Autonomous Pathfinding in Boid-Inspired Swarm Networks

Seth Rossman

University of New Hampshire: Computer Science

1 Abstract

Biomimetic swarm algorithms in past work have largely focused on computer graphics applications to simulate movements in flocks of birds or schools of fish according to pseudo-randomized movement. In this paper, the author proposes a novel variation of these algorithms that enables organic coordinated movement of multiple agents. Base behaviors are implemented derived from "Boid" logic [2] found in Craig Reynold's Paper *Flocks, herds and schools: A distributed behavioral model* while navigational impulses for each Boid are generated from an implementation of the LRTA* algorithm. Analysis of heuristics and performance metrics is explored to gain insight on the limitations of the system.

2 Introduction

Pathfinding, a subfield of artificially intelligent systems, is positioned at one of the most interesting intersections of computer applications combining elements of robotics, computer science, mathematics, and biology. A sub-field of this, online navigation, enables agents to be placed in environments possessing little to no knowledge of their surroundings with the ability to build, and refine, an understanding of the environment through continual spatial exploration, trial and error, and the continuous intake of sensory information. Algorithms enabling computational speed and real-time reaction to dynamic environments where an agent has an incomplete view of the world are vital for real-life robotics applications such as self-driving cars or autonomous aerial/underwater vehicles (AAVs & AUVs respectively).

For almost all traditional path-finding applications, the A* algorithm is at a minimum, a strong candidate to choose. Paths to be taken are formed through a process of exploring best possible states mediated by a cost evaluation based on a heuristic value and the path's cost-so-far. Each potential state is evaluated against all previously cached states by taking the cost of the path so far, denoted as g(s), and adding the calculated heuristic value, denoted h(s), resulting in an f(s) value found via the equation: $f(s) = h(s) + g(s)$. A priority queue is

used to further explore the most promising states. The most promising states chosen are not strictly neighboring the current state and this is where the algorithm falls short for applications where a decision feedback loop is needed. The optimal path in A* is built by expanding past states that seem more promising than the currently considered state even if the optimal next state to investigate is not explicitly connected to the current state. Due to this lack of subsequent 'connected' moves where an agent's next state S' is the direct result of applying action A onto a state S, A* lacks the ability to be applied in online systems. This can be seen in other off-line algorithms such as IDA* where a complete plan must be developed before committing to a single action leading to computational deadlock and exponential time to run [1] incompatible with the fast-paced environments and immediate needs of operating in real-time.

The concept of a "boid", an agent model with inherent behaviors, and boid swarm logic was first introduced in Craig Reynold's 1987 paper *Flocks, herds and schools: A distributed behavioral model* [2] and laid out a scalable behavioral model where a singular agent acts according to innate behaviors. This decentralized idea of navigation removes the need for a leader-follow swarm topology and instead relies on the strict adherence of each boid to three core principles: *flock centering*, *velocity matching*, and *collision avoidance* in order of ascending precedence. The movement velocities of the boid are changed through the summation of the three innate behaviors and as a result, fluid movement resembling the movement patterns of flocks of birds or schools of fish is achieved. Reynold's foundational work on extrapolating organic movement from boid-centric independent movements allowed for randomized swarm motion akin to a flock of birds or school of fish and predetermined scripted movement paths [2]. However, no intelligent pathfinding was introduced or proposed in the paper so scripting paths would fall entirely onto the researcher who would have to plot a trajectory for the entire flock. It should be noted that the need to plot each line segment of a desired path **for each boid** is circumvented by the flocking behaviors but this is still far from autonomous. From a manual path-finding perspective as well, it would be a necessity to have a complete view of a world, i.e. a fully-observable world, in order to create a collision-free path.

A new system for combining the base tenets of boid logic, velocity matching, flock centering, collision avoidance and algorithmic support for real-time pathfinding qualities must be created to support both swarm behavior(s) and pathfinding in a unified fashion.

This paper aims to lay the groundwork for further research on the integration of pathfinding systems in swarm networks that are characterized by a desire for decentralized behaviors contrasted against other more centralized swarm topologies.

