EXPLORING RL TECHNIQUES FOR SPATIAL REGIONALIZATION

1 ABSTRACT

In this project, we look into spatial regionalization, a crucial area in geographic information systems (GIS), urban planning where the main goal is to divide a given space into regions that are not only next to each other but also similar in aspects like how many people live there, how much money they make, or other important characteristics. People have tried to solve this by using smart guessing methods and ways to cut through graphs to group these areas together. the information we're dealing with changes all the time and can get pretty complicated. This makes it really hard for the usual methods to keep up, This project introduces a novel reinforcement learning (RL) approach, leveraging Deep Q-Networks (DQN), to address the p-regions problem, aiming to enhance adaptability, efficiency, and scalability in regionalization tasks. By incorporating RL, our solution dynamically learns optimal partitioning strategies from data, adapting to various constraints and objectives. We utilized the dataset previously employed in the "Region2Vec" study to ensure a relevant and challenging testing ground for our method. Our approach is benchmarked against parameters and evaluation measures derived from the Region2Vec results, allowing for a direct comparison that showcases the advantages of applying RL in spatial regionalization.

2 PROBLEM STATEMENT

Spatial regionalization, the task of clustering spatial units into meaningful regions, is crucial for various applications in geography, urban planning, and related fields. The p-regions problem, which seeks to partition a space into a predefined number of regions while optimizing certain objective functions and ensuring spatial contiguity, poses significant challenges. Traditional and heuristic methods, although useful, often fall short in scalability and adaptability. Recent initiatives have leveraged machine learning to overcome the challenges of spatial regionalization, yet the application of reinforcement learning in this area is still in its infancy. Recent work has started to use machine learning to overcome these challenges, but the use of reinforcement learning in this area hasn't been fully explored yet. Recognizing these challenges, our project proposes a reinforcement learning-based solution, specifically employing a Deep Q-Network (DQN) framework, to tackle the p-regions problem. This approach enables dynamic learning from the environment, offering a flexible, data-driven method that iteratively improves partitioning strategies through interaction with spatial data. The objective is to demonstrate that an RL-based model can outperform traditional methods in terms of adaptability, efficiency, and the ability to handle complex, multi-objective optimization tasks in spatial regionalization. Our study leverages the dataset from the Region2Vec research to ensure a robust evaluation, comparing our results against established benchmarks to validate the effectiveness and innovation of our RL approach in solving the p-regions problem.

3 LITERATURE SURVEY

Community Detection on Spatial Networks Using Graph Embedding [1] introduces an unsupervised Graph Convolutional Network-based method for detecting communities in spatial networks. This approach generates embeddings for regions with common attributes and spatial interactions, utilizing clustering algorithms for community detection. It balances attribute similarities and spatial interactions within communities, offering improved performance over traditional methods.

P-Regions with User-Defined Constraint presents a solution to the generalized spatial regionalization problem, PRUC, which partitions spatial areas into homogeneous regions while considering user-defined constraints. They [2] propose the GSLO algorithm, a parallel stochastic approach that significantly outperforms existing algorithms in speed, quality, and scalability for handling large datasets.

In their 2019 study [3], Di Jin and colleagues developed a novel community detection approach using Graph Convolutional Networks (GCNs) to analyze networks based on both node connections and attributes. Their method promises more accurate community clustering by integrating these dual insights, showcasing potential across various fields like social and biological networks.

4 METHODOLOGY

4.1 Loading and Preprocessing the Feature Matrix: The feature matrix is loaded from a CSV file ‘feature_matrix_f1.csv’ using ‘pandas’. The ‘MinMaxScaler’ from ‘sklearn.preprocessing’ is used to scale the feature values to a range between 0 and 1. This normalization helps in speeding up the training process and improving the model's performance. The input dimension for the DQN model is determined based on the number of features in the scaled feature matrix.

4.2 Creating the DQN Model

A function ‘create_dqn’ is defined to create a sequential model with two hidden layers of 64 neurons each, using ReLU activation, and an output layer with a linear activation function. The number of neurons in the output layer corresponds to the number of possible actions. The model is compiled with an Adam optimizer and a mean squared error loss function.

