Proceedings of the 5th International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/134/2025.22268

Advancing Multi-Agent Pathfinding in Gaming: A Review of Cooperative, Dynamic, and Real-Time Algorithmic Optimizations

Yujie Jin[1*], Yuhan Chen[2], Chris Shen[3]

[1]*University Of Leeds, Leeds, UK*
[2]*Nanjing University, Nanjing, China*
[3]*Dongsheng 1st High School, Erdos, China*
**Corresponding Author. Email: mplau001@email.phoenix.edu*

Abstract: Multi-agent pathfinding (MAPF) is a problem focused on coordinating multiple agents to navigate from starting positions to goals in shared environments while avoiding collisions. This capability is important for applications in areas such as computer gaming and robotics, where efficient and safe navigation in complex environments is required. Although advancements have been made, challenges remain in areas such as multi-agents cooperation, MAPF under dynamic environments, and real-time MAPF. This paper reviews three advanced MAPF algorithms: Cooperative Conflict-Based Search (Co-CBS), which extends the traditional CBS algorithm by introducing cooperative planning; Dynamic Incremental Conflict-Based Search (DI-CBS), which adapts to environmental changes through integration with ECT and SLPA* algorithms; and Bounded Multi-Agent A* (BMAA*), which enables agents to independently plan paths using real-time heuristic search without explicit coordination. Experimental results highlight the distinct characteristics and advantages of each algorithm across different scenarios.

Keywords: Path Planning, Conflict Search, Dynamic Environment, Cooperative Algorithm, Real-Time Search

1. Introduction

Pathfinding algorithms have always been a significant area of research within computer science and artificial intelligence. In gaming, these algorithms encounter uniquely complex challenges not typically found in more static, primarily due to the unpredictable nature of game environments. For example, some algorithm frameworks cannot solve the cooperative tasks of agents, which means all NPCs cannot interact and coordinate with each other from the starting position to the target position during the task. Secondly, the movement of character positions and task updates may require NPCs to move in dynamic environments. Lastly, some non-real-time pathfinding algorithms may result in multiple NPCs crowding and blocking the roads, even leading to deadlocks. That's why Multi-Agent Pathfinding (MAPF) is a key area, involving multiple agents planning paths in a shared environment

DOI: 10.54254/2755-2721/134/2025.22268

and avoiding collisions. Typical algorithms, such as Conflict-Based Search (CBS) [1], Windowed Hierarchical Cooperative A* (WHCA*) [2] and Flow Annotated Replanning (FAR) [3], often suffer from high search costs and are unable to adapt to more complex situations. Therefore, we review improved algorithms such as Cooperative Conflict-Based Search (Co-CBS), Dynamic Incremental Conflict-Based Search (DI-CBS) and Bounded Multi-Agent A* (BMAA*).

The first paper introduces Cooperative Conflict-Based Search (Co-CBS), a novel algorithm designed for Cooperative Multi-Agent Path Finding (Co-MAPF), enhancing the classical Conflict-Based Search (CBS) with elements of cooperative planning. In the context of gaming, one example could involve several agents that must collaboratively navigate a complex environment to complete joint missions, such as moving large obstacles that cannot be handled by a single agent. This scenario underscores the need for cooperative behavior to achieve shared objectives in environments that change dynamically. The Co-CBS algorithm introduces a new search level on top of the classical CBS algorithm, managing to handle the cooperation between agents. It then presents improved versions for efficiency, and demonstrates its effectiveness through experiments. The paper also discusses future directions for Co-CBS and the Co-MAPF framework.

The second paper uses a new algorithm, called DI-CBS, to solve Incremental-MAPF questions. In the game, the environment will change, which requires the agent to refer to the changes in environmental factors to adjust the route when searching for a path. When there are a large number of intelligent agents, the routing algorithm still needs to ensure efficiency. DI-CBS effectively solves these two problems. This algorithm uses ECT for high-level scheduling and SLPA* for low-level computation. This reduces unnecessary calculations, saves memory usage, and improves efficiency.

Finally, real-time algorithms can generate the next move for all agents without the need to find complete paths for them. BMAA* achieves this by allowing each agent to conduct a separate real-time heuristic search and treating other agents as moving obstacles, eliminating the need for coordination and segment sharing between agents. This approach improves completion rates and reduces search time and cost.

2. Background

MAPF is a significant problem in the field of pathfinding. Numerous studies have been carried out to address it. We first provide a definition of the classic Multi-Agent Path Finding (MAPF) problem. We then discuss a classic algorithm developed to address the MAPF problem, followed by several advancements relevant to the current study.

