A TRIDENT SCHOLAR PROJECT REPORT

NO. 508

Analyzing Behaviors of Artificial Intelligence Methods for a Search Game

by

Midshipman 1/C Elana P. Kozak, USN

UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND

USNA-1531-2

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
7/12/21		

4. TITLE AND SUBTITLE
Analyzing Behaviors of Artificial Intelligence Methods for a Search Game

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Kozak, Elana P.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Naval Academy
Annapolis, MD 21402

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Trident Scholar Report no. 508 (2021)

12. DISTRIBUTION / AVAILABILITY STATEMENT

This document has been approved for public release; its distribution is UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Monte Carlo Tree Search (MCTS) is a branch of stochastic modeling that utilizes decision trees for optimization. So far, the method has largely been applied to artificial intelligence (AI) game players. This project imagines a "game" in which an AI player searches for a stationary target within a 2-D grid. We define specific constraints for this search problem and adapt the MCTS method to solve for an efficient path. We analyze its behavior with different target distributions and constraints, including the decision time and domain size. This paper covers both a single searcher scenario and the multi-searcher case. The MCTS player is compared to a simple random walk, a nearly self-avoiding random walk, and the Levy Flight Search, a model for animal foraging behavior. We provide data from simulations and prove theoretical results regarding the convergence of the MCTS when computational constraints disappear. Overall, we conclude that a searcher (or multiple) using MCTS is effective against targets with delta-like distributions but quickly loses its strength when the a priori knowledge becomes more vague.

15. SUBJECT TERMS
Artificial intelligence, Monte Carlo Tree Search, Lévy Flight Search, search and detection, optimal path

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**		27	**19b. TELEPHONE NUMBER** *(include area code)*

U.S.N.A. --- Trident Scholar project report; no. 508 (2021)

ANALYZING BEHAVIOR OF ARTIFICIAL INTELLIGENCE METHODS FOR A SEARCH GAME

by

Midshipman 1/C Elana P. Kozak
United States Naval Academy
Annapolis, Maryland

Certification of Adviser Approval

Assistant Professor Scott Hottovy
Mathematics Department

Acceptance for the Trident Scholar Committee

Professor Maria J. Schroeder
Associate Director of Midshipman Research

USNA-1531-2

ABSTRACT

Monte Carlo Tree Search (MCTS) is a branch of stochastic modeling that utilizes decision trees for optimization. So far, the method has largely been applied to artificial intelligence (AI) game players. This project imagines a “game" in which an AI player searches for a stationary target within a 2-D grid. We define specific constraints for this search problem and adapt the MCTS method to solve for an efficient path. We analyze its behavior with different target distributions and constraints, including the decision time and domain size. This paper covers both a single searcher scenario and the multi-searcher case. The MCTS player is compared to a simple random walk, a nearly self-avoiding random walk, and the Levy Flight Search, a model for animal foraging behavior. We provide data from simulations and prove theoretical results regarding the convergence of the MCTS when computational constraints disappear. Overall, we conclude that a searcher (or multiple) using MCTS is effective against targets with delta-like distributions but quickly loses its strength when the *a priori* knowledge becomes more vague.

Keywords: Artificial Intelligence, Monte Carlo Tree Search, Lévy Flight Search, search and detection, optimal path

Acknowledgements: This research was guided by Professor Scott Hottovy at the United States Naval Academy and Dr. Ira Schwartz at the Naval Research Laboratory in Washington, D.C. The work is partially supported by the National Science Foundation under Grant DMS-1815061. This project is also part of the Trident Scholar program at USNA and received support and feedback from the Trident committee, especially in the Division of Math and Science.

CONTENTS

1 Introduction

From tracking enemy submarines to locating a man overboard, search and detection scenarios are prevalent across many US Navy missions. The problem is greatly variable depending on the type of "target," size of the search space, number of search agents, and other environmental factors. The situation can range from simple and fun, like hide and go seek, to very complex, top secret military movements. Many methods for finding targets have been tested and applied with varying levels of success [5, 6, 12, 14]. In general, they hinge on mathematical optimization techniques [2]. We propose a new method to solve the problem of search and detection: Monte Carlo Tree Search (MCTS).

To test the MCTS method we start with a simple 2-D lattice search problem. The domain is a $M \times M$ lattice with periodic boundaries and discrete locations for the target and searcher to occupy. Each target is placed within the search domain according to some desired distribution. The searcher(s) start at a predetermined location, typically $S_0 = (1,1)$ or $(\frac{M}{2}, \frac{M}{2})$.

While playing the "game" the searchers can make moves in one of four directions: up, down, left, or right. All steps are one unit in length and the boundaries are periodic, meaning that moving left from $(0,1)$ results in $(M,1)$. The target is always stationary.

The solution to this search problem is a path to the target which can be measured by the number of steps. An optimal path is defined as a set of moves which reaches the target in the minimum number of steps. This requires the searcher(s) to always move directly towards the target in either the horizontal or vertical plane. A detection occurs when the distance between any searcher and the target is less than or equal to one. Achieving this optimal path is the best solution to our search problem.

The main focus of this project is a method called the Monte Carlo Tree Search, defined as "a method for finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree according to the results" [3]. The method has so far been used to develop artificial intelligence game players for games such as Go and chess. Important advantages include the ability to set a decision time limit and the lack of domain knowledge needed for success [13]. While very useful in certain board games, the process is at its core a decision making method, allowing us to apply the theoretical concepts to any choice, including search paths.

The Monte Carlo Tree Search depends on a game tree which consists of many nodes that represent possible game states. For our “game” of search and detection, each searcher location is a game state. The MCTS method works by running many simulations to learn about the game tree in order to choose its next move. The general process includes four steps: selection, expansion, playout, and back-propagation.

Definition 1. *The **Upper Confidence Bound for Trees (UCT)** is defined as*

$$UCT(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{log(N(v))}{N(v_i)}}. \tag{1.1}$$

where v represents the root node, v_i represents a child node, $Q(v)$ is the average reward for node v, $N(v)$ is the number of visits to that node, and c is the exploration constant [7].

Definition 2. *The **exploration constant, c,** is a constant that balances exploration and exploitation of the UCT algorithm [3]. Higher values of c prompt the algorithm to choose less visited nodes whereas low values prioritize nodes that have already proven successful.*

The UCT algorithm is the most common selection policy for the Monte Carlo Tree Search method. Within each simulation, the program will choose the v_i with maximum UCT value and play that node.

After selecting a node v_i, the Monte Carlo Tree Search may expand to an unvisited node. This expansion step ensures that each possible choice is tested at least once. Then the program will follow a default rollout policy until it reaches a terminal node. The goal of this policy is to give an accurate representation of how the chosen child node may affect the end result. Additionally, the rollout policy should take minimal computing power so that the simulations can be completed as quickly as possible.

The final step of the Monte Carlo Tree Search method is back-propagation. After reaching a terminal node through the rollout policy, the program determines if that simulation was a "win" or "loss". A reward function can calculate this value based on the environment. The important step is to store the information as a numeric value attached to each node included in the simulation. Thus we "back-propagate" up the tree to store these values along the previous simulation's path. This process repeats for a given number of simulations or loops, each comprised of the selection, expansion, rollout, and back-propagation steps. Finally, the program uses what it's learned from the simulations and chooses the best move by optimizing the UCT algorithm.

Definition 3. *A **reward function** is an equation which calculates one numerical value to represent the success of each choice based on its end result. In a simple win/lose game, this function could be*

$$Y = 1\text{; if win}$$
$$Y = 0\text{; if loss}$$

More complex games require more complex reward functions where the value may be anywhere between 0 *and* 1.

Although Monte Carlo Tree Search is most often applied to games, its foundation is in decision theory. It is beginning to be applied to a much wider array of problems [11]. A similar application of MCTS to the search and detection scenario is the multi-armed bandit problem [1, 8]. Its game structure is similar to that of our search game, providing an example of the UCT algorithm in action. A follow on paper [9], includes a rigorous proof that the algorithm selects the correct move with probability converging to 1. This provides the structure of our analysis in section 4.

In this work we show that MCTS is an efficient method given various amounts of information. For example, given total information about the target (i.e. it is drawn from a delta distribution), the searcher(s) find an almost optimal path given a finite decision time. As the information becomes more uncertain (the distribution of the target widens into a Normal distribution) MCTS out performs, in number of steps to find the target, any default search policy such as a random walk, Lévy flight search, and nearly self avoiding random walk. As the target becomes more uncertain (i.e., it is drawn from a uniform distribution), the MCTS is equally efficient to other, more computationally simple methods. Overall, we conclude that the MCTS method provides insight and useful strategies to better understand efficient search patterns, however, more study is needed to apply the method to real-time decision making.

The theoretical results of this work are two theorems and their proofs. The first states that under the delta distribution assumption, as the number of loops goes to infinity a MCTS searcher converges to the optimal path (Theorem 4.1). The second states that under the uniform distribution assumption, as the size of the search grid increases, a MCTS searcher

converges to a nearly self avoiding random walk (Theorem 4.2). The proof for the delta distributed target could be done by showing that the reward function Q satisfies the assumptions stated in [9]. However, the proof provided here is streamlined and shows insight to which parts of the default policy are key. Namely that the default policy must have a stopping time which, on average, is shorter the closer you start to the target. For the case of an uncertain target the theorem and proof here are novel. They show that the nearly self avoiding random walk, or searching only in sites that have not been searched before, is the optimal case when there is no information about the target.

The outline of the paper is as follows. In section 2 the search problem and related measurements are defined, as well as an overview of the methodology. In section 3, we present simulation data for the single searcher case with varied amounts of information on the target. Our main focuses are the complete information (delta distribution) target, the Gaussian target, and the fully uncertain (uniformly distributed) target. In section 4 the theorems for convergence to the optimal path for the delta distribution target and convergence to a nearly self avoiding random walk for the uniformly distributed target are stated and proved. Section 5 describes the multi-searcher scenario and provides data on the certain and bimodal target distributions. Additionally, we describe behavioral patterns of the MCTS searchers, to include clustering. In section 6 the results are summarized.

