Project III Report for COM S 4/5720 Spring 2025: Hybrid Adversarial A*-Monte Carlo Tree Search with Bayesian Filtering

Christopher Martin[1]

***Abstract*— A Hybrid Adversarial A*-Monte Carlo Tree Search algorithm was created to compete against other agents in a three player grid-based adversarial search game where movements were offset based on game-generated probabilities. Our agent combines the benefits of adversarial A* and Monte Carlo Tree Search with Bayesian estimation for probability deduction.**

I. Game Rules

Three planner agents are placed in a three-way adversarial synchronous search game with the goal of capturing one of the agents while avoiding the other. The game map consists of a 2D grid with obstacles. The agents cannot leave the map bounds and if they make a move on a given turn that results in collision with an obstacle, they lose.

Every game contains probabilities $\{p_1, p_2, p_3\}$ that are unknown to the planner and must be deduced via probability estimation. For every action that the planner attempts to make it has p_1 probability of said action being rotated 90 degrees to the left, p_3 probability of rotating 90 degrees to the right, and p_2 probability of not being affected at all.

Many students and teams will be participating in a competition across all submitted agents so there is a computational limitation. Agents that take too long to execute will be disqualified, with a time budget of 30 seconds per game.

In the event of capture three points are awarded to the agent that successfully captures their target and zero to the other two. If an agent runs into a collision, they receive zero points and a point is awarded to the other two. If none of the agents successfully capture another over a set time frame, all agents receive a point.

Hundreds of games are played and the winner is determined by the highest sum points gained. Each game consists of a semi-randomly generated map where the agents are placed in a random location on the grid and obstacles are randomly placed. Probabilities are also unique to every game.

II. INTRODUCTION

Foundationally speaking our algorithm implements the early game benefits of closing space between targets with A* Search [1] while utilizing the late game risk simulation of Monte Carlo Tree Search [2]. This symbiotic combination also allows us to alleviate some of the weaknesses of both algorithms, such as A*'s greediness and Monte Carlo's early game computation costs.

We then modify the resulting action to comply with our observed movement probability estimate using Bayesian Estimation with Dirichlet Prior [3] knowledge wherein we use Laplace smoothing to converge on the true probability as we play the game over time.

III. Adversarial A* Search

A* Search [1] is a best-first shortest path algorithm that's used to find a goal provided a start and end state on a grid. It's widely popular for its efficiency and flexibility in search problems. A* determines the value of a given action via the following function:

$$f(n) = g + h(n)$$

Where f(n) = estimated total cost from start to n, g = total accumulated cost from start to current point, and h(n) = estimated cost from current point to n. For our heuristic, h(n), we utilize Chebyshev distance [4].

We start from an initial node, then maintain a priority queue of nodes to explore, then repeatedly pick nodes with the lowest f(n) value, returning the path if it finds the goal, and otherwise expanding and updating neighbor values.

For probabilities, obstacle avoidance, adversarial dodging we apply a number of penalties and discounts to our functions to incentivize both capture and defensive exploration.

The Adversarial A* function is then modified to the following:

$$f(n) = g + h(n) + \sum penalties$$

This still provides a complete agent, as the algorithm will still explore all possible options, just de-prioritizing certain routes.

A. Completeness & Optimality

A* will always return a solution in our environment since every action has a cost (current to neighbor is distance 1), there is a limited branching factor (the grid is a set size), and a goal is always reachable. Because our g-value accumulates costs over time and for every distinct node, the algorithm would worst-case explore every possible path.

Our addition of the penalty factor in the value function renders A* no longer optimal, as it will overestimate the costs of a goal. This is intentional however, as it helps ensure some level of safety.

B. Chebyshev Distance Heuristic

Chebyshev distance [4] calculates the distance between two points assuming that we can move in a straight line both diagonally and across, which we can, using the following formula:

$$Chebyshev(start, end) = max(|x_1 - x_2|, |y_1 - y_2|)$$

Because movement from one point in the grid to a neighboring point is cost 1, the minimum number of moves to reach a point is exactly the Chebyshev distance. It is both admissible and consistent ensuring that our A* implementation will return the optimal path.

C. Implementation

Our version correctly implements f(n), tracks nodes in a priority queue, tracks g-value costs, can backtrack to determine the path taken to a goal (and ultimately determine the next best step), expands neighbors to explore possible actions, determines costs with a Chebyshev distance heuristic function, and adjusts for penalties relative to state threats.

IV. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [2] is a heuristic search algorithm that combines tree search with machine learning principles and reinforcement learning. The algorithm builds a tree by simulating games and using their outcomes as a value for future choices. It is split into four phases: Selection, Expansion, Simulation, and Backpropagation.

In selection we traverse the tree from its root value and select the greatest child using a value function, in our case we use Upper Confidence Bound for Trees (UCT) [5]. Expansion takes a leaf node and generates new child nodes to determine possible next moves. Simulation simulates game rounds from a selected child node to determine potential outcomes of particular choices. Backpropagation updates the value of child nodes based on simulation outcomes; this net value determines which route and ultimately choice is the best decision.