3 Environment Setup

The Python programming language is used in conjunction with the Pygame graphics library to visualize a provided map consisting of obstacles (represented

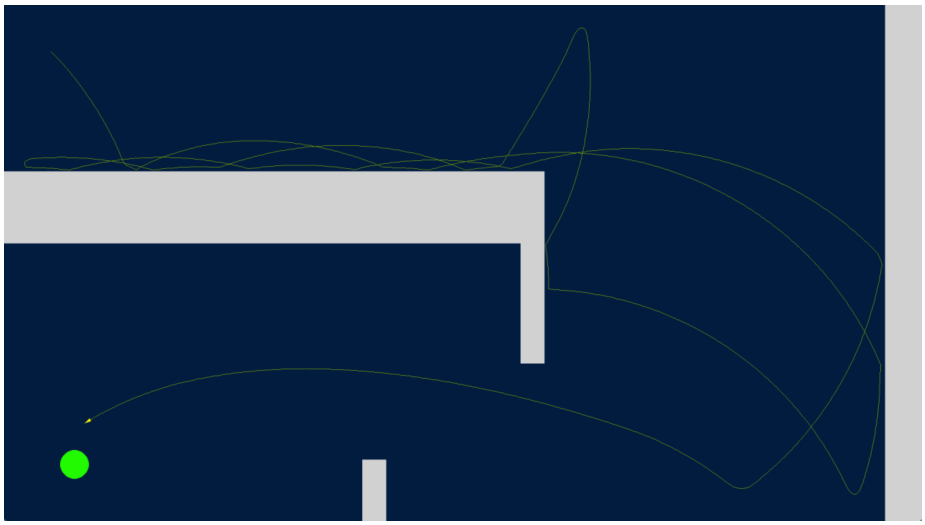

Figure 1: a boid navigating map4.in

by white boxes), an objective (represented by a circle) and the progress of the boids on said map. An example configuration is shown in Figure 1. On creation, a configuration file consisting of two tuples per line (upper left x-coordinate, upper left y-coordinate) and (width, height) are read in and an obstacle is created that has the trait(s) $span_{horizontal} = (x_0, x_0 + width)$ and $span_{vertical} = (y_0, y_0 + height)$. Pygame, like several other graphics libraries, denotes the origin (0,0) as the top-left corner so lower y-values are at the top of the window while higher y-values are closer to the bottom border. The same low-to-high scheme applies for the x-axis. Boid movement can be traced by a line trailing the boid created using a random RGB color code. The actual Boid's shape is represented by an isosceles triangle with the forward movement vector perpendicular to the short of the triangle, extruding through the junction point shared by the longer sides. A boid's life cycle is completed upon touching the circular objective zone and it is immediately removed from the list of boids to be rendered on the next display frame. All movement is tied to an FPS clock (set at 60fps for trials), with each click equaling one execution loop increment. The perceived rotation of the boid during movement is possible through the application of a rotational matrix to a Pygame polygon consisting of 3 vertices denoting the boid's triangular representation. A map is deemed complete when only a quarter of the initial boids remain. Due to the non-pathfinding behaviors influencing the boid's movement, "splitting" of the flock around an obstacle is possible leading to lone boids heavily influencing the average heuristic and time values. For analysis of the overall flock's performance, this cut-off is implemented to reduce skewing of the majority due to misguided individual(s).

4 Approach

Algorithm 1 LRTA*-AGENT with Visited Penalty

Require: s', a percept that identifies the current state
$w_{visited}$, penalty weight for adding 'visited' state bias
Persistent: $results$, dictionary mapping (s, a) to s'
H, dictionary mapping of s to (h_{cost}, $visited$)
s, a, the previous state and action, initially null

1: **function** LRTA*-AGENT($problem, s', h$) **returns** an action
2: **if** s' is $goal$ **then**
3: **return** stop
4: **end if**
5: **if** s' not in H **then**
6: $H[s'] \leftarrow h(s')$
7: **end if**
8: **if** s is not null **then**
9: $result[s, a] \leftarrow s'$
10: $cost_{visited} \leftarrow w_{visited} \cdot number_{neighbors} \cdot H[s]_{update\_counter}$
11: $H[s] \leftarrow \min_{b \in ACTIONS(s)} \text{LRTA*-COST}(s, b, result[s, b]) + cost_{visited}$
12: **end if**
13: $a \leftarrow \arg\min_{b \in ACTIONS(s')} \text{LRTA*-COST}(s', b, result[s', b])$
14: $s \leftarrow s'$
15: **return** a

Given the partially-observable map environment and dynamic nature of flock interactions where an agent's trajectory is influenced by nearby agent's trajectories, the Learning Real-Time A* (LRTA*) algorithm was chosen to add pathfinding ability to each boid. What sets LRTA* apart from simple navigation solely dependent on a heuristic value *h(s)* is the ability to update past heuristic calculations through the re-evaluation of what the previous states best move is. Psuedocode for LRTA*'s implementation is provided in Algorithm 1 and is based off of pseudocode provided in *Artificial Intelligence: A Modern Approach* [3]. Small modifications to the code (particularly on lines 10 and 11) mitigate the risk of entering "movement loops" which are discussed further in the rest of the paper. The calculated LRTA* impulse adjustment to velocity is done by each individual boid instead of a more centralized implementation of a swarm network that determines the ideal heading of a flock, not ideal heading of an individual, by averaging flock member's velocity and position.

