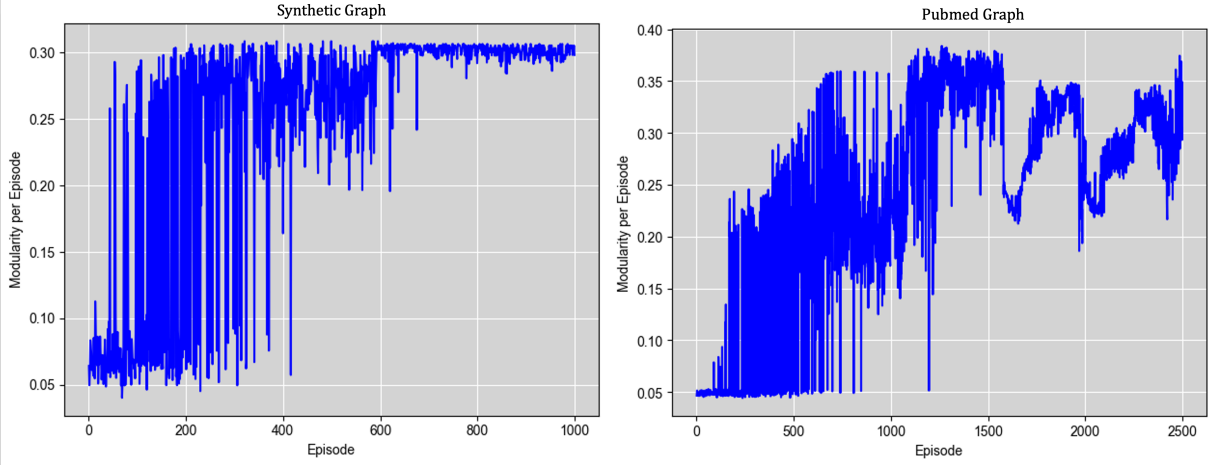


Figure 3: Final modularity score after each training episode for the synthetic dataset (left) and Pubmed dataset (right).

While the RL agent appears to find much fewer communities in the real-world graphs than the comparison algorithms, it is more closely aligned with the ground-truth number of communities (see table 1). Although this result is interesting, it should be noted that the ground-truth community labels are not necessarily guaranteed to represent the true community structure. The labels are applied as the data are collected and are not intrinsically linked to the generating process of each graph.

Figure 2 shows the average reward trace over each training episode for the RL approach on the synthetic dataset and the Pubmed dataset. Figure 3 shows the final modularity trace over each training episode for the RL approach on the synthetic and Pubmed datasets. Both figures suggest that the RL agent does indeed move toward some sort of local optimum during training, even if it is not the optimum found by the comparison algorithms. Figure 4 shows the predicted partitions from the RL agent on the synthetic and Pubmed datasets.

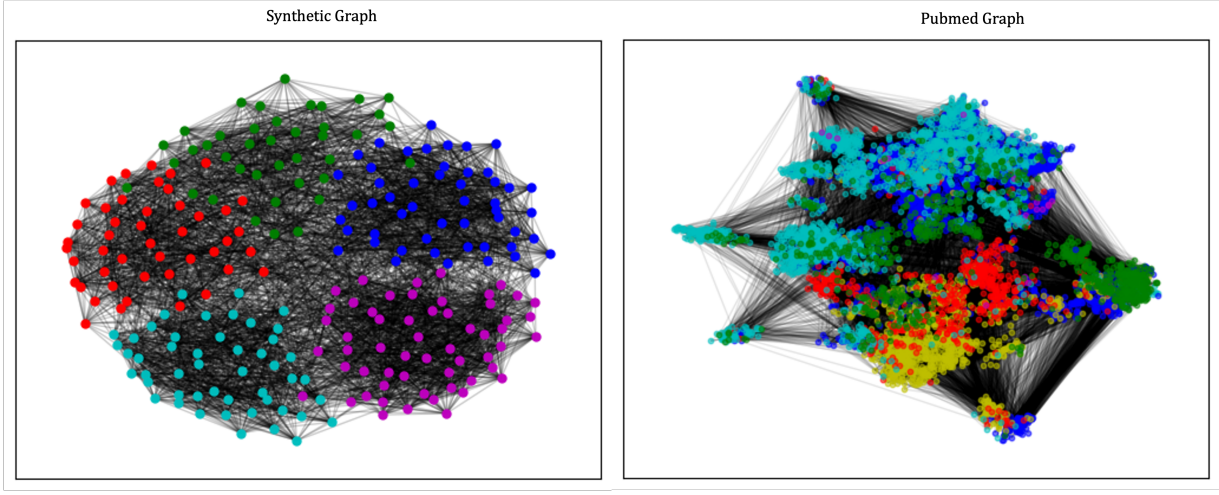


Figure 4: Community partitions for the synthetic dataset (left) and pubmed dataset (right) given by node colors.

