State-space Search Agent for Programmable Matter

Abbass Abdallah, Sleiman El Hajj, Tony Sleiman

**Abstract**—This report focuses on implementing a state-space search agent for programmable matter that finds optimal paths for matter elements to form desired shapes on a two-dimensional grid. Various approaches with different algorithms were tested and evaluated regarding movement efficiency, computational complexity, and shape formation accuracy. Different approaches were used such as a centralized agent with A\* or with BFS search algorithm to control element movement and a Hungarian algorithm for optimal assignment of elements to target positions. Both Von Neumann (4-directional) and Moore (8-directional) neighborhood topologies were evaluated in this report. For visualization and testing, a grid-based simulation environment was created using Streamlit. Experimental results showed that the A\* algorithm with an effective heuristic function performs efficiently in terms of total moves required to form shapes like circles, triangles, and squares. The report discusses the implementation details, challenges faced in handling conflict resolution between multiple elements, and future directions for improving the programmable matter agent.

——————————  ——————————

# Contents

[Contents 1](#_Toc193272085)

[1 Introduction 1](#_Toc193272086)

[2 Background 1](#_Toc193272087)

[3 Proposal 1](#_Toc193272088)

[3.1 Overview 1](#_Toc193272089)

[3.2 State-Space Definition 2](#_Toc193272090)

[3.3 Implementation 2](#_Toc193272091)

[4 Experimental Evaluation 4](#_Toc193272092)

[5 Conclusion 5](#_Toc193272093)

[6 References 5](#_Toc193272094)

# Introduction

I

n recent years, programmable matter has emerged as a highly promising field in materials science and robotics. Programmable matter refers to materials whose properties can be programmed to change in a controlled manner, allowing them to adapt their shape, properties, or functionality.

This paper presents the design process of an intelligent agent for controlling programmable matter elements on a two-dimensional grid surface for Project 1 of the course relying on search solutions. The objective is two-fold:

# Background

Previous work in programmable matter has explored various approaches to control and coordination. One prominentapproach is the cellular automaton model  **known as the** "Game of Life", where each cell follows simple rules based on its current state and the states of its neighbors.

The control of programmable matter can be characterized along several dimensions **including c**entralized vs.distributed control**, g**lobal vs. local sensing**, and s**ynchronous vs. asynchronous actuation**.**

While distributed control with local sensing is often considered more robust and scalable, centralized control with global sensing can achieve more optimal solutions in controlled environments.

The pathfinding problem for programmable matter elements is similar to the multi-agent pathfinding problem, where multiple agents need to navigate from their initial positions to target positions while avoiding collisions. This is a well-studied problem in artificial intelligence and robotics, with various algorithms proposed for efficient pathfinding, including A\* [3] , Conflict-Based Search [3,4], and Priority-Based Search [5]

# Proposal

## Overview

The programmable matter agent developed in this project consists of several key components to consider:

Grid Environment: A two-dimensional grid where each cell can be empty, occupied by an element, or a wall/obstacle.

Elements: Programmable matter units that can move in the grid.

Target Shape: The desired configuration of elements (circle, triangle, square, or even custom shapes).

Assignment Algorithm: Hungarian algorithm for assigning elements to target positions.

Pathfinding/Post-Assignment Algorithms: A\*, BFS, Leader-Follower algorithms for finding optimal paths and moving into position.

The implementation allows for both Von Neumann (4-directional) and Moore (8-directional) neighborhood topologies, giving flexibility in how elements can move through the grid and allowing for evaluation.

## State-Space Definition

**State Representation**

In the implementation, we considered two alternatives for representing the state space:

1. **Shared State Representation**

In this strategy, state S is defined as the overall arrangement of all the pieces on the board:

S = {(x₁, y₁), (x₂, y₂),., (xₙ, yₙ)}

Where the position of the ith point on the grid, denoted as (xᵢ, yᵢ). This representation captures the system state as a whole and enables full reasoning about the interaction of the elements. The combined state space grows exponentially with the number of the elements. In an n × m grid with k points, the theoretical size for the state space is:

|S| = (n×m)ᵏ

2. **Decoupled State Representation**

Due to the computational intractability of the representation for the states that are joined up, we largely utilized the decoupled methodology where the individual's state was represented separately:

Sᵢ = (xᵢ, yᵢ)

We can with this representation determine the size of the individual's state space as:

|Sᵢ| = n×m

This reduces the complexity considerably, making the search problem itself easier. This ease, however, introduces other complexities in managing interactions and conflicts between the elements, which can be managed by applying specific conflict-resolution methods.

