Richman Tic Tac Toe Game

Michael Fan

May 16, 2004

1 Introduction

For the computer science departmental honors project, I created a program for a Richman version of Tic Tac Toe. Richman games, more technically known as combinational games under auction bidding, are a type of game created by David Richman in the 1980's that combine the aspects of combinational and matrix games. The purpose of this project was to understand the theory behind a Richman game, to extend in onto a game of Tic Tac Toe, and develop a winning strategy for the game.

The first part of the project was to understand the theory behind a Richman game and develop a winning strategy for the computer in the Richman Tic Tac Toe game. Lazarus et al. described a winning strategy for Richman games on a graph and I simplified their algorithm to apply to only directed acyclic graphs. Sections 2-4 describe Richman games and the simplified algorithm.

The second part of the project was to code the Richman Tic Tac Toe game. The coding was broken down into a two step process. The first step was to convert the game of Tic Tac Toe into a directed acyclic graph. After the game was reduced into graph form, the second step was to calculate the Richman costs for the entire graph so an optimal strategy for the computer could be used. Section 5 describes the process of coding the Richman game of Tic Tac Toe.

Finally, this project is still a work in progress. Section 6 describes the various improvements in the project that will be made over the next year.

2 Introduction to Richman Games

David Richman in the late 1980s introduced a new type of game theory: Combinational Games under Auction Bidding or Richman Games. This new theory combined characteristics of Combinational Games and Matrix Games. In combinational games, such as chess, checkers, and Nim, two players alternate turns making moves. In matrix games, such as rock, paper, scissors, two players simultaneously make moves. In a Richman game, each player begins with a fixed pool of money. Before each move, each player simultaneously bids to make the move; the player who bids higher makes a move (ties are resolved by a coin flip) and gives the opponent the dollar amount of the winning bid. The loser of the bid does not make a move. This process is repeated until a player wins.

For a Richman game on a graph, the game begins with a token on a vertex of a graph and each player has a separate end vertex. The goal of each player is to move the token onto his end vertex. During each move, each player bids to move the token to an adjacent vertex and the game ends when a player moves the token to his vertex. Lazarus et al. [1,2] determined an optimal algorithm to winning a Richman game on graphs. The purpose of this paper is to elucidate this algorithm and apply it to directed acyclic graphs (DAGs).

3 Richman Cost Function

In this algorithm, the two players known as Ms. Red and Mr. Blue, and respectively their end vertices are r and b. Also, the total amount of money is defined as 1, and Red and Blue possess fractions of this total that add up to the total amount of money. Finally, throughout this paper, G will denote a directed acyclic graph (V, E), with V denoting the set of G and with E denoting the set of edge of G, with vertices r and b, such that from every vertex there is a path to either r or b. The key to the algorithm, is the statement that:

> For every directed acyclic graph G with a token located on vertex v, there exists a critical ration $R(v)$, such that if Blue's money is greater than $R(v)$, he has a guaranteed winning strategy. Respectively, if Red's money is more than $1 - R(v)$, then she has a guaranteed winning strategy.

This $R(v)$ ratio is known as the Richman Cost Function.

3.1 Definition of Richman Cost Function

For each vertex $v \in V$, let $S(v)$ denote the set of successors of v in G. In other words, $S(v) = \{w \in V : (v,w) \in E\}$. Finally, given any function $f : V \to [0,1]$, let

$$f^+(v) = \max_{w \in S(v)} f(w) \text{ and } f^-(v) = \min_{w \in S(v)} f(w)$$

With these definitions in mind, the Richman Cost Function is for $v \in V$

$$R(v) = (R^+(v) + R^-(v))/2$$

with $R(b) = 0$ and $R(r) = 1$. Or in other words, the Richman cost of v is the average of the maximum and minimum Richman cost of v's successors.

3.2 Algorithm for Calculating Richman Costs on a DAG

To keep track of which nodes' Richman value hasn't been set, the algorithm assigns a color to each node (either white, gray, or black.) White nodes haven't been visited by the algorithm, gray nodes have been visited but all their children haven't been visited, and black nodes have been visited and all their children have been visited. The color information for each node is kept in an array named color; for each vertex $v \in V$ in G, color[v] will return the vertex's color.

Richman_Calculate(vertex start_vertex)
for each vertex $v \in V$, intialize $R(v)$ to 1, expect for $R(b) = 0$, and initialize color[v] to white, except color[r] and color[b] are set to black.
for each vertex $y \in S(start\_vertex)$
    Richman_Visit(y)

Richman_Visit(vertex y)
color[y] = gray
for each vertex $z \in S(y)$
    if color[z] = white
    then Richman_Visit(z)
color[y] = black

$R(y) = R^+(y) + R^-(y))/2$

Note: This algorithm is based on the depth-first search algorithm. The correctness of the algorithm can be verified using Theorem 1 (below) and the properties of a depth-first search.

Theorum 1: Each vertex of directed acyclic graph G has only one valid Richman cost
Proof:

For each directed acyclic graph, there are at least one so called sink vertices, vertices for which there are no outgoing edges. For our case, the two sink vertices are r and b. If r and b are removed from the graph, then the resulting graph remains a DAG and still has sink nodes (in this case the nodes that had edges only to r and b) and their Richman cost can be calculated from r and b. This process can be repeated (the sink vertices are removed and the resulting graphs sink vertices Richman costs are calculated) until the start vertex is reached. When the Richman Cost for each vertex is calculated, it is only calculated from vertices that have been removed from previous graphs (otherwise the vertex would not be a sink vertex) and so the Richman Cost will be set to that one valid cost.