Because this process requires some form of randomization, the entire process is simulated many times per turn to get a wider understanding of potential outcomes. MCTS balances both risk and exploration by exploiting high value moves and exploring new routes using UCT. The UCT [5] formula is as follows:

$$UCT = \frac{Q}{N} + c\sqrt{\frac{ln(N_p + 1)}{N}}$$

Where Q is the total reward, N = number of visits, c is a constant, and N_p is the number of parent visits. $\frac{Q}{N}$ allows the algorithm to favor the current-best option while $c\sqrt{\frac{ln(N_p+1)}{N}}$ adds value to nodes that are less explored. $ln(N_p+1)$ ensures all branches are eventually explored and the square root acts as a protection from over-exploration.

A. Completeness & Optimality

MCTS completeness is only constrained by computational budget and simulation depth. To have MCTS simulate all possible game options would require time and memory beyond the constraints of the game. Otherwise our simulations in the rollout phase will eventually reach a terminal state (win or lose), with the selection and exploration phases ensuring all nodes are visited.

MCTS is only optimal in theory. As the number of simulations approaches infinity, the chosen action should converge to the most optimal one, but this is not guaranteed in practice. Randomness vs reward often leads to noise which can mislead the search. Our algorithm satisfies the requirements of MCTS however encounters much of the noise that's generated through randomness.

V. Hybrid Adversarial A*-MCTS

Our algorithm heuristically checks our distance from the pursued and pursuer. If we're both far away from our target and also far away from the pursuer, then we use A* to determine the shortest possible path to our target. We then switch to MCTS in the event that neither of these conditions are met and more high-risk decisions need to be made.

Both algorithms are flexible, where the integration of the two does not interfere with either algorithm's completeness or flexibility, and depending on the game state can switch between the best option.

This hybrid approach maintains the requirements of both algorithms. While A* on its own could benefit from a risk adjustment to its f(n) value function, its purpose in our implementation is to provide an aggressive opener and a value estimator using our probabilities via penalty functions. MCTS is modified to consider the heuristic benefits of A* for game simulation, but otherwise remains the same.

VI. Bayesian Estimation with Dirichlet Prior

Our probability estimator combines Maximum Likelihood Estimation with Bayesian Estimation [3] and Laplace smoothing, where we use previous knowledge to increase our likelihood of correctly estimating probabilities as our agent plays the game.

$$P_i = \frac{N_i}{\sum_j N_j}$$

Where P_i is the estimated probability for category i, N_i the number of times category i is observed, and $\sum_j N_j$ the number of actions observed. In our case, the number of times our action stays the same or deviates relative to the sum number of actions observed so far.

VII. Overrides

Our agent employs two overrides that replace the deciding action in the event of certain game conditions. These attempt to take advantage of the game logic to gain an edge in the competition.

A. Crash

The Crash condition, while rare, allows us to intentionally run our agent into the closest obstacle in the event that either our target cannot be captured (A* fails to return a path) or if the next action would risk capture by the agent pursuing ours and there's an obstacle within range. This provides our agent a point lead in the competition wherein the opposing agents are rewarded a single point rather than three.

B. Dodge

The Dodge condition allows us to take advantage of the lack of in-game physics by overriding our agent's action to phase through and swap positions with the agent pursuing us. This assumes that the opposing agent will choose to not stand still. If successful this not only helps us avoid capture but potentially puts us in a path towards our intended target.

C. Slide

The Slide condition allows us to avoid crashing when adjacent to obstacles by finding the heuristically most favorable outcome with rotation probabilities in mind.

VIII. CONCLUSIONS

MCTS's optimality exists only in theory and the algorithm tends to get lost in noise generation as we increase the simulation count. This causes the algorithm to oftentimes perform best with next to no simulation but with high variance, while increased simulation creates more uniform results that are typically less performant.

Having our MCTS rely on A* for simulation allows us to account for probability risks in a streamlined fashion where accounting for it in our heuristic output allows us to manage the risk in the early more aggressive-facing game and continue doing so once we transition into MCTS without needing to change how we treat the probabilities.

A* is an excellent search algorithm, however it does not provide much in the way of risk mitigation, and instead requires the addition of other value techniques. This change in its value function while still making the algorithm functionally complete, renders it of its optimality.

Combining the two algorithms with Bayes estimation allowed us to securely compete and navigate in an adversarial probabilistic grid-based environment.

REFERENCES

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[2] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," *Lecture Notes in Computer Science*, vol. 4630, pp. 72–83, 2006. [Online]. Available: https://doi.org/10.1007/978-3-540-75538-8$_1$2

[3] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2006.

[4] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010.

[5] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," *Lecture Notes in Computer Science*, vol. 4212, pp. 282–293, 2006. [Online]. Available: https://doi.org/10.1007/11871842$_2$9