4.3 Defining Training Hyperparameters

Several hyperparameters are defined for training, including the number of episodes, maximum steps per episode, batch size, discount factor, initial epsilon for the epsilon-greedy policy, minimum epsilon, and epsilon decay rate.

4.4 Simulating the Environment for Action Taking

A function ‘take_action’ is defined to simulate the environment's response to an agent's action. It randomly selects the next state and calculates the reward based on the difference between the current and next states. A done flag is also randomly generated to simulate episode termination.

4.4.1 State

The ‘state’ is a vector from the ‘scaled_feature_matrix’, which is a normalized version of the feature matrix loaded from the CSV file. The ‘state_index’ is used to select the current state from the ‘scaled_feature_matrix’. The ‘next_state’ is randomly chosen as a new state from the ‘scaled_feature_matrix’ to simulate the transition after taking an action.

4.4.2 Action

The action space size is determined by ‘output_dim’. During training, the action can be chosen randomly or by using the DQN model's prediction, depending on the epsilon-greedy policy. In the ‘take_action’ function, the action is passed as an argument but not used directly, the function is designed to be compatible with environments where actions would affect the transition to the next state and the reward received.

4.4.3 Reward

In the `take_action` function, the reward is calculated based on the mean absolute difference between the current state and the next state. If this difference exceeds a certain threshold, the agent receives a negative reward (-1); otherwise, it receives a positive reward (+1). The reward is then weighted by a factor that increases linearly with the step number within the episode, to give later actions more influence as the episode progresses.

4.5 Training the DQN Model

The training loop iterates over episodes and steps within each episode. Actions are chosen either randomly or by predicting the best action using the DQN model. The state, action, reward, next state, and done flag are stored in a replay buffer. The epsilon value is decayed after each episode to gradually shift from exploration to exploitation.

4.5.1 Objective function

In the context of training a DQN model for reinforcement learning, the objective function, often referred to as the loss function, plays a crucial role in the learning process.The loss function measures the difference between the predicted Q-values by the model and the target Q-values, which are estimated based on the Bellman equation. The specific loss function used in this project is the Mean Squared Error (MSE) loss.

4.5.2 Exploration-Exploitation Balance (Epsilon-Greedy Policy)

The purpose of epsilon in the provided code, and more broadly in reinforcement learning (RL), is to control the balance between exploration and exploitation during the training of an agent. This concept is commonly implemented through an epsilon-greedy policy. Here's how it works and why it's important:

Exploration vs. Exploitation

- Exploration refers to the agent trying out different actions to discover new states and learn more about the environment. It's crucial for finding potentially better strategies that have not been tried yet.
- Exploitation involves the agent using its current knowledge to take actions that it believes will yield the highest reward. This is important for leveraging what the agent has already learned to improve its performance.

Epsilon in the Provided Code

- In the provided code, epsilon is initialized to 1.0, meaning the agent starts by exploring almost entirely, epsilon = 1.0

- Over time, epsilon is decayed (multiplied by epsilon_decay) after each episode, gradually reducing the probability of taking random actions and increasing the reliance on the policy learned by the DQN model:
  epsilon_decay = 0.995
  epsilon = max(epsilon_min, epsilon * epsilon_decay)

- This ensures that the agent explores sufficiently in the early stages of training when its knowledge about the environment is limited. As training progresses and the agent learns more about the environment, the focus shifts more towards exploiting this knowledge to maximize rewards. The epsilon_min parameter sets a lower bound on the value of epsilon to ensure that there's always some degree of exploration:
  epsilon_min = 0.01

- This balance is crucial for effective learning, as it allows the agent to discover new strategies while also making use of its accumulated knowledge to improve its decision-making over time.

4.6 Evaluating the DQN Model with Metrics

The model is evaluated over a number of episodes, and metrics such as median inequality, median cosine similarity, and median Euclidean distance similarity between states are calculated. These metrics provide insights into the diversity and distribution of states visited by the agent during evaluation.

4.7 Visualization comparison of Evaluation Metrics

The results of the DQN model are compared with static "Region2Vec" results using bar charts for each metric. A trend graph is plotted to show the total reward obtained by the DQN model over each evaluation episode, providing insights into the model's performance and learning progress over time.

