Comparative Analysis of Conflict-Based Search Heuristics for Multi-Agent Pathfinding

Saanvi Chugh

Received September 11, 2024
Accepted May 29, 2025
Electronic access July 15, 2025

Multi-agent pathfinding (MAPF) is the NP-Hard task of creating non-conflicting paths for a group of agents, given only start and end locations on a grid. Each agent should be able to move concurrently to their desired destination, without enduring any crashes. MAPF has a wide range of applications, including automated warehouses, robotics, and aviation. To tackle this seemingly intractable problem, numerous MAPF algorithms have been developed, though none has emerged as the best solution. Conflict-Based Search (CBS), a two-level algorithm that employs both a high-level and low-level search, stands as one of the optimal approaches. This research presents four unique heuristic implementations of CBS and evaluates their performance in terms of scalability, pathfinding speed, and efficiency. Each heuristic utilizes a distinct strategy for prioritizing conflict resolution, aiming to reduce the total number of conflicts and accelerate path generation. These heuristics were developed and tested on many randomly generated MAPF instances, with the runtime for each instance recorded and later analyzed for each heuristic. Comparison of the four heuristics based on three metrics—average runtime, success rate, and mode wins—showed that mostCrowded() was most effective in environments with high agent density, with randomConflict() coming in close second. Evaluating the strengths and limitations of each heuristic provides valuable insights into MAPF, guiding future algorithm development and helping select the most suitable approach for specific environments.

Keywords: Single-Agent Pathfinding, A*, Multi-Agent Pathfinding, Conflict-Based Search, Heuristics

Introduction

Single-agent pathfinding is the task of generating a path between two points on a grid, and is best solved by search algorithms like the A* algorithm [1]. A* computes optimal paths by performing the following calculation repeatedly: $f(n) = g(n) + h(n)$, in which $g(n)$ represents the cost, or distance, between the start state and current state (n) and $h(n)$ represents the cost from the current state to the goal state. Using this best-first search approach, A* and other similar single-agent pathfinding algorithms succeed in finding ideal paths. Multi-Agent Pathfinding (MAPF) is the NP-Hard problem of computing collision-free paths for a swarm of agents, given only their start and end locations on a map [2,3]. Following these generated paths, agents should move concurrently without producing any collisions. Given an instance, a working MAPF algorithm determines paths with the lowest cost, in which no two agents conflict at a certain vertex or on a certain edge at the same timestep [3].

MAPF has vast applications including automated warehouses [2], robotics [2,4] and aviation [5]. Many MAPF algorithms have been developed, including Branch-and-Cut-and-Price (BCP) [6], Conflict-Based Search (CBS) [1], and Lazy CBS [7], however, there is currently no dominating MAPF algorithm [3].

The algorithm of focus, CBS, is a bi-level algorithm that harnesses single-agent searches to generate a series of unique paths for all agents [1]. Each agent is initialized with default paths, in which conflicts may arise. The high-level search operates within a conflict tree (CT), where each node contains time and location constraints for all agents, as shown in Figure 1. At each node, the low-level search, typically using A*, generates single-agent paths that satisfy the constraints of that CT node. If additional conflicts are found after this search, the corresponding high-level node is marked as non-goal, and the high-level search continues by adding new CT nodes with updated constraints to resolve the new conflicts. This breadth-first search process repeats until all conflicts are resolved and the final paths are produced.

This research serves to evaluate how different heuristics for conflict prioritization affect the performance of CBS in MAPF. In this study, "heuristic" refers to one of the four unique conflict prioritization strategies integrated into the high-level search of CBS, rather than the admissible cost function traditionally used in search algorithms. To perform this research, the CBS algorithm and four heuristic methods were programmed in Python and tested on randomly generated MAPF instances with ascending agent numbers. For each number of agents, 20 trials were executed across all heuristics, with runtimes recorded for later analysis. The performance of the heuristics was compared using

