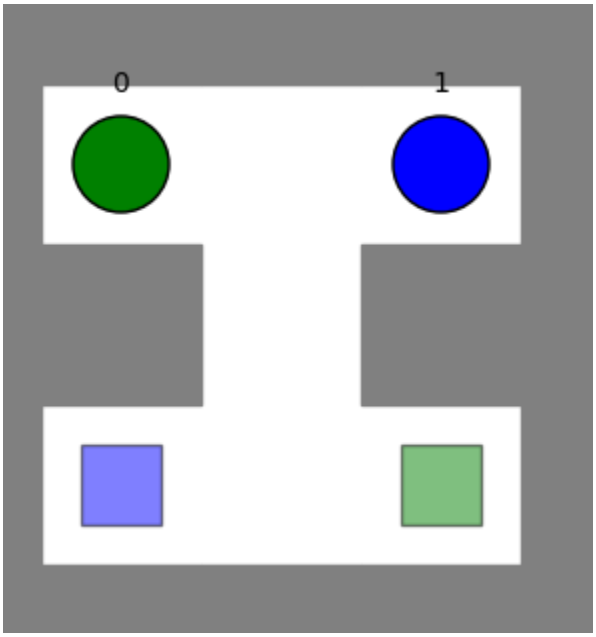
Improved CBS With Heuristics

1. Introduction

Multi-Agent Path Finding (MAPF) is an important topic in many fields, such as robotics, gaming, and AI. In this project we will implement and analyze different algorithms for the MAPF Problem. MAPF is defined as follows: given a set of agents and their respective start and goal locations, determine a set of minimum cost paths that lead the agents to their goals while avoiding collisions between agents and other obstacles.



A commonly used algorithm for solving MAPF problems is the Conflict Based Search (CBS). CBS can be separated into two levels, the high level search and low level search. In the low level search, we will use A* search to find the shortest paths for each agent within a set of constraints. Note, that the low level search does not consider collisions between agents, this is the responsibility of the high level search. The high level search takes the paths returned from the low level search, checks for collisions between agents, and adds new constraints when pairs of paths result in a collision. The algorithm alternates between the low level search and high level search until we find a collision free solution.

2. Implementation

2.1 Improve CBS/CBS with Heuristics (CBSH)

To further improve CBS, we can add heuristics, an approximation of how close a given state is to a collision free solution, at the high level search. The heuristic helps us better determine what constraints to add that will likely lead to a collsion free solution sooner. In this project, we will take a look at three heuristics, namely the CG, DG, and WDG heuristics. The order of the heuristic from most informative to least informative is WDG, DG, CG. The main draw back of a more informative heuristic is that it takes more time and resource to calculate. Hence, we will also analyze and compare the performance of each heurisitc for different types of instances.

2.2 MDD

Before delving into the explanation of the three heuristics, it's important to introduce the concept of Multi-Value Decision Diagrams (MDD). MDD serves as a graphical representation designed to encompass all valid paths that adhere to an agent's constraints. Notably, these paths are structured in such a way that ensures they remain free of cycles.

To efficiently identify conflicts or dependencies between pairs of agents, we leverage a comparative analysis of MDD graphs. These graphs are generated using a modified version of the A* algorithm. A* is a widely-used algorithm for finding the shortest path from an agent's starting location to their goal. In our modified A* version, we go beyond finding a single shortest path; we seek out all shortest paths and employ them to construct the MDD tree, which captures the various shortest paths for the agent.

2.3 CG Heuristic

The CG heuristic, or Cardinal Graph Heuristic, is a valuable approach in the realm of Multi-Agent Path Finding. It focuses on identifying and evaluating conflicts that are of cardinal importance. Cardinal conflicts are pivotal because resolving them is necessary to reach a collision-free solution with a cost similar to the current state. CG provides insights into the critical conflicts that, when resolved efficiently, pave the way for a cost-effective and safe multi-agent path. To find cardinal conflicts we can compare MDDs between agents. A cardinal conflict is represented as a single vertex where both agents are at the same location at the same time and there are no alternative paths for either agent to take, resulting in them being in different locations.

2.4 DG Heuristic

The DG heuristic, or Distance Graph Heuristic, is a heuristic approach that places a strong emphasis on understanding the spatial relationships between agents in Multi-Agent Path Finding scenarios. It aims to identify dependencies between agents, which signifies that the paths each agent chooses can potentially lead to collisions with one another. In other words, two agents are considered "dependent" if, in every pair of optimal paths they might take, there exists the potential for a collision between them.

In essence, the DG heuristic provides a valuable insight by ensuring that, as agents progress along their optimal paths, there will always be critical conflicts to resolve. This helps in guiding the planning process towards addressing these conflicts early, thereby contributing to a more efficient and safe multi-agent path planning strategy.

