A Comparison of Sequential and Parallel Adversarial Search Algorithms in the Pacman Game

Gustavo Lofrese Carvalho
Instituto Federal Catarinense
Blumenau, Santa Catarina, Brazil
gustavolc06@gmail.com

Ricardo de la Rocha Ladeira
Instituto Federal Catarinense
Blumenau, Santa Catarina, Brazil
ricardo.ladeira@ifc.edu.br

Eder Augusto Penharbel
Instituto Federal Catarinense
Blumenau, Santa Catarina, Brazil
eder.penharbel@ifc.edu.br

ABSTRACT

This paper presents a parallel implementation of the adversarial search algorithms Minimax, Alpha-Beta Pruning and Expectimax in the Pacman game, based on the CS188 project from UC Berkeley. The contribution lies in transforming the sequential implementations into parallel versions that distribute subtree evaluations across multiple threads, reducing execution time. Experiments were conducted in a controlled text-mode environment to avoid external interference. The results show substantial reductions in execution time for all algorithms, with Minimax exhibiting the most significant gains. The study also compares the strategic effectiveness of sequential and parallel versions, highlighting how parallelism influences win rates. This work reinforces the practical value of parallel search in educational gaming environments and suggests that future research may combine parallelism with additional optimizations, explore deeper search trees, or adopt alternative parallel models.

KEYWORDS

Adversarial Search, Parallel Programming, Pacman, Artificial Intelligence.

1 INTRODUCTION

Adversarial search is fundamental in competitive multiagent environments [1]. Algorithms such as Minimax, Alpha-Beta Pruning and Expectimax employ game-tree exploration to evaluate paths and determine optimal strategies under zero-sum[1] conditions. However, as the branching factor and search depth increase, the computational requirements of these algorithms grow exponentially, often resulting in infeasible execution times [1].

The Pacman environment, as provided by the CS188 project [2] at the University of California, Berkeley, offers a platform for the study of adversarial search. Its multiagent configuration enables the application of established decision-making algorithms while maintaining clear rules and a controlled state-transition model facilitating empirical evaluation.

Sequential implementations of adversarial search algorithms exhibit performance limitations when deeper search trees are required. This constraint motivates the investigation of parallel computing techniques to accelerate game-tree evaluation through the concurrent expansion of independent subtrees [1].

This study examines the effects of parallelism on the performance and strategic behavior of Minimax, Alpha-Beta Pruning and Expectimax in the Pacman environment. Parallel versions of each algorithm were developed and experiments were conducted in a controlled environment. The focus of this work is the parallel implementation, the sequential versions serve only as baselines for performance and behavioral comparison. They are not the primary subject of analysis, but serve as reference points for measuring execution time and behavioral variation.

The analysis focuses on execution time, win rates, and behavior of each algorithm across varying search depths. The results provide empirical evidence regarding the benefits and limitations of parallel adversarial search and reinforce the relevance of parallel programming in game environments.

2 BACKGROUND

Adversarial search algorithms operate on structured models of decision making in multiagent environments. In these formulations, the interaction between agents is represented as a game tree, where nodes denote states of the environment and edges correspond to the actions available to each agent [10]. This structure provides the basis for algorithms such as Minimax, Alpha-Beta Pruning and Expectimax to evaluate outcomes and propagate utility information across decision layers [4, 5, 6].

Among the foundational algorithms in this domain, Minimax (Figure 1) serves as a baseline strategy in which the utility of a state is maximized for the current player and minimized for the opponent [3]. Its recursive formulation propagates utility values from the leaves to the root, selecting the most advantageous action for the agent [4]. However, its computational cost grows exponentially with search depth, making the algorithm expensive in scenarios with large branching factors [1].

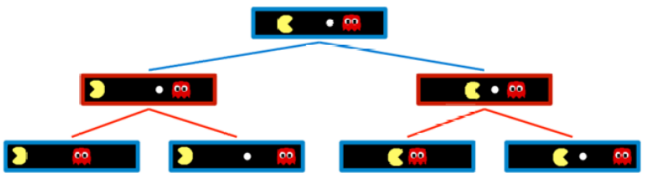

Figure 1: Minimax algorithm using the Pacman model. Adapted from [1].

[1] A *zero-sum game* is a competitive setting in which the gain of one agent corresponds exactly to the loss of another. Classical examples include Chess, Checkers and other adversarial games.

To mitigate this limitation, Alpha-Beta Pruning (Figure 2) extends Minimax by introducing bounding parameters that eliminate branches that cannot affect the final decision. This reduction in explored nodes improves computational efficiency [5].

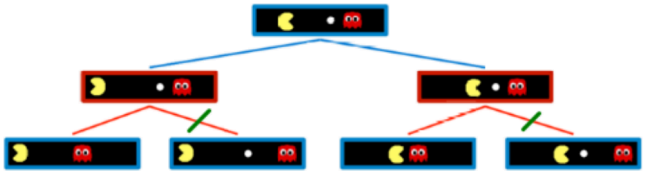

Figure 2: Alpha-Beta Pruning algorithm using the Pacman model. Adapted from [1].