The Multi-Agent Path Finding (MAPF) [4] problem is a critical multi-agent planning challenge which aim at planning paths for multiple agents to move concurrently and reach their respective goals without collisions. The problem input includes an undirected graph G of the environment, the starting points s for the agents, and the target points t. Agents move in discrete time steps, where each agent can choose to stay in place or move to an adjacent vertex at each time step. A solution provides k path plans of action sequences, one for each single agent.

The Conflict-Based Search (CBS) algorithm [1] is an optimal search method for the classical Multi-Agent Path Finding (MAPF) problem. It has a two-level search structure: at the high level, CBS constructs a Conflict Tree (CT) to identify and resolve agent conflicts, with each node recording a set of constrains. at the low level, the algorithm independently searches for optimal paths for each agent that satisfy specific constraints in a node. Each time a conflict in paths is found at the low level, constraints will be added to avoid that conflict, generating sub-nodes with more constraints. Once optimal paths are generated for all agents without any conflicts, these paths collectively form a valid solution to the problem.

DOI: 10.54254/2755-2721/134/2025.22268

Flow Annotated Replanning (FAR) [3] is a path planning algorithm that aims to react to environmental changes in real-time by annotating and modeling the environment. To maintain scalability while keeping CPU and memory requirements low, FAR begins with a preprocessing step that abstracts a grid map into a flow-annotated search graph. This structure is enhanced with flow constraints to avoid head-on collisions and reduce the search's branching factor. Next, a full A* search is performed independently for each agent, calculating a path that disregards other agents on the map. Plan execution starts as soon as a path is computed for each agent. A strategy based on waiting and reservation is employed to mitigate potential deadlocks, which occur when two or more agents cyclically wait for each other, a situation that is not unique to FAR but can arise in any multi-agent system. Examples of deadlocks include agents facing each other in narrow corridors or at intersections without adequate passing room. If deadlocks cannot be avoided, FAR employs a deadlock-breaking procedure to locally adjust plans rather than undertaking a comprehensive replanning step.

3. Cooperative Multi-Agent Path Finding: Beyond Path Planning and Collision Avoidance

In [5] , the paper focuses on the Cooperative Multi-Agent Path Finding (Co-MAPF) problem, which extends the classical MAPF problem by adding cooperation to agents' tasks. To handle this problem, the paper introduces the Cooperative Conflict-Based Search (Co-CBS) algorithm. This algorithm adds a cooperative searching level on top of the classical CBS algorithm. The authors also introduce two advanced Co-CBS algorithms that make computation faster. Experiments are designed and conducted to demonstrate basic and improved Co-CBS's efficiency in solving Co-MAPF instances.

3.1. Problem Defination of Co-MAPF

The Cooperative Multi-Agent Path Finding (Co-MAPF) problem involves finding paths for multiple agents while agents should cooperate to complete tasks in a shared environment while avoiding collisions.

To formally define the Co-MAPF problem, the paper employs a structured framework. The paper refines the cooperative task to a specific problem where two agents work together to accomplish a pathfinding task. Agents are paired as initiators and executors, as (α_i, β_i). A task is assigned to each pair of agents, requiring the initiator to move to the task's start location s_i, both agents to meet at a calculated meeting point, and the executor to move to the task's goal location g_i. When a pair of agents reaches their meeting point, they occupy the same location, but this is not considered a collision. Fig. 1 is an example of four agents with two tasks. The solution to Co-MAPF includes pairs of paths for cooperating agent pairs, one for each pair of agents, to complete the cooperative tasks without collisions. The paper uses the Sum of Costs (SOC) as its objective function, defined as the total number of time steps required for each agent to complete their tasks.

DOI: 10.54254/2755-2721/134/2025.22268

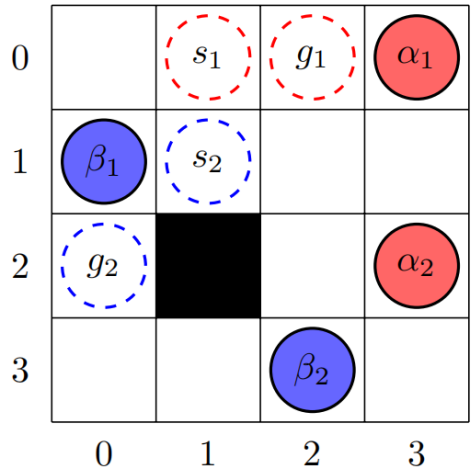


Figure 1: An example instance with two tasks: agents α_1 and β_1 execute task 1 from s_1 to g_1, and agents α_2 and β_2 execute task 2 from s_2 to g_2 [5].