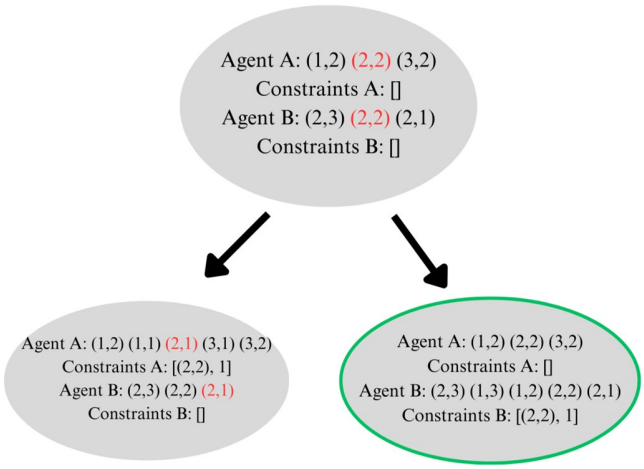

Fig. 1 This displays an example Conflict Tree (CT) tree, for 2 agents, used in CBS. The parent node is initialized with the default paths of each agent. A conflict is found at location (2,2) at time step 1; therefore, two child nodes are created to handle the conflict. In the left child node, Agent A is constrained from being at (2,2) at the time of the detected conflict. The right child node sets the same constraint for Agent B. In this particular case, no further conflicts are found in the right child node, so these final paths would be returned.

three key metrics: average runtime, success rate, and mode wins.

The results of this analysis will direct future algorithm development, help select the most suitable heuristic for specific environments, and provide insights regarding the effectiveness of CBS as a MAPF algorithm. An important limitation to note is the increasing computational expense of running experiments for MAPF instances with much greater numbers of agents. The excessive time required to run experiments and compute nonconflicting paths for large agent swarms limited the number of trials that could be conducted and prevented further experimentation involving swarms of agents beyond 25.

Literature Review

In the context of the papers discussed in this section, "heuristic" refers to a cost estimate in pathfinding, specifically the estimated cost from the current state to the goal state, rather than the strategy used for resolving conflicts.

CBS[1] arbitrarily chooses conflicts to split, and ICBS[8], Improved Conflict-Based Search, prioritizes splitting cardinal conflicts, conflicts that result in the cost of child nodes being larger than their parent node. Moreover, Felner et al. proposed ICBS-h[9], which uses admissible heuristics to better prioritize conflict resolution. At the time of their research, existing CBS variants used only the costs of the (possibly conflicting) paths in the nodes of the CT as the costs of the nodes. ICBS-h is an enhanced algorithm that calculates the admissible heuristics and adds these h-values to the costs of these nodes to better inform the prioritization of conflicts. These admissible heuristics are determined based on aggregating cardinal conflicts among agents. Experiments comparing ICBS-h to standard CBS and ICBS show that ICBS-h outperforms by up to a factor of 5 in terms of efficiency, reduces CT size, and prompts faster searches for conflict-free paths.

Later, Multi-objective CBS (MO-CBS)[10] was developed to extend CBS to simultaneously consider multiple objectives. In Multi-objective Multi-agent path finding (MO-MAPF), agents have to trade off multiple interfering objectives such as completion time, travel distance, and other domain-specific measures. MO-CBS computes the entire Pareto-optimal set of conflict-free paths with respect to multiple objectives. In MO-CBS, each node in the CT corresponds to a set of solutions, instead of a single solution like in standard CBS. When a node is expanded, the solutions are evaluated for Pareto dominance, and only the non-dominated solutions are retained, which are deemed Pareto-optimal. A variant called MO-CBS-t[10] was also developed to speed up the identification of the first solution, and this proved to be beneficial. Experiments comparing MO-CBS against the baseline multi-objective search algorithm NAMOA* (Non-dominated A*) showed MO-CBS's superior performance in scenarios with more agents and objectives, particular in terms of success rates and runtimes.

Continuous-time Conflict-Based Search (CCBS)[11] is a CBS variant that guarantees optimal solutions without discretizing time, however its scalability is limited due to the lack of improvements present in the original CBS framework. Andreychuk A. et al extended successful CBS improvements, specifically disjoint splitting (DS), prioritizing conflicts (PC), and high-level heuristics, to the continuous time domain of CCBS[12]. DS ensures that, for a CT node N, every solution that satisfies $N.constraints$ is exactly one of its children. Their adaptation of PC chooses to split a CT node on the conflict with the greatest cost impact, defined as the increase in total solution cost when the conflict is resolved. Andreychuk A. et al. also developed two admissible heuristics for CCBS. When this improved CCBS was compared against the original, it exhibited enhanced scalability and efficiency, solved problems involving almost twice as many agents as the original, and reduced runtime by up to two orders of magnitude in some cases.