The Expectimax incorporates probabilistic transitions into the search process [6]. Instead of assuming adversarial or optimal behaviors of opposing agents, Expectimax calculates the expected value of successor states (Figure 3), making it suitable for environments where agent actions exhibit random or non-deterministic characteristics. While Expectimax often produces more subtle strategic behaviors, it can also require additional computational resources due to the inclusion of expectation nodes in the game tree.

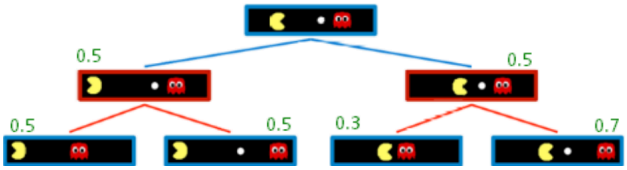

Figure 3: Expectimax algorithm using the Pacman model. Adapted from [1].

The Pacman environment defined in the CS188 project provides a platform for evaluating these algorithms [2]. Its design includes deterministic and adversarial agents, allowing strategies to be tested in a consistent and replicable state space. Previous uses of this environment have focused primarily on sequential implementations of search algorithms for pedagogical purposes.

Parallel computing techniques accelerate computationally intensive tasks [7]. In tree-search problems, parallelism is introduced by distributing independent subtree evaluations across threads or processing units. Although the literature reports gains in game-tree search [4], the magnitude of these improvements depends on the search-space structure, thread-management overhead and hardware characteristics.

In the analyzed algorithms, parallel techniques accelerate branch evaluation without altering the original decision logic. The controlled environment enables a clearer assessment of the impact of parallel execution relative to the sequential versions.

3 THE PARALLEL SOLUTION

The parallel versions of Minimax, Alpha-Beta Pruning, and Expectimax were obtained by modifying their original recursive implementations so that independent branches of the game tree could be evaluated concurrently. All algorithms were developed in Python 3.12 within the Pacman environment, preserving the evaluation functions and decision rules.

In the parallel model, each legal action available to Pacman at a given state initiates the exploration of a distinct subtree. Instead of processing these subtrees sequentially through recursive calls, the algorithm submits their evaluations as independent tasks executed by Python's `ThreadPoolExecutor`. Each task computes the utility associated with its respective branch according to the standard logic of the algorithm. Once all tasks complete, their results are aggregated exactly as in the sequential version: Minimax and Alpha-Beta Pruning select the minimum or maximum value according to the depth layer, while Expectimax computes the expected value of the successor states. No alterations were introduced to the decision-making logic of the algorithms. However, parallel execution modifies the timing and ordering of subtree completion, which can indirectly affect action selection in cases of ties or non-deterministic evaluation order

The experiments were conducted in a text-mode Linux environment (Ubuntu 24.04) without graphical rendering to eliminate its overhead [8,9] and reduce variability caused by competing processes, allowing a clearer comparison of computational performance between sequential and parallel executions. All experiments were performed on the Small Classic map (Figure 4), which provides a stable layout and a consistent branching factor for adversarial search. Although visual output was disabled, the internal game mechanics like movement rules, collision detection, scoring, ghost behavior, and terminal state evaluation remained functional.

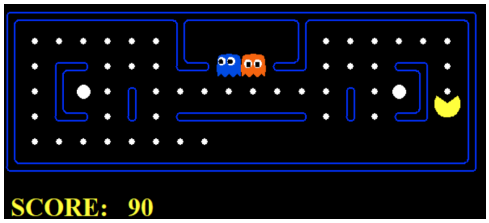

Figure 4: Small Classic map.

To ensure feasibility and consistency, iterations in which any simulation reached 100 seconds were discarded. The threshold, defined in advance and applied uniformly across all algorithms and depths, prevents outliers from distorting the comparison.

By parallelizing only the expansion of child nodes, the approach retains the structural properties of the underlying algorithms while reducing execution times in deeper search levels. The complete source code is available in a GitLab repository[2], ensuring reproducibility of the implementations.

4 RESULTS AND DISCUSSION

The evaluation measured execution time, win rate, and maximum depth in sequential and parallel configurations. Each configuration was executed fifty times, and only successful runs (those in which

[2]https://gitlab.com/GustavoLC/pac-man-ai

Pacman completed the map) were used to compute average execution time, to avoid skewing the comparison with incomplete simulations that provide no meaningful measurement. Depth levels in which any run exceeded one hundred seconds were excluded to maintain practical feasibility of experimentation.

Table 1 presents the results for the sequential algorithms. Minimax and Expectimax achieved the highest win rates at depth 3, with 33 and 27 victories respectively. Alpha-Beta Pruning obtained 20 wins at the same depth. Execution time increased significantly with depth for all algorithms, reflecting the exponential growth of the game tree.

Table 1: Results for sequential algorithms.

