University of Lynchburg
Digital Showcase @ University of Lynchburg

Undergraduate Theses and Capstone Projects | Student Publications

Spring 4-27-2023

# Analyzing and Computing Complete Solution for Dots and Boxes Game

Carl McAninch
*University of Lynchburg*, mcaninc542@lynchburg.edu

Follow this and additional works at: https://digitalshowcase.lynchburg.edu/utcp

Part of the Computer Sciences Commons

## Recommended Citation

McAninch, Carl, "Analyzing and Computing Complete Solution for Dots and Boxes Game" (2023). *Undergraduate Theses and Capstone Projects*. 278.
https://digitalshowcase.lynchburg.edu/utcp/278

This Thesis is brought to you for free and open access by the Student Publications at Digital Showcase @ University of Lynchburg. It has been accepted for inclusion in Undergraduate Theses and Capstone Projects by an authorized administrator of Digital Showcase @ University of Lynchburg. For more information, please contact digitalshowcase@lynchburg.edu.

Analyzing and Computing Complete Solution for Dots and Boxes Game

Carl McAninch

**Senior Honors Project**

**Submitted in partial fulfillment of the graduation requirements of the Westover Honors College**

**Westover Honors College**

May, 2023

Randy Ribler, PhD

M. Zakaria Kurdi, PhD

Edward G DeClair, PhD

*Abstract* - **This thesis improves a process that analyzes all the states of a game of Dots and Boxes. We use retrograde analysis and simulations to create a solution that provides significant performance improvements over our previous best solution. Expanding upon a previous 4x4 solution using rotations, reflections, better optimization, and cloud computing to limit the processing time and gather more data efficiently. We compute a file and the number of states associated with each file and process every state starting with a completely filled board. We optimized the data for cloud computing by running simulations to find the most efficient number of processors and assess potential bottlenecks. The data produced from the results will be able to provide solutions and optimal play for dots and boxes games of different dimensions.**

**Keywords: Dots and Boxes, Retrograde Analysis, Simulation, Cloud Computing**

## I. Introduction

Dots and Boxes is a classic pencil and paper game played between two players. Historically it has been used as a game with a specialization in mathematics. The game begins with an empty grid of dots. Two players take turns drawing lines between the dots [7]. A box can be claimed when all 4 sides are completed and the player that completed the square wins the square. If a player wins a square then they must make another consecutive move. Dots and Boxes is considered a combinatorial game, where players move alternatively and when one player can no longer move the winner is the one with the most boxes [10]. Each move is related to the previous state of the board, so every position is a sub-position of another [10]. General strategy can be applied to a game of dots and boxes such as creating chains that switch the flow of the game in favor of a certain player. If a player is forced into giving away a long chain of boxes they will be at a disadvantage and eventually lose the game [3]. It has been found that the beginning and middle of the game is the most important time to take advantage of this strategy because there are less possible variations at the end of a game [7]. The goal of the game is to be in control of the endgame [2].

But knowing and using this general strategy is no match for a computer program with a complete solution to the game. A program is able to see all possible states of a 4x4 dots and boxes game and make decisions that will result in the most optimal play. Given a certain position, it will choose to make a line that will eventually lead to the maximum number of boxes won. Two perfect machines playing against each other will result in a tie; although, the player that goes first is found to be at a great advantage. The goal of this thesis is to compute a complete solution in the

most optimal way possible. A complete solution, or solved game, provides the best possible move for any position. The best possible move at any given point is also called perfect play, which gives the best move no matter what future or past moves will be.

The issue with a problem such as this is the sheer number of states that could possibly occur during a game and the bottlenecks that happen when files are waiting on the files they are dependent on. Most programs that aim to solve a problem such as this use a search algorithm such as alpha-beta pruning, minimax, and Monte Carlo tree searches to limit the number of states [1,6]. This method does not find a complete solution; however, processing every state and gathering data from this problem can be very costly in terms of processing. Adding to the dimensions of the board for the game increases the processing time exponentially. A previous solution to the 4x4 grid was obtained through the use of 50 different processors running over several hours or even days [9]. This solution suffered from bottlenecks, and boards with different dimensions than the 4x4 game would be too costly to compute. A solution that optimizes the process can reduce these bottlenecks and be more efficient during processing. This allows us to produce results for different dimensions than a 4x4 grid in an efficient manner. Data provided from this process provides indications to what makes a certain move the most optimal and can lead to stronger programs of several different dimensions. The goal of this research is to find the most efficient process to compute a complete solution and collect data from a 4x4 dots and boxes game that may lead to better solutions for games of other dimensions. While the topic for this project is dots and boxes, the techniques that we use to optimize the distributed processing can be applied in any application domain in which the computation time for individual partitions of the problem can be reliably estimated.

## II. Preliminary Data

Fig. 1. 4x4 dots and boxes game shows the result of the possible moves.

The original solution to the 4x4 dots and boxes game used 50 processors to obtain a complete solution after about 5 days of processing. The program can find the line that will result in the boxes won or lost at any given point in the game. Figure 1 shows the value each possible move has during the given game. The worst move is a loss of three boxes and the best move is a gain of five boxes. This will be the result of the game, following the most optimal game play, if it were to make that particular move. [9]

Previous data mining determined interesting data about a 4x4 dimension dots and boxes game. For example, the player who moves first in a game cannot make a losing move until the fifth move of the game, and the player who moves second can lose on their very first move. In

fact, for some first moves there are only two drawing moves out of the 39 available moves. It was also found that the lines towards the center of the board are usually completed first in perfect gameplay. The solution was saved and results of certain moves can be instantly viewed as in Figure 1.

