Chapter 1

“Reasonable” Strategy Selection for Tree Games

Contributed by Justus Hibshman

1.1 Introduction

Ever since awareness of classic games such as The Prisoner’s Dilemma and tree games such as The Centipede Game [3], game theorists have realized that rational play by both agents leads to extremely un-intuitive outcomes.

The Prisoner’s Dilemma is storied as follows: Two fellow criminals are caught and held separately. They must independently decide whether or not to testify against the other. If they both keep quiet, they make off with a light sentence. If one testifies and the other doesn’t, the testifier goes free and the other is in prison an extremely long time. If both testify, they receive a moderately long sentence. What do they do?

This game has the interesting property that no matter what one player does, the other will fare better by testifying against them. Then, according to classic game theory reasoning, they both testify against each other and get the outcome that is net worst.

In The Centipede Game (CG) (Fig: 1.1), players take turns deciding whether or not to stop or continue the game. When the game is stopped, both players receive a “reward.” The longer the game goes, the higher the rewards get, but the rewards are always biased towards the player who stops the game. The game always has a final round, in which the last player may choose between two outcomes.

These conclusions follow directly from the following assumptions:

1. Both players’ goal is to maximize reward for themselves.

2. Both players are rational.

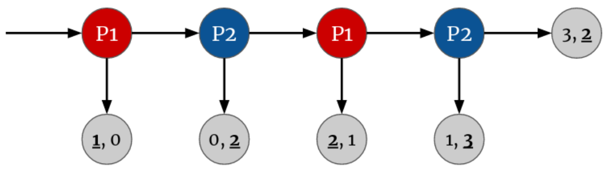

Figure 1.1: A Centipede Game

3. Both players know (or assume) the other player's goal is to maximize their own reward.

4. Both players know (or assume) the other player is rational.

5. Both players know (or assume) 3., 4., and 5. [**Note: Does this make sense?**]

When these assumptions are applied to the CG, a strange but very straightforward "backward induction" applies: If the game gets to the last round, the player whose turn it is, being completely selfish and having complete reign over the outcome, will choose the outcome more favorable for themselves. Thus, the other player can conclude this, and if the game ever reaches the second-to-last round, they will know to end it there. This chain of reasoning follows all the way until the first round, telling the starting player to end the game right away.

In a paper proving the validity of backward induction (given a definition of "rational"), Aumann have hinted that a "solution" in these sorts of cases is to pretend the other player is not rational or at least might not be rational [1]. Ignoring available data hardly seems to be a pleasing solution to a fascinating dilemma; it seems that a satisfactory solution concept requires a *goal* that's different from selfishness.

Some have proposed new solution concepts, such as "Iterated Regret Minimization" [2] wherein the goal is no longer maximizing one's reward but rather to minimize a mathematical notion of possible regret; this particular solution concept leads to more intuitively "correct" behavior but has an "epistemic" flaw. The authors note this flaw but then rebutt it with an argument which fails to be compatible with the idea behind the "iterated" component of their solution concept (In short: the "Regret Minimization" component assumes uncertainty about the other player's strategy; the "Iterated" component assumes the other player is also using Iterated Regret Minimization.).

As a step towards more "Reasonable" goals, in this paper I suggest two goals which may be thought of as behavioral goals; that is, they are expressed in terms of what an agent desires to do rather than what outcome they desire. The goals are assumed to be held by both players and is defined in terms of a *pair* of players' strategies.

Goal 1 - "Avoiding Strategy Modifications with Anticipated Negative Responses": Only use strategy pairs where neither player would desire to deviate to something better for them given that the other player got to adjust their strategy in response (For future work I intend to consider what happens if these back-and-forth responses can continue indefinitely rather than just once.). This can intuitively be thought of as avoiding instances of, "Oh, well, if you were going to upset the nice little agreement we had here [i.e. the original strategy pair], then I'd have done this other thing."

For a game with 2 players let s_i denote the strategy of the ith player and $R_i(s_1, s_2)$ denote the reward player i receives given the strategies s_1 for player 1 and s_2 for player 2. Goal 1 may now be mathematically expressed by saying (s_1, s_2) is valid if

$$\forall s_1' \neq s_1.(R_1(s_1, s_2) \geq R_1(s_1', s_2)) \vee (R_1(s_1, s_2) \geq R_1(s_1', argmax_{s_2'}\{R_2(s_1', s_2')\})) \tag{1.1}$$

and the reverse for all s_2'.

Goal 2 - "Allowing for Trust": Only use strategy pairs where a player couldn't do better by fixing their strategy and then letting the other player change. This can be thought of as rejecting strategy pairs where a player would say, "Wait. Wait. Let's be reasonable now. How about you trust me to keep doing what I'm doing now? Then you can do this thing that's better for both of us."

For a game with 2 players, Goal 2 may be mathematically expressed as saying a strategy pair (s_1, s_2) is valid if

$$R_1(s_1, s_2) \geq R_1(s_1, argmax_{s_2'}\{R_2(s_1, s_2')\}) \tag{1.2}$$

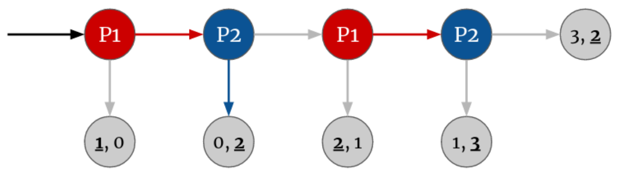

Figure 1.2: A Strategy Pair - Highlighted Edges are the Chosen Edges

and the reverse for all s_2'.