Algorithm	Depth	Time (in *s*)	Number of wins
Minimax	2	1.27	15
Minimax	3	8.80	33
Alpha-Beta	2	1.10	6
Alpha-Beta	3	6.16	20
Expectimax	2	2.21	14
Expectimax	3	12.09	27

The parallel implementations substantially reduced execution time across all algorithms. Minimax demonstrated the most expressive improvement, completing depth 2 in 0.16 seconds and depth 3 in 0.85 seconds, compared to 1.27 and 8.80 seconds in the sequential version. It was also the only algorithm capable of reaching depth 5 in the time constraint, completing it in 49.59 seconds. Alpha-Beta Pruning and Expectimax also reduced execution time, though with smaller gains than Minimax.

Table 2 summarizes the results for the parallel algorithms.

Table 2: Results for parallel algorithms.

Algorithm	Depth	Time (in *s*)	Number of wins
Minimax	2	0.16	23
Minimax	3	0.85	19
Minimax	4	6.39	19
Minimax	5	49.59	20
Alpha-Beta	2	1.85	6
Alpha-Beta	3	3.79	17
Alpha-Beta	4	13.35	30
Expectimax	2	1.28	46
Expectimax	3	6.18	46

Regarding win rates, Expectimax showed the largest improvement when parallelized, rising from 14 to 46 victories at depth 2 and from 27 to 46 at depth 3. Minimax and Alpha-Beta Pruning exhibited modest decreases. These results indicate that parallel execution affects behavior by altering the order in which subtrees complete, influencing tie-breaking and action selection. The win-rate variation is therefore a consequence of scheduling, not of changes to the algorithmic logic.

The experiments demonstrate that parallelism improves computational performance in adversarial search for the Pacman environment, particularly for Minimax at deeper search levels. At the same time, the results highlight that parallel execution can affect win rates depending on the algorithm, indicating the importance of evaluating runtime and strategic performance when applying parallel techniques to game-tree search.

5 CONCLUSION AND FUTURE WORK

This paper evaluated parallel versions of Minimax, Alpha-Beta Pruning and Expectimax, using their sequential implementations strictly as baselines for comparison in the Pacman environment. By parallelizing the evaluation of independent branches of the game tree, the study demonstrated that reductions in execution time can be achieved without altering the decision logic of the original implementations. The parallel Minimax algorithm exhibited the most expressive improvement in runtime. Expectimax obtained the largest increase in win rate when executed in parallel, while Alpha-Beta Pruning showed more modest gains in both metrics.

The results highlight that parallel execution is effective in accelerating adversarial search in multiagent environments. At the same time, the experiments indicate that parallelism can influence strategic behavior depending on the algorithm, suggesting that performance improvements may introduce variations in the ordering and timing of subtree evaluations.

Future works may extend this work by combining parallelism with additional optimization techniques, such as heuristic ordering of actions or selective pruning policies. Further studies may also explore deeper search trees, alternative parallelization models or environments with larger branching factors to assess how parallel search performs under more demanding conditions.

REFERENCES

[1] S. Russell and P. Norvig; Artificial Intelligence: A Modern Approach. 4th Edition. Pearson, 2021.
[2] CS 188 Spring 2024 | Introduction to Artificial Intelligence at UC Berkeley. Available at: https://inst.eecs.berkeley.edu/~cs188/sp24/.
[3] S. G. Diez, J. Laforge, and M. Saerens, Rminimax: An optimally randomized MINIMAX algorithm, IEEE Transactions on Cybernetics, vol. 43, no. 1, 385–393, 2012.
[4] P. Borovska and M. Lazarova, “Efficiency of parallel minimax algorithm for game tree search,” in Proc. 2007 Int. Conf. Computer Systems and Technologies, New York, NY, USA: ACM, 2007, 1–6.
[5] D. E. Knuth and R. W. Moore, “An analysis of alpha–beta pruning,” Artificial Intelligence, vol. 6, no. 4, 293–326, 1975.
[6] L. Bölöni, “Expectimax search and utilities,” Lecture slides for CAP 5636 – Advanced Artificial Intelligence, University of Central Florida, Fall 2023.
[7] R. Robey and Y. Zamora, Parallel and High Performance Computing. New York, NY: Simon & Schuster, 2021.
[8] V. M. Varier, D. K. Rajamani, F. Tavakkolmoghaddam, A. Munawar and G. S. Fischer, “AMBF-RL: A real-time simulation based Reinforcement Learning toolkit for Medical Robotics,” in Int. Symp. Medical Robotics, 2022, 1–8.
[9] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup and D. Meger, “Deep Reinforcement Learning That Matters,” in Proc. Thirty-Second AAAI Conf. on Artificial Intelligence , 2018, 3207–3214.
[10] M. Johanson, N. Burch, R. Valenzano e M. Bowling, “Evaluating state-space abstractions in extensive-form games,” in Proc. 12th Int. Conf. Autonomous Agents and Multi-Agent Systems (AAMAS), Saint Paul, MN, USA, 2013, pp. 271–278.