Expectimax Enhancement through Parallel Search for Non-Deterministic Games

Rigie G. Lamanosa, Kheiffer C. Lim, Ivan Dominic T. Manarang, Ria A. Sagum, and Maria-Eriela G. Vitug

Abstract—**Expectimax, or expectiminimax, is a decision algorithm for artificial intelligence which utilizes game trees to determine the best possible moves for games which involves the element of chance. These are games that use a randomizing device such as a dice, playing cards, or roulettes. The existing algorithm for Expectimax only evaluates the game tree in a linear manner which makes the process slower and wastes time. The study intends to open new possibilities in game theory and game development. The study also seeks to make parallelism an option for enhancing algorithms not only limited to Artificial Intelligence. The objective of this study is to find a way to speed up the process of Expectimax which can eventually make it more efficient. The proponents used the game backgammon, written in Java, to apply Expectimax. The concept of parallel computing using thread pool is used to make the process of Expectimax faster. A game simulation between the existing Expectimax and the enhanced Expectimax is used as the test case for this study. After multiple test runs, the results showed that the enhanced Expectimax has chosen a move faster than the existing Expectimax 90% of the time. This shows that parallel computing speeds up the process of Expectimax.**

Index Terms—**Expectimax, parallel search, non-deterministic games, enhancement**

I. Introduction

Artificial Intelligence is the combination of computer science, physiology and philosophy which is concerned with making computers behave like humans.

Old algorithms like Expectimax were not explored much. Some are not aware that these old programs have a lot more potential if enhanced. When these algorithms (e.g. minimax, Expectimax) were created, powerful hardware were not available at the time. Nowadays, there is a lot of technology that could be used to maximize an algorithm's potential.

This study combines the power of Expectimax and the hardware itself, particularly its processor. In order to combine the two, the principle of parallelism was used.

The study will investigate an enhancement of the Expectimax search algorithm for two-player non-deterministic games. The focus is on zero-sum games with perfect information where the whole state of the game is visible. In this case, backgammon is the perfect game to be used.

The study will not divide the search on the min and max nodes since it will always contain a better choice. By parallelizing on the min and max nodes, lesser values will also be searched. Doing so would be impractical since the proponents are only interested in the best value of a move.

The study will not cover previous enhancements of the Expectimax Search. The proponents only plan to demonstrate the concept of parallelizing the Expectimax Search and how it can generally enhanced the performance of artificial intelligence. Only the form of Expectimax will be subjected to parallelism.

Manuscript received January 15, 2013; revised March 12, 2013. This work was supported in part by the University of Santo Tomas. Incongruity Theory Applied in Dynamic Adaptive Game Artificial Intelligence.

R. A. Sagum is with the faculty of Engineering University of Santo Tomas Philippines. (e-mail:riasagum31@yahoo.com).

R. G. Lamanosa, K. C. Lim, I. D. T. Manarang, and M. G. Vitug are with undergraduates of the University of Santo Tomas (e-mail: gielamanosa@gmail.com, kheiffer12@yahoo.com, ivan.manarang@gmail.com; yelvitug@gmail.com.

II. Related Works

A. Expectimax Enhancements for Stochastic Game Players

According to Veness' study [1], some people argue that the Minimax assumption is perfect and suboptimal, because it assumes that the opponent does not take advantage of the opponent's propensity. This statement is reasonable, but the application of Minimax performs well in domains such as chess and checkers.

Veness [1] mentioned in his study that one of the many improvements in Expectimax is parallel search. This can be applied in any algorithm used in Expectimax. But he specifically stated that the parallel search should be implemented in the negamax algorithm. The succeeding statements discuss studies conducted on parallel search.

B. Implementing a Computer Player for Carcassonne

One of Heyden's [2] computer players in her computer implementation for the game Carcassonne uses an Expectimax search. The computational complexity in this game proved to be relatively high thus, she uses a realistic value of about two or three for the search depth because of the high branching factor. The Expectimax player loses by a great margin to another computer player, which uses a Monte Carlo Tree Search (MCTS). Since Expectimax is a full-width search, the quality of an Expectimax algorithm is dependent on the evaluation function.