With velocity updates coming from the summation of previous velocities with other differing mechanisms for adjusting said velocities, simply having our state naively as solely the (x-coordinate, y-coordinate, x-velocity, y-velocity) would lead to a near uncountable state size. Both coordinate location(s) and velocities in some cases have a decimal precision in the millionths, so saving a

state's representation like this is virtually meaningless in the context of LRTA* heuristic updates. As our state space is limited only by the precision of the data type being used to store the values and as such can theoretically become near-infinite, the odds of returning to a previous state are close to none. This near-infinitesimal probability of revisiting a previous state negates all benefits arising from LRTA*'s heuristic learning property; in fact, it completely eliminates any notion of choosing paths based on updated informed heuristic values, the core motivation for using LRTA*. Subdivision and truncation of the map's area by a constant *nav_mesh_interval* paired with the grounding of hyper-specific angles into n number of angular buckets, reduces the state space from a near-infinite size to being roughly equal to Eq.1, a fraction of the original state space.

For trial runs, the number of buckets corresponding to angles of entry was set to 6 (60^{o} intervals) and *nav_mesh_interval* was set to 50 making the state space size $int(1920/50) * int(1080/50) * (360/6)$. Although not explicitly a multiple of the map's width of 1920 or height of 1080, 50 was chosen through trial and error due to desired results.

$$\frac{width}{height} \cdot \frac{nav\_mesh\_interval}{nav\_mesh\_interval} \cdot n_{angles\_of\_entry}$$

Figure 2: Eq.1

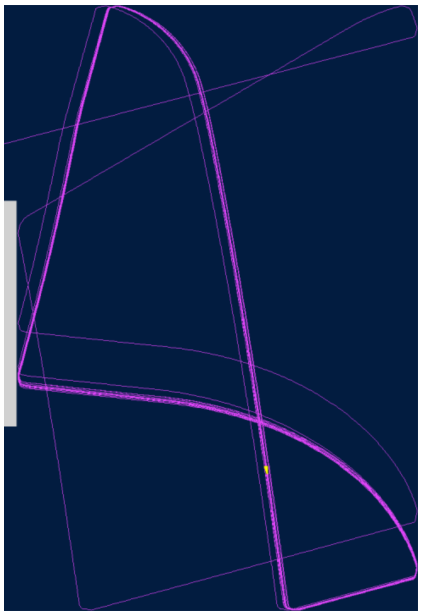

Figure 3: a boid stuck in a movement loop

While revisiting a previous state in simpler implementations such as in a grid-world with discrete movement intervals (i.e. forward/backward/left/right movement all carry an action cost of 1) normally does not impair forward-progress, movement in the Boid-world is inherently more complex due to the summation of many different impulses (velocity matching, centering, collision avoidance, obstacle avoidance, and lastly LRTA*). This combination of factors makes the actual movement change calculated vastly different from the expected movement suggested by LRTA*; it makes the system more closely resemble a non-deterministic system and is inherently more vulnerable to movement loops. This is shown in Figure 3 where the agent fails to take into account how many times a previous state has been visited leading to an inescapable movement loop. This failure to factor in previous states means that the boid's next move is almost-completely dominated by its base distance-from-objective heuristic, whether that heuristic value comes from the Euclidean distance or Manhattan distance from the objective, leading to a loop of choosing the uninformed "best" action.

LRTA*, unlike A*, is unable to "look ahead" to change its movement behavior based on actions several steps down the line, so to dissuade the boid from entering a movement loop, a bias away from previously visited states must be implemented. Line 10 & 11 in Algorithm 1's pseudo-code demonstrate this. The number of times a specific state has been visited is multiplied by a constant scalar $w_{visited}$ and by the number of neighboring boids.

The scalar value must be chosen in relation to the size of the map as the base heuristic's value (Euclidean or Manhattan distance). The value '50' was chosen somewhat arbitrarily for trials as it showed promising results on a 1920 by 1080 map. Further research can explore the behavioral relation of map size to the chosen scalar and more optimally choose a scalar value.

