The George Kingsley Zipf AI Engine : Decision Making

Created by Erin Dahlgren December 1 2011
Modified by Erin Dahlgren January 2 2012

Zipf's goal

$$\min(\frac{losses_{games}}{\forall endgames})$$

Zipf is interested in the statistic above, which essentially maximizes his own wins and ties by minimizing the rest of the endgame space that does not include wins and ties: losses. This boils down to minimizing losses/(wins + ties + losses). To compute this, Zipf recurses through the game tree until he finds an endgame, tallies its status (win, tie, or loss), and moves back up and down the tree until he has traversed it completely. He computes the statistic for all moves he's considering and chooses the move with the lowest value.

Crucially Zipf does not need to do this at every move: for any move he is considering, it is a parent to only *part* of the entire game tree. But more important, there are situations, like at the end of the game, when recursing is just plain unnecessary. This is because there is often only one move that will prevent his death or ensure his win. Zipf only makes statistically minded decisions when he is not in an immediate win situation or a 'do-or-die' situation and has multiple places on the board to choose from.

Zipf first uses a filter. This is `optimals` in the Engine module.

If there is only one move returned from the filter, Zipf takes it immediately. See `choose` in the Engine module.

If there is more than one move returned, Zipf starts to reason statistically. See `recurse_tree`, `get_stats`, and `compute_stats` in the Engine module.

Cases of statistical reasoning (*recursing the game tree*) occur at the beginning of the game, because there is little chance for immediately losing or winning. You will see in the engine that I have fixed the computer (not in the `testgames.py` module but in the user-interactive `main.py` and its dependents) because the computer is then at the disadvantage. I wanted to ensure that even in this role, George Kingsley Zipf could never lose.

1. Statistical Reasoning: First moves

X	.	.
.	.	.
.	.	.

choice	*wins*	*losses*	*ties*	*l/w+l+t*
(0,1)	756	2048	864	0.5583
(0,2)	1132	1832	576	0.5175
(1,0)	756	2048	864	0.5583
(1,1)	1312	1436	720	**0.4140**
(1,2)	936	1902	288	0.6084
(2,0)	1132	1832	576	0.5175
(2,1)	936	1902	288	0.6084
(2,2)	936	1652	1008	0.4593

.	X	.
.	.	.
.	.	.

choice	*wins*	*losses*	*ties*	*l/w+l+t*
(0,0)	1276	1798	864	0.4565
(0,2)	1276	1798	864	0.4565
(1,0)	1080	1868	576	0.5300
(1,1)	1652	1438	720	**0.3774**
(1,2)	1080	1868	576	0.5300
(2,0)	1456	1652	288	0.4864
(2,1)	900	2158	1008	0.5307
(2,2)	1456	1652	288	0.4864

.	.	X
.	.	.
.	.	.

choice	*wins*	*losses*	*ties*	*l/w+l+t*
(0,0)	1132	1832	576	0.5175
(0,1)	756	2048	864	0.5583
(1,0)	936	1902	288	0.6084
(1,1)	1312	1436	720	**0.4140**
(1,2)	756	2048	864	0.5583
(2,0)	936	1652	1008	0.4593
(2,1)	936	1902	288	0.6084
(2,2)	1132	1832	576	0.5175

.	.	.
X	.	.
.	.	.

choice	*wins*	*losses*	*ties*	*l/w+l+t*
(0,0)	1276	1798	864	0.4565
(0,1)	1080	1868	576	0.5300
(0,2)	1456	1652	288	0.4864
(1,1)	1652	1438	720	0.3774
(1,2)	900	2158	1008	0.5307
(2,0)	1276	1798	864	0.4565
(2,1)	1080	1868	576	0.5300
(2,2)	1456	1652	288	0.4864

.	.	.
.	X	.
.	.	.

choice	*wins*	*losses*	*ties*	*l/w+l+t*
(0,0)	792	1830	576	**0.5722**
(0,1)	612	2082	576	0.6366
(0,2)	792	1830	576	**0.5722**
(1,0)	612	2082	576	0.6366
(1,2)	612	2082	576	0.6366
(2,0)	792	1830	576	**0.5722**
(2,1)	612	2082	576	0.6366
(2,2)	792	1830	576	**0.5722**

.	.	.
.	.	X
.	.	.

choice	*wins*	*losses*	*ties*	*l/w+l+t*
(0,0)	1456	1652	288	0.4864
(0,1)	1080	1868	576	0.5300
(0,2)	1276	1798	864	0.4565
(1,0)	900	2158	1008	0.5307
(1,1)	1652	1438	720	**0.3774**
(2,0)	1456	1652	288	0.4864
(2,1)	1080	1868	576	0.5300
(2,2)	1276	1798	864	0.4565