*C. *-Minimax Performance in Backgammon*

The star algorithms were introduced by Ballard [3] early in 1983 but surprisingly did not get much attention from the artificial intelligence community. The surprise comes from the significance of being able to search deeper in two-player perfect information games. Hauk, Buro, and Schaeffer [4], being curious about these reasons, set out to investigate the star algorithms on a game called Backgammon. The study showed how fail-soft alpha-beta enhancement can be adapted to chance nodes. Also, they were able to conclude both Star1 and Star2 outperform Expectimax on single position searches. The Star2 was able to save significant costs even at a depth of five.

Improved Expectimax algorithms were also experimented on and achieved greater success. Optimal among the improved algorithms is Star 2.5, a pruning algorithm like Alpha-beta search, but applied to trees with chance. The improved algorithm won 70% of the simulated games against MCTS and was even able to win against advance human players.

*D. The *-minimax Search Procedure for Trees Containing Chance Nodes*

Ballard [3] introduced the *-Minimax algorithms or more popularly known as Star algorithms which are improvements based on the Alpha-beta pruning technique. The improvements progressed from Star1 to Star2 and then Star2.5. Star1 uses the search windows of the min and max nodes, cutting-off if the sum of the children of the chance node falls outside the Alpha-Beta window. An extension of Star1, Star2 includes a phase where it determines a lower bound for each node by searching only the first child. By searching more than one child to get a more accurate lower bound, Start2 improves to Star 2.5.

E. Parallel Search of Strongly Ordered Game Tree

Marsland and Campbell [5] stated two kinds of parallel search when they noticed that most of the stronger chess programs use simple evaluation functions to give more time to conducting deeper tree searches. But still more search time is spent on move evaluation. They recommended using a number of processors that could be used to evaluate different terms in the scoring function simultaneously which can then be combined to create an overall evaluation of the node. This process is called parallel search. Another parallel implementation is in the alpha-beta window. This is done by dividing the windows into non-overlapping subintervals and applies a processor to each range. This is called parallel aspiration.

F. Comparison of Parallel Adversarial Search Algorithms

In Heim's [6] study of comparing parallel adversarial search algorithms, she stated three parallel algorithms. One of which is Principal Variation Splitting (PV Splitting). Principal Variation in game tree searching is referred to the sequence of moves which is currently believed to be the most advantageous. This sequence of nodes can be compared to a spanning tree. These are referred to as taxonomy. In PV Splitting, the search tree is mapped into an underlying assignment of processors. Each processor is responsible for the evaluation of the node and passing the value up to its parent. It is similar to the way parallel evaluation works and the only difference is it evaluates a sequence of nodes rather than just a single node.

Second is the optimized version of the PV Splitting which is called the Young Brothers Wait Concept (YBWC). The YWBC does not have definite processor assignments based on the tree but instead it develops master/slave relationships between processors. The algorithm starts by assigning the root node to a processor. This processor now "owns" the root node. When a processor owns a node, it is responsible for the node's evaluation and the propagation of the node's value up the tree. The other idle processors start request work from a random processor, if a processor who owns a node and has the PV Branch explored gets this request, it can assign the requesting processor one of its other children that is not in the PV Branch. These are called split-points. This is why it is called "Young Brothers Wait Concept". The younger brothers (siblings in the tree) have to wait for the eldest brother to be evaluated in order for them to be evaluated. These younger brothers are now owned by processors and the algorithm recurses.

The third is almost similar to YBWC which is called Dynamic Tree Splitting (DTS) wherein it also assigns a processor to the root, giving the processor ownership of it. Here, ownership simply means evaluating the node. Once the PV branch is explored, the processor creates a list of split points and adds them to a shared global list of split points (SP List) that all the processors can reference.

The idle processors consult the SP List and choose a node to own from the list. If an idle processor checks the SP List for work and finds no split points, then it sends a broadcast to the other processors requesting work. A processor which receives the broadcast that has work but has not established split points copies the state of the sub-tree in which it is working on into a shared space in memory. The requesting processor then analyzes the sub-tree for split-points. If the processor finds a suitable split-point, it becomes its owner. If the processor cannot find suitable split points from the state it analyzed, it goes idle and rebroadcasts.