3.2. Co-CBS Algorithm

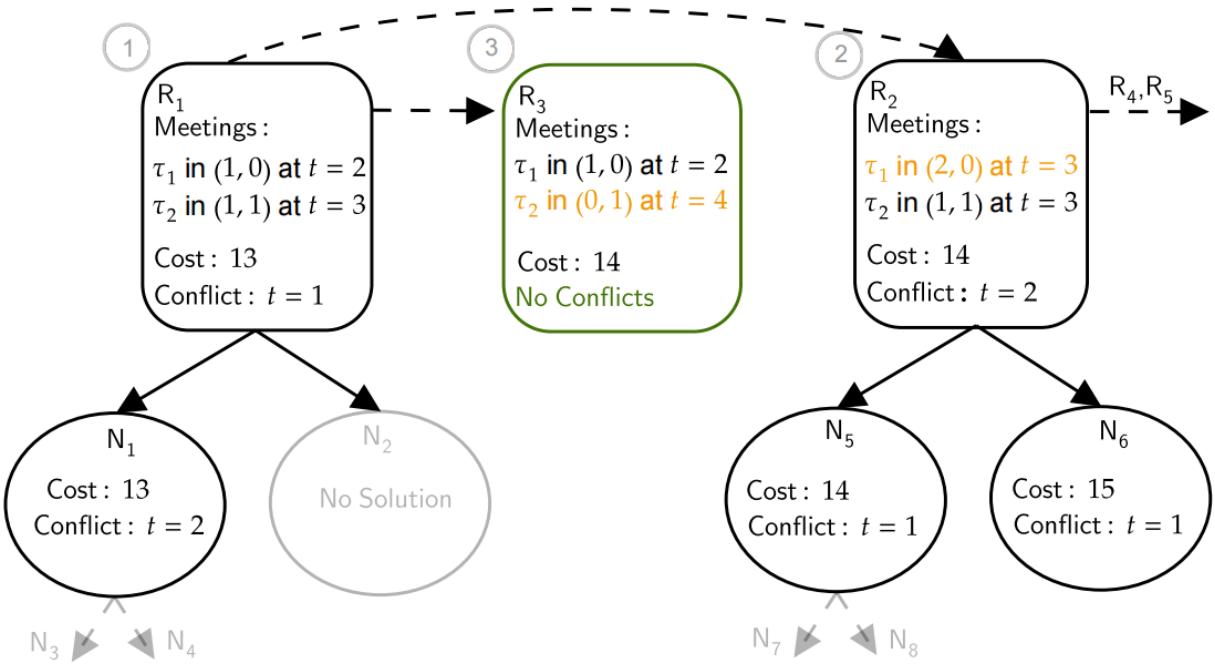
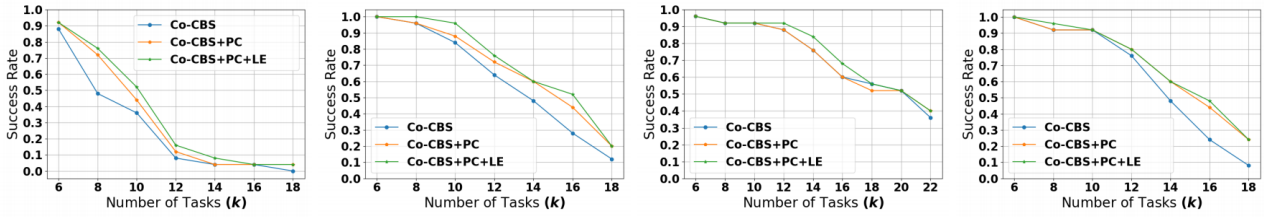


Figure 2: An example of Co-CBS search forest [5].

The paper designed Cooperative Conflict-Based Search(Co-CBS) algorithm to address Co-MAPF problem. Co-CBS is a three-level search algorithm, comprising the meetings level, the conflicts level, and the paths level. At the meetings level, it searches over all situations of meeting arrangements for the agent pairs. At the conflicts level, Co-CBS detects and resolves possible conflicts among agents.

DOI: 10.54254/2755-2721/134/2025.22268



(a) den312d (b) random-32-32-20 (c) warehouse-10-20-10-2-1 (d) warehouse-57-27

Figure 3: Success rates of Co-CBS and Co-CBS improved by PC and LE in four different maps [5].