While existing CBS variants like ICBS[8], ICBS-h[9], MO-CBS[10], and CCBS[11] have focused on heuristic cost functions, Pareto-optimality, and continuous time, this research uniquely evaluates different conflict selection strategies within CBS. Current CBS variants improve PC by incorporating cost-based heuristics (e.g., ICBS-h using h-costs), but they don't explore non-cost-based strategies for conflict resolution. This research fills this gap by systematically comparing different non-cost-related PC strategies, such as resolving conflicts based on earliest occurrence, agent congestion, or most involved agents.

Heuristic	Explanation
`firstConflict()`	Resolves the earliest conflict first.
`mostCrowded()`	Resolves the conflict with the most nearby agents first.
`randomConflict()`	Resolves a randomly selected conflict first.
`mostConflictingAgent()`	Resolves the earliest conflict of the agent involved in the most conflicts first.

Table 1 The table provides brief explanations of each of the four different heuristics being analyzed.

Materials and Methods

Heuristics

This experimental study presents and compares four different heuristic implementations of CBS, described above in Table 1, and discusses their strengths and weaknesses in the context of MAPF. FirstConflict() analyzes the paths at each CT node and determines which of the detected conflicts occur first. This method prioritizes resolving the conflict that occurs at the earliest timestep, based on the reasoning that addressing conflicts in chronological order may reduce the overall number of conflicts and result in quicker generation of non-conflicting paths. FirstConflict() was chosen as a baseline for comparison, representing the default, untouched CBS method for conflict resolution that lacks any prioritization strategy. In contrast, mostCrowded(), declares certain conflicts more dangerous than others, based on how many agents surround the vertex or edge at which the conflict occurs, at that specific time. The more agents found within one grid space up, down, left, or right of the conflict, the larger the crowd and the more high-priority the conflict. MostCrowded() chooses to address the most potentially problematic conflict first, to mitigate future issues with surrounding agents and enhance overall pathfinding efficiency. RandomConflict() serves as a control variable to assess whether more complex heuristics truly offer an advantage or if random selection performs equally as well. In the high-level search, this method identifies all conflicts between the paths at any given CT node and randomly selects one to address first. MostConflictingAgent() selects a conflict to resolve first by identifying which agent is most frequently involved in conflicts. This method begins by finding all conflicts at a CT node and tracking the agents directly involved in them. Based on this assessment, the program is able to determine the most conflicting agent and handle the earliest of that agent's conflicts first. MostConflictingAgent() was created to test whether resolving conflicts involving the most entangled agent first can reduce overall conflicts later. The code developed for the CBS algorithm and each heuristic can be found at the GitHub link, provided at the footnote towards the end of this paper. In this study, the independent variable is one of four different heuristics integrated into the high-level search of CBS, providing a comparative analysis of these unique MAPF solutions and deeper insight into the effectiveness of CBS.

Experimentation Methodology & Data Analysis

The original CBS algorithm, each heuristic method, and the experimentation code were all programmed in Python. For this research, randomly generated MAPF instances on a grid size of 10 x 10 were tested with ascending numbers of agents, ranging from 2 to 25, across all four heuristics. For each different heuristic, 20 identical instances, or trials, are executed for each agent count. For example, when testing MAPF performance for 2 agents, each CBS variant is run with the same 2 start and 2 goal locations, to determine which heuristic can complete the same task in a shorter time. However, running just 1 trial, or 1 set of start and goal locations, would not produce reliable results. Therefore, 20 trials were executed for each increasing number of agents, across all four heuristics, and all runtimes were recorded using the Python time module and used for later analysis. As the number of agents increases, the runtime to complete a MAPF instance grows exponentially. For this reason, parallel processing was employed, and a timeout was integrated that stops any heuristic from continuing path generation after its runtime has exceeded 90 seconds. For timed-out trials, a runtime of 90 seconds is used when calculating average runtime.

The collected runtimes were used to generate three figures, analyzing the performance of each heuristic across the following measures: average runtime, success rate, and mode wins. Average runtime is a widely used metric in existing literature, including works by Felner et al.[9], Ren et al.[10], and Andreychuk et al.[12] Success rate measures how effective each heuristic is at successfully solving MAPF instances within a time limit. Notably, Felner et al.[9] and Ren et al.[10] both use a success rate with a 5-minute time limit, while Andreychuk et al.[12] employ a 30-second limit. Mode wins is included to assess the consistency and frequency of top performance. While this metric is less commonly seen in MAPF papers, it provides a count-based measure of how often a heuristic dominates its competitors. Figure 4 shows the average runtime for each agent count, computed by averaging the durations from 20 trials, and plotted using the Python library matplotlib. Figure 5 compares heuristics by success rate, defined as the fraction of completed MAPF instances out of 20, with an instance considered incomplete if paths could

not be generated within 90 seconds. Figure 6 highlights the mode heuristic winner for each agent count, where a heuristic is deemed to win if it completes a trial the fastest.