III. System Architecture

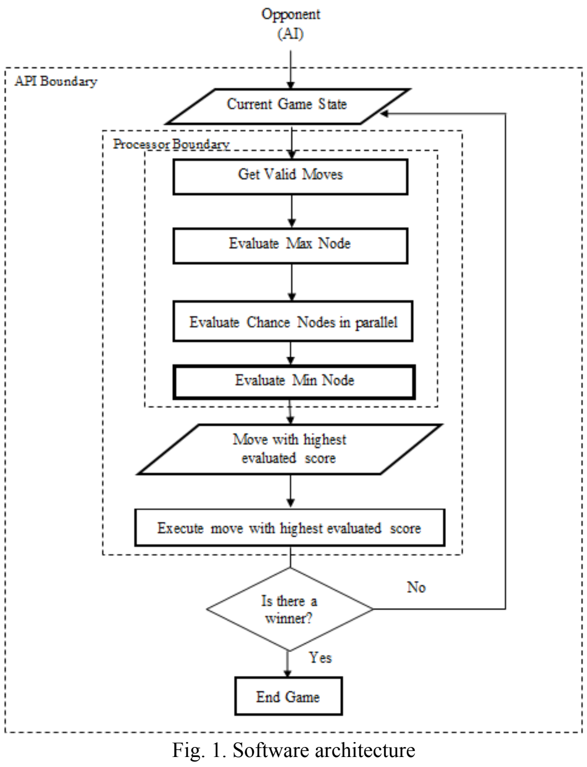

Fig. 1. Software architecture

The software architecture works first by giving the current game state as an input to the artificial intelligence which uses the enhanced algorithm. Based on the current game state, all valid moves are determined and then, evaluated.

The evaluation of the valid moves is initiated by creating a root max node that will only choose the “max” score from the chance nodes. These chance nodes represent the valid moves and needs to be scored before the root max node can get a score.

IV. Problem Statement

The major problem concerning the existing Expectimax is its innate nature to be idle at some point. The whole algorithm employs a linear search and can possibly be enhanced by making the evaluation of chance nodes parallel thus eliminating idle time.

1) On which part of the algorithm should parallelism be done?

2) Will parallelism significantly improve the time an artificial intelligence chooses a move?

V. Solution

To evaluate a chance node in the original Expectimax algorithm, the score of its child nodes is evaluated one by one. The chance node has to wait for the evaluation of one of its child nodes before proceeding to evaluate another child node.

The value of a child node is evaluated in a recursive fashion thus, making the algorithm sequential in nature. Because the algorithm is sequential in nature, it has to wait for the recursive function to finish before going to the next line of code. The value of the chance node is then multiplied by the probability of its move to happen and is added to the variable sum which represents the score of the chance node. Because the chance node score is computed by multiplying probabilities of event occurrences evaluation scores, the evaluation scores need to be directly proportional to the likelihood of winning.

The proponents were able to come up with an idea on which part of the Expectimax algorithm parallelism should be applied and how to implement parallelism to speed up the algorithm. The opportunity to improve the speed of the algorithm lies in its sequential nature of evaluating chance nodes. Originally, a chance node has to evaluate its child nodes one by one and thus, creating an idle time. The idle time can be removed by evaluating all the child nodes in parallel. This parallelizing is possible because the child node of a chance node does not rely on the result of the evaluation of its siblings.

In order to make parallelism possible, the concept of multi-threading is needed where multiple threads are created during evaluation of a chance node. In the enhanced algorithm, a thread pool is created first. Multiple threads are then stored in the thread pool.

After creating multiple threads, array objects are created for storage of values. The reason for using objects is that this type of thread declaration will only process the run() method, which can only exist once in an object. Each thread can only process one run() method at a time. The run() method will also contain the modified Expectimax algorithm. The number of threads and the number of object type val is equal to the number of chance events. Each thread will be assigned to process one object type val.

The enhanced algorithm now enters the loop same as the original algorithm. The difference is that the new algorithm asks a thread to do an evaluation of the node and while that thread evaluates the node, the next iteration is commenced. The command pool.shutdown() prevents the algorithm from executing the return computeSum(val) statement which computes for the sum of the array of an object. It works by waiting for all the threads to finish their work before going to the next line of codes.