And-ing the two goals together, we get that a pair of strategies is considered valid if:

$$\forall s_1'.(R_1(s_1, s_2) \geq R_1(s_1', s_2) \wedge s_1' \neq s_1) \vee (R_1(s_1, s_2) \geq R_1(s_1', argmax_{s_2'}\{R_2(s_1', s_2')\})) \qquad (1.3)$$

and the reverse for all s_2'.

1.2 The Problem as a Graph

While the notions of valid strategy pairs defined above can be applied to all games, I will be looking at how they can be computed for tree games (the simplest version of an Extensive Form Game).

In tree games, all edges are directed and there are two kinds of nodes - decision (internal) and terminal (leaf). Gameplay consists of players selecting a path down the tree to a leaf node. Decision nodes are labeled with a player; that player gets to choose which of that node's outgoing edges is traversed. Terminal nodes are labeled with payouts for all players. Once a terminal node is hit, the players get the payouts from that node and the game is over.

In this setup, strategies may be thought of as subsets of edges. Specifically, a player's strategy consists of a set containing exactly one outgoing edge from each of that player's decision nodes. This represents what the player would do if the game path contained any given node.

To make things cleaner, we will only consider strategies to select edges from nodes in subtrees that the player directs the game path towards. For instance, in Fig: 1.2, player 2's strategy is to end the game as soon as possible, so they don't make any choices about what they "would" do if the game went longer because their earlier action makes such a scenario impossible.

1.3 Some Realistic Data Sets

Currently I am not aware of any data sets. Analysis of these games is generally of the theory, or of gameplay by AI or humans on choice games. Analysis of computational runtime is not frequent.

However, it should be relatively easy to randomly generate games. It should also be relatively easy to generate large versions of a structured game such as the centipede game or a game with a high branching factor.

1.4 RSS-A Key Graph Kernel

The simplest kernel computes all pairs of strategies that satisfy the mathematical definitions of the goals described in the introduction:

$$\forall s_1'.(R_1(s_1, s_2) \geq R_1(s_1', s_2) \wedge s_1' \neq s_1) \vee (R_1(s_1, s_2) \geq R_1(s_1', argmax_{s_2'}\{R_2(s_1', s_2')\})) \qquad (1.4)$$

and

$$\forall s_2'.(R_2(s_1,s_2) \geq R_2(s_1,s_2') \wedge s_2' \neq s_2) \vee (R_2(s_1,s_2) \geq R_2(argmax_{s_1'}\{R_1(s_1',s_2')\},s_2')) \tag{1.5}$$

As a relatively straightforward computation, this can be done in several phases:

1. Compute $R_1(s_1,s_2)$ and $R_2(s_1,s_2)$ for all combinations of s_1, s_2.
2. Using 1., compute $argmax_{s_2}\{R_2(s_1,s_2)\}$ for all s_1 and vice versa. Store the results in a lookup format.
3. Using 1. and 2., compute

   $$\forall s_1'.(R_1(s_1,s_2) \geq R_1(s_1',s_2) \wedge s_1' \neq s_1) \vee (R_1(s_1,s_2) \geq R_1(s_1', argmax_{s_2'}\{R_2(s_1',s_2')\}))$$

   and vice versa for every (s_1,s_2).

The number of possible strategy pairs is upper-bounded by the product of the out-degrees of each node. This is then exponential in the number of decision nodes. In practice it may be smaller.

This problem already can be seen to have a lot of dynamic programming potential and a lot of opportunity for parallelism over strategies at each phase. If a more efficient algorithm can be found, parallelism might become harder.

1.5 Prior and Related Work

This is space to add in discussion of **prior work** - word on the same problem or kernel that your paper assumes, and **related work** - work on the same application but using different approach or kernel, or a different but similar application..

1.6 A Sequential Algorithm

Discuss here the outlines of a sequential algorithm. What programming paradigms might make the most sense? What are the key data structures? Does the computational complexity differ from that in the Section 1.4?

1.7 A Reference Sequential Implementation

Discuss here your implementation of the basic sequential code. Include what language/paradigm you used for the code.

1.8 Sequential Scaling Results

Discuss here results from your sequential implementation. Include software and hardware configuration, where the input graph data sets came from, and how input data set characteristics were varied. Did the performance as a function of size vary as you predicted?

1.9 An Enhanced Algorithm

Discuss here the outlines of an enhanced algorithm. This could be a parallel code, a code with some significant heuristics, or a code written in a non-traditional programming paradigm. Pseudocode is fine. Discuss what you think is the computational complexity.

1.10 A Reference Enhanced Implementation

Discuss here an implementation of the enhanced algorithm. Include what language/paradigm you used for the code.

1.11 Enhanced Scaling Results

Discuss here results from the enhanced algorithm. Include software and hardware configuration, where the input graph data sets came from, and how input data set characteristics were varied. Ideally plots of performance vs BOTH problem size changes AND hardware resources are desired. Did the performance as a function of size vary as you predicted?

1.12 Conclusion

Summarize your paper. Discuss possible future work and/or other options that may make sense.

1.13 Response to Reviews

This will be included only in the second and third iterations, and will be a summary of what you learned from the reviews you received from the prior pass, and how you modified the paper accordingly.

Bibliography

[1] Robert J Aumann. Backward induction and common knowledge of rationality. *Games and Economic Behavior*, 8(1):6–19, 1995.

[2] Joseph Y Halpern, Rafael Pass, et al. Iterated regret minimization: A new solution concept. *Games and Economic Behavior*, 74(1):184–207, 2012.

[3] Robert W Rosenthal. Games of perfect information, predatory pricing and the chain-store paradox. *Journal of Economic theory*, 25(1):92–100, 1981.