2 Single Agent Search

The main scenario we analyze here is a single searcher, single target problem. This is a baseline example from which all other search problems can be built up to. The assumptions are general yet lead to clear and significant results.

2.1 Problem Set Up

To test the MCTS method we start with a simple 2-D lattice search problem. The domain is a $M \times M$ lattice with periodic boundaries and discrete locations for the target and searcher to occupy, denoted $\mathbb{T}^2_M$. That is, the searcher at time $t \in \{0,1,2,...\}$ has position

$$S_t = (x,y), \text{ such that } x,y \in \{0,1,2,3,...,M\}. \tag{2.1}$$

Typically the searcher starts at the initial position $S_0 = (1,1)$. The target, T, is placed within the search domain according to some desired distribution $P(x,y)$ on the lattice. The target is stationary.

While playing the "game" the searcher can make moves in one of four directions: up, down, left, or right. That is, if $S_t = (x,y)$ then

$$S_{t+1} = S_t + \xi, \tag{2.2}$$

where $\xi \in \{(0,1),(0,-1),(1,0),(-1,0)\}$ is the selected move. All steps are one unit in length and the boundaries are periodic, i.e. if $S_t = (x,M)$ and $\xi = (0,1)$ then $S_{t+1} = (x,0)$. Similarly for the other boundaries.

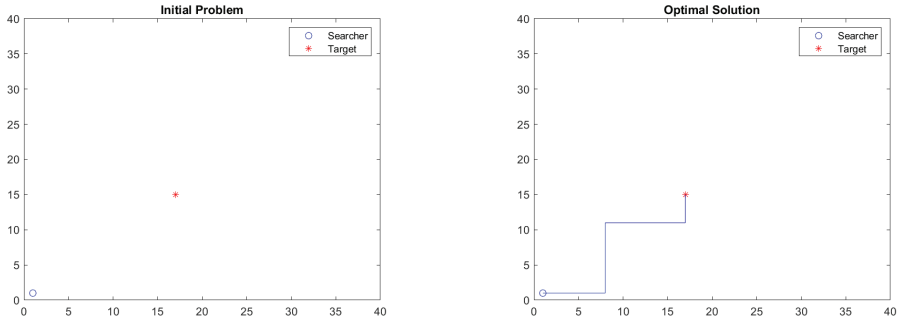
The goal for our search problem is to find a series of directional moves that results in a target detection. Further, the number of moves in this series should be as close to possible as the optimal step count. To measure the efficiency, we define metrics such as the distance and optimal step count. An example of this search problem and one solution is shown in figure 2.1.

Definition 4. *The **distance** between two locations, Q and R, on $\mathbb{T}^2_M$ is defined as $d(Q,R) = \|(Q-R)\|_{\ell^1}$ where the $\ell^1(\mathbb{T})$ metric on a torus is calculated as:*

$$\|(Q-R)\|_{\ell^1} = \begin{cases} \left|Q_x - R_x\right| + \left|Q_y - R_y\right| & \text{if } \left|Q_x - R_x\right| \le \frac{M}{2}, \quad \left|Q_y - R_y\right| \le \frac{M}{2}, \\ \frac{M}{2} - \left|Q_x - R_x\right| + \left|Q_y - R_y\right|, & \text{if } \left|Q_x - R_x\right| > \frac{M}{2}, \quad \left|Q_y - R_y\right| < \frac{M}{2} \\ \left|Q_x - R_x\right| + \frac{M}{2} - \left|Q_y - R_y\right|, & \text{if } \left|Q_x - R_x\right| < \frac{M}{2}, \quad \left|Q_y - R_y\right| > \frac{M}{2} \\ M - \left|Q_x - R_x\right| - \left|Q_y - R_y\right|, & \text{if } \left|Q_x - R_x\right| > \frac{M}{2}, \quad \left|Q_y - R_y\right| > \frac{M}{2} \end{cases}$$

Definition 5. *A target **detection** is when the distance between searcher and target is less than or equal to one. This occurs when $\|(S-T)\|_{\ell^1} \le 1$.*

Definition 6. *The **optimal step count** is defined as $O = \|(S_0 - T)\|_{\ell^1}$, the distance between the searcher's initial location and the target.*



(a) Initial problem with $S_0 = (1,1)$ and a randomly generated target on $\mathbb{T}^2_{40}$.

(b) One solution of the optimal step count, $O = 30$.

Figure 2.1: Example search problem on $\mathbb{T}^2_{40}$ and one of its optimal solutions.

There are three general cases of this search problem based on the information known about the target and the degree of certainty. For each case the target is drawn from a predefined distribution, $P(x,y)$. The most specific scenario is the certain target and the least specific is the completely uncertain. In between, we study the Gaussian target which varies in certainty depending on the initial parameters. The target distribution is important to know because all of the simulated targets will be drawn from it ($T = (T_x, T_y) \sim P(x,y)$). A more well defined target distribution will result in more accurate and useful simulation loops.

Definition 7. *A **certain target** is drawn from the delta distribution.*

$$T_x = X$$
$$T_y = Y$$

There is no variation between targets; throughout all simulations the target will always be drawn with location $T = (X,Y)$.

Definition 8. *A **completely uncertain target** is drawn from the uniformly random distribution.*

$$T_x \sim U(0,M)$$
$$T_y \sim U(0,M),$$

where (T_x, T_y) are independent. For each simulation the target could be anywhere in $\mathbb{T}^2_M$ with equal probability.

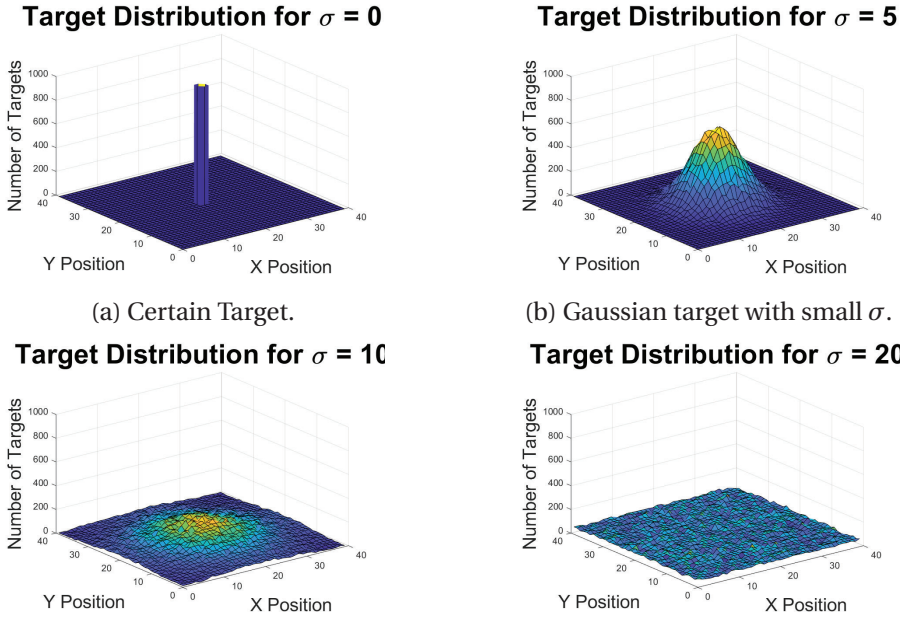
Definition 9. *A **Gaussian target** is drawn from the jointly independent Gaussian distribution with standard deviation, σ, and mean $\frac{M}{2}$.*

$$T_x \sim N\left(\frac{M}{2}, \sigma^2\right)$$

$$T_y \sim N\left(\frac{M}{2}, \sigma^2\right).$$

Depending on the value of σ the Gaussian distribution may resemble the certain target ($\sigma = 0$), completely uncertain target ($\sigma = \infty$), or somewhere in between. A smaller σ value results in a more predictable target location whereas larger values create a more uncertain distribution.

Figure 2.2 shows the heat maps of four different target distributions, each with 100,000 sample targets in $\mathbb{T}^2_{40}$.



(a) Certain Target.

(b) Gaussian target with small σ.

(c) Gaussian target with large σ.

(d) Completely uncertain target.

Figure 2.2: Example Gaussian target distributions on $\mathbb{T}^2_{40}$.

2.2 Search Methods

Throughout this project we apply four different methods (with variations) to the search problem. The main focus is on the Monte Carlo Tree Search with two rollout policies defined by the processes below.

Definition 10. *A **symmetric random walk (RW)** is a series of directional moves chosen with equal probability. That is, if $X_t = (x, y)$ then*

$$X_{t+1} = X_t + \xi, \tag{2.3}$$

where $\xi \in \{(0,1),(0,-1),(1,0),(-1,0)\}$ and ξ is chosen uniformly randomly.

Definition 11. *A **Lévy Flight Search (LFS)** is a series of directional moves which mimics animal foraging behavior and consists of many small steps with a few large jumps [5]. A point is chosen by selecting a distance $\theta \sim U(0,2\pi)$ and a length ℓ from a stable distribution:*

$$P(l_j) \sim \ell^{-\mu} \quad 1 < \mu \leq 3$$

such that the destination is

$$P_j = (l_j \cos(\theta), l_j \sin(\theta)).$$

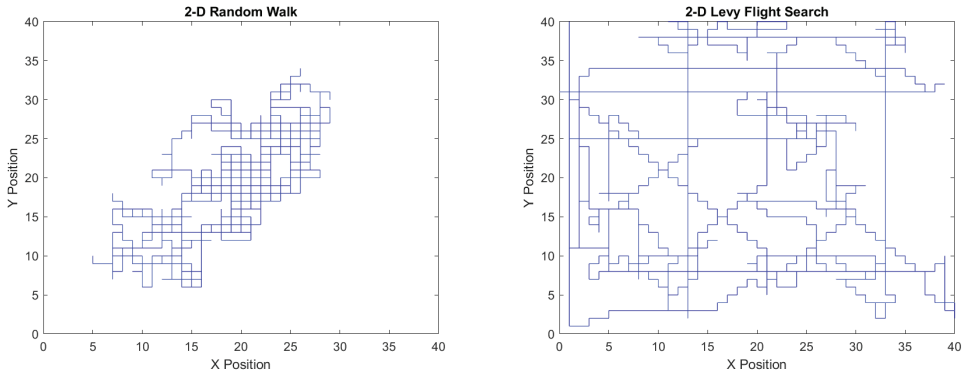
Starting from L_t, the searcher moves to the new location $L_{t+k} = P_j$ where $k = \|(S_t - P_j)\|_{\ell^1}$ is the optimal number of steps.