Visually, there does seem to be some community structure recovered by the predicted partition.

# 5 Related Work

In Paim et al. (2020), the authors use a Multi-Agent Reinforcement Learning strategy to optimize modularity by assigning each node to an agent and allowing agents to decide whether to form a community with one-another. Modularity is also optimized using reinforcement learning in Wei et al. (2024).

In Costa (2021), Zhao et al. (2025), and Costa & Ralha (2023) Reinforcement Learning is applied to the problem of community detection on dynamic social networks.

# 6 Conclusion

This paper explores a Reinforcement Learning solution to modularity optimization for the task of community detection. The environment defined is similar to the Label Propagation

algorithm and the agent proposed is motivated by Q-learning. Optimizing alternative metrics, such as Stochastic Block Model likelihood, is a direction for future research in this area. The results of the proposed framework do not exceed state-of-the-art approaches, but do offer an interesting starting point for future work.

# References


Blondel, V. D., Guillaume, J.-L., Lambiotte, R. & Lefebvre, E. (2008), 'Fast unfolding of communities in large networks', *Journal of Statistical Mechanics: Theory and Experiment* **2008**(10), P10008.

**URL:** *http://dx.doi.org/10.1088/1742-5468/2008/10/P10008*

Costa, A. R. (2021), 'Towards modularity optimization using reinforcement learning to community detection in dynamic social networks'.

**URL:** *https://arxiv.org/abs/2111.15623*

Costa, A. R. & Ralha, C. G. (2023), 'Ac2cd: An actor–critic architecture for community detection in dynamic social networks', *Knowledge-Based Systems* **261**, 110202.

**URL:** *https://www.sciencedirect.com/science/article/pii/S0950705122012989*

Dhillon, I. S., Guan, Y. & Kulis, B. (2004), Kernel k-means: spectral clustering and normalized cuts, *in* 'Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining', pp. 551–556.

Fortunato, S. & Barthélemy, M. (2007), 'Resolution limit in community detection', *Proceedings of the National Academy of Sciences* **104**(1), 36–41.

**URL:** *http://dx.doi.org/10.1073/pnas.0605965104*

Giles, C. L., Bollacker, K. D. & Lawrence, S. (1998), Citeseer: an automatic citation indexing system, *in* 'Proceedings of the Third ACM Conference on Digital Libraries', DL

'98, Association for Computing Machinery, New York, NY, USA, p. 89–98.

**URL:** *https://doi.org/10.1145/276675.276685*

Halko, N., Martinsson, P.-G. & Tropp, J. A. (2010), 'Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions'.

**URL:** *https://arxiv.org/abs/0909.4061*

Karrer, B. & Newman, M. E. J. (2011), 'Stochastic blockmodels and community structure in networks', *Phys. Rev. E* **83**, 016107.

**URL:** *https://link.aps.org/doi/10.1103/PhysRevE.83.016107*

Kipf, T. N. & Welling, M. (2017), 'Semi-supervised classification with graph convolutional networks'.

**URL:** *https://arxiv.org/abs/1609.02907*

Lee, C. & Wilkinson, D. J. (2019), 'A review of stochastic block models and extensions for graph clustering', *Applied Network Science* **4**(1), 1–50.

Li, X., Yang, W., Liang, J., Zhang, Z. & Jordan, M. I. (2023), 'A statistical analysis of polyak-ruppert averaged q-learning'.

**URL:** *https://arxiv.org/abs/2112.14582*

McCallum, A., Nigam, K., Rennie, J. D. M. & Seymore, K. (2000), 'Automating the construction of internet portals with machine learning', *Information Retrieval* **3**, 127–163.

**URL:** *https://api.semanticscholar.org/CorpusID:349242*

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. (2013), 'Playing atari with deep reinforcement learning'.

**URL:** *https://arxiv.org/abs/1312.5602*

Newman, M. E. J. (2006), 'Modularity and community structure in networks', *Proceedings*

*of the National Academy of Sciences* **103**(23), 8577–8582.

**URL:** *http://dx.doi.org/10.1073/pnas.0601602103*

Newman, M. E. J. & Girvan, M. (2004), 'Finding and evaluating community structure in networks', *Physical Review E* **69**(2).

**URL:** *http://dx.doi.org/10.1103/PhysRevE.69.026113*

Paim, E. C., Bazzan, A. L. C. & Chira, C. (2020), Detecting communities in networks: a decentralized approach based on multiagent reinforcement learning, *in* '2020 IEEE Symposium Series on Computational Intelligence (SSCI)', pp. 2225–2232.