**Actions**

For Von Neumann Neighborhood (4-directional), the possible actions for each unit of matter is:

A = {UP, DOWN, LEFT, RIGHT}

For Moore Neighborhood (8-directional):

A = {UP, DOWN, LEFT, RIGHT, UP-LEFT, UP-RIGHT, DOWN-LEFT, DOWN-RIGHT}

To avoid adding further complexities, we assumed in our implementation that all movements have uniform cost regardless of the chosen topology.

**Goal Test**

The goal test determines when elements have reached their targets and have formed the required shape. Success is achieved when all agents have reached their respective targets.

**Heuristic Function**

Manhattan distance

Number of neighbors

## Implementation

The report distinguishes three approaches to the programmable matter pathfinding problem, based on different algorithmic strategies and element coordination mechanisms. Each approach addresses specific challenges in the state-space search for programmable matter shape formation.

**Hungarian Algorithm for Optimal Assignment**

The assignment of the elements to their target locations is a basic assignment problem, for which we aim to minimize the aggregated Manhattan distance covered by all the elements. We solve the assignment problem by using the Hungarian algorithm.

It begins with the creation of a cost matrix with objects as rows, target positions as columns, and the cells as the Manhattan distance between the two cells. Pre-computing the shape-specific priority rearrangement so that the assignment considers the global distance optimality as well as the shape-specific formation method precedes the creation of this cost matrix.

The Hungarian algorithm proceeds to work on this matrix with a sequence of column and row operations to determine the optimum assignment with the minimum possible total cost. The algorithm has the guarantee to provide a globally optimum solution in polynomial time (O(n³ where n = number of elements), faster than would be found by checking all the possible assignments.

For cases where the number of elements differs from the number of target positions, we employ an extended version of the Hungarian algorithm that handles rectangular cost matrices. This allows for handling of scenarios where not all target positions need to be filled or where there are more elements than required for the shape.

**Breadth-First Search with Eight-Directional Movement**

Our first implementation adds eight-direction movement via a Breadth-First Search algorithm using the Moore neighborhood. This implementation results in the agents moving diagonally, as well as the cardinal directions, which reduces the final path length as well as the formation time.

The BFS solution traverses the grid in systematic expanding circles centered on the start point, with the shortest path in terms of the number of steps. Unlike the implementation in the solution provided, where the solution order that the graph gets visited depends on a heuristic, the BFS solution goes through all the points at the current distance prior to considering more distant points. This guarantors optimality when the graph is unweighted and performs favorably with the grid-based setting where all the moves are the same cost.

An important feature of the implementation here is the dynamic obstacle handling, where the positions of the other entities are considered as temporary obstacles while planning the path. This enables each entity to plan the path according to the current location of the other entities, but with the requirement to replan with high frequency since the environment continuously changes with each step of movement.

Eight-direction motion increases the mobility of the objects, as the objects are able to select more direct routes towards their destinations. On terrain without obstacles, it reduces the length of the path by about 30% relative to four-direction motion. This higher mobility is offset by higher conflict complexity, as more opportunities exist for the points where simultaneous moving objects interact with each other.

**A\* Search with Four-Directional Movement**

Our second implementation has a centralized controller with complete knowledge about the environment, where we employ an adapted version of the A\* algorithm designed for grid-based motion. Our implementation employs four-directional motion (Von Neumann neighborhood), as this produces a clear model for motion that directly facilitates conflict resolution with path optimality being retained due to the constraints on motion.

The A\* algorithm was used because of its optimality guarantees with the use of admissible heuristics and its relative efficiency compared to other search algorithms. The Manhattan distance is used as the heuristic function in our implementation, which is admissible and exactly matches the actual minimum distance in four-directional movement models. This results in all of the paths found being optimal in terms of the number of steps taken.

It begins considering the current cell position and explores the grid systematically so that the target position is reached or all accessible cells are visited. During the process, the algorithm maintains a priority queue of the places to visit, ordered by their estimated total cost. For each cell position visited, the algorithm examines the four cardinal directions (up, down, left, right), and the valid direction is added to the queue to be visited in the future.

A primary optimization in the implementation we use is a visited set to prevent visiting already visited locations and therefore having to revisit already visited locations. This saves computational cost significantly on hard cases involving a high number of obstacles or narrow passages. On finding a path, the algorithm backtracks the path from the goal node to the start location using parent pointers held during the traverse.