Lastly, at the paths level, the algorithm generates specific paths for the agents, satisfying constraints imposed by higher levels. A meeting table is used to record a set of certain meetings and their cost, which varies with meeting times and locations.

The algorithm begins with constructing a initial meeting table with meetings of the minimal cost, the total number time steps required to meet, for each pair of agents, and generating a root node containing the initial meeting table. Subsequently, the algorithm loops to expand nodes, and conducts its three-level search, Fig. 2 as an example of Co-CBS process. Each time step, the algorithm selects the cost-least node to expand. When the algorithm encounters a ”root node”, it signifies a point where a decision is made regarding the initial meeting plans for each agent pair. At this juncture, the algorithm generates a new root node for each pair, substituting the existing meeting plan with the subsequent most favorable meeting from the meeting table. This method allows Co-CBS to comprehensively explore all potential optimal combinations of meeting plans across the network of agents. As the search progresses to the two lower levels, the algorithm actively monitors for any collisions among the agents. Upon detecting a collision, Co-CBS intervenes by dividing the affected constraint tree node into two child nodes. This split is designed to introduce new constraints that effectively prevent the recurrence of the same collision in future paths, while maintaining the originally planned meetings from the root node intact. The process of expanding nodes involves examining each node's associated costs and constraints, adjusting the search path to navigate around identified conflicts through the newly created child nodes. This iterative expansion and adjustment continue until Co-CBS identifies the first valid solution that meets all constraints without any collisions, deeming it the optimal solution for the scenario.

3.3. Advanced Co-CBS

The article introduces two improvements to the basic Co-CBS. The first is Prioritizing Conflicts(PC) [6], a significant improvement for classical CBS. In PC, conflicts are classified into different types, and conflicts that are more likely to lead to optimal solutions are given priority for expansion. Due to the meeting requirements of Co-CBS, implementing PC in Co-CBS requires modifying the construction of the Multi-Value Decision Diagram (MDD), which is done by eliminating invalid nodes through breadth-first searches.

Additionally, this paper introduces a unique improvement for Co-CBS called Lazy Expansion (LE), which utilizes the special characteristics of root nodes. Applying LE, the cost of root nodes is precomputed and recorded. Co-CBS then creates root nodes without immediately calculating their low-level searches, only recording the cost for expansion. Low-level paths are only computed when nodes are chosen as the current least-cost node to be expanded. Since most generated nodes are not expanded, LE may save significant runtime while maintaining optimality.

DOI: 10.54254/2755-2721/134/2025.22268

3.4. Experiment

To evaluate the performance of the Co-CBS algorithm in addressing the Co-MAPF problem, the authors conducted a series of experiments. These experiments were performed on four different 2D grid maps, including dense game maps, random maps, and two warehouses of different sizes. Each map was assessed with varying numbers of tasks from 6 to 22, within a two-minute timeout constraint.

The experimental results, illustrated in Fig. 3, show that the basic Co-CBS algorithm achieves a high success rate in scenarios with a smaller number of tasks. However, as tasks increase on denser maps, the success rate of basic Co-CBS decreases significantly due to its limitations in handling complex conflicts and meeting arrangements. In contrast, the enhanced algorithms, Prioritizing Conflicts (PC) and PC combined with Lazy Expansion (PC+LE), demonstrate improved performance, especially in environments with a large number of tasks. These improvements suggest that PC and PC+LE better manage the additional complexity introduced by increased task numbers. Results from Fig. 4 highlight that in sparser warehouse environments, Co-CBS quickly finds feasible solutions using the initial meeting plans. However, in smaller and denser settings, a more exhaustive search is necessary to discover the optimal solution, increasing computational demands. This exhaustive search indicates the trade-off between solution quality and computational cost. While PC and PC+LE improve success rates, they also introduce higher computational loads. Specifically, PC and PC+LE require more processing to prioritize conflicts and manage expanded search trees effectively, which can lead to increased memory usage and processing time. These factors are critical considerations, as the improved outcomes come at the cost of higher resource utilization and potentially slower performance under high-demand conditions.

