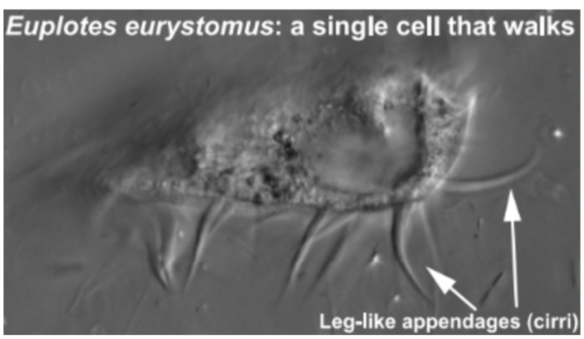
Determining an Optimal Gait for a Unicellular Walker

Eric Carson

ABSTRACT

Monte Carlo Tree Search (MCTS) is a heuristic search algorithm widely used in decision-making processes for complex domains, including games, robotics, and optimization problems. The algorithm combines the principles of random sampling with tree-based search, iteratively building a search tree by exploring promising actions while balancing exploration and exploitation. MCTS consists of four main steps: selection, expansion, simulation, and backpropagation, which work together to identify optimal strategies by simulating potential outcomes and updating the search tree based on empirical results. Its ability to handle large, unstructured decision spaces without requiring domain-specific heuristics makes it a versatile tool for solving problems with high computational complexity.



Euplotes cell architecture (Larson)

INTRODUCTION

This research project, which is in collaboration with Agung Julius from Rensselaer Polytechnic Institute and Ben Larson from the University of California, Berkeley, builds upon prior work by Larson on the locomotion of the unicellular organism Euplotes eurystomus (pictured above) [1]. Larson's earlier study focused on identifying patterns in the movement of E. eurystomus's 14 leg-like appendages, known as cirri, which enable the organism to navigate its environment. Unlike multicellular organisms with central nervous systems that facilitate coordinated appendage movement, unicellular organisms like E. eurystomus lack such centralized control mechanisms [1]. This raises intriguing questions about how coordination and effective locomotion are achieved at the unicellular level. The study aims to explore these mechanisms further, advancing our understanding of unicellular locomotion and its underlying principles.

Since the publication of Larson's study, a simulator replicating the gait of Euplotes eurystomus has been developed. This simulator revealed that the organism's leg movements are governed by microtubules and that the positions and movements of the cirri can be effectively modeled using a finite-state machine. Building on this foundation, a potential avenue for further exploration in the context of this research is to investigate whether a pattern of cirri movement exists that results in optimal locomotion. Specifically, given the simulator and the finite-state machine representation of cirri movement, this study seeks to determine the optimal sequence of movements that maximizes the distance Euplotes eurystomus can travel. This inquiry could yield valuable insights into the principles of unicellular locomotion and contribute to the broader understanding of efficient gait generation in biological systems.

To address the question of optimal cirri movement, Monte Carlo Tree Search (MCTS) will be employed as the learning method. MCTS is particularly well-suited for this problem due to its capacity to balance exploration and exploitation within a large and complex search space. By iteratively simulating potential movement sequences and updating a search tree with the outcomes, MCTS can effectively identify patterns that maximize the distance traveled. The context of the problem is best modelled as an online learning problem since the only parameters known about the problem initially is the starting leg positions and their locations relative to the body of the organism. MCTS is an ideal choice to solve this problem since it effectively balances exploration and exploitation.

The structure of this paper is outlined as follows: it begins with a review of the literature and previous work related to the study in the subsequent section. This is followed by an in-depth explanation of the implementation, detailing the assumptions made for the problem and the modifications made to the classic MCTS algorithm to better suit the problem. Next, the results are presented, with a focus on analyzing how certain parameters such as the search depth were selected. The paper concludes with a summary of the findings and a discussion of potential future directions for this research.

RELATED WORK

Reinforcement learning is frequently used in finding an optimal gait, and many different implementations of such algorithms exist. The paper "Optimal Gait Control for a Tendon-driven Soft Quadruped Robot by Model-based Reinforcement Learning" by Xuezhi Niu, Kaige Tan, and Lei Feng details an algorithm to optimize gait control for a tendon-driven soft quadruped robot using model-based reinforcement learning (MBRL) to develop a control framework capable of handling the nonlinear dynamics and high flexibility of soft robots [2]. Specifically, the authors used a Soft Actor-Critic (SAC) algorithm since the robot's state and action space were represented using continuous time dynamics. Overall, their results showed the effectiveness of MBRL in achieving stable and efficient gait control. Compared with traditional reinforcement learning methods, both the learning efficiency and locomotion performance improved. Fewer training iterations were required to converge to an optimal gait, showcasing the model's ability to learn complex dynamics with reduced computational costs.