5 RESULTS & PERFORMANCE EVALUATION

5.1 Comparison of Evaluation Metrics with the Reg2Vec baseline

The bar charts provided offer a comparison between a Deep Q-Network (DQN) and Region2Vec across three metrics. DQN exhibits significantly lower median inequality than Region2Vec, indicating that DQN tends to create communities with more homogeneous attributes. This suggests a stronger similarity among nodes within the same community for DQN. However, Region2Vec has a higher median cosine similarity, meaning that it is more effective at grouping nodes with similar attributes, as it emphasizes attribute closeness within communities.

The median Euclidean distance similarity further distinguishes the two methods. Region2Vec shows a lower median Euclidean distance between nodes within communities, indicating that it creates tighter and possibly more geographically cohesive communities. This contrast suggests that Region2Vec may be more suitable for applications where the closeness of node attributes in the feature space is a priority.

The line graph reflecting the DQN's performance over time shows considerable variability in the total rewards per episode. The average reward over 20 episodes provides a baseline for DQN's performance but lacks a direct comparison to Region2Vec, as the latter's performance is not measured in terms of rewards. Nonetheless, the fluctuations in DQN's rewards indicate the learning process's exploratory nature and the dynamic challenges of the environment it is interacting with. In choosing between DQN and Region2Vec, the decision would hinge on the specific requirements of the community detection task,
whether it prioritizes attribute homogeneity within communities (favoring DQN) or the attribute closeness of nodes (favoring Region2Vec).

```python
# Evaluation Functions
def calculate_cosine_similarity(v1, v2):
    if np.all(v1 == 0) or np.all(v2 == 0):
        return 0
    return np.dot(v1, v2) / (norm(v1) * norm(v2))

def calculate_median_cosine_similarity(all_states):
    similarities = []
    for i in range(len(all_states)):
        for j in range(i + 1, len(all_states)):
            similarities.append(calculate_cosine_similarity(all_states[i], all_states[j]))
    return np.median(similarities) if similarities else 0

def calculate_median_euclidean_distance_similarity(all_states):
    distances = []
    for i in range(len(all_states)):
        for j in range(i + 1, len(all_states)):
            distances.append(euclidean(all_states[i], all_states[j]))
    return np.median(distances) if distances else 0

def calculate_median_inequality(all_states):
    variances = [np.var(state) for state in all_states]
    return np.median(variances) if variances else 0
```

Fig 5.1(a) Evaluation Metrics

```python
def take_action(state_index, action, step, max_steps_per_episode):
    next_state_index = np.random.randint(0, scaled_feature_matrix.shape[0])
    next_state = scaled_feature_matrix[next_state_index]
    difference_threshold = 0.5
    difference = np.abs(state - next_state).mean()
    if difference > difference_threshold:
        reward = -1
    else:
        reward = 1
    reward_weight = 1 + (step / max_steps_per_episode)
    weighted_reward = reward * reward_weight
    done = np.random.rand() < 0.1
    return next_state, weighted_reward, done, next_state_index
```

Fig 5.1(b) Reward Function

```python
for episode in range(num_episodes):
    state_index = np.random.randint(0, scaled_feature_matrix.shape[0])
    state = scaled_feature_matrix[state_index]
    episode_reward = 0
    for step in range(max_steps_per_episode):
        if np.random.rand() < epsilon:
            action = np.random.randint(output_dim)
        else:
            action = np.argmax(dqn_model.predict(state.reshape(1, -1))[0])
        next_state, reward, done, next_state_index = take_action(state_index, action, step, max_steps_per_episode)
        replay_buffer.append((state, action, reward, next_state, done))
        state = next_state
        state_index = next_state_index
        episode_reward += reward
        if done:
            break
    epsilon = max(epsilon_min, epsilon * epsilon_decay)
    print(f"Episode {episode + 1}: Reward = {episode_reward}")
```

Fig 5.1(c) Simulation loop

```python
# Training hyperparameters
num_episodes = 50
max_steps_per_episode = 50
batch_size = 32
gamma = 0.99
epsilon = 1.0
epsilon_min = 0.01
epsilon_decay = 0.995
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
loss_fn = tf.keras.losses.MeanSquaredError()
replay_buffer = []
```

