Analyzing Strategies for Voronoi Area Game
CS460-2024 Group Project: Endterm Report

Samir Dileep
School of Biological Sciences
National Institute of Science Education and Research
Jatni, Odisha-752050

Sandipan Samanta
School of Mathematical Sciences
National Institute of Science Education and Research
Jatni, Odisha-752050

Abstract

This project aims to design and train a Machine Learning model to master the 2-player Voronoi Area Game in 2D. An in-house replica of the game was made in python and the training process was run on this interface. After training, we expect the model to make the most optimal placement of points to win an instance of the game when placing as Red or as Blue. The model also helps answer which of Red and Blue have the advantage when both play optimally.

1 Introduction

1.1 Voronoi Diagrams

A Voronoi diagram is a method of geometrical partitioning of a space where the division is based on the distance of every point in the space from a specific set of points. The distance metric used here is Euclidean, but any other metric can be used, depending on the application.

Let X be a metric space with a distance metric d. Let $P = \{P_1, P_2, ..., P_n\}$ be the set of points to be considered as sites based on which the division is done. Then, the region R_i assigned to a point $P_i \in P$ will be given by:

$$R_i = \{x \in X \,|\, d(x, P_i) \leq d(x, P_j) \,\forall\, P_j \in P;\; P_i \neq P_j\} \tag{1}$$

Put simply, the Voronoi diagram can be defined as the collection of R_i from all P_i in P.

This version of the Voronoi game can be used to simulate and find solutions for multiple cases of **Competitive Facility Location** problems, where multiple players place facilities inside a fixed area such that their region of influence is maximised. Changes can be made in no. of dimensions, turns and players to suit each problem. Due to this being a rather simple representation, other required factors like time can be easily incorporated.

1.2 The Voronoi Area Game in 2D

The Voronoi area game (hereafter referred to as the **Voronoi game**) is one where a finite space is divided according to a Voronoi diagram and the winner is decided based on which subset of points

cover the higher proportion of area. Players take turns to add points to a finite space which will act as sites for the Voronoi diagram. For this project, we consider only the two-player variant in 2D with 5 turns; complex variants having more players in higher dimensions with more turns may be studied with some variations in the code, although the computational power required for the same will exponentially increase.

The steps involved in an instance of the Voronoi game with two players (**Red** and **Blue**) are given below. Without loss of generality, we take Red to make the first turn in every game.

1. Red places its point in the given limited playing space. The corresponding Voronoi diagram is obtained, where each region is coloured with that of its site. Here, all of the area would be assigned to Red. The proportion of area covered by each player can be obtained at every turn.
2. Blue places its point in the space such that it is at least a certain unit of distance d away from the point. Once again, the corresponding Voronoi diagram is made, and we see both red and blue regions along with the proportion of the total area they cover.
3. Both players keep adding their points to the space such that every point is at least d units away from each other till 5 turns are complete. The player covering the most area wins.

1.3 Optimal Strategies using ML

In this project, our aim is to create an ML model which would make the most optimal placement of points to win after 5 turns as either Red or Blue. For this, two versions of the model is to be made, each for playing as Red and Blue respectively.

Through applications of this model, we hope to answer some fundamental questions about the game, some of which are:

1. **Do any of the players hold an automatic advantage?** We plan to answer this by making the two fully trained models play against each other multiple times.
2. **Is a greedy approach the most optimal strategy?** Here, greedy refers to a player placing a point such that the highest possible area is secured for the current turn. Comparing a greedy placement of points with that of the trained models, both against the same placement of opposite points should give a clear answer to this question.

2 Execution of Midway Goals

Two goals were planned to be completed by the midway period. One was to **engineer an interface** on which the model would play multiple iterations of the Voronoi game during the learning process. The second goal was to start **implementation of ML algorithms** on the interface.

2.1 The game interface

The interface models a simplified version of the Voronoi game where a 100×100 grid is taken as the playing area. Points can be plotted with at least 10 units of distance between them. The code gives the set of sites in the area as well as the proportion of area covered by both players for all turns. It also shows the final Voronoi diagram which determines the winner. The python code for the interface can be found here.

2.2 ML Implementation

Unfortunately, we were not able to start implementing an ML algorithm for the project yet. We planned to use Q-learning for the learning process, but the Q-matrix involved turned out to be too large to feasibly work with, even for a 100×100 grid. This was due to the fact that placements of points in every previous turn have to be included in the Q-matrix for one single turn. Hence, we have to change the state representation so that decisions can be taken in a reasonable amount of time. Also, we had also tried to code a model using a greedy approach to beat the gaem, hoping to make it a benchmark that our ML model has to beat, and even though we were successful at it, the program was too slow to run enough time to make relevant observations.

