Describing Algorithms for The Voronoi Game

Abstract:

The Voronoi Game models real-world competition in resource allocation, such as companies deciding optimal locations for stores in a competitive marketplace. In this project, I explored strategies to maximize revenue on a graph where each vertex represents a district with an associated value, and edges represent distances between districts. To better understand this problem, I developed and analyzed an algorithm that consistently outperforms both random and greedy approaches by strategically selecting vertices based on calculated influence on neighboring districts. This project utilized core computer science concepts such as graph theory, algorithm design, and heuristic evaluation to efficiently determine optimal moves under strict time constraints. My key findings demonstrate the usage of weights for parameters to better optimize an algorithm to make it more efficient.

Results:

My Algorithm **VoronoiMaxAlgorithm** goes through all the vertices in the graph. For each of the vertices, it will loop through the remaining vertices with the key factors described below to get the "value" of that particular vertex. The vertices with the higher value are preferred for selection. The value is calculated in the following way:

<u>Calculating Three Key Factors:</u>

- Value Factor:
  This is based on the intrinsic value of the vertex itself. Initially, the value is multiplied by 2 to emphasize its importance. In the late stages of the game (when fewer turns remain), the value is further amplified to prioritize direct gain.

- Edge Factor:
  This considers the connected neighbors of the vertex. For each neighboring vertex, it divides its value to the square root by the distance to the current vertex. This factor reflects the potential influence of the chosen vertex on nearby districts, especially unclaimed ones.

- Left Neighbors:
  This counts the number of neighbors the player can still "win over" by choosing the current vertex. If a neighboring vertex is already owned by the opponent, it is not considered. This factor encourages selecting vertices that can dominate unclaimed or contested areas.

<u>Adding weights</u>:

- As described earlier, Value Factor is weighted higher in late-game scenarios.

- Left Neighbors is weighted with a multiplier of 50. This is done because the average vertex has the value of 50 and so the average value from the remaining vertices is assumed to be 50.

- Edge Factor contributes based on nearby vertex value and distance. The weight of 1 assumed to be best for this case.

Thought Process:

The thought process for this was very simple. Knowing the value of the vertex had a big role, I immediately increased the factor of the value by two. Next, using the distances to individual vertices also was not so difficult to think about. When I divided by distances, I thought of the inverse square law so I initially tried that. But the results were bad, so experimenting with the powers led me to use 0.5 as it provided the best results.

Results from Testing:

Iterating over 2000 games, these were the results:

Against random algorithm:

P1Wins: 61

P2Wins: 1939

Ties: 0

Against greedy algorithm:

P1Wins: 360

P2Wins: 1639

Ties: 1

Time Complexity:

The implementation uses a double for loop over the vertices and the methods inside it run at constant time. So, the overall time complexity is **O(V^2).** The get distances method runs at O(V^3) however since it runs only once in the constructor, we can ignore its use in the algorithm, regardless due to my implementation of a HashMap, the return distance also runs at constant time.

Extensions:

The **VoronoiBlockerAlgorithm** uses a straightforward approach to select vertices based on two main factors: maximizing the value of the vertex and blocking the opponent's potential expansion. As done previously, each vertex is scored based on its "value" using the key factors described below, and the vertex with the highest score is chosen. The factors contributing to the score are:

Calculating Two Key Factors:

1. Value Factor:

This is simply the value of the vertex, and as done previously, in the late-game the value is doubled to prioritize direct gain.

2. Blocking Score:

   This is an interesting metric. What I am trying to implement is vertex's proximity to opponent-controlled neighbors. For each of the neighbors, the blocking score increases proportionally with its value and inversely proportional to distance.

Adding Weights:

- Value Factor: The weight of the value factor increases in the late game, as mentioned earlier, by doubling the inherent value.
- Blocking Score: It is weighted based on the value of neighboring vertices divided by their distance. This will prioritize close neighbors that can be won over to our side.

Thought Process:

The algorithm is driven by the goal of disrupting the opponent's expansion by placing the tokens very close to vertices owned by the opponent. These are seen as areas where the opponent can expand further unless blocked. This strategy is designed to win contested areas, even if they are temporarily controlled by the opponent. This is a much simpler and less efficient algorithm but is an interesting algorithm.

Results from Testing:

Random Algorithm:

P1Wins: 314

P2Wins: 1685

Ties: 1

Greedy Algorithm:

P1Wins: 970

P2Wins: 1029

Ties: 1

This is a much simpler and less efficient algorithm but is an interesting algorithm which given time could probably be optimized for better defense.

Time Complexity:

The outer loop iterates over all vertices, and the inner loop evaluates their neighbors. Let V be the total number of vertices and N the average number of neighbors per vertex. The time complexity is O(V×N).

Acknowledgements:

In completing this project, I consulted with Max and Allen, but did not look at any code. I attended the evening TA sessions on Sunday and Monday.