To find these results the program breaks the board into 4 quadrants with 10 lines in each quadrant, as shown in Figure 2. The game is partitioned into files, named by the number of lines in each of the 4 quadrants, represented from 0_0_0_0 to 10_10_10_10. 0 represents a quadrant that is completely empty, while 10 is a quadrant that is completely filled. Each partition has a certain number of game states. The partition 10_10_10_10 and 0_0_0_0 only have one possible state, while the partition 5_5_5_5 has 4 billion possible states. Each game state is dependent on the previous state for retrograde analysis. A previous state is the result of removing one line from one of the quadrants. For example game state 10_10_10_10 is dependent on states 9_10_10_10, 10_9_10_10, 10_10_9_10, and 10_10_10_9.

It was found that this program was not very optimal in terms of the time it took to process each of the files. 50 processors were used on the University of Lynchburg campus over the course of 5 days to complete the solution [9]. The program did not take advantage of possible redundancies present such as reflections and rotations. It was also found that a relative few of the files make up a majority of the processing time. These features created bottlenecks that made the processing time inefficient. Implementing the changes we describe should result in a speedup in the processing time.

Fig. 2. Line quadrants

At most, each partition depends only on the partitions that are identical in all but one number, where that number is exactly one larger. The number of dependencies is often reduced when reflections and rotations are considered. Consider each partition represented as letters a, b, c, and d, where each letter represents the number of lines in each quadrant. Specifically, partition a_b_c_d is a rotation of partition b_c_d_a, c_d_a_b, and d_a_b_c. Partition a_c_b_d is a reflection of a_b_c_d, which itself has 4 rotations. As a result of these redundancies, we can find the complete solution computing only approximately 1/8th of the states that were needed to compute the original 50 processor solution.

In order to create a simulation for this program we generated a series of dependency graphs to keep a record of which partitions were required to complete processing before the next partition is able to be computed. The first dependency graph took into account all reflection and rotations by getting the normalized value for each partition. The second dependency graph added

variations of optimizations we created as shown in Figure 3. For example, we created a new dependency graph that broke larger partitions into smaller ones based on the number of states that those partitions had within them as shown in Figure 4.

```
5_5_5_5::4_5_5_5
5_5_5_6::4_5_5_6:5_5_5_5
5_5_5_7::4_5_5_7:5_5_5_6
5_5_5_8::4_5_5_8:5_5_5_7
5_5_5_9::4_5_5_9:5_5_5_8
5_5_5_10::4_5_5_10:5_5_5_9
5_5_6_6::4_5_6_6:5_5_5_6
5_5_6_7::4_5_6_7:5_5_5_7:5_5_6_6
5_5_6_8::4_5_6_8:5_5_5_8:5_5_6_7
5_5_6_9::4_5_6_9:5_5_5_9:5_5_6_8
5_5_6_10::4_5_6_10:5_5_5_10:5_5_6_9
5_5_7_7::4_5_7_7:5_5_6_7
5_5_7_8::4_5_7_8:5_5_6_8:5_5_7_7
5_5_7_9::4_5_7_9:5_5_6_9:5_5_7_8
5_5_7_10::4_5_7_10:5_5_6_10:5_5_7_9
5_5_8_8::4_5_8_8:5_5_7_8
5_5_8_9::4_5_8_9:5_5_7_9:5_5_8_8
5_5_8_10::4_5_8_10:5_5_7_10:5_5_8_9
5_5_9_9::4_5_9_9:5_5_8_9
5_5_9_10::4_5_9_10:5_5_8_10:5_5_9_9
```

Fig. 3. A section of the dependency graph using reflections and Rotations

```
5_5_5_5::4_5_5_5_0:4_5_5_5_1:4_5_5_5_2:4_5_5_5_3:4_5_5_5_4:4_5_5_5_5:4_5_5_5_6:4_5_5_5_7:4_5_5_5
_8:4_5_5_5_9:4_5_5_5_10:4_5_5_5_11:4_5_5_5_12:4_5_5_5_13:4_5_5_5_14:4_5_5_5_15:4_5_5_5_16
5_5_5_6::4_5_5_6_0:4_5_5_6_1:4_5_5_6_2:4_5_5_6_3:4_5_5_6_4:4_5_5_6_5:4_5_5_6_6:4_5_5_6_7:4_5_5_6
_8:4_5_5_6_9:4_5_5_6_10:4_5_5_6_11:4_5_5_6_12:4_5_5_6_13:4_5_5_6_14:5_5_5_5_0:5_5_5_5_1:5_5_5_5_
2:5_5_5_5_3:5_5_5_5_4:5_5_5_5_5:5_5_5_5_6:5_5_5_5_7:5_5_5_5_8:5_5_5_5_9:5_5_5_5_10:5_5_5_5_11:5_
5_5_5_12:5_5_5_5_13:5_5_5_5_14:5_5_5_5_15:5_5_5_5_16:5_5_5_5_17:5_5_5_5_18:5_5_5_5_19:5_5_5_5_20
5_5_5_7::4_5_5_7_0:4_5_5_7_1:4_5_5_7_2:4_5_5_7_3:4_5_5_7_4:4_5_5_7_5:4_5_5_7_6:4_5_5_7_7:4_5_5_7
_8:5_5_5_6_0:5_5_5_6_1:5_5_5_6_2:5_5_5_6_3:5_5_5_6_4:5_5_5_6_5:5_5_5_6_6:5_5_5_6_7:5_5_5_6_8:5_5
_5_6_9:5_5_5_6_10:5_5_5_6_11:5_5_5_6_12:5_5_5_6_13:5_5_5_6_14:5_5_5_6_15:5_5_5_6_16
5_5_5_8::4_5_5_8_0:4_5_5_8_1:4_5_5_8_2:4_5_5_8_3:5_5_5_7_0:5_5_5_7_1:5_5_5_7_2:5_5_5_7_3:5_5_5_7
_4:5_5_5_7_5:5_5_5_7_6:5_5_5_7_7:5_5_5_7_8:5_5_5_7_9
5_5_5_9::4_5_5_9:5_5_5_8_0:5_5_5_8_1:5_5_5_8_2:5_5_5_8_3
5_5_5_10::4_5_5_10:5_5_5_9
5_5_6_8_0::5_5_6_8
5_5_6_8_1::5_5_6_8
```