Definition 12. *A **nearly self avoiding random walk (NSARW)** is a series of directional moves chosen randomly based on visit count for each location. The method proceeds with the following steps:*
Let I be the set of four directional moves (up, down, left, or right). For each $i \in I$ there is a corresponding location on $\mathbb{T}^2_M$ denoted v_i. Each of these four locations has a visit count, $N(v_i)$ which represents the number of times the searcher has previously been located at node v_i. Let $Z = \min_{i \in I} N(v_i)$. Then define J as the set of moves $j \in I$ such that $N(v_j) = Z$. Choose move $j \in J$ uniformly randomly.

A series of moves chosen with the NSARW method results in a random walk which avoids previously visited locations unless there are no other options, in which case it moves to one of the least visited locations.

The random walk and Lévy Flight Search are used as our rollout policies for the MCTS. For comparison, we test the RW and LFS methods by themselves in addition to the NSARW.

Examples of a random walk and Lévy Flight Search are shown in figure 2.3. Note that these are example paths but not searches since there is no target in the domain. The nearly self avoiding random walk is shown in figure 2.4, where the color bar denotes the number of visits at each location. This example does contain a target, drawn from the uniformly random distribution and in this case located at $T = (15,30)$.



(a) Random Walk. (b) Lévy Flight Search.

Figure 2.3: Example paths for two different rollout policies in $\mathbb{T}^2_{40}$.

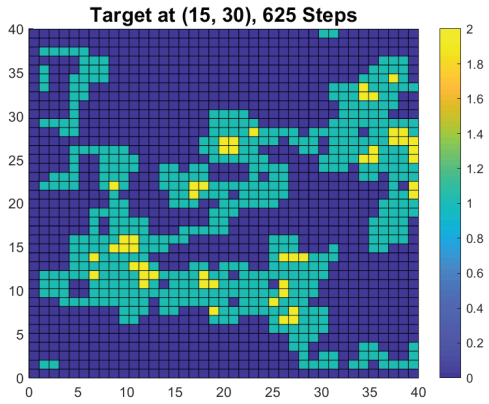


Figure 2.4: Example step counts for a nearly self-avoiding random on T^2_{40}.

2.3 Numerical Methodology

To test each search method we ran numerous simulations through MATLAB™. The Monte Carlo Tree Search requires the most complex program, outlined below [10].

First, we set the parameters: grid size (M), number of trials (T), number of loops/simulations (L), and exploration constant (c). Next, we place the target at $T = (T_x, T_y) \in \mathbb{T}^2_M$ according to the desired distribution. We also start the searcher at $S_0 = (S_{x,0}, S_{y,0})$ and calculate the optimal step count ($O = \|(S_0 - T)\|_{\ell^1}$).

Then the Monte Carlo Tree Search section of the code is repeated until the searcher detects the target. For each move, the program plays out L loops which can be viewed as practice games. Within each loop, the searcher starts at location $v = (S_x, S_y)$ and a practice target ($T_1 = (T_{x1}, T_{y1})$) is placed according to the same distribution as T. A move $j \in$ {up, down, left, right} is chosen and then the default policy plays out until the target is detected. For each location (v_i) in the search grid a new value of $Q(v_i)$ (the reward function) will be added to the Q matrix. Our reward function is the inverse of the average step count ($Q(v_i) = \frac{1}{Y(v_i)}$ where $Y(v_i)$ is the mean step count from that location). Additionally, we update the visit count matrix, N, such that $N(v_i) = 1$ for all locations touched by that simulation and $N(v_i) = 0$ for all other locations in the search space. Then the UCT matrix is calculated using the searcher's original location (v) as the root node and all other locations in the search space (v_i) as child nodes.

$$UCT(v, v_i) = \frac{Q(V_i)}{N(v_i} + c\sqrt{\frac{\log N(v)}{N(v_i)}}.$$

In order to choose the next move, we restrict the UCT values to a matrix with only the adjacent nodes: $v_j = \{v + [1,0], v - [1,0], v + [0,1], v - [0,1]\}$. Thus the program can maximize this UCT vector in order to choose which move (up, down, left, right) to play next. The process repeats for L loops such that at the end of the cycle, $N(v) = 100$ and $N(v_i) \geq 0$ for all locations in the search grid. Once again, the UCT matrix is calculated and restricted to the four potential moves, then the program chooses the optimal value and the searcher's location is updated to $S_1 = (S_{x,1}, S_{y,1})$. The searcher continues to follow this process until it is within one space of the target ($||S_t - T||_{\ell^1} \leq 1$). We record the number of steps (t) and then compute the steps over optimal ($t - O$). After many trials, these values are averaged to get the ASOO.

Definition 13. *The **average steps over optimal (ASOO)** is a measurement of efficiency for a search method calculated as*

ASOO = mean(steps taken before decision - optimal step count).

3 Numerical Results

The results from our MATLAB™ simulations are broken into three sections based on the target distribution. All data is from an $M = 40$ search grid with the searcher starting at $S_0 = (1,1)$ and one stationary target. The efficiency of each search method is mainly measured in average steps over optimal (ASOO).

3.1 Certain Target

The first case is that of the certain target, where the target is drawn from a delta distribution and is therefore always located in the exact same place. This results in simulated trials that form a very useful representation of the real problem, making the MCTS a highly efficient search method.

Figure 3.1 shows how the MCTS compares to other methods under these conditions. It is interesting to see that the Lévy Flight Search default policy results in a more efficient search than the random walk policy. We hypothesize that this is a result of the LFS covering more of the search space and back-tracking less across its own path.

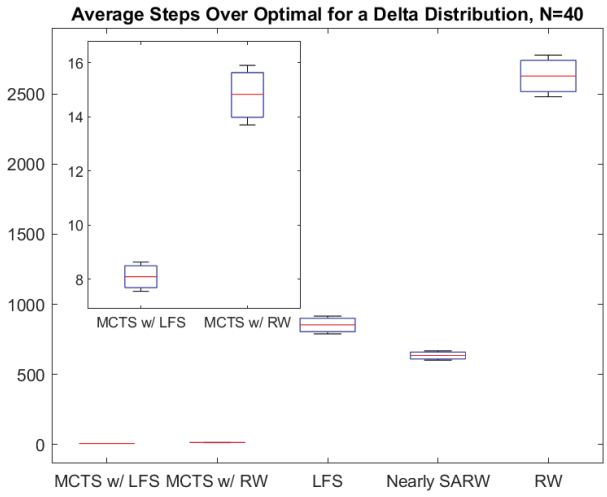
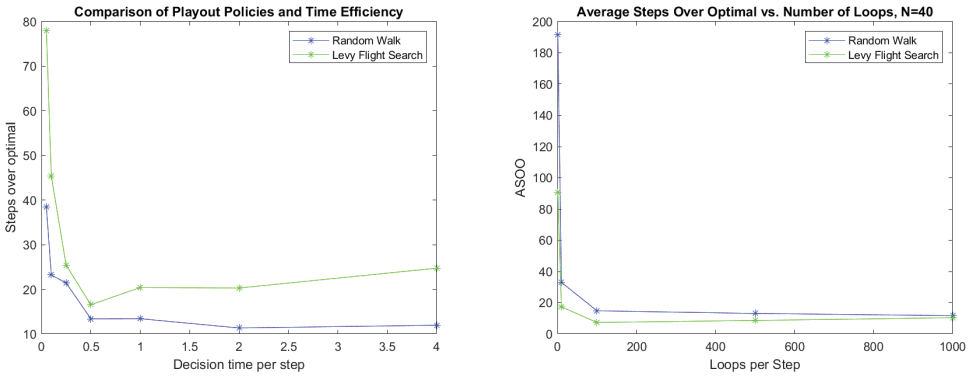


Figure 3.1: Performance with a Delta Target Distribution, 95% Confidence Interval.

In section 4.1 we will prove that in this scenario the MCTS converges to the optimal step count. Supporting data is shown in figure 3.2 with both the number of loops per step and the decision time as independent variables.



(a) Effect of decision time for MCTS.

(b) Effect of loops for MCTS.

Figure 3.2: Effect of resource limits on MCTS with each default policy.

3.2 Completely Uncertain Target

Next we analyze the completely uncertain target case, where each target is drawn from the uniformly random distribution. Thus, each simulated target is in a new location, making it the most difficult scenario for the MCTS program because it knows nothing useful about the target. Nevertheless, for both LFS and RW default policies the MCTS does outperform a basic LFS and a basic RW. The supporting data is shown in figure 3.3.

Additionally, we show in figure 3.4 that the MCTS with random walk default policy will converge to a nearly self-avoiding random walk as the size of the search grid gets larger, assuming the target is drawn from a uniformly random distribution. We provide a proof of this in section 4.2.

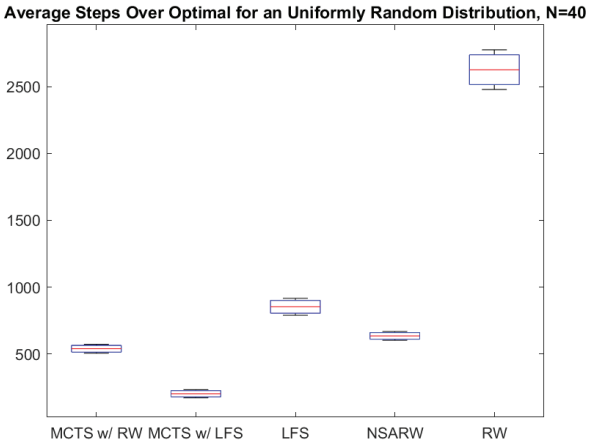


Figure 3.3: Performance with a Uniform Target Distribution, 95% Confidence Interval.