In practice, one effective way to identify these dependencies is to combine the MDDs of different agents. By examining the MDDs together, you can identify situations where the paths of two or more agents intersect or overlap, potentially leading to conflicts or dependencies.

2.5 WDG Heuristic

The WDG heuristic, or Weighted Distance Graph Heuristic, builds upon the DG heuristic by introducing weighted edges into the distance graph. These weights reflect the significance of specific agent-to-agent relationships in terms of collision avoidance. By assigning different weights to edges, the WDG heuristic acknowledges that not all agent interactions carry the same level of importance. This allows for a more nuanced evaluation of conflicts and the prioritization of certain paths or edges over others in the pursuit of efficient collision avoidance.

3. Research Plan

Our goal is to examine how the performance of the Conflict-Based Search with Heuristics (CBSH) algorithm degrades as the complexity of problem instances increases due to higher agent density. We aim to understand how the agent density, calculated as the ratio of the number of agents to the number of open locations in the instance, influences the efficiency and effectiveness of CBSH. By analyzing CBSH's performance across different levels of agent density, we seek to determine whether increasing density through the addition of more agents or obstacles affects the algorithm differently.

Definition of Agent Density: Agent density is calculated as follows: Agent Density = Number of Agents / Number of Open Locations.

Instance Generation: We will create problem instances with varying levels of agent density. These instances will be of varying sizes and complexities, with random initialization of agent start and goal locations.

Density Subcategories: We will categorize instances into three density subcategories:

- **Low Density**: Agent Density in the range of [0.05, 0.10)
- **Medium Density**: Agent Density in the range of [0.10, 0.15)
- **High Density**: Agent Density in the range of [0.15, 0.20)

Time Limit: For each instance, a time limit of 10 minutes is set for algorithm execution. We have chosen an upper bound of 0.20 for the highest density category because, beyond this threshold, we were unable to consistently solve the problem within the time limit for any algorithm.

Performance Metrics: We will measure the performance of CBSH in terms of execution time and the number of states examined. Execution time is crucial, given the 10-minute limit, and the number of states examined provides insight into the computational efficiency of CBSH. This metric reflects the algorithm's ability to find the optimal solution within the given time constraint and is particularly relevant to improving time performance by reducing the number of states checked.

Comparison with Heuristics: We will employ the CG, DG, and WDG heuristics in our analysis and compare their performance at different agent densities. This will help us assess how CBSH with different heuristics adapts to increasing complexity.

4. Hypothesis and Prediction

We predict that in low-density instances, the choice of a more expensive heuristic will result in relatively worse performance. The simplicity of these cases allows CBSH to efficiently find solutions, even with less accurate heuristics.

As the agent density increases, we anticipate that the more accurate heuristics (CG, DG, and WDG) will become more valuable. The growing complexity introduces numerous conflicts and dependencies, making these heuristics crucial for efficient problem-solving. This demonstrates the trade-off between expanding more states and the runtime needed to compute the heuristic.

5. Experimental setup

This project was implemented with Python 3.9.5 64-bit. Tests were conducted on a Windows 10 PC with an AMD Ryzen 7, 8-core processor (CPU) running at 3.9GHz base clock speed. The computer also has 32GB of RAM available.

6. Experimental results

In this section, we present our findings and analysis based on the performance of the Conflict-Based Search with Heuristics (CBSH) algorithm in Multi-Agent Path Finding (MAPF) scenarios. Our investigation focuses on the impact of increasing agent density on CBSH performance, whether achieved by adding more agents or obstacles.

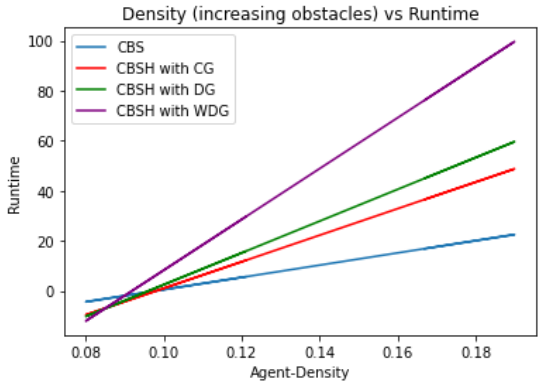


Figure 1.1: Runtime of different heuristics for increasing levels of density (by **increasing obstacles** in the instance)