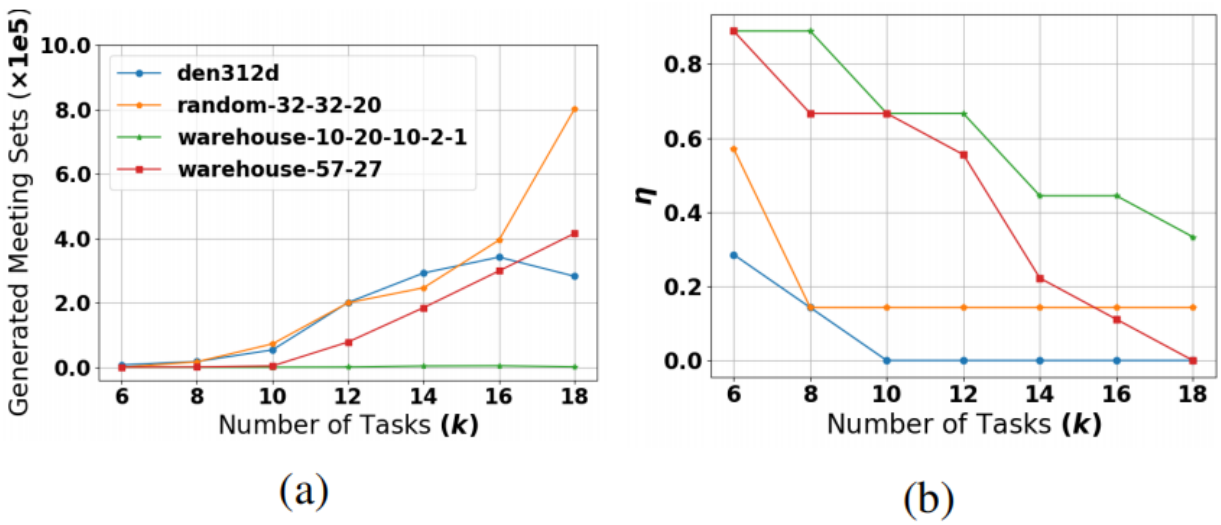


Figure 4: (a) Number of generated sets of meetings. (b) Ratio η between the number of instances solved using the first set of meetings, and the total number of instances [5].

3.5. Conclusion

The paper introduces the Cooperative Conflict-Based Search (Co-CBS) algorithm, addressing the Cooperative Multi-Agent Path Finding (Co-MAPF) problem by which extends classical CBS with

DOI: 10.54254/2755-2721/134/2025.22268

a cooperative layer to handle agents' cooperative actions. Two advanced improvements, Prioritizing Conflicts (PC) and Lazy Expansion (LE), enhances the algorithm's scalability and performance. Experimental results demonstrate that the basic CBS successfully completed the Co-MAPF tasks, while the improved algorithms performed better in complex environments and under higher task loads.

The paper also explores future developments for Co-CBS and the Co-MAPF framework. Reusing information between constraint trees (CTs) can reduce computation time, and using efficient meeting-level algorithms, like CF-MM*, offers further improvements. Advanced CBS techniques, such as heuristic search, conflict splitting, and symmetry breaking, may also enhance Co-CBS. Adapting the meeting-level search to support adjacent-location meetings improves real-world applicability. Expanding the framework to handle more agents, introduce temporal constraints, and incorporate lifelong planning could further extend its practicality for complex, real-world scenarios.

4. Incremental Conflict-Based Search for Multi-agent Path Finding in Dynamic Environment

This article introduces an algorithm based on incremental CBS to solve the I-MAPF problem. The article uses a binary environment conflict tree for high-level scheduling and SLPA* for low-level scheduling, which improves efficiency by inheriting the global scheme and only changing the routes around obstacles. The article compares the method proposed in the article with CBS planner through experimental design, demonstrating that their approach can maintain high efficiency in scenarios with frequent environmental changes.

4.1. I-MAPF Problem Definition

Incremental Multi-Agent Pathfinding (I-MAPF) is the full name for this approach, which involves agents continuously adjusting their paths in response to changes in their environment to avoid collisions and other issues. This flexible approach is necessary because static pathfinding methods often fail in dynamic scenarios where the environment can unpredictably change. Among the various solutions addressing this problem, conflict search-based solutions are particularly notable. This method stands out because it can dynamically adapt to changes without the need for restarting the path search, which makes it more efficient in environments where changes occur frequently. Several algorithms have been developed to enhance the capabilities of Conflict-Based Search (CBS). For example, the CBS-planner and CBS-D*-lite schemes, which are designed to manage dynamic environments by reconstructing conflict trees[7], allow for more granular control over the pathfinding process, thereby improving the resolution of conflicts and the efficiency of pathfinding.

The article provides an image to better illustrate the I-MAPF problem. As shown in Fig. 5, the map consists of undirected and unweighted squares. Solid circles represent the initial positions of agents, while hollow circles represent their goals. The dashed line illustrates the path of an intelligent agent under global planning. Blue squares indicate newly emerging obstacles, and red squares mark the locations where collisions have occurred, affecting the agents' paths.