3 Endterm Goals

The primary focus for endterm was on developing an effective state representation and implementing a robust ML algorithm to make optimal moves in the Voronoi game for both players. We planned to use a KNN-based approach for starters, then move on to another approach hopefully using specialised RL algorithms like PPO, DQN or A2C.

4 Execution of Endterm Goals

4.1 kNN-based Approach

In our project, we've employed a K-Nearest Neighbors (KNN) model to develop a game-playing AI for a Voronoi area game, where the strategic placement of markers by two players alternately (red and blue) determines control of the game area. The approach is based on historical game data, utilizing patterns from past games to inform future gameplay strategies. Key Features of the KNN-Based Model Training:

- Data-Driven Approach: The model is trained on a dataset generated from simulated games, where each game's moves and outcomes are recorded. This dataset includes the positions of markers placed by each player and the final area percentages controlled by each player.
- Feature Utilization: For each turn in the game, the model considers the current state of the board (i.e., the positions of all markers placed so far) to predict the most advantageous next move. The features used in the KNN model are the positions of the markers, while the labels could be the subsequent move or the game outcome in terms of area control.
- Iterative Learning: The model is refined through an iterative process. Initially, it learns from randomly played games (baseline data). After each training phase, the model is used to play against a randomly moving opponent, generating new game data that is presumably of higher strategic quality than its predecessors. This new data is then used in the next training cycle, progressively enhancing the model's ability to predict effective moves.
- Adaptation to Game Dynamics: The KNN algorithm is particularly well-suited for this application due to its simplicity and efficacy in classification tasks based on feature similarity. In the context of the Voronoi game, it can effectively determine which moves lead to winning outcomes based on historical similarities.
- Performance Improvement: Each iteration aims to improve the model's predictive accuracy and game-playing proficiency, making the AI a more formidable opponent.

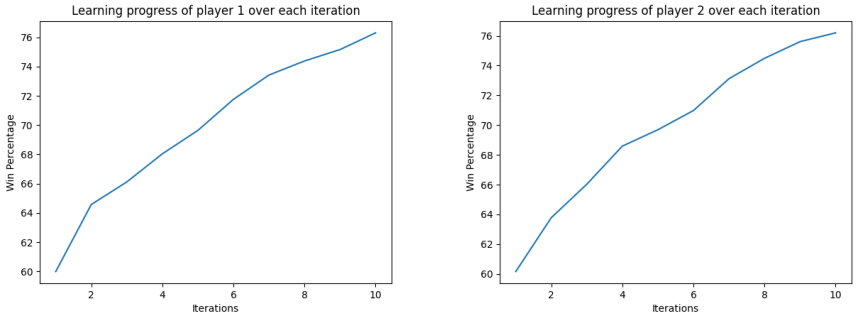

Figure 1: Increase in win percentage over training sessions with 10,000 games in each iteration.

The pseudocode for the kNN based algorithm we have used is as follows:

```
// Define constants and initial settings
Set num_iterations to the total number of iterations desired (e.g., 10)
Set num_games to the number of games to simulate in each iteration (e.g., 10000)
Set data_path to the path of the initial dataset (e.g., "data_0.csv")

// Define the main function to run iterations
Function run_iterations(num_iterations, num_games, data_path):
    // Loop through each iteration
    For i from 0 to num_iterations - 1:
        // Load the dataset from the current iteration
        Load dataset from the file named "data_{i}.csv"

        // Train the KNN model using the current dataset
        model = train_knn(dataset)

        // Initialize an empty list to store results of new games
        Initialize new_games as an empty list

        // Simulate games using the trained model
        For j from 1 to num_games:
            // Simulate a game using the current model and dataset
            new_game = simulate_game(model, dataset)

            // Store the results of the game
            Append new_game to new_games

        // Convert the list of new games into a DataFrame
        Create a DataFrame from new_games

        // Save the new dataset for the next iteration
        Save DataFrame to a file named "data_{i+1}.csv"

// Define the function to simulate a single game
Function simulate_game(model, dataset):
    // Initialize the current game state as an empty list
    Initialize current_state as an empty list

    // Initialize game_data as an empty list to collect game results
    Initialize game_data as an empty list

    // Loop through each turn in the game (assuming each player plays 'size' turns)
    For t from 1 to 2*size:
        // Check if it's the KNN model's turn to play
        If t is even:
            // KNN model's turn
            If current_state is not empty:
                // Predict the next move based on the current state
                Predict the next move using model and current_state
            Else:
                // If no moves have been made, select a random move from the dataset
                Select a random move from the dataset column t

            // Record the move in the current game state
            Append the move to current_state

        // If it's the random player's turn
```

```
        Else:
            // Select a random move from the dataset
            Select a random move from the dataset column t

            // Record the move in the current game state
            Append the move to current_state

    // After all turns are completed, calculate the areas controlled by each player
    Calculate areas for final positions based on the current_state

    // Record the final areas in game_data
    Append area results to game_data

    // Return the collected data for this game
    Return game_data

// Start the iterative training and simulation process
Call run_iterations(num_iterations, num_games, data_path)
```

This KNN-based method represents a practical approach to developing an AI that not only mimics human-like strategic thinking observed in historical game data but also adapts and improves through continuous learning and interaction with varying game scenarios.