4 The Optimal Strategy

So, the optimal algorithm begins by calculating the Richman cost for each vertex of the graph. Once the Richman costs are calculated, when the token is at v, the optimal bid for both players is $R(v) - R^-(v)$ (this optimal bid is equivalent to $R^+(v) - R(v)$). If Blue wins, Blue should move the token to the successor of v with the minimum Richman cost. If Red wins, Red should move the token to the successor of v with the maximum Richman cost.

Theorem 2: If Blues share of the money exceeds $R(v)$, then Blue has a deterministic winning strategy. If Blues share is less than $R(v)$, then Red has a deterministic winning strategy.
Proof by Strong Induction:

Base Case: If the token is on b and Blue's share of the money exceeds $R(b) = 0$, and so Blue wins since the token is on Blue's end vertex.

Induction Hypothesis: For all successors w of v assume if Blue's share of the money exceeds $R(w)$ when the token is on w, then Blue has a winning strategy. Prove for v.

Blue bids the optimal bid of $R(v) - R^{-}(v)$.

If Blue wins the bid:

> Blue's share of the money is subtracted by $R(v) - R^{-}(v)$.
>
> Blue's share of the money was greater than $R(v)$, so Blue's share of the money is now greater than $R(v) - (R(v) - R^{-}(v))$.
>
> Using some simple algebra,
>
> $$R(v) - (R(v) - R^{-}(v)) = R(v) - R(v) + R^{-}(v) = R(v)$$
>
> Blue's share of the money is greater than $R^{-}(v)$. Blue moves the token to the successor of v with the minimum Richman cost, which has a Richman value of $R^{-}(v)$. By the Induction Hypothesis, since Blue's share of the money is greater than $R^{-}(v)$ and the Richman cost of the vertex the token is on is equal to $R^{-}(v)$, Blue retains a winning strategy.

If Blue loses the bid:

> Using the alternate form of the bid, Blue's share of the money is increased by $R^{+}(v) - R(v)$.
>
> Since, Blue's share of the money was greater than $R(v)$, so Blue's share of the money is now greater than $R(v) + (R^{+}(v) - R(v))$.
>
> After some simple algebra, Blue's share of the money is greater than $R^{+}(v)$. Since the maximum Richman value of any of v's successors is by definition $R^{+}(v)$, no matter which successor Red moves the token to, Blue's share of the money is greater than the Richman value of that successor. So, by the Induction Hypothesis, Blue retains a deterministic winning strategy.

This proof can easily be reversed for Red and the Richman function replaced with $1 - R(v)$. If Blue's share of the money is less than $R(v)$ then Red's share of the money is greater than $1 - R(v)$ and repeat the proof above to show that Red has a deterministic winning strategy.

There is a third case however, that Blue and Red's share of the money is equal to that of $R(v)$. If both players play optimally, then Blue's share of the money will continue to be equal to $R(v)$ and the resolution of tie bids is by a coin flip, so the winning of the game is now by chance.

5 Process of Implementation

As shown in the previous sections, it is simple to devise a winning strategy for a Richman game if the game can be reduced into a graph format. To implement the Richman game of Tic Tac Toe, the first step was to reduce Tic Tac Toe into a directed acyclic graph game and then to run a function to determine the Richman costs of the entire graph.

The Richman Tic Tac Toe game was written entirely in Java. Since the game is eventually going to have a visual interface and be web-accessable, Java was the language best suited for these two implementing these two features.

The original design was to include all the possible Tic Tac Toe board configurations as vertices in the graph. The vertices were connected with edges that connected each vertex to the vertices that were one move away further into the game. This design would result in a directed acyclic graph that represented all the possible ways a Tic Tac Toe game could be played.

After some testing, it was discovered that this design was untenable. Originally, the program would attempt to create the graph from scratch each time it was run. This lead to a lengthy start-up time. Afterwards, a change was made to pre-process the Tic Tac Toe graph into a file format and load the program from a file. This, however, still took a lengthy amount of time, since the Tic Tac Toe graph would contain 3^9 different vertices.

A second design was created utilizing reflections and rotations to minimize the number of vertices and edges in the graph. For instance, the empty board has edges to eighteen separate board configurations, two marks possible for each square on the board. However, with reflections and rotations, only six vertices are necessary: two center, two side, and two corners. Rotations or reflections of each of these six vertices can represent each of the original eighteen vertices. By utilizing reflections, the number of edges and vertices were decreased by a factor of ten.

However, Tic Tac Toe allows draws to occur when the board is filled and there is no winner. To resolve the problem in computing Richman costs for draw scenarios, draws were decided to be wins for the human player. This had the effect of the human player having a slight advantage if both the human and the computer start with the same amount of money. With this resolved, the Richman costs were calculated and the optimum computer playing strategy could be easily determined.

The code for the game is currently posted at www.wam.umd.edu/~mfan.

6 Improvements

This project is still in the development stages. As of now, the interface for the game is very primitive and relies on the user reorientating reflected and rotated board configurations in his mind. In the future, a visual interface will be developed for the game. Also, there are plans to introduce an error into the computer bidding process and compare the difference in performance between a small error and larger error in bidding.

References

[1] Lazarus, Andrew et al. "Richman Games." In *Games of No Chance: Combinational Games at MSRI, MSRI Publication Series* 29, edited by Richard J. Nowakowski, 439-450. Cambridge University Press, 1998.

[2] Lazarus, Andrew et al. "Combinatorial Games under Auction Play." *Games and Economic Behavior* 27 (1999): 229-264.