Visualization

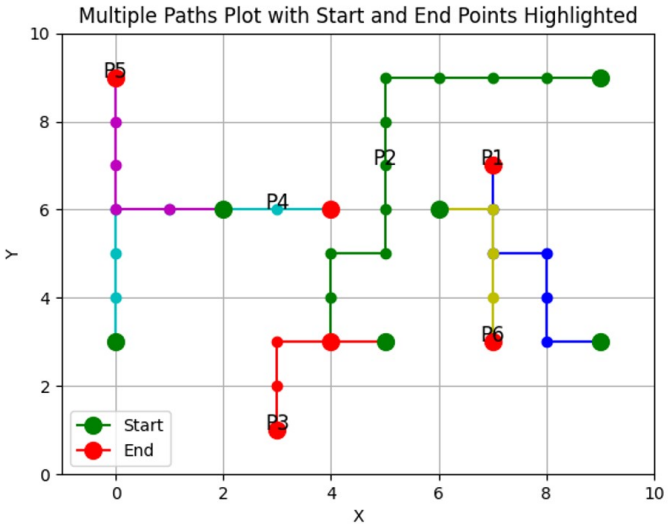

Fig. 2 Visualization tool developed using Python library matplotlib to view completed paths on a grid. Each colored line represents a different agent.

To visualize the final nonconflicting paths, each agent's trajectory was plotted on a 2D grid. Figure 2 illustrates all generated paths for a randomly generated MAPF instance involving 6 agents. This illustrates the positions of all agents on the grid, although it does not indicate the specific timestep at which each agent occupies their respective locations. To further ensure the accuracy of the CBS algorithm, computed paths were flown on Crazyflie drones in the Automatic Coordination of Teams (ACT) Lab at Brown University, as shown in Figure 3. These visualization methods confirmed the correctness of the default CBS algorithm, which served as a foundation for developing and integrating the four unique PC strategies. These were useful in initially programming, debugging, and testing CBS.

Results

Figure 4 displays the results of 20 trials in ascending order of agents, 2 to 25, and compares computational performance and scalability between heuristic methods. The shaded regions show 95% confidence intervals for each average runtime. It is observed that up to 5 agents, all heuristics perform at relatively similar average runtimes, with low variability, and indicate roughly equal performance. However, as the number of agents grows, differences between the various heuristics

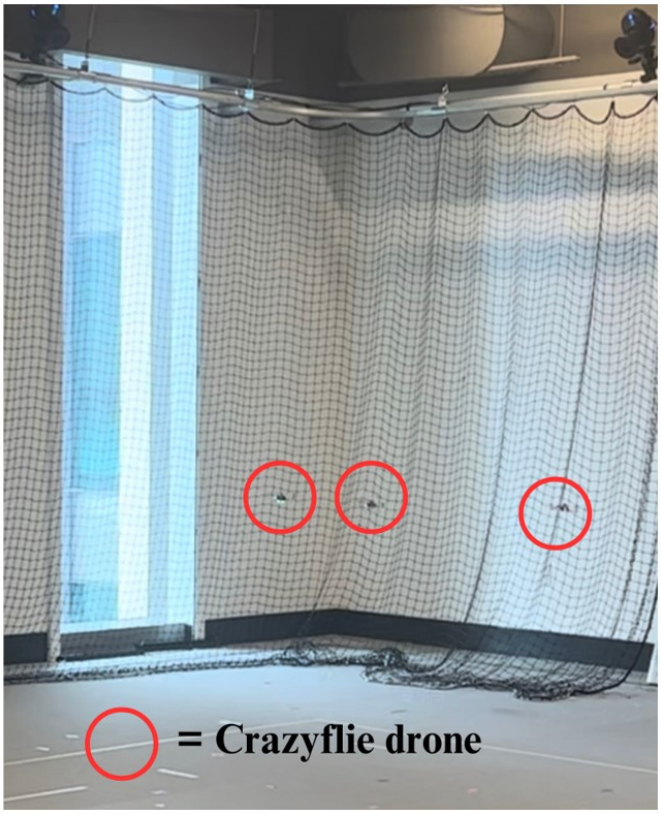

Fig. 3 Photograph taken of a tested flight on 3 Crazyflie ® drones in the ACT Lab.

