Chris Meill
Bharath Sitaraman
CS 229 – Fall 2009

RISK: A Case Study in Applying Learning Algorithms to Strategy Board Games

Chris Meill, Bharath Sitaraman
cmeill, bharath@stanford.edu

Abstract

Our goal was to create a learning algorithm that can play the classic board game, Risk, a game in which players control territories that produce armies, which the players can use to conquer the territories of others. It offers an intriguing environment in which to deploy an AI player, with many different decisions to be made, and a wide variety of gameplay aspects affecting those decisions. We first implement several deterministic AI players that will focus on maximizing separate elements of the game, therefore responding formulaically to isolated game situations. Our learning algorithm used these AIs as training partners. However, it adapted to game situations based off of a decision tree where we adaptively weighted our possibilities of achieving future situations and the relevancy of distances to surrounding territories. This AI played multiple games in order to generate more diverse data, and is optimized to increase its effectiveness.

1. Introduction

1.1 – Motivation

Risk tends to be a very entertaining game due to the constantly evolving situations that unfold during repeated sessions. However, from personal experience, we have found that most AI players in computerized versions of Risk tend to fall into predictable (and thus, exploitable) patterns, making these versions of the game rather trivial to beat. Granted, the game is somewhat cyclical in nature (spend a few turns building up an army, go out on an attack, fix up defenses, build up another army, etc.), so even human players can tend to fall into patterns. However, many of the best of them do not. The best strategies in Risk are adaptive, reacting to game situations as they unfold, rather than rigidly adhering to some sort of master plan.

Additionally, there are many "unwritten rules" to Risk that any human player figures out very quickly, which many AI's tend to struggle with. For example, while conquering the entire continent of Asia provides an appealingly large unit production bonus, it is virtually impossible to maintain control of the entire continent. It simply has too many borders and its centralized location frequently makes it a staging ground for conflict. On top of this, when a player controls Asia, that same appealing unit production bonus makes him an immediate top-priority threat that other players will concentrate on dealing with. Our hope is that a learning algorithm could identify these types of counterintuitive situations and behave more intelligently than a deterministic AI, without being explicitly programmed to do so.

1.2 – Problem Definition

The problem can somewhat be boiled down to a classification task, in which our algorithm will be choosing it's set of actions depending on the current status of the board. If it's in a powerful position, it should tend to attack more. When troops are low, it will want to defend and increase troops at its borders in order to stave off impending attacks from opposing players.

1.3 – Dataset

We actually did not start with any sort of dataset. We gave our algorithm arbitrary weights, which it will gradually adjust and converge using logistic regression over time. Our dataset will actually be generated through playing numerous games. While pitting our deterministic AIs against one another, we will be generating data and storing it into a file. It may be prohibitively slow to re-train the AI every game, we'll probably have it play games in clusters, and then re-train to apply what it learned after each set of games. It then starts with it's optimized weights from the previous games, and will adjust them in accordance with the current game.

2. Approach

Currently, we have recreated Risk as a text-based game played through the consol, with a Player super-class that is queried throughout the game for decisions (for example, the Human Player extension of this class simply prompts the user for input). We have implemented a system of reading preconfigured game boards from a file, so that we can easily create desired situations, rather than hoping that they unfold as the game progresses. This also allows us to play on various board sizes, and scale the game up and down in complexity.

2.1 – Deterministic ("Dummy") AIs

We have produced 6 "dummy" deterministic AI players, which basically respond formulaically to the current state of the board. Each AI's behavior is an exaggeration of certain tactics or playing styles frequently employed by human players. They are listed below:

1) Aggressive – attack everything possible
2) Bully – attack whenever victory is essentially guaranteed
3) OCD – aggressively maintain current territorial holdings
4) Border Patrol – just defend borders
5) Wanderer – put all armies in one "horde" that wanders around the board
6) Distributor – distribute all armies evenly across territories

Originally, our dummy AIs were going to help our learning algorithm train based off of different game situations and help it act appropriately depending on what kind of position it was in. Eventually, they just ended up acting more like sparring partners, allowing the AI to retrain it's weights to adjust to each different AI, which in turn would affect it's actions. Studying their patterns and habits also enabled us to observe and create a mix-and-match strategy We tried to cover as many bases as possible in terms of playing style with each AI so that after multiple games, the AI would be able to adapt much more quickly to future opponents.

The Aggressive AI and the Bully AI definitely surprised us. We expected in a set of trials that they would probably win the majority of them, but the speed of the games and the adeptness with which they defeated their opponents reinforced our belief that the AI should attack when given the opportunity rather than wait. This actually helped simplify our process somewhat.

