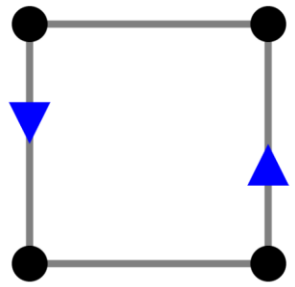
Identifying Winning Strategies for the Game of Cycles Using Computer Programs and Game Trees

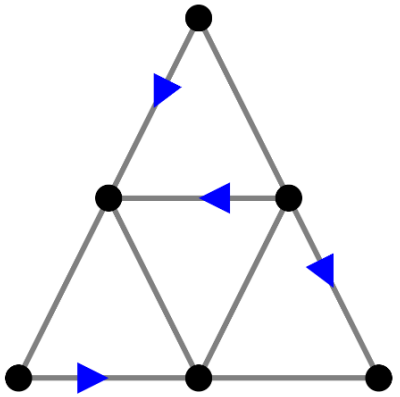
Nicholas Bozentko

Iona College

Abstract: The Game of Cycles (Su, 2020) is an impartial combinatorial game played on a connected planar graph. Each player takes turns marking an edge of an initially undirected graph with a direction subject to certain rules, with the goal of completing a cycle cell. Our research involves finding winning strategies for different classes of game boards. We begin by identifying either first or second player winning strategies for relatively simple boards. To study more complex boards, we implement a playable version of the game that can be run on a computer. We then use this to create a program that evaluates every possible game state of a given board and finally determines which player has a winning strategy. Currently, we have our playable computer game and can generate game trees for smaller boards. Going forward, we plan to implement more boards in our playable game and optimize our program to efficiently identify winning strategies in larger boards.

Background: The goal of the Game of Cycles is to complete a cycle cell on a connected planar graph board. Two players take turns marking edges with the goal of either completing a cycle cell or marking the last markable edge. A move is not allowed to result in a sink or source node in the graph. The goal of our research is to identify winning strategies for different classes of boards. To assist in this goal, we have created a computer playable version of the game and are writing programs to evaluate every possible state of a game board.





Designing Our Application

Major Design Considerations

The challenge in designing the application was deciding how to accurately represent the game board, game state, and determine if a player has won the game.

A naïve approach to representing the game board would be to use a simple graph implemented with adjacency list. With this approach, existing and well-known graph algorithms could be used to derive the game state. There are several problems with this approach.

The first issue that comes up with this approach is our unique idea of a "cell." Each game board consists of cells that, when formed into a cycle, would result in the end of the game. In regular graph theory, this idea of a "cell" does not exist. Simply moving around the nodes / edges in a graph does not change the graph. However, moving the nodes / edges in a game board does change the game board. For this reason, creating and maintaining the game board is slightly more complex than simply constructing a graph.

The next issue that arises with this approach is "outer cycles", meaning a cycle in the graph that does not complete a cycle cell. These outer cycles do not constitute a win. Therefore, we must define which sets of edges (forming cells) would count as winning cycles.

Finally, it is possible to have an edge that is shared by multiple cells. This is not necessarily a problem with the naïve graph representation, but it is something we need to keep in mind when creating our structure.

Representing the Game Board

Our solution to these problems is to create custom Node, Edge, Cell, and Board data structures. In this representation, Nodes are given a unique id, Edges are made up of a tuple of Nodes, Cells are made up of an ordered list of Edges, and a Board is made up of a set of Cells.

