**Analysis of heuristic scoring functions used for playing isolation**
**Christopher Rivera**

As part of the isolation project, I generated three 'families' agents using related heuristics. These families employed the following three scoring function forms:

1. $score = m_p - w * m_o$
2. $score = \lambda(m_p - w * m_o) - (1 - \lambda)\sqrt{(x_p - x_c)^2 + (y_p - x_c)^2}$
3. $score = sign(m_p - w * m_o) * (m_p - w * m_o)^k$

where $m_p$ and $m_o$ are the number of moves available for the player and opponent, $\lambda$ is a tuning factor that controls a tradeoff between the strategy the favors more moves and a strategy that favors being in the center, w is a weight that discount the opponents number of moves, x and y are positions on the isolation grid and k is an exponent. These heuristics are based on the heuristics presented in the class.

To start with I tried setting w to either 0.5 or 2, k to 2 and $\lambda$ to 0.5 and used a modified version of the tournament code. I noted that using these agents that the agent with scoring function $m_p$-0.5*$m_o$ seemed to play best.

To explore how altering these parameters would affect the play, I next wrote a series of scoring function constructors that allowed me to parameterize the scoring function so I that could investigate how varying the parameters affected win rate. I chose the parameters randomly drawing from a uniform distribution and had the resulting agents play the standard CPU agents for 10 (20) games each. I generated 20 agents of using the heuristics for form 1 and 2 and 10 agents using form 3. Overall it appeared that agents of the second form did the best (figure 1), where as agents using the second form did worse than the base line player.

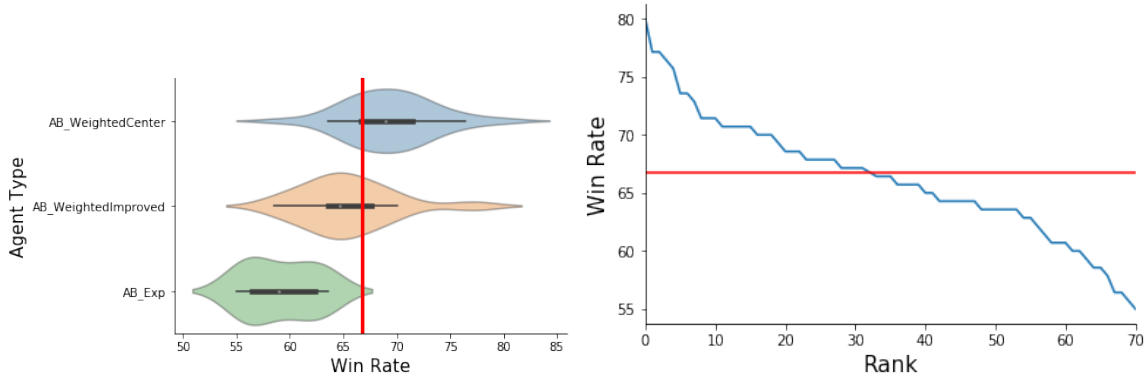


**Figure 1. a) Violin plot showing distribution of win rates for the three classes of agents b) The win rate versus the agent rank.**

To understand how the changing the weight affects the agents using equation 1, I plotted the win rate versus w and performed linear regression. Although extremely noisy, it seems that increasing w resulted in decreased win rate, indicating that more aggressive players

tended to do well (figure 2a). However the win rate of the agents that used a variant of equx2 did not appear to depend on w (data not shown), yet it decreased with increasing λ (figure 2b).

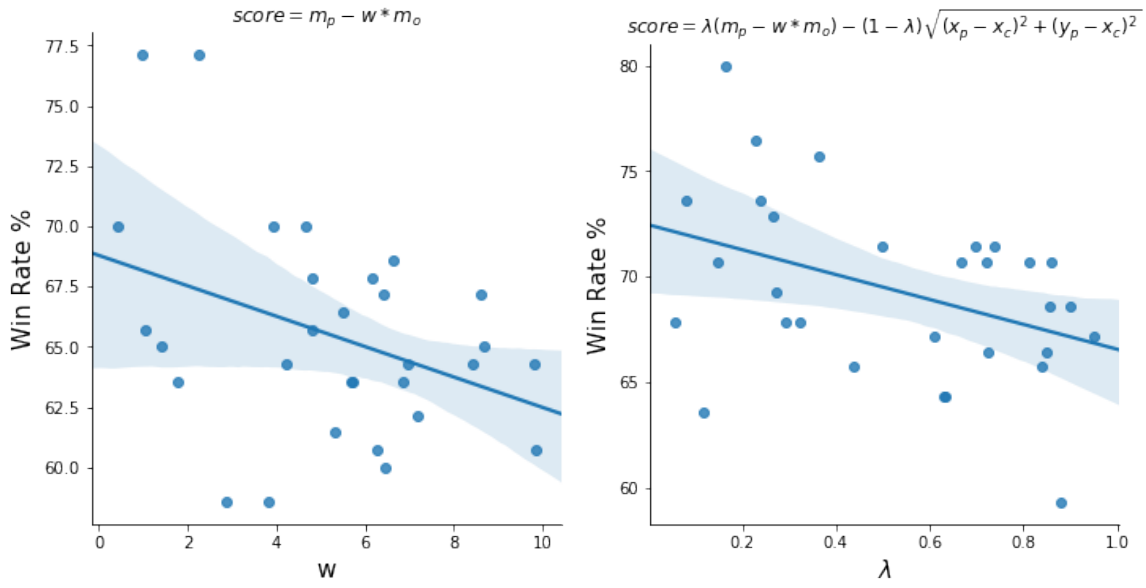


***Figure 2. a) The win rate tends to decrease with increasing w for agents employing first scoring function. b) The win for agents with form 2 tends to decrease with increasing*** λ.

In the experiments, my top 3 agents outperformed the baseline alpha beta agent (win rates: 80, 77.14, 77.14 versus 66.79 %).

To directly compare my best performing agent (which used the second scoring function with a w of 8.25 and a λ of.16), I competed it against the baseline agent in 200 matches in which the first player alternated and was allowed to choose its first move (unlike the tournament). It beat the baseline in 65% of the matches. Because this custom agent is drawn to the center, I reasoned that it would tend to have the same starting place for both its first move. Compared to he baseline agent, which started at either (2,2) or (1,0) depending on whether it went first or second, the custom agent always started at (2,1). This starting position may be superior, however further analysis and experiments should be done to confirm that.

In the future, I would do the following:

1. As described in the lectures, use a hash to generate the first moves.
2. Try using a mixed scoring function that varied upon the move number (I think that agents should be aggressive at staying in the center initially but the start to explore).
3. Try using machine learning or reinforcement learning to approximate the value/scoring function as in Alpha Go (this would require many more simulations than I conducted)

In conclusion, I generated a series of scoring functions based on 3 different forms, played random agents against the standard CPU agents and found various heuristics that are able to beat the baseline agent.