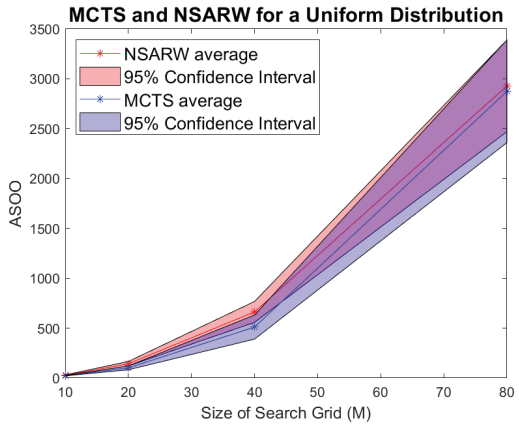


Figure 3.4: Convergence of MCTS to a Nearly Self-Avoiding Random Walk.

3.3 Gaussian Target Distribution Simulation Results

The Gaussian distribution is the most general target distribution and can be changed by varying the standard deviation (σ). We can view this as a range, where $\sigma = 0$ corresponds to the certain target (delta distribution) and $\sigma = \infty$ corresponds to a completely uncertain target (uniformly random distribution). These distributions are shown in figure 2.2. Accordingly, the simulated data shows that as σ increases, our performance decreases, leveling out to match the completely uncertain target case (a nearly self-avoiding random walk). This data is shown in figure 3.5. It is also important to note that for any value of σ the Monte Carlo Tree Search with random walk default policy outperforms both a basic Lévy Flight Search (LFS) and a random walk (RW).

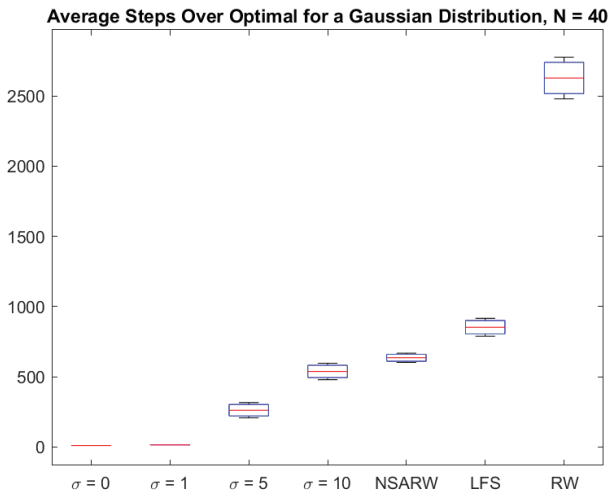


Figure 3.5: Performance with a Gaussian Target Distribution, 95% Confidence Interval.

We collected data from the MCTS against a target from the Gaussian distribution with $\sigma \in \{1, ..., 10\}$ on $\mathbb{T}^2_{40}$. The measurement of interest is the average steps over optimal (ASOO) for 1000 trials of each distribution. Based on the pattern of data points, we hypothesized that

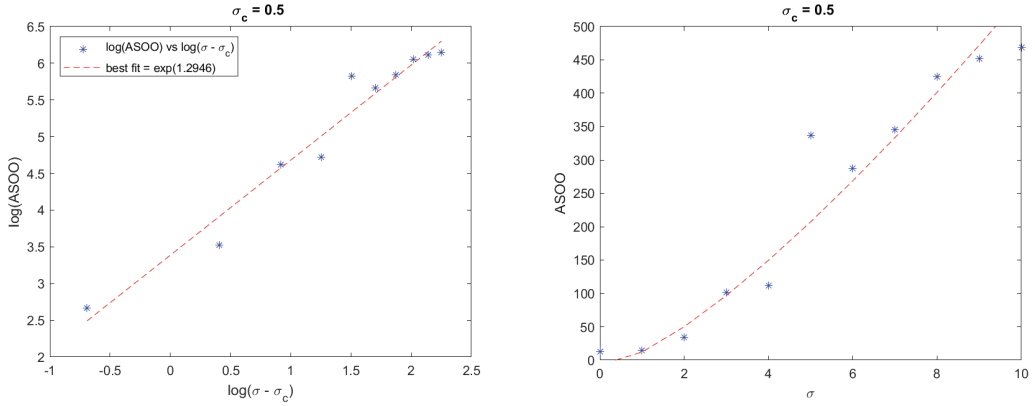
the behavior might fit a power law. This would be useful in predicting the efficiency over many values of σ so we ran the following tests to determine the appropriate fit.

Definition 14. *A **power law** is a relationship in which the data fits an equation of the form:*

$$Y(\sigma) = \alpha(\sigma - \sigma_c)^{\beta}, \ \sigma > \sigma_c$$

To test this relationship we looked at a variety of critical values, $\sigma_c \in \{0.5, 1.5, ..., 6.5\}$ and plotted a line of best fit between $log(ASOO)$ and $log(\sigma_c - \sigma)$. Comparing the r^2 values for these best lines of fit we found that the maximum r^2 was for $\sigma_c = 0.5$. Using the slope, m and y-intercept, b of this graph we calculate $\alpha = e^m$ and $\beta = b$, resulting in the power law fit of equation 3.1. The log-log plot and power law fit are shown in figure 3.6.

$$ASOO(\sigma) = e^{3.3867}(\sigma_c - 0.5)^{1.2946} \tag{3.1}$$



(a) Log-log plot of data.

(b) Original data with fitted curve.

Figure 3.6: Line of best fit for the power law relationship.

Note that this power law fit is dependent on the size of the search grid (M). All of the data above was from $\mathbb{T}^2_{40}$ but the important relationship is between σ and M. For a larger search grid the scale of distributions change. We suggest that for the power law relationship, $\beta \sim \frac{\sigma}{M}$.

While this method of identifying a power law fit is not exact, the implications of this behavior are wide. Many unexpected data sets such as the populations of cities or the sizes of power outages are thought to follow a power law [4]. Search behavior for Monte Carlo Tree Search agents may be another such example. Further study is required to better understand the way in which this relationship affects the searcher behavior.

4 Theoretical Results

Here we prove two theorems regarding the behavior of a single searcher following the MCTS method to find a single target. The results are dependent on the certainty of the target distribution and are supported by the data above.

4.1 Convergence to the Optimal Path

Assumptions: We start with a finite search lattice of size $M \times M$ with periodic boundaries, denoted $\mathbb{T}^2_M$. A target is placed randomly within, at location $T = (T_x, T_y)$. The searcher is

started from $S_0 = (1,1)$. The optimal step count is calculated as $O = \|(S_0 - T)\|_{\ell^1}$ by using the $\ell^1(\mathbb{T})$ metric on a torus (see definition 4).

Definition 15. *Define the **stopping time** $\tau_{i,j}$ of a symmetric random walk X on $\mathbb{T}^2_M$ as*

$$\tau_{i,j} = \min_k \{k \in \mathbb{N}, \|X_0 - T\|_{\ell^1} = i, \|X_k - T\|_{\ell^1} = j\}.$$

That is, $\tau_{i,j}$ is the stopping time (measured in number of steps) for a symmetric random walk to move from i units to j units away from the target. For the walk to move from n units to 0 units away from the target it must pass through locations that are $n-1$, $n-2$, etc., units away from the target. Therefore we can write the stopping time $\tau_{n,0}$ as a sum of stopping times,

$$\tau_{n,0} = \tau_{n,n-1} + \tau_{n-1,n-2} + \cdots \tau_{1,0}.$$

Each of these stopping times is non-negative and has an expected value, $E[\tau_{i,j}]$. Therefore when $\frac{M}{2} \geq n \geq 1$,

$$E[\tau_{n,0}] > E[\tau_{n-1,0}].$$

Thus for any distance D from the target,

$$E[\tau_{D,0}] < E[\tau_{D+1,0}].$$

Note that this assumption of the finite stopping time is satisfied for a symmetric random walk.

Definition 16. *A move in the **optimal direction** is one which minimizes $\|S - T\|_{\ell^1}$. Due to the lattice structure, there may be up to two optimal directions at any given location.*

Definition 17. *The **number of loops, l**, of a Monte Carlo Tree Search is the number of simulations run before each move is chosen. This parameter affects the time each decision takes and the amount of the decision space that the program explores.*

Theorem 1. *If a certain target is placed according to the delta distribution in $\mathbb{T}^2_M$ and the number of loops (l) of a Monte Carlo Tree Search with random walk default goes to infinity, then the MCTS program will choose a move that leads the searcher in an optimal direction.*

Proof. Start with the searcher at location $S = (S_x, S_y)$ which is some distance $D = \|(S - T)\|_{\ell^1}$ from the target. Let $i \in \{1,2,3,4\}$ represent the four possible moves: up, right, down, and left. Define v_i as the location corresponding to node i (i.e., if $i = 1$ (up) then $v_i = (S_{x,t}, S_{y,t} + 1)$). Let N be a matrix that stores the visit count for each node on $\mathbb{T}^2_M$. Let l be the number of loops played within each simulation. Define $R_{i,l}$ as the number of times move i has been played after l loops. Let $\tau_{D,0}$ be a stopping time that represents the number of steps taken in a random walk from the searcher's location to the target, where $D = \|S_t - T\|_{\ell^1}$. Let $\tau^{(k)}_{D,0}$ denote this value for the kth simulation of that move. Then calculate the average step count after move i has been chosen $R_{i,l}$ times as

$$X_{i,R_{i,l}} = \frac{\tau^{(1)}_{D,0} + \tau^{(2)}_{D,0} + \ldots \tau^{(R_{i,l})}_{D,0}}{R_{i,l}}.$$

Next, define the average reward for move i as

$$Y_{i,R_{i,l}} = \frac{1}{X_{i,R_{i,l}}}.$$

Using this notation, the UCT algorithm can be written as

$$UCT(v_i, S) = \frac{Y_{i,R_{i,l}}}{M(v_i)} + c\sqrt{\frac{\log N(S)}{N(v_i)}}.$$

Before continuing this proof, we provide a lemma about the number of times the optimal and sub-optimal choices are simulated.