Fig. 4. The same section of the dependency graph using subpartitions

We created a discrete event simulation to analyze the process of creating a complete solution. The simulation uses a priority queue to track the completion of each partition and the flow of data between distributed processors. The priority queue represents the current files that

are being processed. No file can enter the priority queue until each of its dependent states have exited. The simulation time is represented by the number of states that are processed. The first simulation was run on an updated dependency graph that accounts for reflections and rotations. A reflection or rotation eliminates the need to process the same state multiple times. This addition brings the total number of states from $2^{40}$ to $2^{37}$ resulting in a substantial speedup. This first simulation showed that the maximum number of processors in use at one time is 25, and the average number of processors in use is between 6 and 7 as shown in Figure 5.

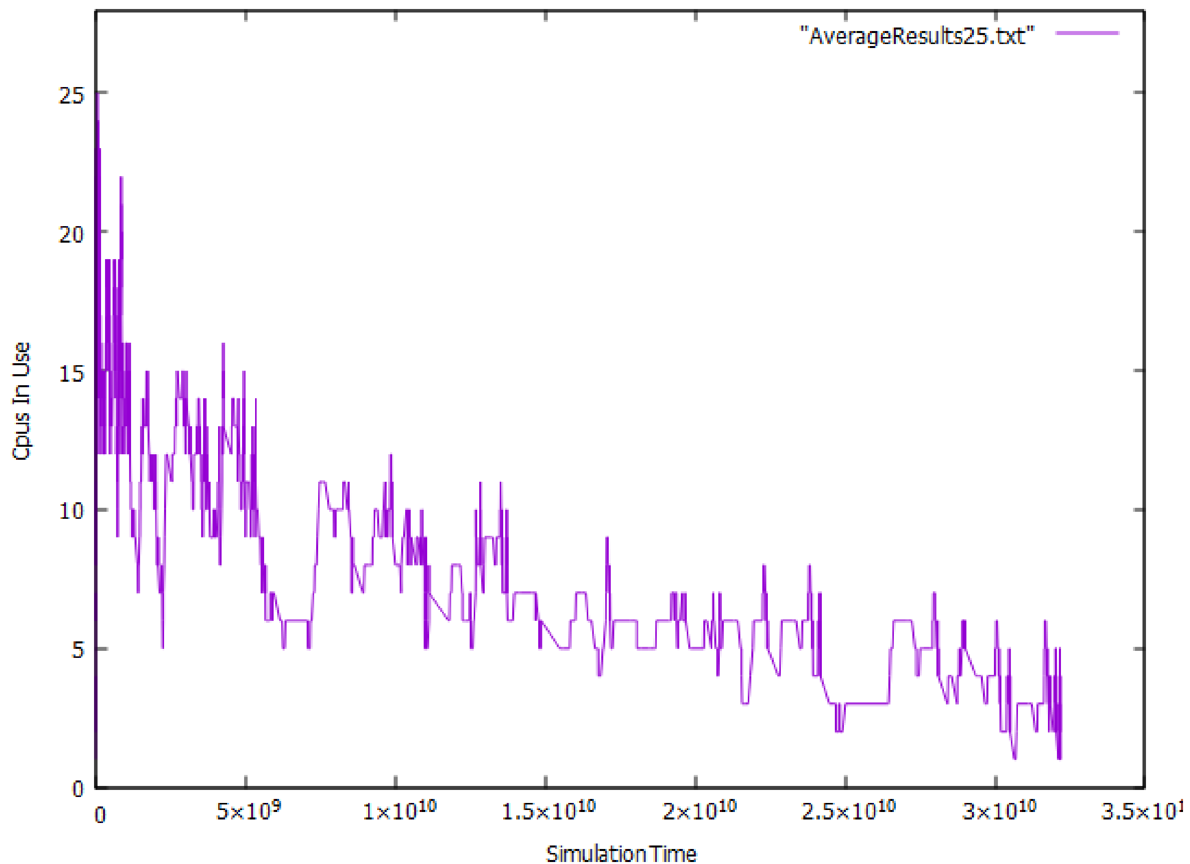


Fig. 5. Number of processors in use by Processing time

Further simulations tested the speedup between the number of available processors and total processing time. We added parameters to the simulation that limits the number of processors available. As the simulation ran, dependent partitions had to wait for an available processor in order to enter the priority queue. Waiting for an available processor adds extra time to the simulation and shows a realistic representation of what can happen based on the number of processors we are able to allocate. It was found that limiting the number of processors did not drastically change the simulation time after a certain point. Figure 6 shows the simulation run with 25 available processors compared to 1 and Figure 7 shows 10 available processors compared to 25. There is a drastic speedup if there is an unlimited amount of processors available. With only one processor there is no opportunity for parallel processing. An unlimited number of processors allows processing to begin as soon as dependent files are available. As the number of processors increases the wait time continues to decrease. Even with more than half of the available processors taken away there is not a drastic change in the simulation time. These results show the bottlenecks from certain dependencies and that the larger files that are taking up the bulk of processing time. Even though there are a large amount of processors free to take on a job there is no job to provide to them because the next jobs up are waiting for their dependent partitions to complete.