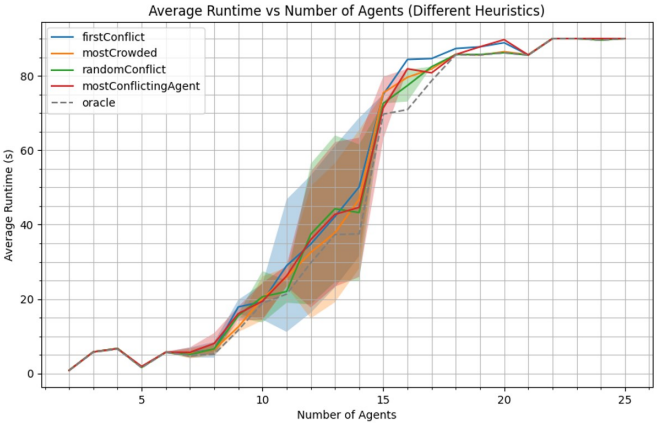

Fig. 4 Compares average runtime of 20 trials against agent count.

begin to emerge. MostCrowded() ranked as either the highest or second-highest performing heuristic for nearly all agent counts between 8 and 20, specifically at 8, 9, 10, 11, 12, 13, 16, 17, 18, and 20 agents. From 6-16 agents, all 4 heuristics exhibit increased variability, however, the confidence intervals for mostCrowded() remain consistently either the smallest or the

second smallest in this range. Another feature to note is the high performance of randomConflict(). At certain agent counts, randomConflict() was identified as either the fastest heuristic or tied with mostCrowded(), specifically at 8, 11, 14, 16, 18, 19, and 20 agents. However, within the 6-16 agent range, randomConflict() demonstrates the most significant variability, with the widest confidence intervals and greatest standard deviation occurring at 6, 10, 12, 13, and 16 agents. In Figure 4, it is also clear that beyond 20 agents, all four heuristics have nearly identical average runtimes. This is due to increased congestion, causing more instances to reach the 90-second timeout and leading to a leveling out of runtimes. This is further corroborated by the confidence interval graph, which shows shrinking confidence intervals beyond 20 agents, confirming that the solution times are increasingly determined by the timeout threshold rather than heuristic efficiency.

The dashed oracle line represents the best-case scenario or average minimum runtimes. It is determined by first identifying the minimum runtime in each of the 20 trials, regardless of whichever heuristic this minimum came from. The mean of these 20 minimum runtimes is plotted. This process is repeated for each number of agents. The oracle line signifies time taken for the most ideal path generation technique, combining the four heuristics. The strength of a certain heuristic can be assessed based on how much its corresponding line deviates from the oracle line.

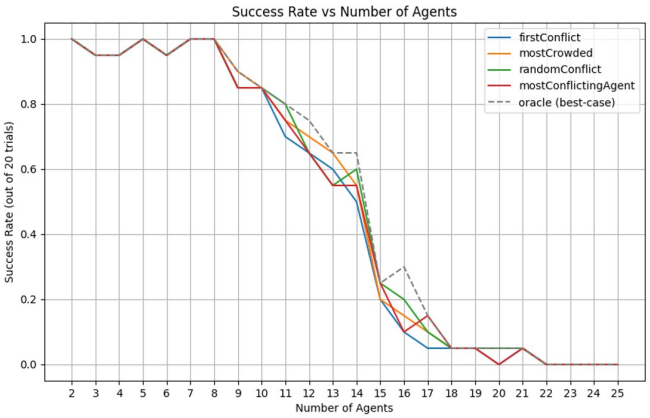

Fig. 5 Compares success rates over 20 trials against agent count.