4.2 New game interface

We had also managed to remake our interface and environment on which the Voronoi game is played. We previously used the `matplotlib` library to make the interface; we have removed this library entirely and used the `numpy` library to make a grid of size 100×100, which acts as the area on which the Voronoi game is played. This change helped in faster state generation for the ML models to get information from. We have also used the `gamepy` library to make a platform for human opponents to play in. The code for the same has been attached in GitHub page given at the end.

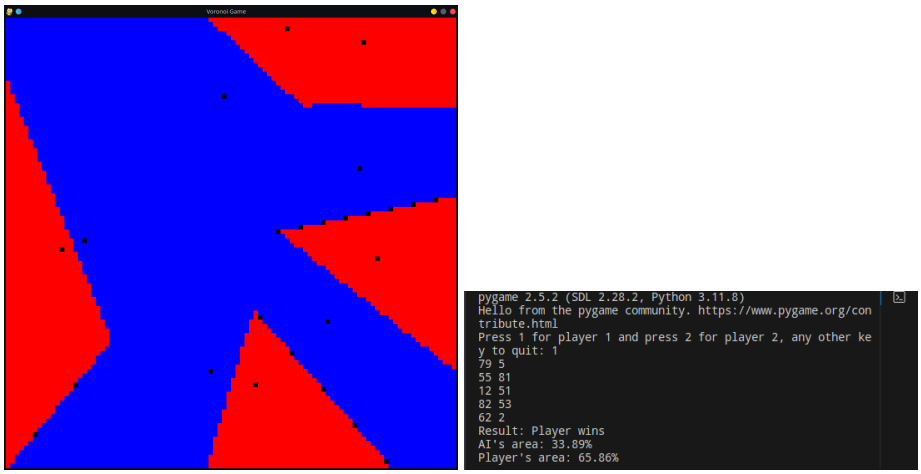

Figure 2: An example game showing the gaming interface made using pygame and numpy libraries. The interface gives the human player the chance to choose which player to play as, and gives out the points placed by the ML model, along with the percentage of area accumulated at the end of the game.

5 Caveats & Further Work

5.1 Limitations

- Model Selection: Initial experiments with PPO, DQN, and A2C were promising at first, but actual implementation over the environment we created turned out to be very difficult and gave little to no results.
- The KNN Model: KNN has given out satisfactory results, providing models for both players 1 and 2 with a win percentage of above 70% against random opponents. Nevertheless, we feel the previously mentioned algorithms would fare even better, provided we achieve proper implementation.

5.2 Performance Issues

- Greedy Method Drawbacks: The implemented greedy method was too slow to generate multiple games, so some further steps in our approach like pitting greedy VS KNN and the proper statistics for greedy strategy were not obtained.

5.3 Future Directions

- Towards Other Variants: Future efforts will focus on adapting our approach for more complex scenarios like weighted Voronoi games to add more real-life applications.
- Optimizing Computational Efficiency: We plan to enhance the speed and efficiency of our models, particularly improving the greedy method for faster decision-making capabilities. Also, we plan to debug and optimize the codes for PPO, DQN and A2C we wrote earlier in hopes of better performance.

6 Conclusion

To summarise, the 2-D Voronoi Area game is a simple and highly malleable model for instances of competitive location problems in any context, we have managed to make a decently successful kNN-based model trained to make optimal places to maximise winning chances for each player with a win percentage of over 75% against random opponents. kNN is a very basic approach to this problem due to which we observed that win percentage dips against human opponents and the greedy approach, so we aim to use more advanced and specialised RL algorithms, and make optimisations to the current codes, in the near future.

7 Resources

- Codes related to this project have been attached in the following GitHub repository: https://github.com/Sandipan04/voronoi

References

[1] Aritra Banik, Bhaswar B. Bhattacharya, Sandip Das, and Satyaki Mukherjee. The discrete voronoi game in r2. *Computational Geometry*, 63:53–62, 2017.

[2] Francesco S. Bellelli. F.s.bellelli: The fascinating world of voronoi diagrams, 2021.

[3] Bruno Bouzy, Marc Métivier, and Damien Pellier. Mcts experiments on the voronoi game. 2012.

[4] Frederik Brasz. Voronoi diagram area game, 2015.

[5] Wikipedia contributors. Voronoi diagram — Wikipedia, the free encyclopedia, 2024.