Fig 5.1(d) Training parameters

```
--- Evaluation Results ---
Average reward over 20 evaluation episodes: 16.375
Median Inequality: 0.09408084565045066
Median Cosine Similarity: 0.9468790097698856
Median Euclidean Distance Similarity: 0.42070397897575157

--- Region2Vec Results ---
Median Inequality: 0.351
Median Cosine Similarity: 0.973
Median Euclidean Distance Similarity: 0.250
```

Fig 5.1(e) Evaluation

5.2 Visualization of the Evaluation Metrics:

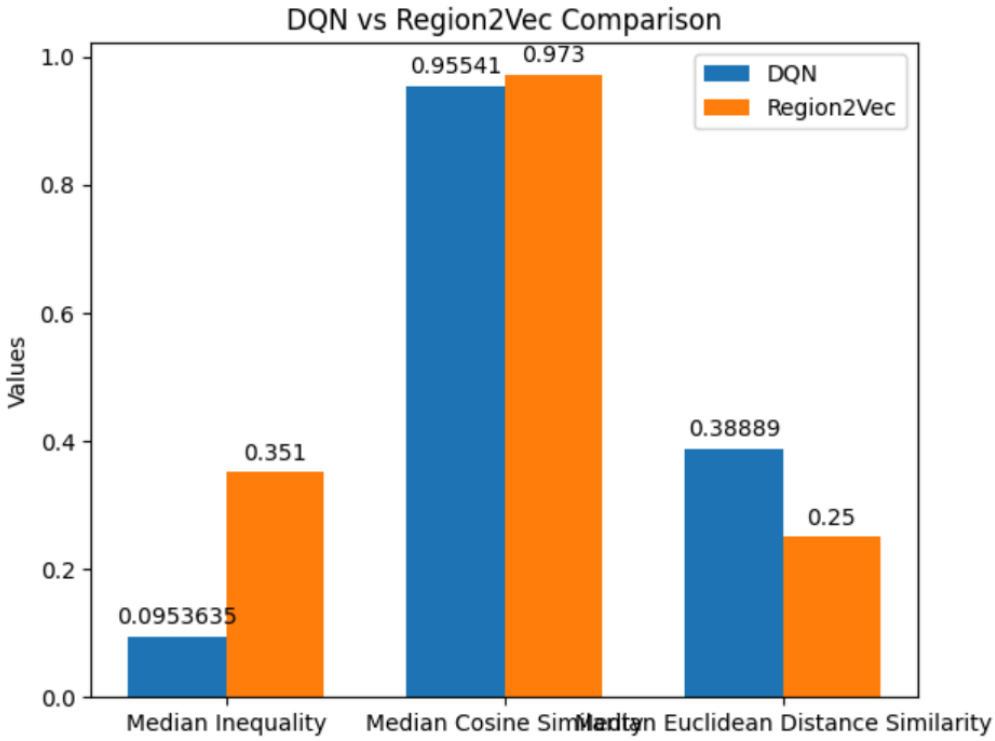

Fig 5.2(a) DQN vs Reg2Vec using a Bar Graph

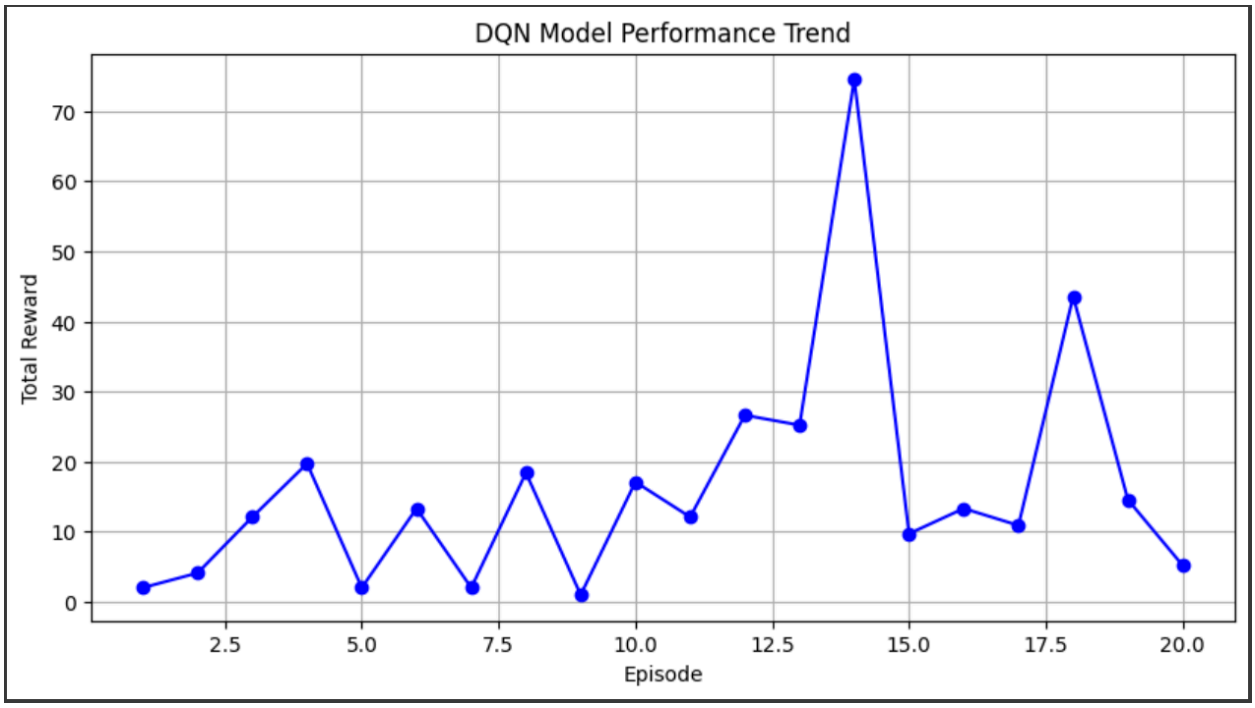

Fig 5.2(b) DQN Model Performance Trend Graph for 20 Episodes

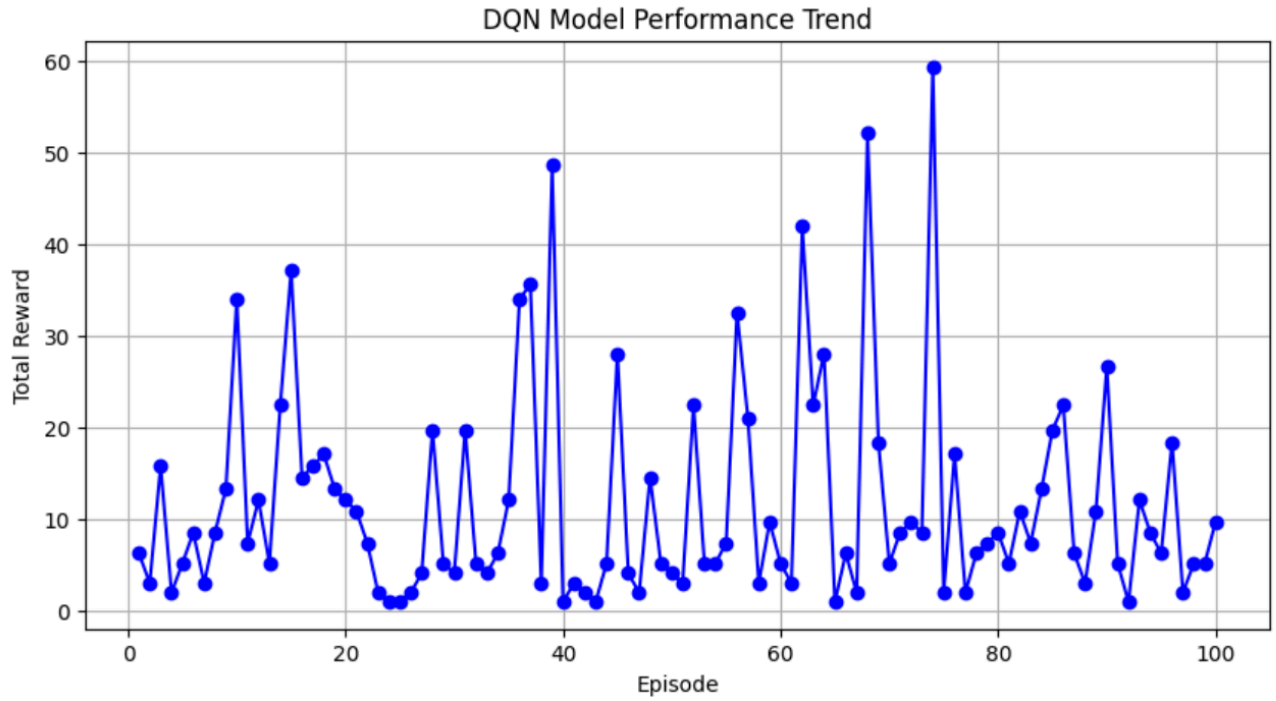

Fig 5.2(b) DQN Model Performance Trend Graph for 100 Episodes

5.3 Quantitative Metrics Comparison

- Median Inequality:
  reg2vec: 0.351 vs. RL: 0.0937.
  Lower values indicate better equality among clusters. RL has a lower median inequality, suggesting more balanced clusters.

- Median Cosine Similarity:
  Reg2vec: 0.973 vs. RL: 0.945.
  Higher values imply better similarity within clusters. Model 1 has a slightly higher score, indicating potentially better intra-cluster similarity.

- Median Euclidean Distance Similarity:
  reg2vec: 0.250 vs. RL: 0.431.
  Higher values imply better similarity within clusters. RL has a significantly higher score, suggesting better intra-cluster similarity.