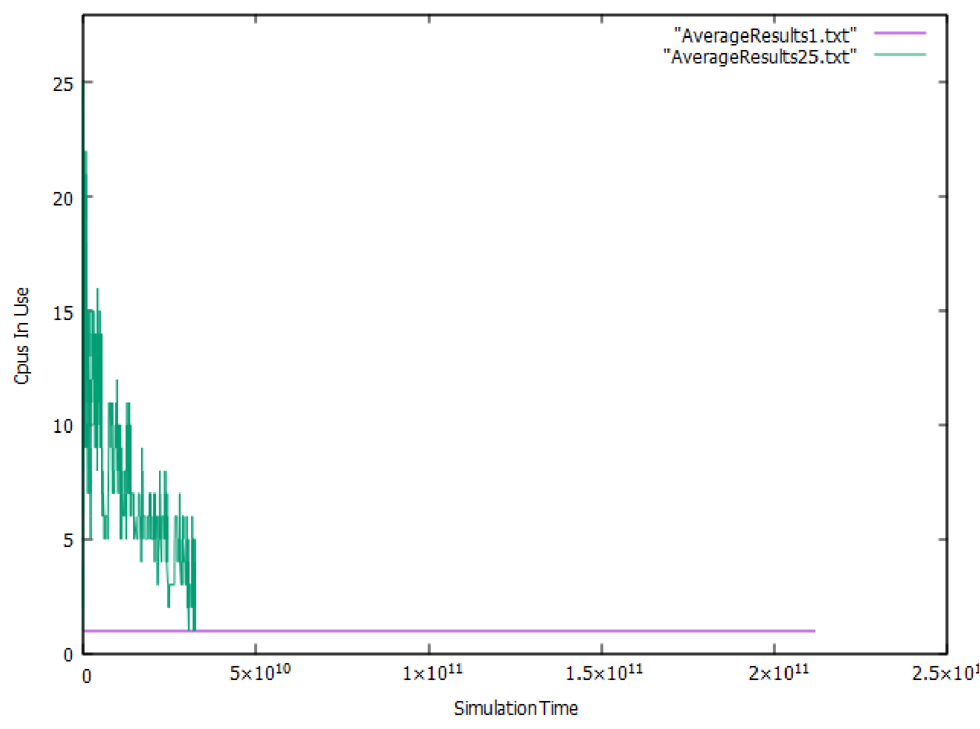


Fig. 6. 25 processors compared to 1

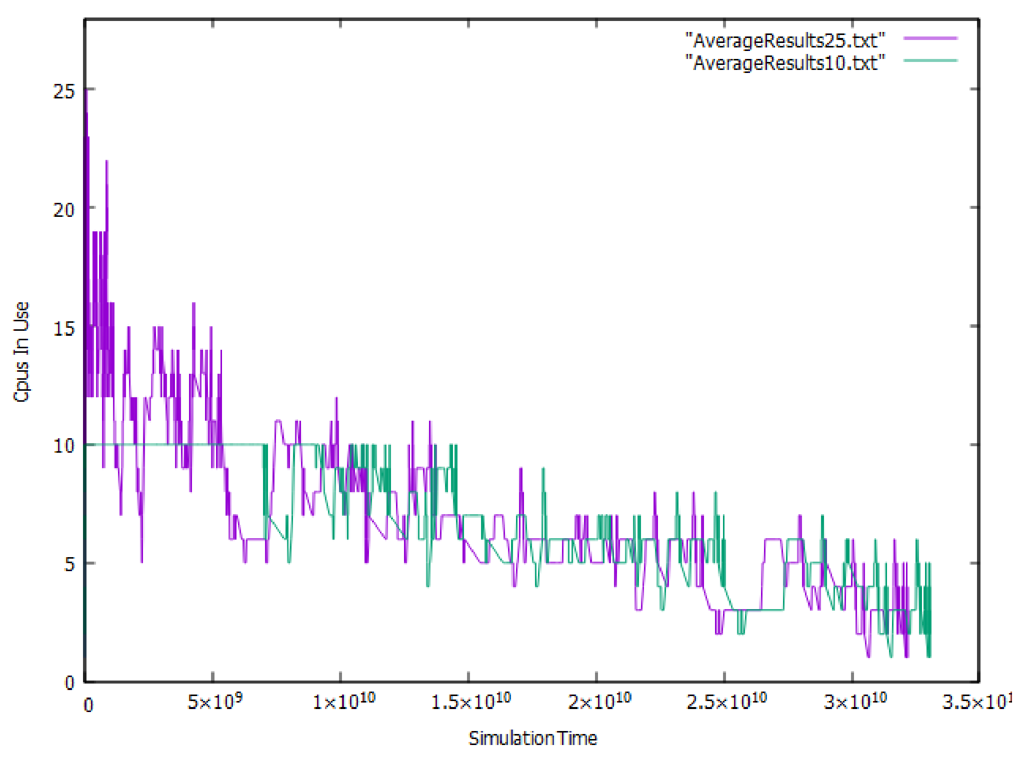


Fig. 7. 25 Processors compared to 10 (without subpartitions)

## III. Methodology

*Optimizations*

In order to find the most optimal process for computing all possible states of a dots and boxes game we applied certain optimizations. The first of which was including rotations and reflections in the dependency graph. Most game states are the exact same as others if the board were to be rotated or reflected. This eliminates the need to process those states multiple times. The number of states that need to be processed is reduced by approximately 1/8th, from $2^{40}$ to $2^{37}$. This is applied by designating one of the 8 representations as the "normalized" representation, and only normalized files are included in the dependency graph.

Even with rotations and reflections the simulations showed bottlenecks due to the dependencies. Large files, like 5_5_5_5 with over 4 billion states, take a long time to process. The dependent partitions are forced to wait for the completion of such files. In order to combat this, partitions with over 200 millions states are broken up into smaller subpartitions. Each partition is split up between 1 and 20 new partitions. We regenerated the dependency graph allowing for a large increase in parallelism between the partitions.