Lemma 2. *For all moves $i \in \{1,...,4\}$ the number of times move i has been played after l loops goes to infinity; i.e., $R_{i,l} \to \infty$, as $l \to \infty$.*

Proof. We proceed by contradiction. Assume at lease one move (denote as j) is chosen a finite number of times, i.e., $R_{j,l}$ is bounded. Thus we can find some constant, R such that for all l,

$$R_{j,l} \leq R.$$

Based on the UCT method, the program will chose its next move by maximizing

$$UCT(v_i, S) = \frac{Y_{i,R_{i,l}}}{N(v_i)} + c\sqrt{\frac{\log N(S)}{N(v_i)}}.$$

Using the assumption that one move is bounded, we know that at least one of the other moves must not be bounded since the total number of simulations (l) goes to infinity. Define this move(s) as i^*.
Since $0 \leq Y_{i,R_{i,l}} \leq 1$ and $1 \leq M(v_i) \leq l$ for all $i \in \{1,2,3,4\}$, we can say that for the j move

$$UCT(j) \leq 1 + c\sqrt{\frac{\log l}{R_{j,l}}}, \ or$$

$$UCT(j) \geq c\sqrt{\frac{\log l}{R}}.$$

Note that for the unbounded move i^*, we can choose an L such that for all $l \geq L$, $R_{i^*,l} > R^2$. Thus

$$\frac{1}{R_{i^*,l}} < \frac{1}{R^2} \leq \frac{1}{R}.$$

Multiplying by $\log l$ and taking the square root we get

$$\sqrt{\frac{\log l}{R_{i^*,l}}} \leq \sqrt{\frac{\log l}{R^2}} \leq \sqrt{\frac{\log l}{R}}.$$

Since R is a constant and $\log l$ is monotonically increasing, we can choose $L' > L$ such that for all $l \geq L'$ we get

$$c\sqrt{\log l}\left(\frac{\sqrt{R}-1}{R}\right) > 1, \ or$$

$$c\sqrt{\log l}\left(\frac{1}{\sqrt{R}} - \frac{1}{\sqrt{R_{i^*,l}}}\right) > 1.$$

Thus for all $l \geq \max\{L, L'\}$ we conclude that

$$UCT(i^*) \leq 1 + c\sqrt{\frac{\log l}{R_{i^*,l}}} < c\sqrt{\frac{\log l}{R^2}} < c\sqrt{\frac{\log l}{R}} \leq UCT(j).$$

Therefore $UCT(i^*) \leq UCT(j)$ so move j will be chosen and played out.
This contradicts the assumption that $R_{j,l}$ is bounded, so we know that as $l \to \infty$ $R_{j,l} \to \infty$. The same logic can be applied to all other moves $i \in \{1,...,4\}$ so we conclude that for all i, as $l \to \infty$, $R_{i,l} \to \infty$.

□

Now we proceed with the proof of Theorem 1.

For the searcher choosing between 4 possible directions let $i \in \{1,...,4\}$ represent a choice in the sub-optimal direction and i^* represent a move in the optimal direction. The searcher starts at a distance $D = \|(S-T)\|_{\ell^1}$ from the target. For a sub-optimal move, i, the searcher initially moves one step away from the target, increasing this distance to $D+1$. Therefore the average step count for move i is defined as

$$X_{i,R_{i,l}} = \frac{\tau^{(1)}_{D+1,0} + \tau^{(2)}_{D+1,0} + ...\tau^{(R_{i,l})}_{D+1,0}}{R_{i,l}}.$$

For the optimal arm, i^*, however, the searcher moves toward the target, decreasing the distance to $D-1$. Thus we define the average step count for i_* after l loops as

$$X_{i^*,R_{i^*,l}} = \frac{\tau^{(1)}_{D-1,0} + \tau^{(2)}_{D-1,0} + ...\tau^{(R_{i^*,l})}_{D-1,0}}{R_{i^*,l}}.$$

As $\ell \to \infty$, by Lemma 2, $R_{i,\ell} \to \infty$ and the above quantity converges to the expected value of $E[\tau_{D,0}]$. Since farther distances will on average take longer to reach the target, $E[\tau_{D-1,0}] < E[\tau_{D+1,0}]$. By the law of large numbers we can say that for a sub-optimal move, i,

$$\lim_{l\to\infty} \frac{\tau^1_{D+1,0} + \tau^2_{D+1,0} + ...\tau^{R_{i,l}}_{D+1,0}}{R_{i,l}} = \lim_{l\to\infty} X_{i,R_{i,l}} = E[\tau_{D+1,0}]$$

with probability 1. Similarly, for the optimal move we have

$$\lim_{l\to\infty} \frac{\tau^1_{D-1,0} + \tau^2_{D-1,0} + ...\tau^{R_{i^*,l}}_{D-1,0}}{R_{i^*,l}} = \lim_{l\to\infty} X_{i^*,R_{i^*,l}} = E[\tau_{D-1,0}].$$

Using this expected value we want to show that the probability of the step count being significantly larger or smaller is small. For each real move of the searcher, the MCTS method chooses the direction with the best UCT value:

$$I_l = \max_{i\in\{1,2,3,4\}} \Big\{ \frac{Y_{i,R_{i,l}}}{N(v_i)} + c\sqrt{\frac{\log N(S)}{N(v_i)}} \Big\}.$$

Fix $\epsilon > 0$. We want to show that if l is sufficiently large, $\mathbb{P}(I_l \neq i_*) \leq \epsilon$. For any sub-optimal move, i, let $p_{i,l} = \mathbb{P}(Y_{i,R_{i,l}} \geq Y_{i^*,R_{i^*,l}})$. We substitute the inverse of our average reward to get $p_{i,l} = \mathbb{P}(X_{i^*,R_{i^*,l}} \geq X_{i,R_{i,l}})$. Then we know $\mathbb{P}(I_l \neq i_*) \leq \sum_{i\neq i_*} p_{i,l}$. Therefore it suffices to show that $p_{i,l} \leq \frac{\epsilon}{K}$ is true for all sub-optimal moves when l is large enough.

Using the convergence for large l, if $X_{i^*,R_{i^*,l}} < E[\tau_{D-1,0}] + \epsilon$ and $X_{i,R_{i,l}} > E[\tau_{D-1,0}] - \epsilon$ then $X_{i^*,R_{i^*,l}} < X_{i,R_{i,l}}$. Therefore,

$$p_{i,l} \leq \mathbb{P}(X_{i^*,R_{i^*,l}} \geq E[\tau_{D-1,0}] + \epsilon) + \mathbb{P}(X_{i,R_{i,l}} \leq E[\tau_{D+1,0}] - \epsilon).$$

By the law of large numbers, as $n \to \infty$, $\mathbb{P}(|E[X] - X_n| > \epsilon) \to 0$. We apply this to the inequality above to say that

$$\lim_{l\to\infty} p_{i,l} \leq \lim_{l\to\infty} P(X_{i^*,R_{i^*,l}} - E[\tau_{D-1,0}] \geq \epsilon) + \lim_{l\to\infty} P(E[\tau_{D+1,0}] - X_{i,R_{i,l}} \geq \epsilon) = 0.$$

Then, since $\lim_{l\to\infty} p_{i,l} = 0$ we can choose an L sufficiently large such that for all $l \geq L$

$$p_{i,l} \leq \frac{\epsilon}{K}.$$

□

4.2 Convergence to a Self-Avoiding Random Walk

Theorem 3. *If a completely uncertain target is placed according to the uniform random distribution in $\mathbb{T}^2_M$ and the size of the search grid, $M \to \infty$, then a Monte Carlo Tree Search program will converge to a nearly self-avoiding random walk on $\mathbb{T}^2_M$.*

Proof. We want to show that the UCT algorithm, defined by

$$UCT(v, v_i) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{log(N(v))}{N(v_i)}},$$

chooses a child node (v_i) randomly as the number of trials goes toward infinity, unless one direction has been picked more frequently than the others.

Let T be the randomly selected target location and S_t be the location of the searcher at time t. Assume that $S_0 = (1,1)$.

Choose $\epsilon > 0$. Then set K_ϵ as the radius of a region E around the origin such that $\frac{1}{K_\epsilon} < \epsilon$. Thus K_ϵ is the minimum number of steps required to exit this region.

Choose $\delta > 0$. Based on the ϵ chosen and the resulting radius of region E, set M to be the dimension of the search space such that $P(T \in E) < \delta$. Therefore $P(\|(S_0 - T)\|_{\ell^1} > K_\epsilon) = 1 - \delta$ where $\|(X - Y)\|_{\ell^1}$ is the distance on $\mathbb{T}^2_M$ between locations X and Y. This is represented pictorially in Figure 4.1.

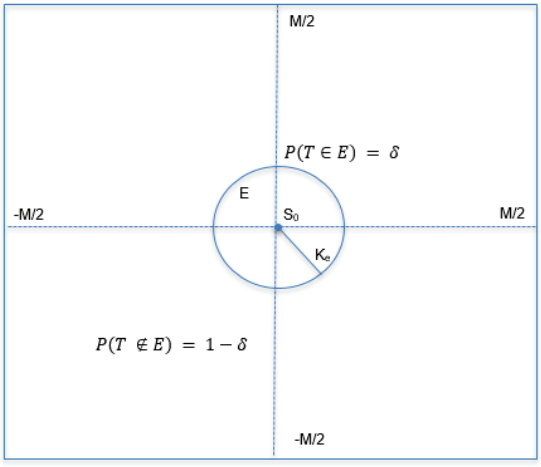


Figure 4.1: Search grid given ϵ.

Let v_i, $i \in \{1,2,3,4\}$ represent each of the four child nodes (up, down, left and right) of the root node v. Then $Q(v_i)$ represents the success rate of each child node, where $Q(v_i) = \frac{1}{X_{i,l}}$ and $X_{i,l}$ is the average number of steps after l loops. Recall that K_ϵ is the optimal step count to exit the circle and T is located outside the circle with probability $1-\delta$. Therefore, on average $X_{i,l} \geq K_\epsilon$. This means that as the size of the search grid increases we get $Q(v_i) \ll \frac{1}{K_\epsilon} < \epsilon$. Therefore as ϵ gets smaller, $Q(v_i)$ approaches zero. Hence the first term of the UCT algorithm approaches zero.