.	.	.
.	.	.
X	.	.

choice	*wins*	*losses*	*ties*	*l/w+l+t*
(0,0)	1132	1832	576	0.5175
(0,1)	936	1902	288	0.6084
(0,2)	936	1652	1008	0.4593
(1,0)	756	2048	864	0.5583
(1,1)	1312	1436	720	**0.4140**
(1,2)	936	1902	288	0.6084
(2,1)	756	2048	864	0.5583
(2,2)	1132	1832	576	0.5175

.	.	.
.	.	.
.	X	.

choice	*wins*	*losses*	*ties*	*l/w+l+t*
(0,0)	1456	1652	288	0.4864
(0,1)	900	2158	1008	0.5307
(0,2)	1456	1652	288	0.4864
(1,0)	1080	1868	576	0.5300
(1,1)	1652	1438	720	**0.3774**
(1,2)	1080	1868	576	0.5300
(2,0)	1276	1798	864	0.4565
(2,2)	1276	1798	864	0.4565

.	.	.
.	.	.
.	.	X

choice	*wins*	*losses*	*ties*	*l/w+l+t*
(0,0)	936	1652	1008	0.4593
(0,1)	936	1902	288	0.6084
(0,2)	1132	1832	576	0.5175
(1,0)	936	1902	288	0.6084
(1,1)	1312	1436	720	**0.4140**
(1,2)	756	2048	864	0.5583
(2,0)	1132	1832	576	0.5175
(2,1)	756	2048	864	0.5583

Observe that George Kingsley Zipf reasons that the center is his best move if it is available, and if not, a corner.

2. Crucial Second Moves: Potential forking

.	X	.
X	O	.
.	.	.

(*and permutations*)

choice	*wins*	*losses*	*ties*	*l/w+l+t*
(0,0)	40	24	36	**0.24**
(0,2)	48	34	12	0.3617
(1,2)	32	52	24	0.4814
(2,0)	48	34	12	0.3617
(2,1)	32	52	24	0.4814
(2,2)	56	32	0	0.3636

O	.	.
.	X	.
.	.	X

(*and permutations*)

choice	*wins*	*losses*	*ties*	*l/w+l+t*
(0,1)	28	40	12	0.5
(0,2)	28	34	24	**0.3953**
(1,0)	28	40	12	0.5
(1,2)	16	50	36	0.4901
(2,0)	28	34	24	**0.3953**
(2,1)	16	50	36	0.4901

3. The Granddaddy of boards: Double Potential Forking

If you inspect the statistics below, you will notice that Zipf's minimizing will not choose the correct output. This is not surprising: 'X' can fork the edges of the board using either the top-right corner or the bottom-left!

Meet the 'double fork'.

I found this situation extremely interesting, because it was the *only* edge case to my statistic. What ended up being the most elegant solution was performing a 45 degree transformation of the board and projecting the transformation downwards, such that 'X O X' rotated to (1,0), (1,1), (1,2). See `rotate()` in the Engine module for the specific map. This correctly positioned the statistics such that the minimum could be correctly selected for this board.

X	.	.
.	O	.
.	.	X

(*and permutations*)

choice	*wins*	*losses*	*ties*	*l/w+l+t*
(0,1)	40	34	12	0.3953
(0,2)	44	32	12	**0.3636**
(1,0)	40	34	12	0.3953
(1,2)	40	34	12	0.3953
(2,0)	44	32	12	**0.3636**
(2,1)	40	34	12	0.3953

transformation such that (0,1) or (2,1) are selected

↓

.	O	.
X	O	X
.	.	.

What this means for the board

↓

X	O	.
.	O	.
.	.	X

Improvements

1. Speed

   At the moment, George Kingsley Zipf takes one full second to make a decision about his first move. This is because he must search every node under each second possible move in the tree to accurately compute his statistic. I explored the possibility of alpha-beta pruning, and even wrote a version of `recurse_tree` (not included in the code package) that would break out of the tree searching early upon meeting a condition. Unfortunately, I only have wins, ties, and losses attached to each node, and these grow at different rates (see Tests.pdf for a visual). I tried only using one: minimizing bare loss counts, but this was not robust enough to make the right decisions.

2. Language block: more recursion

   Recursion makes sense for traversing trees, especially when the depth of your tree searching is variable. I have tried to show my recursive strategies in all corners of the application, not just in the Engine module. I would like to extend my solution to a language where recursion is not only more natural, but often necessary, like in Haskell.