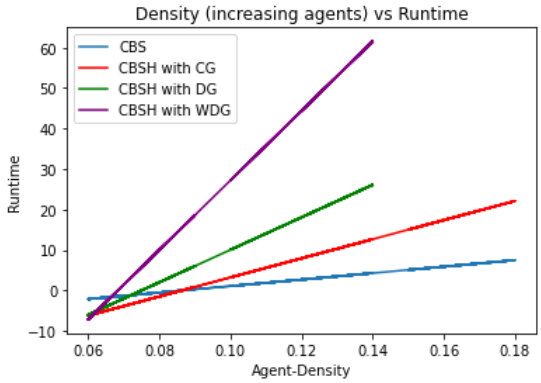


Figure 1.2: Runtime of different heuristics for increasing levels of density (by **increasing agent count** in the instance)

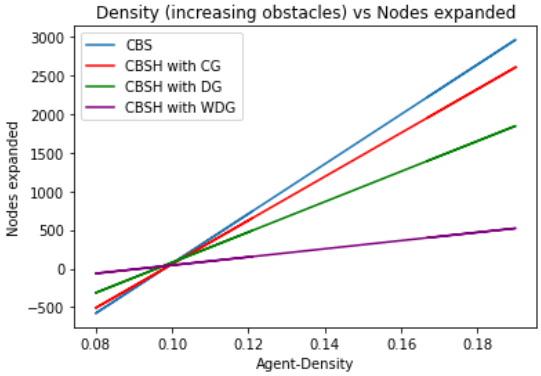


Figure 2.1: Number of nodes expanded of different heuristics for increasing levels of density (by **increasing obstacles** in the instance)

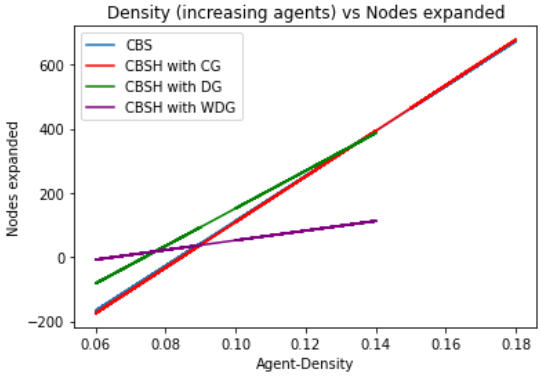


Figure 2.2: Number of nodes expanded of different heuristics for increasing levels of density (by **increasing agent count** in the instance)

The linear regression lines shown in the graphs provide a visual representation of the data. It is important to note that negative values on the lines are artifacts and can be disregarded.

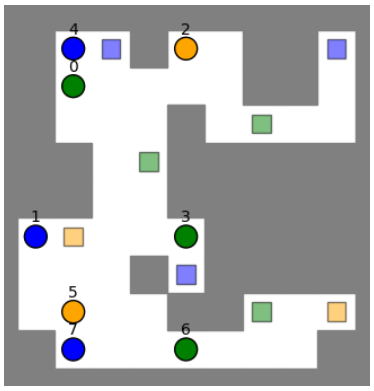
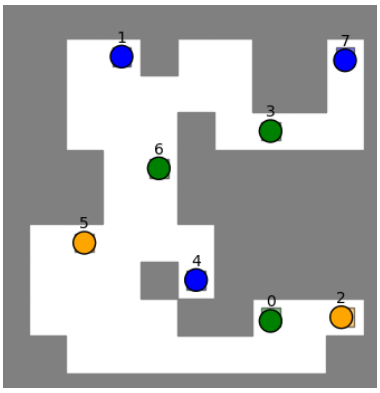
As anticipated, more accurate heuristics tend to increase runtime, while reducing the number of nodes expanded. Notably, maps with a higher number of agents prove to be significantly more resource-intensive than maps with a higher density of obstacles, as indicated by Figure 1.1 and 2.1.

For CBSH, particularly with the Weighted Distance Graph (WDG) heuristic, the reduction in the number of nodes expanded becomes more pronounced as the density increases with added obstacles. Denser maps limit agents' freedom of movement, leading to a higher likelihood of collisions and dependencies. Heuristics enable CBS to identify these collisions and choose paths with fewer of them. However, it's crucial to mention that the runtime also increases considerably with increasing density. This suggests that, at least in our implementation of WDG, there is no significant runtime advantage over Distance Graph (DG) or Cardinal Graph (CG) heuristics when the density exceeds 0.10.