IMPLEMENTATION DETAILS

Monte Carlo Tree search was be the learning method used since the number of iterations would be equivalent to the depth parameter and good to use for exploring different cirri movements (exploration vs exploitation)

State and Action Space Definition

Several assumptions are made to simplify the analysis and simulation of the organism's movement. First, it is assumed that any attempted leg movement is always successful, with a probability P(switch)=1. Second, only one leg is allowed to move at a time, ensuring sequential rather than simultaneous leg movements.

The organism is assumed to be moving on a substrate to which its cirri can attach. Each individual leg can exist in one of two states: 0, indicating that the leg is attached to the substrate and not moving, or 1, indicating that the leg is free from the substrate and returning to its starting position. The state space for the organism is defined as ranging from 0 to $2^{14} - 1$, encompassing all possible configurations of the 14 legs at any given time. For example, a leg configuration such as [0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1] can be represented in binary as b01100100001011, which corresponds to the decimal value 6411. This representation enables systematic exploration of potential gait patterns within the state space.

The action space is defined as [a1, a2, …, a14], where each action corresponds to switching the state of an individual leg. This framework allows for a straightforward exploration of movement strategies by focusing on single-leg transitions within the defined state and action spaces.

Transition Model Definition

The transition model defines how the system evolves based on the selected action. Specifically, it modifies the state of the leg corresponding to the action taken. For example, if the current state s=6411 and the current action a=move leg 4, the transition would determine the new state by converting the state back to a binary array representation and toggling the fourth element. In this case, 6411 can be represented as [0, 1, 1, **0**, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1] before the action to move leg 4 is taken. Producing the next state results in toggling the fourth element, resulting in the array representation [0, 1, 1, **1**, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1]. Converting this back to a decimal value results in the new state value 7435.

Since all 2^{14} states can be affected by the 14 actions, the resulting transition model has a size of $14*2^{14}$. For each state action pair, there is a unique state which occurs with a probability P(switch)=1 since the paper assumes attempts made to move a leg are successful.

Reward Model Definition

The reward model evaluates the performance of a sequence of leg movements based on the magnitude of the distance traveled in the x-direction. It is important to note that assessing a single leg movement in isolation does not provide an accurate representation of the overall distance traveled, as certain leg movements may not contribute directly to locomotion but instead position the organism for a subsequent movement that results in significant displacement. The reward model takes as input the current sequence of leg movements and outputs the total distance traveled as a result of those movements. This approach allows the evaluation of cumulative effects over a series of actions, emphasizing the importance of coordinated and strategic leg movements to maximize forward progress.

MCTS Modifications

Instead of using a predefined reward model, the current "best path" is provided to the Monte Carlo Tree Search (MCTS) as a parameter. This approach avoids the computationally expensive task of preemptively calculating the distances traveled for all possible leg movement sequences, which would effectively turn the problem into brute force computation. By incorporating the current path into the evaluation, the algorithm can dynamically assess potential new states in context, leading to more accurate results.

When calculating the reward for the new state, the potential addition is appended to the existing sequence of leg movements to evaluate its impact on the overall distance traveled. The sequence of leg movements is then given as an input to Larson’s model which returns the distance traveled in the x-direction. While this method provides greater accuracy by considering the broader context of movements, it is also significantly more computationally expensive, as the number of steps in the sequence increases with each iteration. Ultimately, this modified reward calculation was used since there was a significant improvement in results.

PERFORMANCE ANALYSIS

In the initial implementation, ties in the reward values for determining which leg to move next were resolved by defaulting to the Python max() function, which always picked the lowest-numbered leg with the maximum reward. This approach resulted in a bias, where movements primarily involved legs 1 through 7, leading to unrealistic and unevenly distributed leg usage.

To address this issue, randomness was introduced as a tie-breaking mechanism when multiple legs share the maximum reward value. By randomly selecting among the tied legs, the updated method ensured more realistic and balanced leg movements, preventing overuse of certain legs and promoted diversity in movement patterns while making the final sequence of leg movements more realistic.

A parameter which is also toggled with frequently in MCTS is the depth parameter. The selection of a depth parameter plays a critical role in balancing computational cost and the quality of results. Through experimentation, a depth of 4 was identified as an optimal balance. MCTS was run with depth values ranging from 1 to 10 for 15 steps, and it was observed that after a depth of 4, the maximum distance traveled did not significantly improve.

