The Resistance Probabilistic Game Solver: Probabilistic Analysis Tool for Resistance

Matthew Brennan (brennanm)
Vincent Kee (vkee)

Overview: The Resistance Probabilistic Game Solver

The Resistance Probabilistic Game Solver is designed to assist users playing the Mafia-esque game, Resistance (rules at http://en.wikipedia.org/wiki/The_Resistance_%28 game%29). Players are divided into two opposing groups: spies and resistance. The spies know who the other spies are but the resistance do not and their goal is to find out who the spies are in five rounds of gameplay.

We decided to implement this because our living group enjoys playing Resistance. Everytime we play, the math majors end up trying to compute the probabilities that each person is a spy and determining who is a spy based on how logically people act. This ends up taking a significantly long time, making rounds take long. We decided that our project would benefit our living group by giving everyone the game probabilistic analysis and increase the fun we have when we play Resistance in the future.

Our analysis tool assumes that all players are playing in some way to their own benefit. We predict that the tool should work best if players are playing close to optimally, but because of the way we set up the probabilistic tool, it should also not produce unreasonable analyses even if players are making some arbitrary decisions. The tool outputs an estimate of the probability each player is a spy given the actions that have occurred in the game and also can output probabilities from each players perspective (assuming he or she is part of the resistance). Our implementation makes all deterministic observations that can be made. For example, if the failing missions so far imply that one player must be a spy then our tool will detect this and output a probability of one for this player. It also is intended to respond to other parts of the game such as how players vote on teams to be part of the missions.

The Code: Main and Simple Main Methods

The code is divided into several components: main.cpp, simple_main.cpp, game_spec.cpp, game_tree.cpp and r_game.cpp.

1. main.cpp contains the main loop where the user input is entered and probabilistic information is displayed for the more complicated method.
2. simple_main.cpp contains the main loop where the user input is entered and probabilistic information is displayed for the simpler method.
3. game_spec.cpp contains the game specifications. It generates all the possible subsets of players that can go on each round.

4. game_tree.cpp contains the components used to generate trees used to estimate the weighted fraction of outcomes resulting in wins for each possible game state for both the resistance and the spies.
5. r_game contains methods that update overall game statistics and player statistics as well as returning them.

The Method: Probabilistic Model of Decision Making in Resistance

The objective of our method is to estimate the probability P(X|G) where X is random variable denoting the set of spies among the players and G is the sequence of events that have transpired so far in the game. Formally, X takes on values equal to the k-subsets of {0, 1, … , n - 1} where k is the number of spies and n is the number of players. The prior probability P(X = x) is initialized to 1/(n choose k) for all subsets x of {0, 1, ... , n - 1}.

We implement two methods to estimate P(X|G), both of which update P(X|G) with the estimates of the conditional probabilities P(A|X = x) for the next action A in the game given each possible subset x of spies. The update is carried out by computing P(A|G) = sum of P(A|X = x) P(X = x | G) over all subsets x, and by then computing P(X|A,G) = P(A|X) P(X|G) / P(A | G). The simpler first method estimates P(A|X = x) directly using several parameters and is implemented in simple_main.cpp. The more complicated second method estimates P(A|X = x) by constructing a tree of the possible sequences of actions in the game and basing P(A|X = x) using the fraction of outcomes at each node in the tree that lead to a victory for the spies or the resistance. This method is implemented in main.cpp.

The simpler first method uses two constants coord_confidence and points_confidence, which are initialized to 0.8 and 0.6, respectively, to determine P(A|X = x). These confidences represent the bias towards spies not being able to coordinate and plan different actions if voted to go on a mission together and the bias that spies will attempt to acquire points in their favor rather than pass missions. The more complex second method bases P(A|X = x) on coord_confidence, mission_confidence and voting_confidence. The latter two constants represent the confidence that spies and resistance members will play optimally when voting on whether or not to pass a mission or vote on a team. The method also bases P(A|X = x) on a tree of all possible sequences of outcomes given the current state G of the game. Specifically, it computes a weighted estimate T(v, x) of the fraction of the outcomes in the subtree rooted at node v that will lead to a win for the resistance assuming x is the set of spies. Leaf nodes v in the tree in which the resistance win have T(v, x) = 1 and those in which the spies win have T(v, x) = 0. The estimates T(v, x) are then computed as the average of the values T(c, x) for all children nodes c of v. The probabilities P(A|X = x) over all possible actions A are modelled based on the win estimates T(A, x) for all possible next actions A in game state G. The actions A considered are selecting a team for the next mission and failing or passing the next mission.

From the probabilities P(X|G), we compute an estimate of the probability that each player is a spy with the formula: P(m is a spy | G) = sum of P(X = x | G) over all sets x such that m is in x. We output these values at each step of the game as well as similar person-specific statistics.

Alternative Approach: Potential Future Topic

One more extensive idea is create a tree that takes into account additional actions A such as which players vote in favor of teams chosen for each mission. This would enable the probabilities of all actions to be modelled from the tree by directly computing the probabilities of winning (rather than giving a uniform estimate as we do). This would require many more layers in the tree and much more computation time and memory. As our process is already memory-intensive, we chose not to investigate in this direction. However, we believe that this would be an interesting potential future topic.