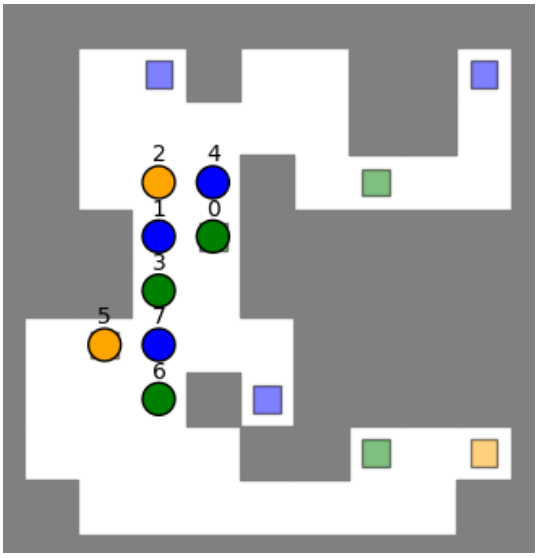
Adding more agents to instances revealed some intriguing results, particularly in high-density scenarios. In instances with a density of 0.20, we encountered challenges in consistently solving problems within the allocated 10-minute time limit. This limitation is reflected in the absence of data points for DG and WDG heuristics in Figures 1.2 and 2.2. As a result, the linear runtime line appears flatter than it would be in scenarios where all problems were solvable. This situation highlights the potential for combinatorial explosion when dealing with an increased number of agents, especially in computing the joint Multi-Value Decision Diagrams (MDD) and edge weights between new dependencies.

7. Case Study: High-Density Bottleneck Scenario

An interesting case, "more_walls_instances/high_density/instance_7," exemplifies the challenges posed by high agent density.

 Start	 Goal

In this instance, we have many agents that start from one side and their goal is on the other side. Agents that want to go from one side of the map to the other must pass the center of the map. This pathway is very narrow and only two agents can fit side by side, thus creating a bottleneck.



In this timestep, we see that all agents are gathered at the bottleneck. If the traffic was controlled properly, with agents moving from top to bottom on one side and agents moving from bottom to top on the other side, we'd have no conflicts. Unfortunately our A* does not

support this, hence we have many conflicts that are dependent on each other. In the table below we see that wdg generates and expands much less nodes. Although wdg expands less nodes than cbs, it is still very slow, this is likely because of our mdd. Creating mdd trees and joining them are very expensive operations. If we can further optimize our code, it should be much better than cbs.

Method	Nodes Generated	Nodes Expanded	Time (s)
Regular cbs	40289	25483	200.725
Cbs with Wdg	3505	2224	409.136

8. Conclusion

In the culmination of our experiments, we have discerned that the introduction of more agents significantly amplifies the complexity of Multi-Agent Path Finding problems, resulting in a more pronounced increase in solution depth. Notably, our findings underscore a critical distinction between the effects of augmenting agent count versus adding obstacles. Increasing agent count substantially exacerbates runtime, especially evident when employing the Distance Graph (DG) and Weighted Distance Graph (WDG) heuristics.

DG and WDG heuristics necessitate the calculation of MDDs for every pair of agents, and this demand becomes increasingly formidable as the number of agents escalates. The MDD construction and joining processes are computationally expensive and memory-intensive. As such, the proliferation of agents compounds the challenges associated with heuristic computation.

Our endeavors, though not culminating in runtime improvements, have achieved notable reductions in both the number of nodes generated and expanded for specific instances. Instances characterized by narrow passageways that compel numerous agents through bottleneck regions tend to engender an abundance of conflicts and dependencies. In scenarios where agents travel in opposing directions, the Weighted Distance Graph (WDG) heuristic emerges as a viable choice to diminish the generation and expansion of nodes, given the heightened interdependencies.

To enhance the outcomes of our project, we envision a future focus on the optimization of our heuristic implementation. Targeting improvements in the efficiency of MDD calculation and joining processes is imperative, as it holds the potential to significantly enhance heuristic computation times. Such optimization

endeavors are poised to extend the applicability of the Conflict-Based Search with Heuristics (CBSH) algorithm, rendering the use of expensive heuristics more viable in practice.

Our research contributes valuable insights into the realm of Multi-Agent Path Finding, shedding light on the trade-offs between problem complexity, runtime, and heuristic effectiveness. The significance of these findings extends to real-world applications, where efficient path planning is a critical consideration.

9. References

Li, J., Felner, A., Boyarski, E., Ma, H., & Koenig, S. (2019). Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence. doi:10.24963/ijcai.2019/63

Vertex cover problem: Set 1 (introduction and approximate algorithm). GeeksforGeeks. (2021, August 18). Retrieved December 21, 2021, from https://www.geeksforgeeks.org/vertex-cover-problem-set-1-introduction-approximate-algorithm-2/

Python tutorial. Python Tutorial: Graph Data Structure - 2021. (n.d.). Retrieved December 21, 2021, from https://www.bogotobogo.com/python/python_graph_data_structures.php