Figure 5 shows the proportion of MAPF instances each heuristic can successfully complete in under 90 seconds. When the CBS algorithm, with an integrated heuristic, is unable to generate paths within this timeframe, the task is considered incomplete. Success rate is the proportion of completed MAPF instances out of the 20 instances per agent count. At 8 or fewer agents, all heuristics had nearly identical success rates, successfully generating all or nearly all non-conflicting paths due to the lack of congestion present. As agents are added, Figure 5 reveals that mostCrowded() provides more promising success rates. At 9 or more agents, mostCrowded() is consistently within the highest two success rates, specifically at 9, 10, 11, 12, 13, 14, 16, and 17 agents. This superior performance is highlighted between 8 and 10 agents, where mostCrowded() has equal success rates to the oracle line, indicating a high proportion of completed instances there. It is also important to note the success of randomConflict(). RandomConflict() yielded a success rate within the top two at 10, 11, 12, 14, 15, 16, and 17 agents. Beyond 18 agents, all four heuristics had nearly identical success rates, except randomConflict, which had a higher success rate specifically at 20 agents. As shown in Figure 5, none of the heuristics can generate non-conflicting paths within 90 seconds beyond 22 agents, leading to success rates of 0 at 22, 23, 24, and 25 agents.

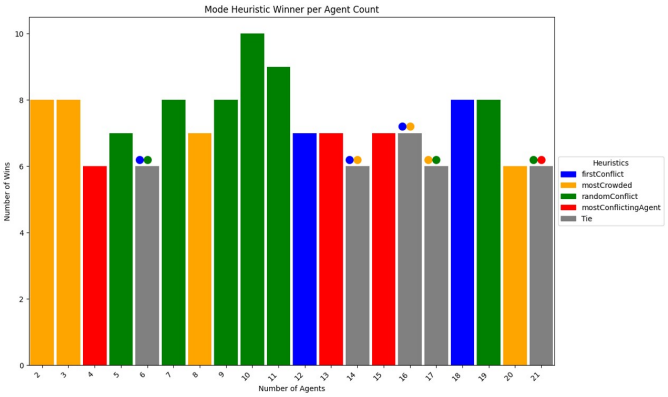

Fig. 6 Shows the number of wins by the mode-winning heuristic over 20 trials against agent count. The gray bars indicate ties and the colored dots above the bars indicate which heuristics tied for most wins.

Figure 6 illustrates which heuristic achieved the most wins and how many wins it obtained, indicating which heuristic was most often the fastest across 20 trials for each agent count. A heuristic is credited with a ”win” when it has a faster runtime than the other three heuristics in a given trial. For example, for swarms of 2 agents, mostCrowded() was the fastest heuristic in 8 out of the 20 trials. All other heuristics had fewer than 8 wins out of 20 trials, making mostCrowded() the mode winner. The x-axis was truncated at 21 agents because beyond this point, the runtimes of all four heuristics converge due to the 90-second timeout. MostCrowded() performed strongly, achieving the most wins or tying for the most wins at agent counts 2, 3, 8, 14, 16, 17, and 20. Additionally, randomConflict() frequently achieved or tied for the highest number of wins at agent counts 5, 6, 9, 10, 11, 17, 19, and 21.

Discussion

In environments with low agent density, particularly fewer than 8 agents, the choice of heuristic has little impact, as all heuris-

tics perform almost identically. In environments with medium to high agent density, mostCrowded() consistently computes nonconflicting paths in shorter average runtimes, particularly for instances involving between 8 and 20 agents. This is supported by Figure 4, in which mostCrowded() is ranked as either the fastest or second fastest heuristic at 10 agent counts within this range. Between 8 and 20 agents, a close competitor, randomConflict(), ranked within the top two fastest heuristics at only 7 agent counts. Additionally, while randomConflict also generated paths in shorter amounts of time than firstConflict() and mostConflictingAgent(), this heuristic displayed greater variability and wider confidence intervals than mostCrowded(). This suggests that while it can be among the fastest heuristics, it is more inconsistent than mostCrowded(), sometimes performing well and other times struggling.

Moreover, the mostCrowded() heuristic consistently ranked among the top two in success rates for nearly every agent count (8 agent counts) between 9 and 17. RandomConflict also achieved high success rates, though it ranked among the top two in success rate at only 7 agent counts within this range, compared to mostCrowded's 8.

In Figure 6, it is shown that mostCrowded() achieved the most wins or tied for most wins at 7 agent counts between 2 and 21. This reveals that the strategy of prioritizing densely populated conflicts proved beneficial primarily in situations involving a larger number of agents, but it is shown to be less critical in scenarios with fewer agents. However, randomConflict() achieved the most wins or tied for most wins at 8 agent counts, slightly more than mostCrowded().