4.2. Method Description

In high-level scheduling, the article adopts a binary environment conflict tree to solve the problem. Each node on this tree stores constraints, solutions, and costs. The expansion scheme of the node is shown in the Fig. 5. When a new obstacle appears or a collision point is estimated, the node will record the position of the obstacle and let the agent avoid this position. At the same time, perform low-level scheduling and record the resulting solutions and their costs. This process will continue until the lowest cost solution is found.

DOI: 10.54254/2755-2721/134/2025.22268

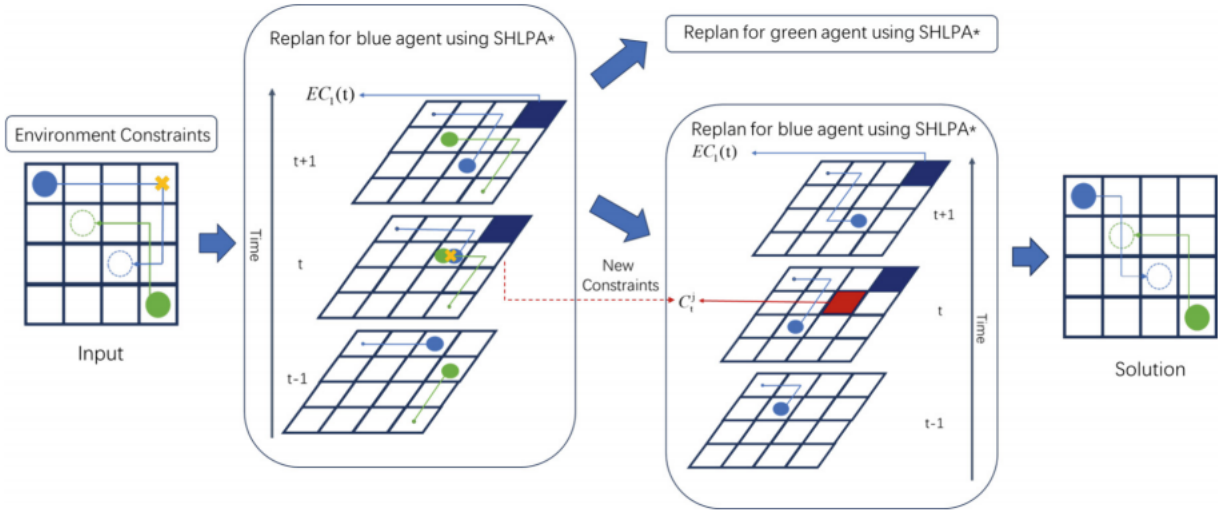


Figure 5: Node Expansion Process Diagram [8]

At the lower level of scheduling, the author used a new algorithm based on LPA* improvement - SLPA*. This algorithm uses a three-dimensional data (a_i, v_i, t) to represent the arrival of a_i at vertex v_i at time t. If the agent moves to an adjacent vertex in the next time step, its state becomes (a_i, v_j, $t+1$); if it waits at the current vertex, its state becomes (a_i, v_i, $t+1$). Whenever a node is expanded, SLPA* will simulate the actions of the agent: up, down, left, right, and stationary, and check for collisions between the agent and the environment and other agents. If a collision occurs, the point will be expanded as a child node, and the starting point of SLPA* simulation will backtrack n steps. (n is an adjustable parameter, usually proportional to the distance between the current position of the agent and the collision point.) If the new collision point is located after the new starting point, inherit the path cost of the new starting point, and then perform incremental retrieval.

The article also compared their methods with CBS planner. The main differences between DI-CBS and CBS planner are as follows:

1. DI-CBS inherits the global scheme and starts calculating based on it. The CBS planner directly initializes the conflict tree.

2. DI-CBS ensures the continuity of paths and vertices during Environmental Conflict Tree (ECT) construction by outputting a complete path containing all agents before resolving all current conflicts and any new environmental changes occur, combining the new solution with the traversed path.

3. Upon the occurrence of environmental changes, DI-CBS updates the ECT by modifying the timestamp of each node, represented as EC_i, where i denotes the index of the node. EC_i is then re-initialized as the node for the next set of constraints to be expanded. The conflict C_i, where i refers to the specific instance of conflict, is addressed by adding new constraints to the Node during the lower-level search. This node is continually updated throughout the ECT expansion. EC_i remains unchanged unless new environmental changes are introduced.