The second term, $c\sqrt{\frac{log(N(v))}{N(v_i)}}$, is a relationship between the visit counts of the parent and child nodes. For the first four trials, each child node gets visited once (chosen in a random order as per MCTS protocol). It is important to note that after the first step is taken, the previous root node (now a child node) will have a very large visit count because it was included in all of the simulations from the past decision. Thus, after 4 trials, $N(v_i) = 1 \ \forall \ i \in \{1,2,3,4\}$ except the direction from which it came, which will be significantly higher (equal to the number of loops). Note that $N(v)$ is the same for all v_i since this new root node's visit count increases after each and every trial.

Since $N(v_i)$ has equal entries for all directions except one, and $N(v)$ is a constant, the UCT vector will have equal values for three moves, excluding the direction from which it came. Thus the UCT program picks one of the three directions randomly, avoiding its previous path. Call this direction v_j. The next trial will be run using v_j as the first move. Regardless of the outcome, we now get $N(v_i) = 1 \ \forall \ i \neq j$ and $N(v_j) = 2$. Since the UCT value has $N(v_i)$ in the denominator of the second term, the UCT value will decrease for v_j which has been visited twice. Therefore $max(UCT)$ will randomly return one of the three remaining values (these are all equal).

The process will repeat until all four nodes are visited twice, then three times, etc. Regardless of the time this program runs it will continue randomly cycling through the nodes, trying to avoid hitting one node more times than the others. At the end, all four nodes will have an equal visit count so the final decision will be chosen uniformly random.

Therefore the program will choose a random direction unless one or more of the directions have been chosen an unequal number of times, in which case the program will choose randomly between the options visited the minimum number of times. Thus the path becomes a nearly self-avoiding random walk. □

5 Multi-Agent Search

To further explore the 2-D search space we propose a scenario in which K searchers look for one target within the same domain. We carry over the assumptions from section 2 but with the searchers, S, in the form of a $K \times 2$ matrix where each row represents a searcher.

There are two different ways to view this problem: simultaneous or alternating choice. For the simultaneous choice we have 4^K child nodes, each representing a combination of the searchers' moves (e.g. searcher one moves up, searcher two moves right, ..., searcher K moves down). This method becomes computationally expensive since the game tree and data matrices expand rapidly with the size of the search grid. No further study of this method is provided due to the computational limits of our available resources.

The alternating choice method is more similar to the single searcher problem and thus faster to execute. In this case, searcher one runs through all L loops of the MCTS method, collecting data for Q and N and calculating the UCT matrix as it goes. Searcher one chooses a move, then searcher two goes through the same process, adding to and rewriting the same

Q, N, and UCT matrices, then choosing its move. This process is repeated, cycling through all K searchers until the target is found. Since the searchers all use the same data matrices and continuously update one UCT matrix, they are able to learn from the other's choices and make more efficient decisions.

The focus of our study is to determine how multiple searchers using the MCTS method behave within the same domain. We will show through simulations and examples how the search patterns change based on the target distribution and number/placement of searchers.

5.1 Multiple Searchers for a Certain Target

First we analyze the certain target situation (see section 2.1 for definition), where the target is drawn from the delta distribution. For the two-searcher scenario we want to determine how the searchers work together based on their initial placement. To do this we develop a simple two searcher problem in a $(M \times M)$ lattice with periodic boundary conditions $(\mathbb{T}^2_M)$. The target is placed at $(\frac{M}{2}, \frac{M}{2})$ and a circle of radius $\frac{M}{4}$ is drawn around this target. Searcher 1 always starts at the angle $\phi = 0$ and searcher 2 is placed around the circle at the angle θ. An example for the $M = 40$ case is shown in figure 5.1.

Using the unit circle we test $\theta = k * \frac{\pi}{4}$ with $k \in 1, 2, ..., 8$. To accommodate for the 2-D lattice we round each the x and y values to the closest integer. The goal of this problem is to find the target in as few steps as possible. Since both searchers start equidistant from the target in the center, the optimal number of steps is always $\frac{M}{4}$. We measure the searchers' efficiency in Average Steps Over Optimal (ASOO). The results from MATLAB ™ simulations are shown in figure 5.2.

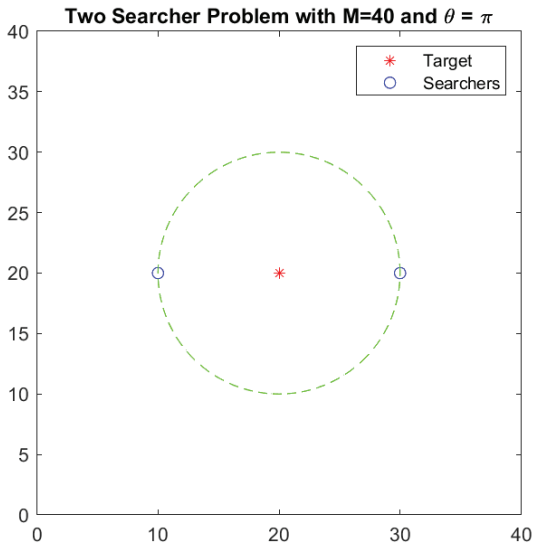


Figure 5.1: Initial set up for 2 searchers, 1 target.

Our data shows that when searchers start close together ($\theta = 0$) they find the target more quickly than when they start farther apart ($\theta = \pi$). We investigate this by looking at the UCT selection algorithm (equation 1). In general, the algorithm prevents a searcher from revisiting the same location twice (exploration term). In our multi-searcher scenario, this results in the searchers trying to stay away from each other and not revisiting each other's previous

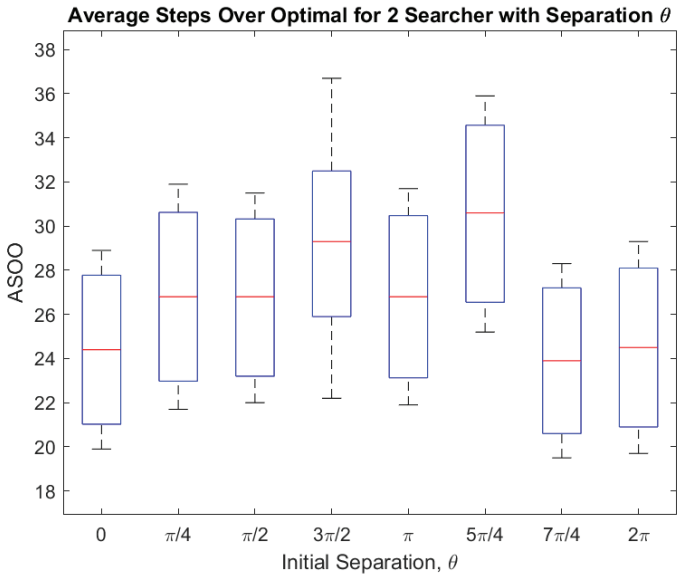
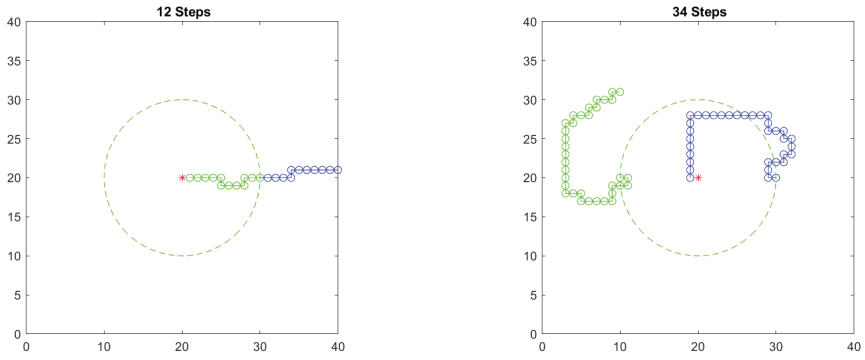


Figure 5.2: 95% confidence intervals for a certain target on $\mathbb{T}^2_{40}$; 100 trials with $L = 100$.

locations. However, both searchers are still trying to move towards the target (exploitation term) so there must be a balance between separation of searchers and their paths to the target. Thus searchers who start close together will spread out and at least one will move directly towards the target, resulting in an efficient search. The searchers who start on opposite sides of the target, however, struggle to find balance between a direct path to the target (moving towards each other) and staying separated. This results in a less optimal search. Example paths are shown in figure 5.3.



(a) Example path for two searchers, $\theta = 0$.

(b) Example path for two searchers, $\theta = \pi$.

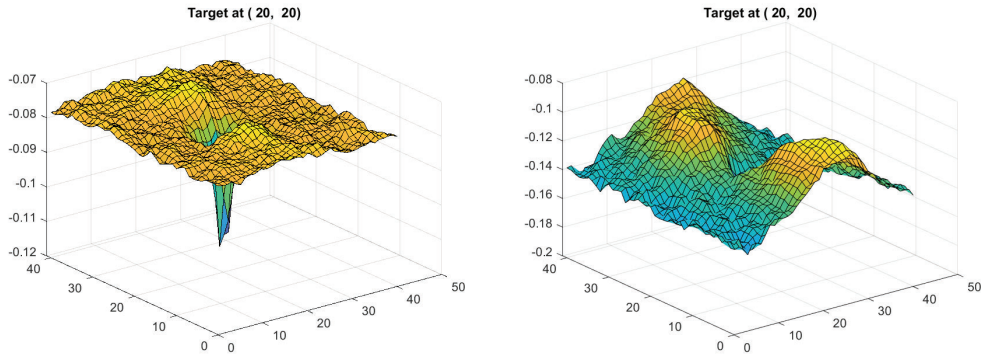
Figure 5.3: Results of two searcher, certain target scenario. These example plots show how searchers starting closer together have a more efficient path than those who start on opposite sides of the target.

Another metric that is useful to our understanding of the multi-searcher case is the UCT gravity plot. Recall that the UCT selection algorithm is

$$UCT(v, v_i) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{log(N(v))}{N(v_i)}}$$

where v is the starting location and v_i are the other spaces in the search grid. To create the gravity plot we use the final searcher location, $v_f = S_t$ as our root node and calculate the UCT matrix for all other locations, v_i. Then we negate this value and graph $-UCT(v_f, v_i)$ as a surface plot, as shown in figure 5.4.