This indicates that increasing the depth beyond 4 yields diminishing returns in terms of performance, while substantially increasing computational costs. By using a depth of 4, the algorithm achieves efficient exploration of the state space while maintaining a manageable computational overhead, ensuring both accuracy and practicality in identifying optimal movement strategies. The figure below shows the numerical results of the MCTS depth and the resulting distance traveled.

```
Depth: 1, Distance: 0.607274099662532
Depth: 2, Distance: 0.2133298526129473
Depth: 3, Distance: -0.02739009004908355
Depth: 4, Distance: 1.5688058799017406
Depth: 5, Distance: 1.5741929361730238
Depth: 6, Distance: 1.6143900553268453
Depth: 7, Distance: 1.7136427276961492
Depth: 8, Distance: 1.56844012136148 8
Depth: 9, Distance: 1.605148460225943
Depth: 10, Distance: 1.6942961499919185
```

MCTS Depth vs Distance Traveled

SUMMARY & RESULTS

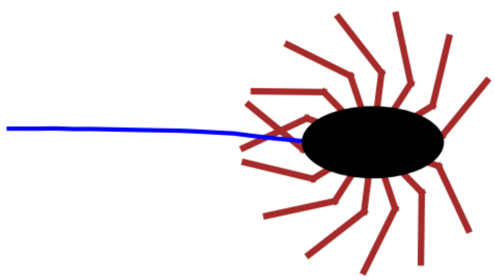
Upon completion, the program writes the optimal sequence to a file, each step providing the starting state and the leg which is moved to reach the next state. Afterwards, the list of (x, y) coordinates are given to show the progression of the simulation over time. The figures below show example results:

```
Trajectory MCTS:
State: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], Action: leg7
State: [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0], Action: leg2
State: [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0], Action: leg3
State: [0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0], Action: leg8
State: [0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0], Action: leg14
State: [0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1], Action: leg9
State: [0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1], Action: leg10
State: [0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1], Action: leg12
State: [0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1], Action: leg6
State: [0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1], Action: leg1
State: [1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1], Action: leg4
State: [1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1], Action: leg4
State: [1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1], Action: leg4
State: [1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1], Action: leg13
State: [1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1], Action: leg4
State: [1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1], Action: leg5
State: [1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1], Action: leg4
State: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1], Action: leg11
State: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], Action: leg11
State: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1], Action: leg11
```

```
Coordinates:
(0.000000, 0.000000)
(0.000000, 0.000000)
(0.000000, 0.000000)
(0.050000, 0.000000)
(0.090089, -0.000088)
(0.143161, -0.002145)
(0.205301, -0.005247)
(0.273335, -0.003228)
(0.356309, -0.002164)
(0.456500, -0.000120)
(0.579387, 0.003293)
```

```
(0.579387, 0.003293)
(0.738716, 0.002687)
(0.943296, 0.001551)
(1.211868, -0.000409)
(1.324634, -0.003077)
(1.485222, -0.009790)
(1.790037, -0.009133)
(1.953215, -0.008505)
(2.317605, -0.013729)
(2.743359, -0.019165)
```

In the above example, the organism traveled a net x-distance of 2.743359 units over 20 steps. To help visualize what this may look like, the program uses this information to create an animation of an object and its path of motion. The figure below shows what the end result of the animation could look like:



The predicted optimal movement pattern for the cirri involves transitioning them from a fixed state to a free state in a front-to-back order, which generally matches the results.

FUTURE WORK

The model itself can be modified in the future in two ways: increasing complexity by allowing multiple legs to move in a single iteration and improving the reward function to decrease computational complexity.

Allowing multiple legs to move simultaneously would exponentially increase the number of elements in the transition model since the more complex model means all states can be accessed by another state. With 2^{14} state options, this would bring the size of the transition model to 2^{28} states, a size much larger than the current $14*2^{14}$. A potential modification is to calculate the transitions in real time and cache the results. This both reduces the computational requirement since all 2^{28} combinations do not need to be computed while simultaneously saving already calculated values for a further reduction in the computational requirement. For the caching, a Least Recently Used (LRU) cache can be utilized. This modification would more accurately reflect the complex, coordinated movements that might occur in real-world locomotion, potentially leading to faster and more efficient movement strategies. By enabling parallel leg actions, the model could better simulate natural gait patterns, offering a more dynamic and realistic approach to gait optimization.

Redesigning the reward model to improve computational efficiency without sacrificing significant accuracy can be achieved by limiting the amount of context given when calculating the reward. The current model, which computes the total distance traveled based on the entire sequence of movements, can be computationally expensive and increases in the computation time since the sequence of movements increases as the iterations increase. Rather than providing the entire sequence of movements, a maximum of the five most recent steps can be used to calculate the reward. This modification allows context to be provided about the current state action pair while eliminating the growing computation cost. Theoretically, the resulting distance traveled should not be significantly affected since context about preceding movements is still provided.

References

1. B. T. Larson, J. Garbus, J. B. Pollack, and W. F. Marshall, "A unicellular walker controlled by a microtubule-based finite-state machine," Current Biology, Aug. 2022, doi: https://doi.org/10.1016/j.cub.2022.07.034 [Accessed: Dec. 6, 2024].
2. V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "Optimal Gait Control for a Tendon-driven Soft Quadruped Robot by Model-based Reinforcement Learning," arXiv:2406.07069v1, Jun. 2024. [Online]. Available: https://arxiv.org/abs/2406.07069. [Accessed: Dec. 6, 2024].

GitHub Link:
https://github.com/carsoe2/UnicellularWalker