*Simulations*

Given the time and cost required to calculate a complete solution it is not practical to run this computation multiple times as we are making changes. Our simulation runs in a few seconds and is able to estimate the results of running and making changes to the program [5]. It provides an estimate of what we can expect when the program is actually running [5]. After a simulation is created we are able to run it many times over in a matter of milliseconds, instead of a period such as 5 days like the previous solution took. This allows us to make small changes to the

program and analyze the data in an efficient manner. The simulation created for this thesis simulates the process that will happen through HTCondor and cloud computing. The simulation time we use to estimate the time it will take to compute the solution is represented by the number of states in each file.

The simulation allowed us to constantly add stipulations and new methods to improve the processing time. The simulation receives a dependency graph containing each partition's dependent partitions. The dependency graph was used to create several different data structures including the number of states for each partition, all dependent partitions of a given partition, and the number of dependencies each partition had. The first simulation ran with no stipulations on the number of processors available. Beginning with the last possible state of 10_10_10_10, each partition was added to a queue as soon as every dependent partition exited the queue. The next simulation limited the number of processors that would be available. Assume the number of processors available was set to 10, if all 10 processors were occupied, represented by the number of items in the queue, then the partition to be added will be put in a separate queue to wait for cpu availability, consequently affecting the simulation time. Further simulations continued to add optimizations that would decrease the simulation time.

*Retrograde Analysis*

The program used in this thesis uses retrograde analysis to calculate the value of all the possible states that can occur in a given game. Retrograde analysis starts at the end of a game where the value of the game is known to be 0 and moves backwards through the previous states. In dots and boxes, the value of every state with k completed lines can be computed if we know the value of every state with k+1 complete lines. We know the value of the game with all the lines

filled in is 0, because there are no more boxes to win at that point. So in the case of the 4x4 dots and boxes game, we know the value of all states, the only state with k=40 lines is zero. To compute the value of a state with 39 lines we use the following logic:

```
if (the k+1 line completes any boxes)
        Value = nBoxesCompleted + value of the resulting state with k+1 lines.
else
        Value = the value of the resulting state * (-1)
```

Because of the requirements of having all possible resulting states available, each partition of the form a_b_c_d might be dependent on the partitions (a+1)_b_c_d, a_(b+1)_c_d, a_b_(c+1)_d, and a_b_c_(d+1). So each partition is dependent on a maximum of four other partitions. Because no partitions have numbers larger than 10, and because of redundancies caused by reflections and rotations, the average number of dependencies is less.

Each file and the states associated with that file is dependent on the previous file(s). The results of retrograde analysis tells the computer which line will give the greatest advantage. Retrograde analysis is a common method in games with many possible different states like chess and Go. Ken Thompson's "Retrograde Analysis of Certain Chess Endgames" provides a similar approach that analyzes files of chess states to find the best moves in endgames of chess[8]. Thompson's research analyzes certain positions at the end of a chess game, working backwards to find the most optimal play through those situations[8]. Chess games have many more possible states than a game of Dots and Boxes because of the number of pieces and board dimensions. The program in this thesis is able to use retrograde analysis to analyze every possible state from a completely filled board to an empty one.

*HTCondor*

In order to run this program concurrently on many different machines we use HTCondor. HTCondor controls the scheduling and execution of the individual partitions. It schedules and runs jobs as they are available to run, and transfers files as they are ready to be computed or when they complete. This method allows us to take advantage of the available processors on the cloud, creating an economical environment for the large number of states we have to compute.We wrote a program to provide condor with the dependency graph. This program must run from the last state all the way to the first and each dependent partition is required to be completed first, HTCondor is able to handle these dependencies very well. [4]

*Microsoft Azure Cloud*

This prepares us to run on Microsoft Azure Cloud, providing us with network speeds much higher than that of the University of Lynchburg's campus. This change in the process will create a substantial speedup even without the current optimizations. Computing the solution on the cloud also has several other advantages.

## IV. Results

We learned in our initial simulations that bottlenecks were created by several very large files. We broke those files down into smaller subpartitions and generated a new dependency graph. The new dependency graph including subpartitions showed a drastic decrease in the simulation time from the graph without these subpartitions. The new graph takes advantage of

the available processors to process states from the larger files. Figure 8 shows the average number of processors in use increases to 22 from the simulation without subpartitions. It also shows approximately a 24 times speedup from the original program and a 3 times speed up from the initial simulation as shown in Figure 9.