4.3. Experimental Explanation

The evaluation utilized the MAPF benchmark, which includes 32 grid maps with varying attributes such as city maps, random obstacle grids, mazes, and warehouse maps. Each grid can support sim-

DOI: 10.54254/2755-2721/134/2025.22268

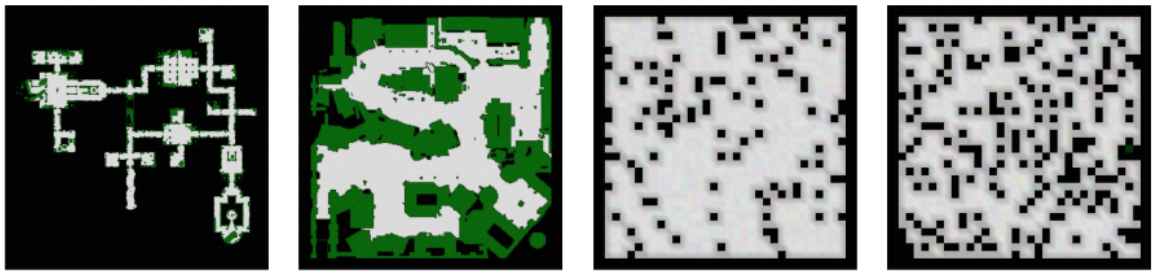


Figure 6: Part of the experimental map [8]

ulations with up to 7000 agents, referring to the entities that navigate the maps, such as robots or virtual characters, as well as various starting vertices and targets. Some examples of datasets used in this study are illustrated in Fig. 6. In this field, algorithms commonly employ CBS-H2 for global path search to determine the initial path of each instance. CBS-H2, or Conflict-Based Search with Heuristics, is an enhancement of the original CBS algorithm that incorporates heuristic strategies to optimize pathfinding efficiency by reducing the search space and expediting conflict resolution. When calculating path cost, both movement and waiting will incur cost 1 unless the agent reaches the destination. The maximum running time limit for each test is 300 seconds. Firstly, the author used DI-CBS, CBS-planner, and CBS-D*-lite as pathfinding algorithms and controlled the frequency of environmental changes in each map, ranging from 4 to 24 changes per unit time. Partial results are shown in Fig. 7 and Fig. 8, where the x-axis represents the number of environmental changes and the y-axis represents the calculation time. The number of agents is always 10. The experiment was conducted on a 32x32 small map and a 100x100 large map. For a 100x100 large map, the experiment generated 2000 examples of random starting vertices and targets. By comparing these two images, the author found that the efficiency of CBS planner slightly exceeded DI-CBS (less than 10%) when there were fewer environmental changes. However, as the number of changes increases to 8 times per unit time or more, the running time of DI-CBS is of that of CBS planner. Especially when the environmental changes exceed 16 times, the running time of CBS-planner and CBS-D* - lite increases exponentially, reaching near 10^5ms (which is equal to 100 seconds), while this algorithm keeps about 10^3ms. Thus, there is a significant time efficiency in DI-CBS.

DOI: 10.54254/2755-2721/134/2025.22268

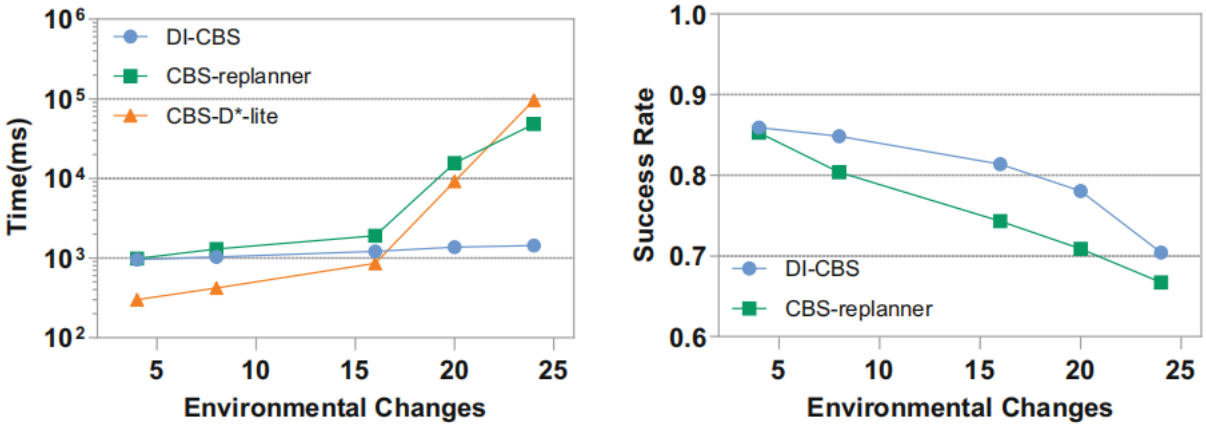


Figure 7: Experimental results of DI-CBS and CBS-Replanner in map den-520d [8].