- Considering the available metrics, it's challenging to definitively declare one model as superior. However, based on the provided metrics, RL seems to perform better overall due to its lower median inequality and higher median cosine similarity and median Euclidean distance similarity scores. These metrics collectively suggest that RL produces more balanced and internally similar clusters compared to reg2vec.

6 FUTURE WORKS

Enhanced State Representation: Explore the use of graph-based representations to capture the complex spatial interactions between regions more effectively. To extend the algorithm to incorporate additional datasets such as Flow Matrix and Spatial Distance Matrix csv files.
Advanced Reward Function: Develop a more sophisticated reward function that not only considers attribute similarity and spatial interactions but also takes into account other factors.
Evaluation and Validation: Develop a comprehensive evaluation framework that includes a variety of metrics to assess the quality of the detected communities from different perspectives.

7 CONCLUSION

In conclusion, the comparison between the Deep Q-Network (DQN) and Region2Vec algorithms in community detection highlights their distinct characteristics and performance metrics. DQN shows a tendency to create communities with more homogeneous attributes, as indicated by its lower median inequality compared to Region2Vec. However, Region2Vec excels in grouping nodes with similar attributes, demonstrated by its higher median cosine similarity. Moreover, Region2Vec tends to create tighter and more geographically cohesive communities, as reflected in its lower median Euclidean distance similarity. To further validate these findings and draw more definitive conclusions about the algorithm's performance, it is essential to experiment with a wider range of datasets.

8 REFERENCES

[1] Li, P., Ruan, X., Zhu, X., & Chai, J. (2019). A Regionalization Navigation Method Based on Deep Reinforcement Learning. In Proceedings of the IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC 2019) (pp. 803–807). Beijing, China: IEEE

[2] Yongyi Liu, Ahmed Mahmood, Amr Magdy, and Sergio J. Rey. 2021. PRUC: P-Regions with User-Defined Constraint. In Proceedings of the VLDB Endowment, Vol. 15, No. 3, (2021), 491–503

[3] Yunlei Liang, Jiawei Zhu, Wen Ye, and Song Gao. 2022. Region2Vec: Community Detection on Spatial Networks Using Graph Embedding with Node Attributes and Spatial Interactions. In Proceedings of the ACM SIGSPATIAL 2022, Article 39, (2022), 1–4

[4] https://www.geeksforgeeks.org/implementing-deep-q-learning-using-tensorflow/

[5] https://www.tensorflow.org/agents/tutorials/1_dqn_tutorial

[6] https://lukasschwarz.de/dqn

[7] https://www.tensorflow.org/agents/tutorials/0_intro_rl