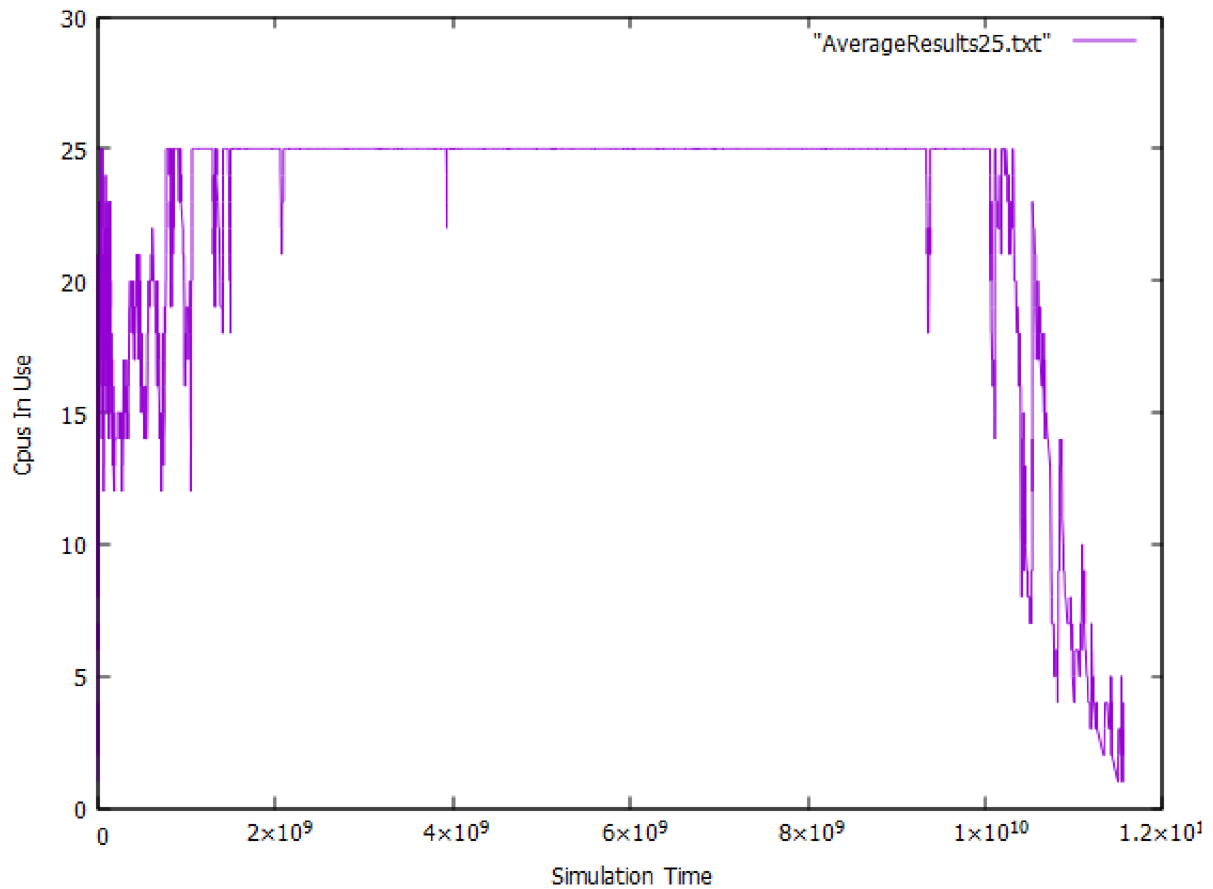


Fig. 8. 25 available processors with subpartitions

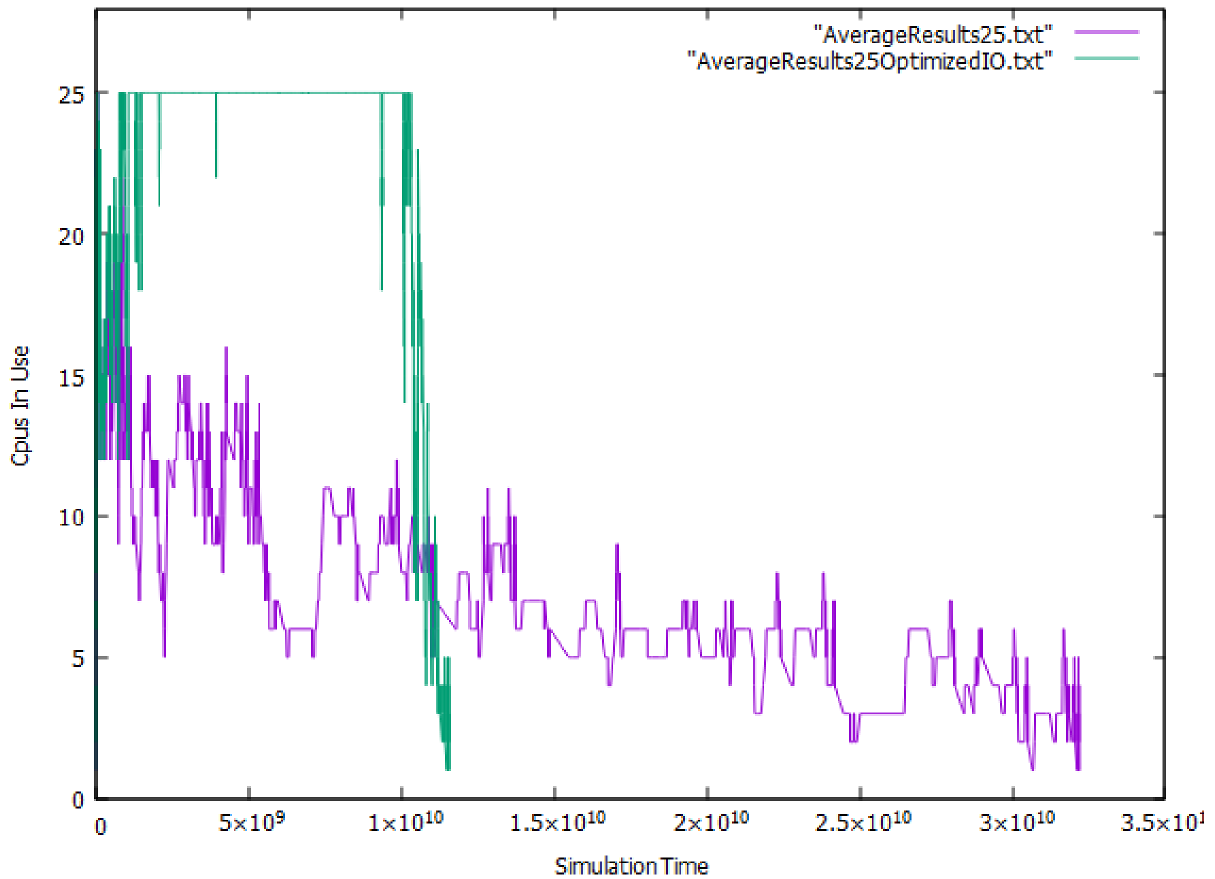


Fig. 9. 25 Processors with Subpartitions Compared to Without Subpartitions

The new dependency graph also shows that we can use even more than the original 25 maximum processors. There is a substantial speedup when the total number of available processors is increased from 25 to 50. As the number of available processors increases the rate of speedup begins to decrease. (Fig. 10)