The relatively strong performance of mostCrowded() in medium-high density environments may be attributed to its ability to target and resolve conflicts in the most congested areas of the map. This likely reduces the chance of new conflicts emerging later in the search, making the overall process more efficient. In contrast, randomConflict() does not account for congestion or conflict impact, which may explain its wider variability. In some cases, the randomly selected conflict happens to be important, while in others, it delays the resolution of more critical bottlenecks.

At very high agent counts, the performance of all heuristics converges due to widespread congestion and an increased likelihood of reaching the timeout limit. While the heuristic may still influence performance, the timeout prevents those differences from becoming apparent, making the strategies appear similarly effective under high congestion.

Additionally, it is important to discuss that the differences in average runtimes between the heuristics were minor. The premise of mostCrowded() led it to be believed that it would outperform all others in scenarios with high agent densities. While the data supports this line of reasoning, the improved computational performance of mostCrowded() is within the order of a few seconds of the other heuristics.

The randomConflict() heuristic serves as a control variable in this experiment, helping to determine whether the strategy for conflict resolution within a heuristic directly influences performance, or if the choice of heuristic itself has little impact. RandomConflict() generally yielded shorter average runtimes than firstConflict() and mostConflictingAgent(), achieved high success rates, and frequently recorded the most wins. This demonstrates that randomly shuffling detected conflicts at a CT node and selecting one can sometimes lead to fewer overall conflicts, indicating that the conflict resolution strategy may be less critical than expected, given that the random approach provided results comparable to the succeeding mostCrowded() heuristic.

Conclusion

This study analyzed the impact of different heuristics on CBS performance across varying agent densities. In mid-high density environments, mostCrowded() consistently generated nonconflicting paths more efficiently, achieving shorter average runtimes and high success rates. However, its advantage was marginal, typically within a few seconds of competing heuristics. RandomConflict() also demonstrated strong performance, though with much greater variability. Its success suggests that the choice of conflict resolution strategy has little impact on algorithm efficiency.

An important limitation to note was the high computational cost of running experiments with larger agent swarms. To validate these findings, it is recommended that additional trials with even higher agent counts and extended timeouts be executed. More importantly, these results indicate that non-cost-based conflict selection strategies contribute little to CBS efficiency, suggesting that future algorithm development should focus on refining cost-based heuristics rather than optimizing purely prioritization-based conflict resolution strategies.

Acknowledgments

I would like to express my deepest gratitude to Assistant Professor Eric Ewing for his guidance and mentorship throughout the course of this research. I am also immensely thankful to the Automatic Coordination of Teams (ACT) Lab at Brown University for providing me with excellent facilities that enabled the success of this project.

References

1 G. Sharon, R. Stern, A. Felner and N. Sturtevant, *Conflict-based search for optimal multi-agent pathfinding*.

2 R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar, E. Boyarski and R. Bartak, *Multi-agent pathfinding: definitions, variants, and benchmarks*.

3 J. Ren, V. Sathiyanarayanan, E. Ewing, B. Senbaslar and N. Ayanian, *MAP-FAST: A deep algorithm selector for multi-agent path finding using shortest path embeddings*.

4 M. Bennewitz, W. Burgard and S. Thrun, *Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots*.

5 L. Pallottino, V. Scordio, A. Bicchi and E. Frazzoli, *Decentralized cooperative policy for conflict resolution in multivehicle systems*.

6 E. Lam, P. Bodic, D. Harabor and P. Stuckey, *Branch-and-cut-and-price for multi-agent pathfinding*.

7 G. Gange, D. Harabor and P. Stuckey, *Lazy CBS: Implicit conflict-based search using lazy clause generation*.

8 E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel and E. Shimony, *ICBS: improved conflict-based search algorithm for multi-agent pathfinding*.

9 A. Felner, J. Li, E. Boyarski, H. Ma, L. Cohen, T. Kumar and S. Koenig, *Adding heuristics to conflict-based search for multi-agent path finding*.

10 Z. Ren, S. Rathinam and H. Choset, *Multi-objective conflict-based search for multi-agent path finding*.

11 A. Andreychuk, K. Yakovlev, D. Atzmon and R. Stern, *Multi-agent pathfinding with continuous time*.

12 A. Andreychuk, K. Yakovlev, E. Boyarski and R. Stern, *Improving continuous-time conflict based search*.