5 Analysis

Flocks consisting of a single boid, three boids, and nine boids were run and analyzed *a posteriori*. The results are found in Figure 4 and Figure 5. The Euclidean distance base heuristic performed much better than the Manhattan distance in almost all tests, with the Manhattan distance heuristic timing out on two different tests (map3.in and map4.in at 9/1 boid(s) respectively) as the boids failed to complete the course (reach the objective) within a reasonable amount of time.

With both heuristics, a generalized pattern is difficult to extract as an increase in boids (or decrease in boids) does not seem to linearly affect the proportional computational time. The lack of a pattern, even in the better performing heuristic of Euclidean distance, indicates that despite tuned heuristic scalars and the incorporation of a weighted penalty for revisiting previous states, more rudimentary confounding navigational factors are hindering the pathfinding performance. These factors include the impulses *velocity matching*, *flock centering*, and *collision avoidance*, in addition to the lower-level movement constraints such as turning radius and max/min

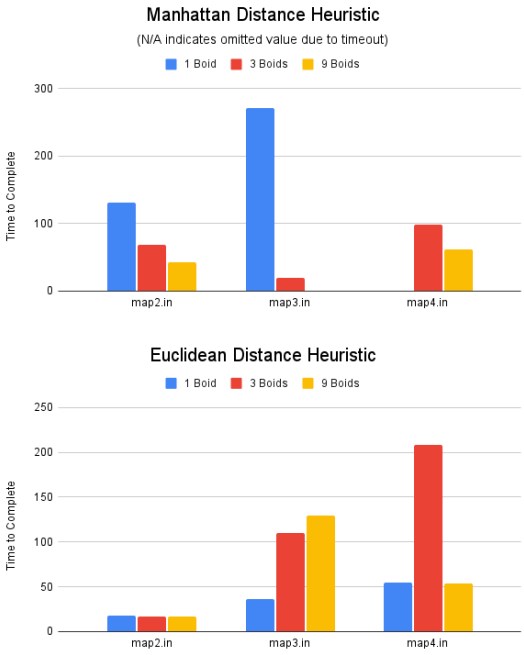

Figure 4: computational time of heuristics on different maps

speed.

This is corroborated in Figure 5 as areas with high heuristic update values (indicated by a darker color) are continually revisited without significant action changes to avoid this looping in the future. As the objective is situated in the lower left-most region of the map, further problems can be observed as boids seem to consistently narrowly miss the objective before continuing in a direction opposite of the positive progress it made up to that point. This is most likely due to collision avoidance and turn radius constraints specifically in corners.

6 Discussion

This paper mainly explored if decentralized, coordinated swarm pathfinding was possible but failed to implement robustness with behaviors. Basic map-building and less-comprehensive wall collision avoidance leads to boid 'clipping' through obstacles at times and less-than realistic steering avoidance of walls. Reynold's mentions this in his paper and proposes a complicated but more organic steering mechanism called 'steer-to-avoid' that proactively finds the closest border of a complex obstacle such as a convex shape or plane [2] before a collision happens and steers beyond that border. Additionally, in the current implementation the 'neighborhood' of nearby boids is calculated using Squared Euclidean Distance for simplicity and potential computational speed-ups but it fails to reflect how close the actual boid distance is, instead using a singular threshold value. The true magnitude of distance between boids can be calculated using the inverse exponential of the distance between boids leading to more realistic movement vector changes. The need for a previously visited weight scalar, $w_{visited}$, is a potentially less-elegant solution to avoiding specific states and actions that decay into movement loops. Although more computationally expensive, implementing LSS (local search space) functionality to help the boids 'see ahead' by some n number of states could help avoid these loops but that is outside the scope of this project.

7 Future Work

This work has many different applications from physical robotic swarm navigation to more advanced simulation or goal-based heuristic development. Beyond what is proposed in the 'Discussion' section of the paper, some next steps could include 1. Applying pathfinding to swarm networks in 3 dimensions rather than just 2 dimensions, 2. Using a more realistic movement framework that supports actions such as temporary speed-ups and intentional separation from the flock, 3. Communication between boids to signal approaching obstacles, and 4. Implementing more biomimicy-inspired behaviors such as different vision models depending on the type of boid or different movement actions

8 Conclusion

Biologically-inspired swarm networks have been implemented in the past using innate behaviors for each agent. The author proposes a new velocity impulse generated by the LRTA* algorithm to 'guide' the agents in a swarm towards a common objective enabling pathfinding in partially-observable environments. The author finds that the confounding factors tied to the non-pathfinding behaviors affecting velocity are, in some cases, overpowering and hinder all progress. The author discusses modifications that can be made to this preliminary exploration of the subject that could aid future research into this topic.