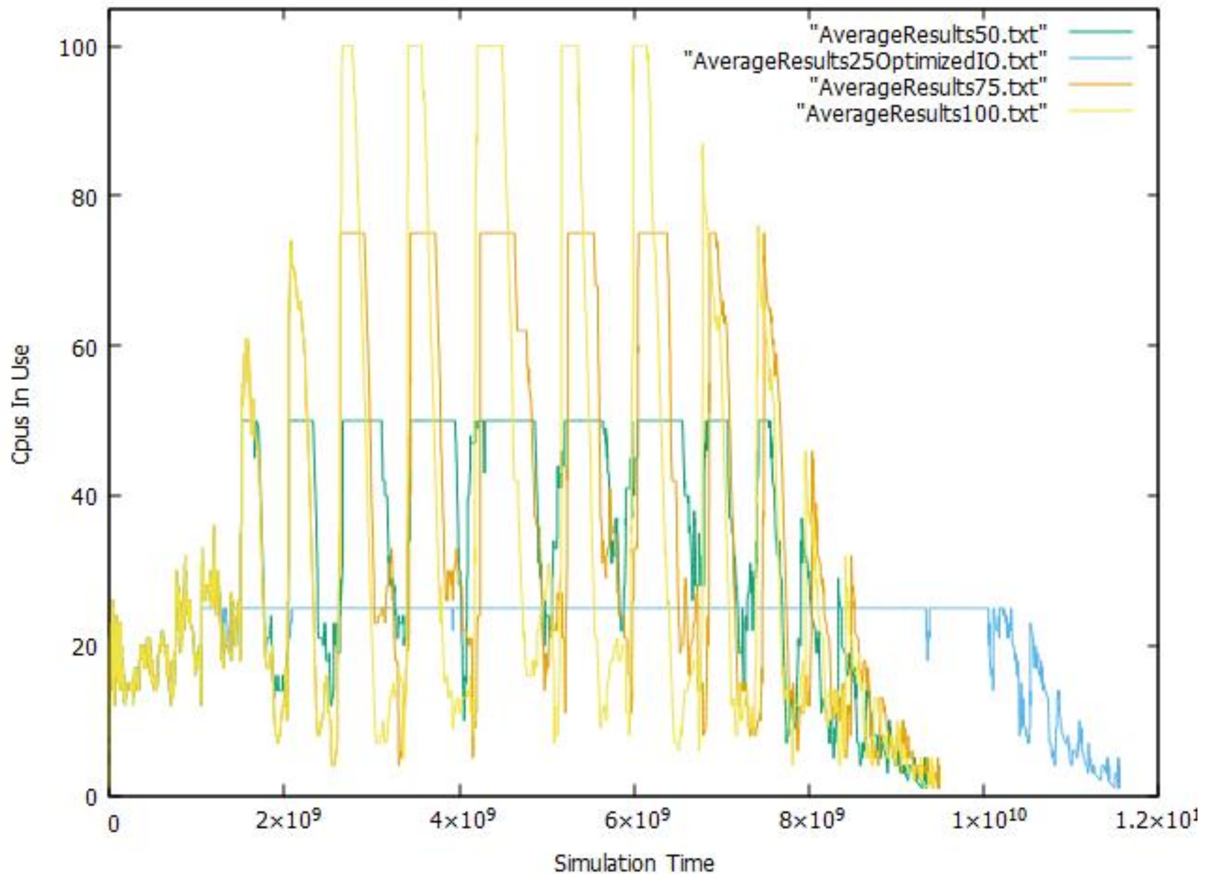


Fig. 10. 25, 50, 75, and 100 available processors

Results show that the optimum number of processors to use is approximately 60 processors. Even though the number of possible processors that can be in use at one time is over 130, the partitions are small enough during those spikes that it takes very little time to compute. The simulation was run from 1 to the max number of processors that could be used, 137, and Figure 11 shows the speedup leveling off near 60 processors. The maximum speedup we would achieve is approximately a 5x speedup to the initial simulation and over 30x the original program before adding more processors has no benefit.