VI. Analysis

The assessment of effectiveness was done by getting the time (in milliseconds) each algorithm takes to evaluate and make a move. After acquiring the data, min, max and standard deviation was computed from the results to compare the original Expectimax and the enhanced Expectimax. Standard deviation was used to determine the percentage difference of the algorithms.

The following formula was used to evaluate the data gathered:

To get the overall min, look for the lowest time taken to yield a move. To get the overall max, look for the highest time to yield a move. To get the average mean, sum up all the means and divide by 50 which is the number of tests conducted. Below is the computation for the mean of each test.

$$X_{mean} = (1/n)(X_1 + X_2 + X_3 + \ldots + X_n)$$

where n = number of values

For the average median, sum up all the medians and divide by the number of tests conducted. To get the median for each test, calculate for the location of the median first then use linear interpolation. Below is the formula for the location of median.

$$I_{median} = (n+1)/2$$

where n = number of values

The average standard deviation was computed by the summation of all the standard deviations over the total number of tests conducted. Below is the formula for standard deviation.

$$\sigma = \left(\frac{\sum (X - X_{mean})^2}{N}\right)^{1/2}$$

where σ = standard deviation

X = each value in the population

N = the number of values (the population)

The percentage difference was computed by getting the difference of the old value and the new value over the old value multiplied by 100 to convert it to percentage.

$$p = \frac{v_o - v_n}{v_o} \times 100$$

where p = percentage difference

v_o = old value

v_n = new value

VII. Simulation and Experimentation

Two test cases were conducted in testing the software. Fifty games were conducted where the first several moves were taken from the original Expectimax and the enhanced Expectimax. Each has the same dice roll.

After conducting the tests, the results show that both AIs have the same minimum time to evaluate and make a move which is 0. Overall maximum time for the enhanced Expectimax to make a move is 2088ms which is significantly faster compared to the original Expectimax at 8905ms. Average mean and average median shows that the enhanced Expectimax has more occurrences where it evaluates and makes moves rated at 30.496ms and 9.04ms, respectively. Standard deviation was computed to be able to calculate the percentage difference.

With the results obtained in calculating for the percentage difference, it shows that the enhanced Expectimax is faster than the original Expectimax by 90.504%.

TABLE I: Overall Computed Test results

	Original Expectimax (ms)	Enhanced Expectimax (ms)
Overall Min	0	0
Overall Max	8905	2088
Average Mean	568.781	30.496
Average Median	350.760	9.04
Average Standard Deviation	610.451	57.965

VIII. Conclusion

The parallelized algorithm proved to be significantly faster than the original Expectimax algorithm in the speed department. To improve the speed of the original Expectimax algorithm, parallelization is introduced in the evaluation of chance nodes. Idle time is present while evaluating a chance node. The sequential nature of this algorithm is the cause of idle time. Child nodes 1 level below of a chance nodes are evaluated sequentially. This is unnecessary because the result of evaluation of a child node is independent of other child nodes.Z With this, the sequential nature can be eliminated by evaluating all child nodes below of a chance node concurrently.

The parallel search of child nodes which are 1 level below the chance node reduces the time complexity to $O(n^{u-v})$, where u is the depth of the game tree and v is the number of levels consisting only of chance nodes. Memory usage while using the enhanced algorithm was noted to be greater but this issue was not formally investigated on, with the proponents only interested in the change of speed.

IX. Future Work

The following is a list of recommendations that the proponents might find essential and relevant to be able to contribute to the field of game development.

- Parallelization of Min Nodes and Max nodes. It is possible to parallelize all types of nodes in the Expectimax algorithm. This might potentially improve the time complexity of the algorithm to O(1), running at a constant time.
- Investigation of methods to parallelize the star1 algorithm (Expectimax with alpha-beta pruning). Concepts such as Move Ordering, Principal Variation Splitting, Young Brothers Wait Concepts and Dynamic Tree Splitting will be involved while trying to achieve this. The idea of parallelization of star1 is theoretically more complex than implementing it with the Expectimax algorithm though it will most likely achieve marked performance gains in terms of the speed in evaluating and executing moves than the parallelized Expectimax algorithm as it involves cut-offs. Having cut-offs eliminates the possibility of searching redundant nodes thus, reducing the amount of creating threads and improving both speed and memory usage.