Peixoto, T. P. (2023), *Descriptive vs. Inferential Community Detection in Networks: Pitfalls, Myths and Half-Truths*, Elements in the Structure and Dynamics of Complex Networks, Cambridge University Press, chapter 3.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Gallagher, B. & Eliassi-Rad, T. (2008), Collective classification in network data, *in* 'The AI Magazine'.

**URL:** *https://api.semanticscholar.org/CorpusID:62016134*

Traag, V. A., Van Dooren, P. & Nesterov, Y. (2011), 'Narrow scope for resolution-limit-free community detection', *Physical Review E* **84**(1).

**URL:** *http://dx.doi.org/10.1103/PhysRevE.84.016114*

Traag, V. A., Waltman, L. & van Eck, N. J. (2019), 'From louvain to leiden: guaranteeing well-connected communities', *Scientific Reports* **9**(1).

**URL:** *http://dx.doi.org/10.1038/s41598-019-41695-z*

Tsitsulin, A., Palowitch, J., Perozzi, B. & Müller, E. (2023), 'Graph clustering with graph neural networks'.

**URL:** *https://arxiv.org/abs/2006.16904*

Wang, Y., Huang, H., Feng, C. & Liu, Z. (2015), Community detection based on minimum-cut

graph partitioning, *in* 'International Conference on Web-Age Information Management', Springer, pp. 57–69.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M. & de Freitas, N. (2016), 'Dueling network architectures for deep reinforcement learning'.
**URL:** *https://arxiv.org/abs/1511.06581*

Wei, W., Meng, Y. & Li, Q. (2024), A novel reinforcement learning multi-objective community detection algorithm with $\epsilon$-gradient-greedy strategy, *in* '2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC)', pp. 3683–3688.

Zhao, H., Huang, H., Cui, Z., Li, J. & Liu, J. (2025), 'Multiagent reinforcement learning aided multiobjective evolutionary algorithm with local higher-order information for community detection', *IEEE Transactions on Computational Social Systems* pp. 1–13.

Zhu, X. & Ghahramani, Z. (2002), 'Learning from labeled and unlabeled data with label propagation', *ProQuest number: information to all users* .

# A Configuration

For all datasets, the environment is configured to run for a maximum of 100 steps per episode (truncation). If the change in modularity is less than $\epsilon$ for 10 consecutive steps, the episode is terminated (convergence). For the synthetic and Citeseer environments, $\epsilon = 0.003$. For the Cora environment, $\epsilon = 0.005$. For Pubmed, $\epsilon = 0.001$.

For the synthetic graph, the agent is trained for 1,000 episodes. The exploration rate $\eta$ starts at 0.999 then decays to 0.0001 according to the schedule $\eta_{t+1} = \frac{\eta_t}{0.00005t+1}$. The agent's memory buffer holds 1,024 experiences. The hidden dimension in the DQN is 200 and the output dimension is 10. The discount factor in the loss function is $\gamma = 0.9$. The agent is

trained with batches of size 16 for 9 iterations at a time and a learning rate of 0.0001. The target network is updated after every 3 training iterations. The Polyak averaging coefficient for updating the target network is $\tau = 0.01$.

For the Cora dataset, the agent is trained for 2,000 episodes. The exploration rate $\eta$ starts at 0.999 then decays to 0.0001 according to the schedule $\eta_{t+1} = \frac{\eta_t}{0.00001t+1}$. The agent's memory buffer holds 2,048 experiences. The hidden dimension in the DQN is 400 and the output dimension is 10. The discount factor in the loss function is $\gamma = 0.95$. The Polyak averaging coefficient for updating the target network is $\tau = 0.05$.

For the Citeseer dataset, the agent is trained for 2,000 episodes. The exploration rate $\eta$ starts at 0.999 then decays to 0.0001 according to the schedule $\eta_{t+1} = \frac{\eta_t}{0.00001t+1}$. The agent's memory buffer holds 2,048 experiences. The hidden dimension in the DQN is 400 and the output dimension is 10. The discount factor in the loss function is $\gamma = 0.99$. The Polyak averaging coefficient for updating the target network is $\tau = 0.05$.

For the Pubmed dataset, the agent is trained for 2,500 episodes. The exploration rate $\eta$ starts at 0.999 then decays to 0.0001 according to the schedule $\eta_{t+1} = \frac{\eta_t}{0.00001t+1}$. The agent's memory buffer holds 2,048 experiences. The hidden dimension in the DQN is 400 and the output dimension is 10. The discount factor in the loss function is $\gamma = 0.9$. The Polyak averaging coefficient for updating the target network is $\tau = 0.01$.

The Python implementation is available at https://github.com/willarliss/RL-Community-Detection.