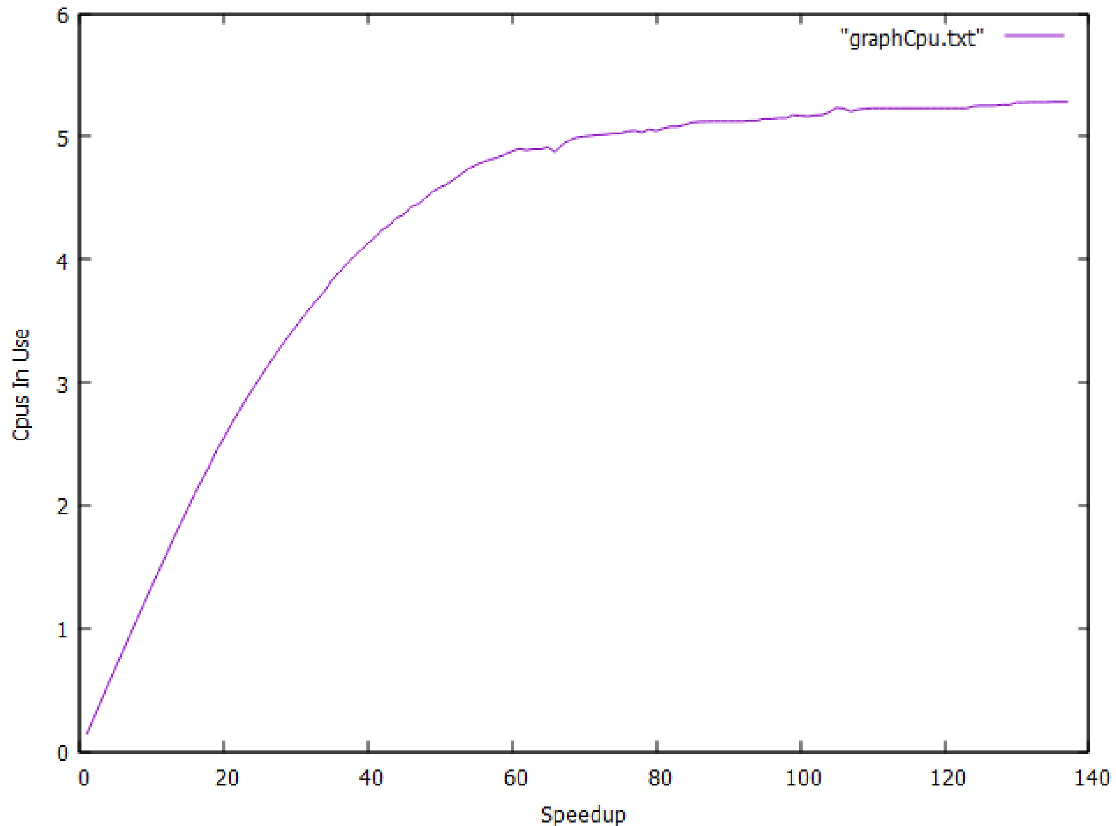


Fig. 11. Speedup from original simulation by number of processors

Another factor that changes the processing time in this project is the transferring of files to and from the different processors as jobs complete. Some files will have to be transferred several times to route them to all processors that need them. When a job is ready to run it needs all of the files it is dependent on. When a job is complete it must transfer the output file back to the controller node. Even though transferring files is fast compared to the computation times, the number of times files must be transferred, and the fact that processing is dependent on the completion of these transfers, can cause a slowdown in the process.

## V. Conclusion

This project used simulations to find the most efficient method to compute a complete solution for the 4x4 Dots and Boxes game. The original solution took 5 days and 50 processors to compute, and our goal was to significantly reduce that execution time. The refined simulation provided a method for computing the solution on Microsoft Azure Cloud that can be completed much faster than the previous solution that was computed on the University of Lynchburg campus. The program is able to run every possible state in a 4x4 Dots and Boxes game, and can be applied to games of larger dimensions. The results from the simulations show that the optimizations provide a substantial amount of speedup, even with fewer processors available. The best possible speedup is well over 10x faster than the original solution, and can be even greater depending on the results from the cloud computing.

Further work will include running the program created from the simulations on Microsoft Azure Cloud. The cloud services will provide us with crucial data that can advance this project much further. Continued research on this project will include computing solutions of Dots and Boxes on boards of different dimensions, and gathering data and trends on these games. The optimizations from this thesis will be able to create these solutions in a much more realistic timeframe than the original program created. Increased parallelism of dependencies enables the problem to decrease the amount of bottlenecks due to dependencies.

The techniques used in this thesis can also be applied outside of creating solutions for board games. In fact the simulator used can be applied to any problem for which it is possible to determine the computation time for each job. Each method used is important in the field of Computer Science and can be applied to many different kinds of problems. Simulations are an

extremely useful tool to analyze problems in an efficient, cost effective manner. As research continues more techniques will be applied to create solutions for Dots and Boxes, and many other similar problems.

**VI. Acknowledgements**

I would like to thank the entire computer science department of the University of Lynchburg for their support throughout this project and over the course of the past four years. A special thanks to Dr. Ribler for his help and mentorship on this project as well as the students that have worked to develop this project in previous years.

**VII. References and Work Cited**

[1] A. Cotarelo, “Improving Monte Carlo Tree Search with Artificial Neural Networks without Heuristics,” Applied Sciences, vol. 11, no. 5, 2021.

[2] D. Allcock, “Best Play in Dots and Boxes Endgames,” International Journal of Game Theory, vol. 50, no. 3, pp. 671-693, sep 2021.

[3] E. Berlekamp, “The Dots and Boxes Game: Sophisticated Child’s Play,” Mathematics Magazine, vol. 73, no. 4, pp. 331, Oct. 2000.

[4] HTCondor, “HTCondor Manuals,”, http://research.cs.wisc.edu/htcondor/manual/index.html

[5] J. Banks, J. Carson, “Discrete-Event System Simulation,” Englewood Cliffs, NJ: Prentice-Hall, 1984.

[6] J. K. Barker and Richard E. Korf, “Solving dots-and-boxes”, Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, Toronto, Canada, 2012

[7] K. Buzzard, M Ciere, “Playing simple loony Dots and Boxes endgames optimally”, May, 2014, arxiv:1305.2156.

[8] K. Thompson "Retrograde Analysis of Certain Endgames," ICCA Journal vol. 9, pp 131-139, 1986.

[9] R. L. Ribler, T. Percario, and C. Ware, "Computing, Publishing, and Analyzing a complete Solution to the 4x4 Dots and Boxes Game," Dept. Computer Science, University of Lynchburg, Lynchburg, Virginia, USA.

[10] U. Larson, "Games with guaranteed scores and waiting moves," International Journal of Game Theory, vol. 47, pp. 653-671, 2018.