X. Acknowledgment

The proponents would like to express their sincere gratitude to the following persons who made possible this research undertaking:

Ms. Ria A. Sagum, the researchers' adviser, who provided guidance and motivated the researchers to think beyond the usual and shared her expertise which made possible the fine-tuning of this study.

Ms. Arne B. Barcelo, Mr. Jose S. Seno and Ms. Rochelle L. Lopez, panel members during the defense, who gave valuable insights and suggestions for the improvement of the paper.

For the hospitality of the Lamanosa family who graciously accommodated the proponents at their residence.

To the families of the proponents, for their love, support and helping hand in all ways possible.

To the proponents' colleagues at the Faculty of Engineering, who contributed to the achievement of their goal.

Lastly, to God Almighty, who made possible all things, far beyond what we ever dreamed or imagined, we bow our hands in humble thanksgiving and offer everything for His greater glory in time and in eternity.

References

[1] J. Veness, "Expectimax Enhancements for Stochastic Game Players," B.S. thesis, School of Computer Science and Engineering, University of New South Wales, 2006.

[2] C. Heyden, "Implementing a Computer Player for Carcassonne," M.S. thesis, Dept. of Knowledge Engineering, Maastricht University, Maastricht, Netherlands, 2009.

[3] B. W. Ballard, "The *-minimax search procedure for trees containing chance nodes," *Artificial Intelligence*, vol. 21, Elsevier Science Publishers B.V., 1983, pp. 327–350.

[4] M. Buro, T. Hauk, and J. Schaeffer, "*-Minimax Performance in Backgammon," Dept. of Computing Science, University of Alberta, Edmonton, Alberta, 2004.

[5] T. A. Marsland and M. Campbell. Parallel Search of Strongly Ordered Game Tree. [Online]. Available: http://webdocs.cs.ualberta.ca/~tony/OldPapers/strong.pdf.

[6] E. Heim. Comparison of Parallel Adversarial Search Algorithms. (2009). [Online]. Available: http://www.cs.pitt.edu/~eth13/papers/ParallelAdversarialSearch.

Rigie G. Lamanosa was born in Manila, Philippines on the August 22, 1992. He became an intern in IT&T Department of Regus plc in The Fort Bonifacio, Tagiug City, Philippines from April to June 2012. He is currently taking up his Bachelor's Degree in Computer Science at the Faculty of Engineering, University of Santo Tomas.

Kheiffer C. Lim was born in Quezon City, Philippines on October 5, 1992. He is currently taking up a degree in Bachelor of Science in Computer Science at the University of Santo Tomas. He underwent internship at the Social Security System as a programmer.

Ivan Dominic Manarang was born in Manila on August 20, 1993. He is currently taking his degree in Bachelor of Science in Computer Science at the University of Santo Tomas and was awarded the Dean's List twice. He underwent internship at the National Power Corporation as a junior programmer.

Maria-Eriela G. Vitug was born in Quezon City, Philippines on July 6, 1993. She graduated high school at Montessori Integrated School where she received the award "3rd Honorable Mention." She is currently taking up Bachelor of Science major in Computer Science at the University of Santo Tomas. She completed her internship training at TrendMicro, Inc. where she was assigned in file sourcing. Ms. Vitug has been a member of TomasinoWeb, the official online student publication and organization of the University of Santo Tomas, and is now a senior member of the organization.

Ria A. Sagum was born in Laguna, Philippines on August 31, 1969. She took up Bachelor of Computer Data Processing Management from the Polytechnic University of the Philippines and Professional Education at the Eulogio Amang Rodriguez Institute of Science and Technology. She received her Master's degree in Computer Science from the De La Salle University in 2012. She is currently teaching at the Department of Computer Science, College of Computer and Information Sciences, Polytechnic University of the Philippines in Sta. Mesa, Manila and a lecturer at the Information and Computer Studies, Faculty of Engineering, University of Santo Tomas in Manila. Ms. Sagum has been a presenter at different conferences, including the 2012 International Conference on e- Commerce, e-Administration, e-Society, e-Education, and e-Technology and National Natural Language Processing Research Symposium. She is a member of different professional associations including ACMCSTA and an active member of the Computing Society of the Philippines- Natural Language Processing Special Interest Group.