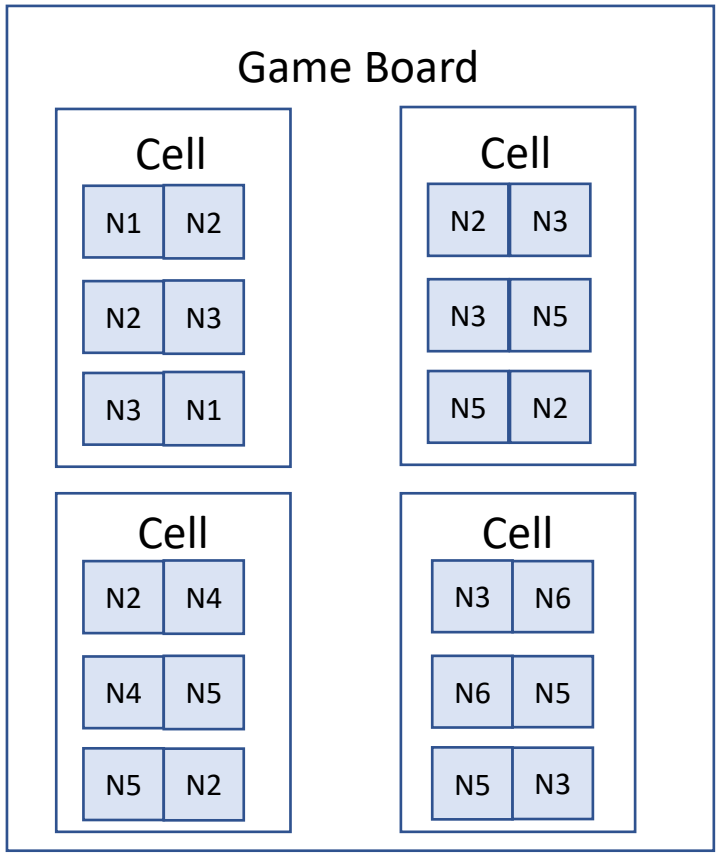
Nodes are a simple class that have a unique identifier, a list of adjacent nodes, a list of inbound Nodes, and a list of outbound Nodes. These Node lists are used to determine when an almost-sink or almost-source is present.

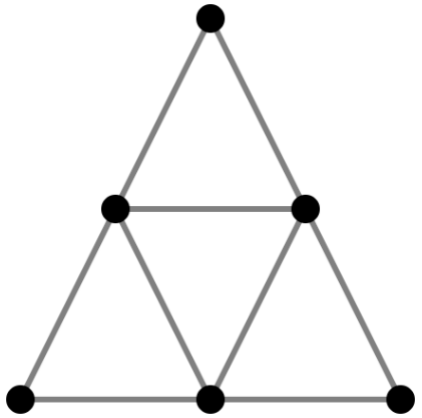
Edges are simply a tuple of Nodes where the first Node is the source Node and the second Node is the target Node.

The Cell structure is more complex. The main part of the Cell class is the ordered list of Edges that make up the Cell. The list must be ordered so that we can check if the Cell contains a cycle.

The Board structure is the final piece that brings all of the structures together. The important part of the Board structure is the set of Cells that make up the board. We also keep an underlying graph structure for the sake of creating a human-friendly visual representation of the game board.

When a move is made, several actions must be taken. We must check if the move is valid by checking if the move would result in a sink or source and that the edge selected is not already marked. This involves determining which cells and edges are involved with the move and performing appropriate checks. When a move is determined to be valid, all of our structures are updated starting with the Nodes and bubbling all the way up to the Board structure.