The purpose of negating the values is to create a more intuitive graph, referred to as the gravity plot. Here we can imagine the behavior of our searchers in terms of physics. Like balls on a curved surface, our searchers on the gravity plot will naturally move towards the points of lowest energy, i.e. the troughs. Deeper troughs are more likely for the searchers to find but other low points on the graph may attract the searchers as well. Once the searcher has found a low point, it is unlikely to move to a point of higher energy, just like a ball at the bottom of a canyon would not roll to the top. As expected, a more accurate gravity plot will result in a more efficient search.



(a) More defined UCT plot generated from a $\theta = 0$ example run. This will result in a more optimal search.

(b) Less defined UCT plot generated from a $\theta = \pi$ example run. This will result in a less efficient search.

Figure 5.4: Gravity plots for the two searcher, certain target scenario with 100 loops per move. These examples show that the more well-defined plots with a sharper point at the target's location result in a more accurate representation of the search space and thus a more efficient search.

Continuing with the certain target scenario, next we explore the behaviors of K searchers working together in the same domain. To do this, we use the same setup shown in figure 5.1 but start K searchers at $(\frac{3M}{2}, \frac{M}{2})$ (separation angle $\theta = 0$). By increasing the number of searchers we show that the search becomes more efficient, actually converging to zero steps over optimal when enough searchers are present for the defined grid size. Data for this is shown in figure 5.5. To explain this behavior, we use the same argument as the two searcher scenario. When the searchers all start in the same location they have a tendency to spread out in order to avoid visiting the same locations multiple times. When there are enough searchers in the domain this results in a large area being covered thoroughly by the searchers, increasing the chances of an efficient search. Once the searchers have spread out, the one

closest to the target is likely to find the optimal path. Figure 5.6 shows the percent of locations visited after the first three steps of a MCTS program. After the first move there are 4 available locations, after the second there are 12, and after the third there are 24. A higher percent of spreading means that the searchers covered a larger area of the search grid. As explained above, this will naturally result in a more efficient search. Finally, figure 5.7 shows an example path for five searchers in $\mathbb{T}^2_{40}$. Searcher 2 finds the target in only six steps over optimal. Our data in figure 5.5 shows that with more searchers, this number would be zero.

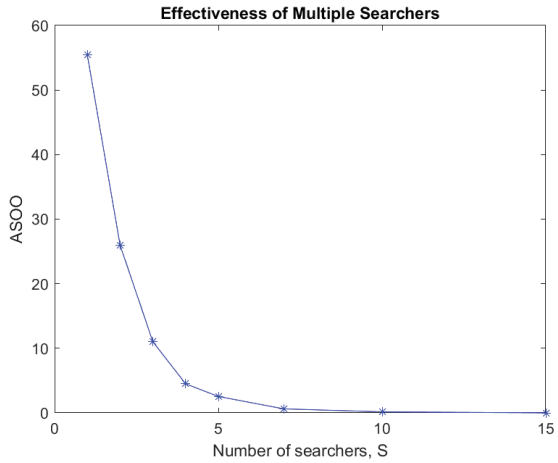


Figure 5.5: Average steps over optimal for a $M = 40$, $c = 2$ MCTS with K searchers, averaged over 100 trials. The data shows a convergence to zero, i.e., a scenario with enough searchers and a certain target will always find the optimal path.

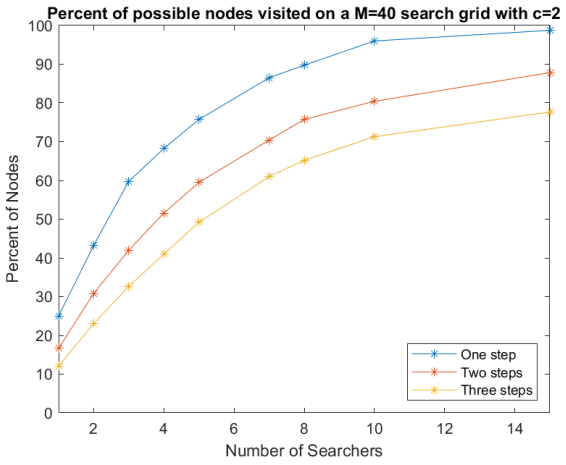


Figure 5.6: Average percent of the possible locations visited after one, two, and three steps of the K searcher scenario with a certain target. For this data, $M = 40$ and $c = 2$ with 100 trials. The data shows that more searchers results in more spreading, covering a larger area of the search space.

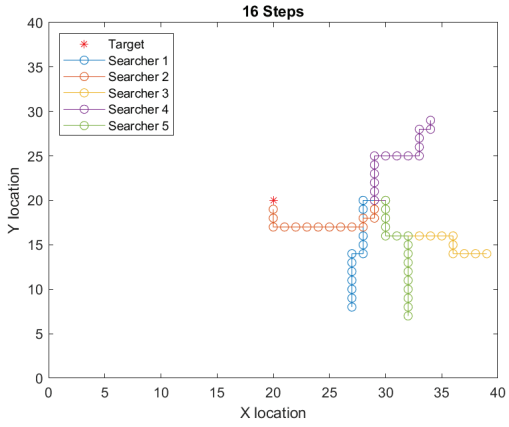


Figure 5.7: Example paths for 5 searchers.

In conclusion, we provide a conjecture about the behavior of multiple MCTS searchers in $\mathbb{T}^2_M$ with a certain target.

Conjecture 3.1. *For a finite search grid of size $M \times M$ (denoted $\mathbb{T}^2_M$) with one certain target (drawn from the delta distribution), there exists a minimum number of searchers, K, such that for any game with $k \geq K$ searchers using the MCTS method using L loops per decision, at least one searcher will find the optimal path to the target.*

5.2 Two Searcher Bimodal Distribution

To further explore the behavior of two searchers using the MCTS decision method we develop a problem with a bimodal target distribution. We start with the same search area ($\mathbb{T}^2_{40}$) and place both searchers at a central starting location, $(\frac{M}{2}, \frac{M}{2})$. The target is drawn from a bimodal Gaussian distribution with one mode at $(\frac{M}{4}, \frac{M}{4})$ and the second at $(\frac{3M}{4}, \frac{3M}{4})$. We define a common standard deviation, σ for both modes and a percent, p, which determines the bias between the two modes (i.e., if $p = 0.2$ then 20% of the targets will be drawn from mode 1 and 80% will be drawn from mode 2) An example distribution is shown in figure 5.8.

Definition 18. *A **bimodal target** is drawn from the distribution*

$$P(x,y) \propto p * e^{\frac{((x-x_1)^2+(y-y_1)^2)}{2\sigma}} + (1-p) * e^{\frac{(((x-x_2)^2+(y-y_2)^2)}{2\sigma}}$$

where p is the bias toward mode 1 at (x_1, y_1) and the second mode is at (x_2, y_2).

When the target distribution is heavily weighted towards one of the modes ($p = 0.9$ or 0.1) an interesting behavior emerges. In most simulations one target will move directly towards the most likely mode and start to cluster around that area, looking for the target while the other searcher performs more of a random walk around the rest of the search space. This behavior is most prominent in distributions where $\sigma > 0$ but not large enough for the distributions to overlap. An example is shown in figure 5.9.

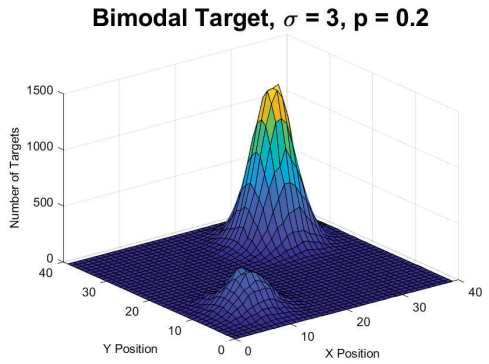


Figure 5.8: Example bimodal distribution on $\mathbb{T}^2_{40}$ with one mode centered at $(10,10)$ and the other at $(30,30)$. The distribution is weighted towards the first mode with $p = 0.2$, meaning there is only a 20% chance of the target being drawn from the Gaussian distribution with $\mu = (10,10)$ and $\sigma = 3$ but an 80% chance of it being drawn with $\mu = (30,30)$.

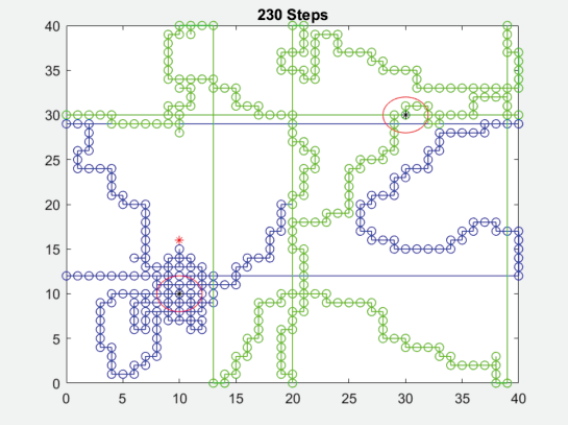


Figure 5.9: Example search path for two searchers in $\mathbb{T}^2_{40}$ with a bimodal distribution, $\sigma = 2$, $p = 0.9$, modes at $(10,10)$ and $(30,30)$. Both searchers started at $(20,20)$. Clustering behavior is shown for searcher one (blue path) while meandering behavior is shown for searcher two (green path). Searcher one found the target in 230 steps.