**Shape-Specific Priority Strategies**

A novelty in our solution is the usage of shape-dependent priority heuristics for the target-assignment decision. Different shapes of the target (square, rectangle, circle, and triangle) require different formation strategies in the attempt to prevent conflicts and economically disperse the elements. We employed the following special priority schemes:

Center-First Priority: In rectangle, square, and circle shapes, objects are organized based on their distance from the center point of the shape. This technique begins with the center region of the shape and fills outward. Overcrowding is minimized because objects naturally expand outward towards the center.

Bottom-First Priority: For the triangular shapes, the bottom row up priority is assigned, with the base row being occupied by the elements, and the subsequent rows on top. This approach considers the natural stability the triangular shapes possess, with the base needing to be sturdy enough for the subsequent rows to be set on top. With the bottom rows prioritized, we reduce path conflicts as the elements stack up bottom to top.

**Conflict Resolution through Prioritized Movement**

**Dynamic Replanning with Conflict Avoidance (BFS Implementation)**

1. Elements are initially assigned priority based on their distance to their destinations, with closer elements having higher priority. This minimizes deadlocks as it enables nearer elements to their destinations to make their routes earlier.

2. Every agent calculates its path dynamically at each step using the current locations of other agents as obstacles, with this making constant adaptation to the changing environment possible.

3. Conflicts are resolved through a two-step mechanism:

* Initially, elements keep their favored positions in reserve, with contention when more than one element try to move to the same position.
* When the conflict is met, the lower priority item remains there.

4. An exception handler addresses the case of direct exchanges (where two objects would exchange places), so that the two objects during that step do not move in order to prevent potential collisions.

This replanning proves to be most valuable in dense situations where the paths need to be frequently altered as objects shift about. Replanning on each step, objects are able to react to the current environment, potentially using different paths when their preferred paths are blocked.

**Prioritized Movement with Waiting (A\* Implementation)**

This implementation utilizes a conflict resolution method based on prior planning enhanced with considerations about shapes:

1. The entities are ordered based on both their distance to their destination and based on their shapes. For the triangular shapes, the entities that are directed towards the apex are given more priority so that the formation goes on smoothly from the ground level upwards.

2. For each unit, the path towards the target is calculated by the A\* algorithm, the other units' current locations as obstacles.

3. If conflicts happen (multiple members attempting to go to the same position), the lower-priority member remains where it currently is during that time step.

4. "Collision conflicts" are handled specially where two objects exchange places, with the two objects remaining stationary in the attempt to reduce potential collisions.

# Experimental Evaluation

Live data is presented in the implementation of each algorithm

# Conclusion

# This work aimed to develop a state-space search agent for programmable matter that efficiently forms target shapes on a two-dimensional grid. By evaluating different algorithms and methodologies, we explored solutions for pathfinding, optimal assignment, and coordination of multiple agents. Our findings highlight the effectiveness of A\* with an appropriate heuristic function, the advantages of BFS in certain scenarios, and the role of the Hungarian algorithm in minimizing movement cost.

# Moving forward, we plan to enhance the system in the following ways:

# Leader-Follower Algorithm Improvements: Improve the leader-follower strategy by optimizing movement coordination and ensuring efficient formation of complex shapes.

# Handling Edge Cases in Blocked Paths: Address scenarios where paths are completely blocked, enabling better rerouting and avoiding deadlocks.

# Adversarial and Stochastic Search Techniques: Explore adversarial search methods and stochastic approaches such as Simulated Annealing or Genetic Algorithms to improve adaptability.

# Gradient-Based Techniques: Implement gradient-based approaches for movement planning, enabling more dynamic and efficient shape formation.

# Supervised Learning Integration: Investigate the use of supervised learning techniques to enhance decision-making in complex environments.

# References

[1] P. Norvig and S. Russel,Artificial Intelligence: A Modern Approach. S.l.: s.n., 2010.

[2] "Solving assignment problem using min-cost-flow". Algorithms for Competitive Programming. 17 July 2022. Retrieved 14 May 2023.

[3] R. Stern, “Multi-Agent Path Finding – an overview,” in Lecture notes in computer science, 2019, pp. 96–115. doi: 10.1007/978-3-030-33274-7\_6.

[4] Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. Artificial Intelligence, 219, 40-66.

[5] Silver, D. (2005). Cooperative pathfinding. In Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (pp. 117-122).