1 Summary

We intend to implement an efficient game time optimal (GTO) solution to heads up (1v1) no limit texas hold em post flop (after the first 3 cards have been shown). More specifically, given both players ranges, stack sizes, and possible bet sizes, return the best possible move for everything in your range.

2 Background

A naive approach to this algorithm would be to build out every possibility in the tree, and pick the path that maximizes your expected value. At every step it is either your turn or your opponents, so enumerate every possible move at each step and calculate the outcome of each. This tree grows exponentially with every decision, and to allow for every card combination the tree size will grow to an unsustainable level. Our goal is to find a way to prune the tree from going down impossible/unfavorable paths, and even memoize the paths that we have seen before. Additionally, alot of the paths on the tree do not depend on one another, so we can compute these in parallel.

3 The Challenge

If we limit ourselves to just raise, check call, and fold, this tree would contain roughly $9 \times (44^3)^2 \times (43^3)^2 \times \text{sizeofrange}$ nodes. This is such a large number that it was previously considered impossible to calculate in any reasonable amount of time. However, in recent years people have found ways to reduce the amount of computation to construct such a tree in minutes. We want to be able to come up with our own way to prune paths and reduce computation without losing alot of accuracy, and also be able to make our tree calculation as parallel as possible to maximize speedup. Additionally, we are going to only use one computer for our project, which means we will have a limited amount of memory available to us. Although caching tree results will speedup similar computations, we will not be able to store the entire tree in memory because of its size.
4 Resources

Because the end (possibly not achievable) goal of this project is to compute the GTO strategy in real time at home, we want to only use one computer for this project. However, because of the immense amount of computation that is needed, we want to use a machine that is very powerful. We are wanting to utilize the cuda to maximize performance, so we want a machine with a powerful GPU, like the Nvidia GTX 1080. If we find that cuda isn’t the best way to optimize this code, then we would want a machine with alot of RAM and a large amount of execution contexts because there is alot of independent work that would be best parallelized.

Although we have no starter code to work with, there already exists an application that does what we are wanting to do (piosolver), however this application is designed for a common laptop. Because we will be using a more powerful machine we hope to beat the benchmarks set by piosolver.

5 Goals and Deliverables

There is already a project that has an implementation for what we are doing, but it is designed to be run on an ordinary laptop. For a common case, it can calculate the tree within 0.25% accuracy in around 760 seconds. Because we are going to be using a more powerful machine than a common laptop, we hope to beat this benchmark.

One stretch goal we have is to make this project fast enough for real time use. In order to make this usable in a real game, you would need to have the computations be done in under 30 seconds. This number is a huge stretch goal because if a program was made that was efficient enough for real time use it would break online poker. We hope that with the use of parallelism on a GPU and a machine with much higher RAM than a common laptop we would be able to come much closer to real time efficiency than has been shown in modern applications of a GTO poker solver.

Depending on our efficiency, we hope to be able to show a live demo of our project. We would have the user input a flop and the ranges and stack sizes for both players, and show the computation being done and the final output. We would also compare our solution to a brute force implementation of the tree computation, and also the leading product on the market.

6 Platform Choice

Because ideally this application would be used on a home computer to play poker, we do not want to utilize a cluster of machines for this computation. Additionally, because the computational complexity is so huge we will need a
powerful enough machine to be able to handle that in a reasonable amount of time. Furthermore, we are choosing to use C++ so that we can use the tools that we have learned throughout the class for parallelism so far.

7 Schedule

Monday, April 17: At this point we want to have fleshed out the implementation details for computing the tree. Additionally, we want to have a baseline tree computation finished for only post river (final card shown).

Monday, April 24: By April 24th we want to have finished the naive implementation for the post flop tree calculation. This does not have to be optimized, but it should be able to make the final calculations.

Monday, May 1: Finish the implementation of our parallel version. This does not have to be finalized, but should be a good idea of how fast is practical for our project.

Monday, May 8: Finish the final implementation, begin working with benchmarking a naive solution and construct a ui for demo purposes.

Friday, May 12: Final product with ui finished along with proper analysis of speedup compared to our benchmarks and piosolver.