The best way to measure this occurrence is through histograms of the clustering vs. meandering (random walk) searchers. To do this, we look at each individual trial and the average distances from each searcher to the most probable node. At any time $t \geq 0$ searcher i is at location $(S^i_{x,t}, S^i_{y,t})$. Assuming our distribution is weighted 90% towards the node at $(10,10)$ and it takes the searchers T steps to reach the target, our average distance equation is:

$$D_i = \frac{\sum_{t=0}^{T} ||(S^i_{x,t}, S^i_{y,t}) - (10,10)||_{\ell^1}}{T},$$

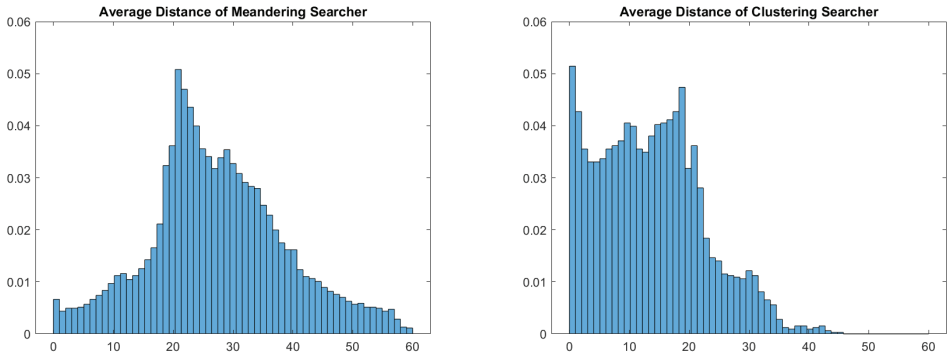
using the ℓ^1 metric on the torus. Then, we separate the distances into two bins based on visual inspection and the general guidelines:

$$D_i \in M \ \ if \ \ D_i \geq 20$$

$$D_i \in C \ \ if \ \ D_i < 20$$

This criteria is based on the size of our search grid such that M is the set of data for meandering searchers and C is the set of data for clustering searchers. The boundary at 20 is only a guideline; some searchers with average distances over 20 exhibit clustering behavior and vice versa. The visual inspection can help determine which behavior was performed by each searcher.

Histograms of the average distances for meandering and clustering searchers is shown in figure 5.10. Here we can see a very clear difference between the two types of searchers. The meandering searcher is on average much farther away from the center of the target distribution mode, whereas the clustering searcher is almost always within 20 steps of it.

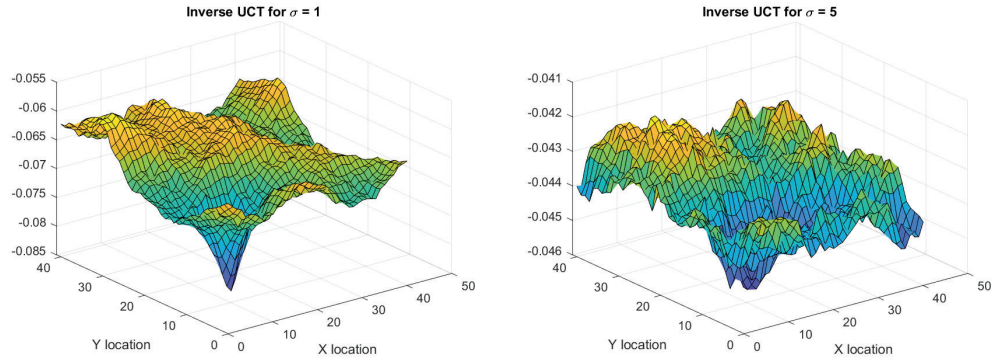


(a) Accumulated data from 112 meandering searchers.

(b) Accumulated data from 88 clustering searchers.

Figure 5.10: Frequencies of average distance from each searcher to the center of the most likely mode in a bimodal distribution on $\mathbb{T}^2_{40}$, separated between clustering and meandering behavior; 100 trials total.

To further describe this behavior, we show the gravity plots for different values of σ. As you can see in figure 5.11, the UCT values are highest (i.e., the lowest point in the graph) around the more probable target distribution mode. This explains why one searcher almost always "falls" into the area around the high-probability mode and then clusters around the center of that distribution.



(a) Smaller σ; more certain target.

(b) Larger σ; less certain target.

Figure 5.11: Example UCT gravity plots on $\mathbb{T}^2_{40}$ with a bimodal distribution 90% skewed towards the target at $(10, 10)$.

Overall, we determine that the Monte Carlo Tree Search method finds an intuitive search pattern for the bimodal target distribution. We can compare this to human search behavior by imagining a scenario where you leave your car in a crowded parking lot at an amusement park. If you are 90% sure that you left it in lot A but think that maybe it could be in lot B ($p = 0.9$), the most logical solution would be for one friend to go quicly look around lot B while you thoroughly search lot A. This is exactly how the MCTS method divides the work between two searchers: one does a "quick look" around the grid by performing a meandering walk while the other does a "thorough search" by clustering around the area where the target is most likely to be. This behavior shows how the MCTS is able to adapt to the scenario, making it a promising area of study for real life search problems.

6 Conclusions

In this paper we studied a search problem on a 2-D finite lattice with periodic boundaries. The level of certainty about the target's location ranged from complete information (delta distribution) to completely unknown (uniform distribution). The searcher followed a Monte Carlo Tree Search algorithm with the UCT selection policy, using a reward function based on the inverse of the number of steps it took the searcher to find the target. In our study we varied the target certainty, number of searchers, and target distribution. Additionally, two default policies for the MCTS were compared: a random walk and the Lévy flight search.

From numerical simulations we discovered that the MCTS program learns to find the target efficiently for all target distributions. There are, however, diminishing returns for less certain targets, especially when $\sigma > 10$ (corresponding to 95% of the targets being found within a 20×20 subsection of the 40×40 grid). In the case where the target is certain (drawn from the delta distribution), the searcher converges to the optimal path as the number of simulation loops (or decision time) grows. Interestingly, we found that between the two default policies, Lévy flight search out performs the random walk based on loops per decision where as the random walk default policy is better for a restricted decision time. This could be because the random walk is more efficient at searching the moderately sized grid (40×40) that we were using. We expect that a Lévy flight default policy would be optimal in both cases on larger grids.

Theoretically we prove two theorems for the single searcher scenario. The first theorem shows that for a certain target (delta distributed), a MCTS searcher will converge to the optimal path under mild assumptions. The key to this proof is that the default policy must have a stopping time which, on average, is shorter the closer you start to the target. This result shows that MCTS can be used to find an optimal solution to our certain-target search problem. The second theorem shows that for a uniformly selected target, i.e. no information given, as the grid grows larger the MCTS with a random walk default policy converges to a nearly self avoiding random walk. This result shows us what the optimal strategy is for an uncertain target and allows us to execute that in a more computationally efficient way, searching randomly except to avoid looking in the same place twice.

For the multi-searcher scenario we provide data that shows a few interesting behavioral patterns for Monte Carlo Tree Search searchers. First, we discovered that two searchers starting equidistant from a certain target perform better together when they start near the same location. We infer this as a result of the exploration-exploitation balance of the UCT algorithm which pushes the searchers to stay far away from each other while also moving towards the target. When the searchers start on opposite sides of the target they are unable to balance these two goals without taking suboptimal paths.

These conclusions about a two searcher program with a certain target are expanded to explain the behavior of K searchers under the same assumptions. We provide data on the spreading behavior of these searchers, showing that more searchers will cover the search grid more thoroughly and thus are more likely to find the optimal path.

Finally, we look at a bimodal target distribution and study the behavior of two searchers which start together between the two modes. When one mode is far more likely than the other we find that the searchers will work together, one moving directly towards the most likely area and performing a clustering pattern while the other moves in a meandering path towards and around the less likely target. This is very similar to human intuition on how to find a similarly defined target, showing that the MCTS method is a potential tool for real-life scenarios.

Since our results are so promising we suggest that the method be applied to a continuous search space. By continuing to work towards more complex problem set-ups we may learn more about the MCTS and how it can inform human decisions. Ultimately, we hope that the behavior shown through Monte Carlo Tree Search programs will provide human searchers with more efficient behaviors to adapt their search patterns and find targets more efficiently. The uses could range from search and rescue of hikers in the woods all the way to acquisition of military targets. Search problems are present in nearly all areas of life, so the patterns and behaviors shown by the MCTS method will no doubt be useful in many applications.

Overall, we conclude that Monte Carlo Tree Search can be effectively applied to a range of search problems and that the efficiency varies depending on the initial parameters. These results can be generalized by saying that MCTS searchers perform efficiently with a well-defined target but the method is not computationally beneficial for a random target. Once multiple searchers are introduced to the problem their behavior shows an organized team effort, also dependent on the target distribution. For now, these results provide useful strategies and insight into effective search behaviors, however, more study is required to be able to apply the MCTS method to real-time decision making.

REFERENCES

[1] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[2] Richard Bellman. *Mathematical optimization techniques*. Univ of California Press, 1963.

[3] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[4] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[5] Vincenzo Fioriti, Fabio Fratichini, Stefano Chiesa, and Claudio Moriconi. Levy foraging in a dynamic environment–extending the levy search. *International Journal of Advanced Robotic Systems*, 12(7):98, 2015.

[6] Douglas W Gage. Randomized search strategies with imperfect sensors. In *Mobile Robots VIII*, volume 2058, pages 270–279. International Society for Optics and Photonics, 1994.

[7] Richard Kelly and David Churchill. Comparison of monte carlo tree search methods in the imperfect information card game cribbage.

[8] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[9] Levente Kocsis, Csaba Szepesvári, and Jan Willemson. Improved monte-carlo search. *Univ. Tartu, Estonia, Tech. Rep*, 1, 2006.

[10] Elana. Kozak. Trident-2021. https://github.com/ekozak7/Trident-2021, 2020.

[11] Jacek Mańdziuk. Mcts/uct in solving real-life problems. In *Advances in Data Analysis with Computational Intelligence Methods*, pages 277–292. Springer, 2018.

[12] Daniel Marthaler, Andrea L Bertozzi, and Ira B Schwartz. Levy searches based on a priori information: The biased levy walk. Technical report, CALIFORNIA UNIV LOS ANGELES DEPT OF MATHEMATICS, 2004.

[13] G Roelofs. Monte carlo tree search in a modern board game framework. *Research paper available at umimaas. nl*, 2012.

[14] GM Viswanathan, V Afanasyev, Sergey V Buldyrev, Shlomo Havlin, MGE Da Luz, EP Raposo, and H Eugene Stanley. Lévy flights in random searches. *Physica A: Statistical Mechanics and its Applications*, 282(1-2):1–12, 2000.