9 Acknowledgments

I want to thank Professor Wheeler Ruml for encouraging ideas like the one in this paper, and for providing the tools needed to actually pursue that idea. I would also like to thank Cornell's electrical engineering department for going through the painful process of finding desirable heuristics for boid behavior.

References

[1] Richard E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2):189–211, 1990.

[2] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34, August 1987.

[3] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 4 edition, 2020.

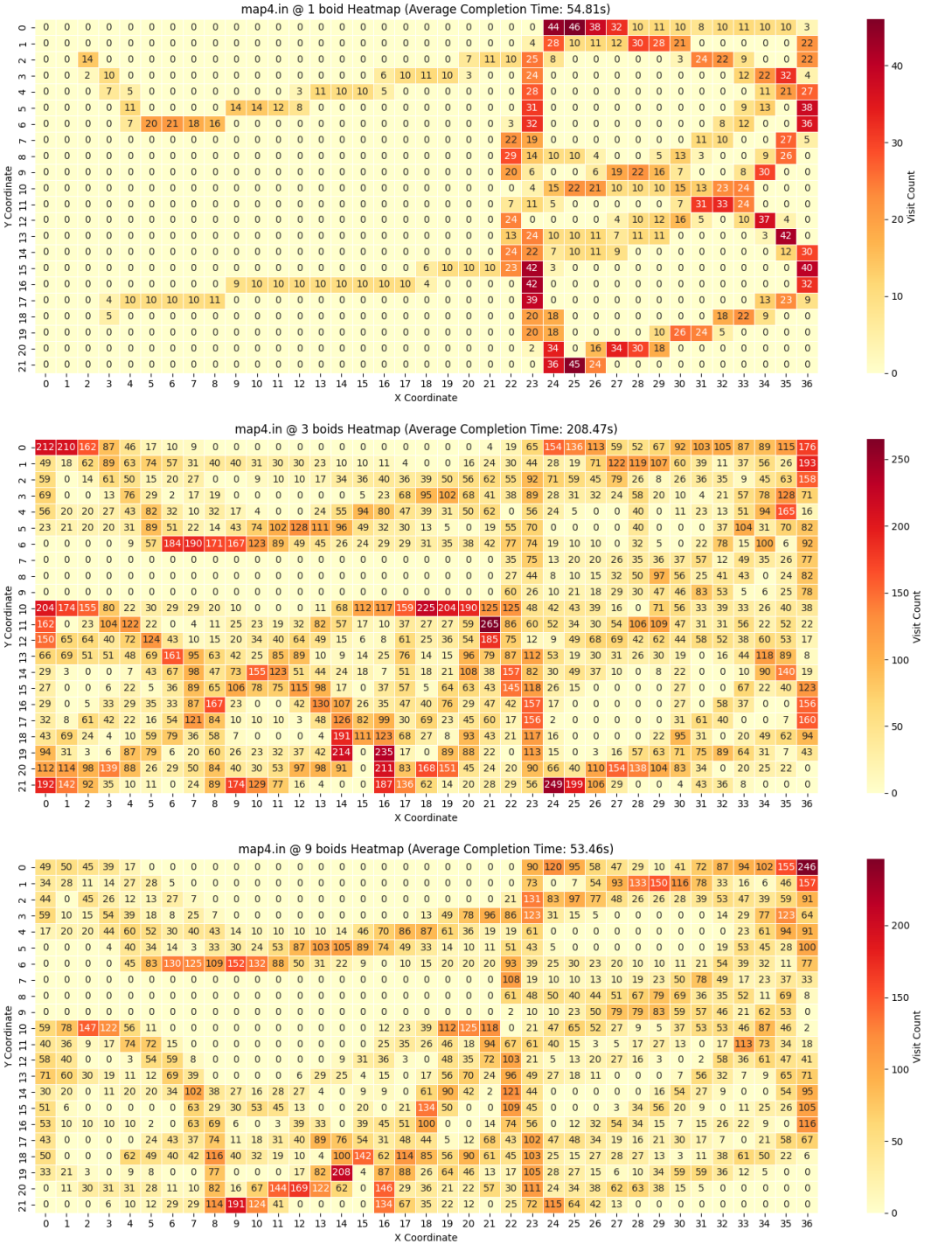

Figure 5: heatmaps of grid cell heuristic updates with differing boid numbers