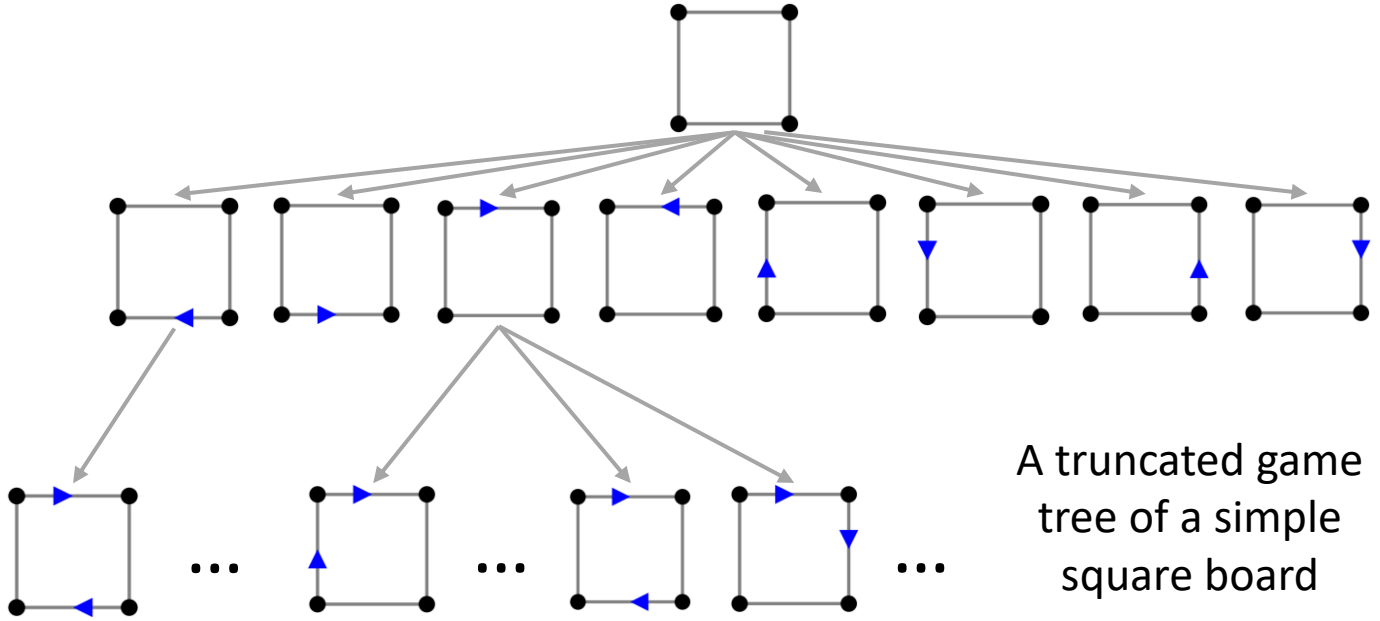


A playable game board along with its internal representation

Creating a Game Tree

Motivation

After creating the necessary structures to represent our game board, we then set out to write a program that accepts a game board and generates every possible game state for that board. With this, we can then analyze the states and determine if player one or player two has a winning strategy.



A truncated game tree of a simple square board

Optimizations

One problem that we face is the inefficiency in generating these boards. We plan on making two optimizations that should dramatically increase the size of game trees that we can feasibly generate. The first strategy is to take advantage of symmetry in the boards. In the example shown, one can see that every first possible move on the square board is logically the same move. With our current structure, we can not take advantage of this symmetry. However, we clearly see the value in using this symmetry down the line, as even in this simple example, we would dramatically decrease the number of game states that we need to generate.

The next optimization that we can make involves pruning branches of the game tree with the Minimax Algorithm and Alpha-Beta pruning. This will allow us to prune entire branches that definitely do not lead to viable strategies. This, again, will significantly cut down the number of game states that we need to generate.

Outcomes

Our work so far has produced a playable implementation of the Game of Cycles. This allows us to play and reset games easier than drawing them each time, especially while playing remotely. We also do not make accidental illegal moves while playing anymore since the game will catch these mistakes for us.

With our generated Game Trees, we are able to run through and evaluate every possible state of a game. Optimizing our methods to handle larger boards will be our next priority.

Future Work

Our future work will be focused on generating Game Trees more efficiently and deriving greater insights from them. We plan to utilize the Minimax Algorithm and Alpha-Beta pruning to avoid making unnecessary calculations, which would dramatically increase our efficiency. After this stage of generating sufficiently large game boards that are past the size that we already have answers for, we can determine which player has a winning strategy. This will make is easier for us to proceed with finding what that winning strategy is.

References

Francis Su. *Mathematics for Human Flourishing*. Yale University Press, 2020.

R. Alvarado, M. Averett, B. Gaines, C. Jackson, M. L. Karker, M. A. Marciniak, F. Su, and S. Walker. *The Game of Cycles*. 2020. arXiv: 2004.00776

"Dash Cytoscape." *Plotly*, dash.plotly.com/cytoscape.

Acknowledgements

The project was based on work begun at the 2019 Research Experiences for Undergraduate Faculty program, which was made possible by the support from the National Science Foundation (NSF) through Grants NSF-DMS 1620073 to AIM and NSF-DMS 1620080 to ICERM

This project was made in collaboration with Dr. Benjamin Gaines who is conducting research into winning strategies for the Game of Cycles and contributed toward the development of the work discussed here.

We would also like to thank Iona College for the resources provided that helped make this project possible.