2.2 – Attacking Strategy

Our algorithm learns by optimizing a set of two different discount factors α and γ. During a game when it is the AI's turn to attack, it will consider the full set of available actions A (including A_0: the "do nothing" action that indicates the end of a turn), where each action A_i is a tuple containing the territory used to attack, the target of this attack, and the number of attacking dice rolled. For each of these actions, there is some uncertainty about the resulting game state due to the randomness of the dice, so the perceived value of this action is assessed to be the sum of each possible resulting state reward weighted by the probability of ending up in this state.

The reward for a resulting state is calculated using several different factors. The first is $F_a^{(0)}$, the current fighting force of the attacking AI, which is the sum of armies from all its territories weighted by relevancy to the current action. This relevancy is determined by the distance away from the territories involved in the action using the discount factor α. For example, if there were n troop in a territory d hops away from our action, it would contribute $\alpha^d * n$ to $F_a^{(0)}$. Similarly we have $F_d^{(0)}$, the current fighting force of the defending opponent. There is also $P_a^{(0)}$ and $P_d^{(0)}$, the number of armies produced by each player at the beginning of their next turn given the current board state. The resulting state will have values $F_a^{(1)}$, $F_d^{(1)}$, $P_a^{(1)}$, and $P_d^{(1)}$, which are simply these values recalculated for the potential resulting state. The reward is then:

$$\frac{F_a^{(1)} + P_a^{(1)}}{F_d^{(1)} + P_d^{(1)}} - \frac{F_a^{(0)} + P_a^{(0)}}{F_d^{(0)} + P_d^{(0)}}$$

or the difference between the ratios of how "well off" the attacker AI is compared to its opponent after taking this action.

The resulting state's reward, however, need be calculated only if the AI's turn were to end after that action. Since an AI can take many actions over the course of one turn, these resulting states can be further expanded by considering the new set of actions available at that state. This creates a tree structure terminating in A_0 actions nodes whose values are calculated using the formula above. This value is passed upwards through the tree to the state node S that was considering this action. S determines its reward to be the maximal value of the A's available to it, weighted by the discount factor γ. In this sense, γ determines the relative importance that our AI places on future possibilities when considering actions.

2.3 – Learning

The learning comes in the two discount factors α and γ. Essentially, they are found to the optimal values for the effectiveness of the AI (the percentage of games it is observed to win). We update them alternating between the two. For some given α, the learning process plays through many games to find the optimal value for γ, and then the resulting γ is then used to assess the effectiveness of the algorithm. In this way we create a series of training examples [$\alpha^{(i)}$, $\omega^{(i)}$], where $\alpha^{(i)}$ is the discount factor used and $\omega^{(i)}$ is the observed effectiveness of using this $\alpha^{(i)}$. Initially a set $\alpha^{(i)}$ is chose across the range [0,1], which yields a set of training examples that we can apply a polynomial regression to. The resulting function W can then be used to find the optimal α that maximizes effectiveness. For this optimal α we can find an optimal γ, which together can be assessed by the new effectiveness ω. Together with α, this ω creates a new training example, which when added to the training set produces a new *W*, which in turn can be used to find an updated optimal α. This process is repeated until the updates to α and γ converge.

At each iteration of this learning, the optimal γ is found for a given α in a similar fashion. A set of [$\gamma^{(i)}$, $\omega^{(i)}$] training examples is computed which is used to train a polynomial regression function *W*. The value of γ that optimizes this *W* is then assessed to find a new ω, which creates a new training example that is added to the training set.

2.4 – Other Decisions

Defending:
This action was strictly deterministic. If the attacker were to roll one die, we would defend with two die (if possible) no matter what. Rolling aggressively against the attacker's single die gives us a strict advantage in that we have more chances of winning the roll.

In the case that the attacker rolled multiple die, given a choice, we would defend with two die only when their lower dice roll was a 3 or less. Rolling on a 3 was somewhat of a judgment call that we made deterministically since it has approximately a 50% probability of either player winning the roll. Otherwise, we would only defend with one dice.

Fortifying:
For this algorithm, we actually would apply our own attacking learning algorithm in terms of our opponent's troops and fortify our troops accordingly in order to minimize our expected losses.

3. Future Considerations

1) Be able to read and train on game situations using somewhat of a reinforcement-learning algorithm. Through playing multitudes of games, we would be able to see favorable outcomes (conquering a continent, eradicating an opponent's strong troop base, etc.) and be able to use those in our current learning scheme.
2) We made modifications in order to make the game of Risk simpler. For example, we created randomized balanced board states at the beginning of games. We also did not introduce the concept of risk cards in the game since that would add an extra element to the game.
3) Currently, we have a deterministic defending strategy. We made a judgment call based on the result of an attacker's roll that human players frequently use in order to simplify the problem. However, we would like to have let the AI come to this decision itself based on the board state.