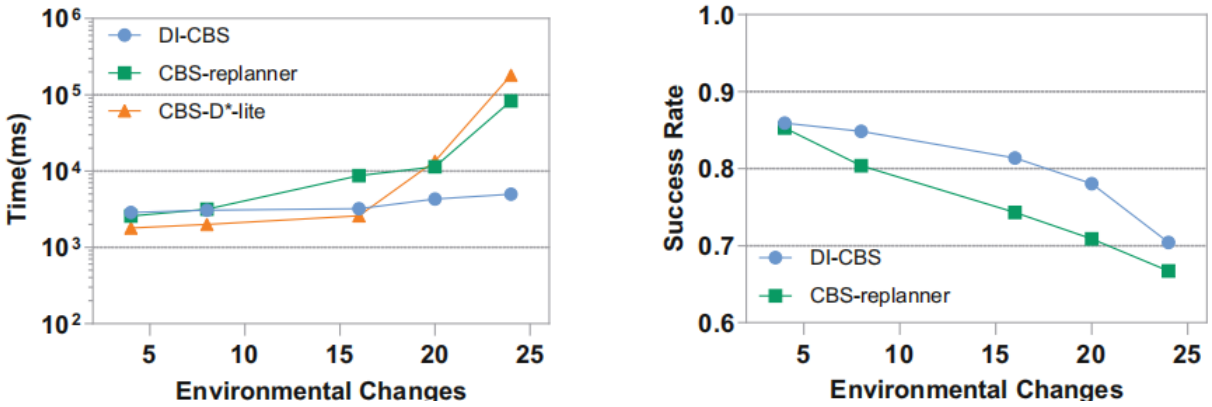


Figure 8: Experimental results of DI-CBS and CBS-Replanner in map brc-202d [8].

In addition, the author conducted experiments using DI-CBS and DI-CBS without SLPA* to compare the effectiveness of ECT and SLPA* in accelerating efficiency. The results are shown in Fig. 9 and Fig. 10. The author found that DI-CBS using SLPA* can still maintain high efficiency even in rapidly changing environments, while DI-CBS without SLPA* has a runtime variation that is basically consistent with CBS planner. It is obvious that expanding the environment conflict tree nodes can accelerate retrieval efficiency, and SLPA* has a significant effect in frequently changing scenarios.

DOI: 10.54254/2755-2721/134/2025.22268

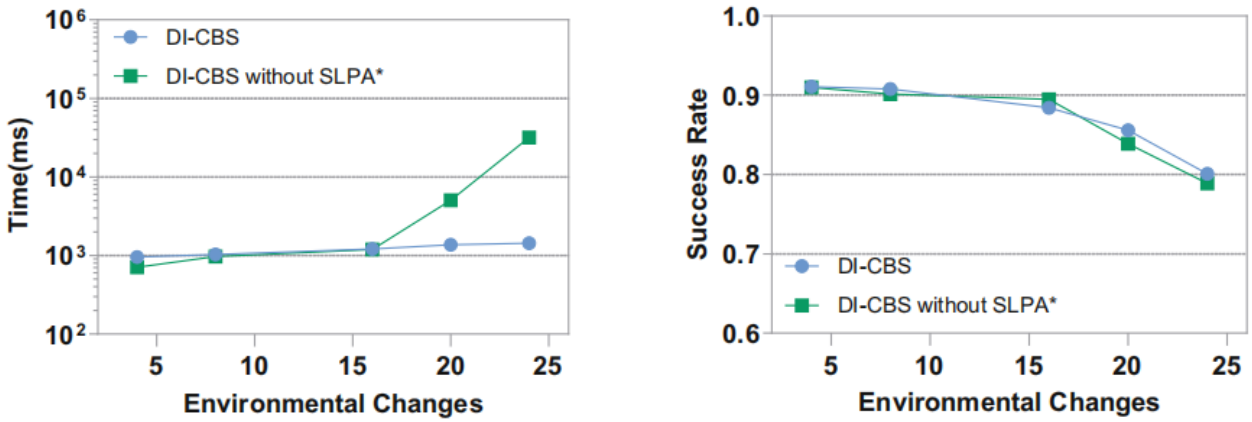


Figure 9: Experimental results of DI-CBS and DI-CBS without SLPA* in map den-520d [8].

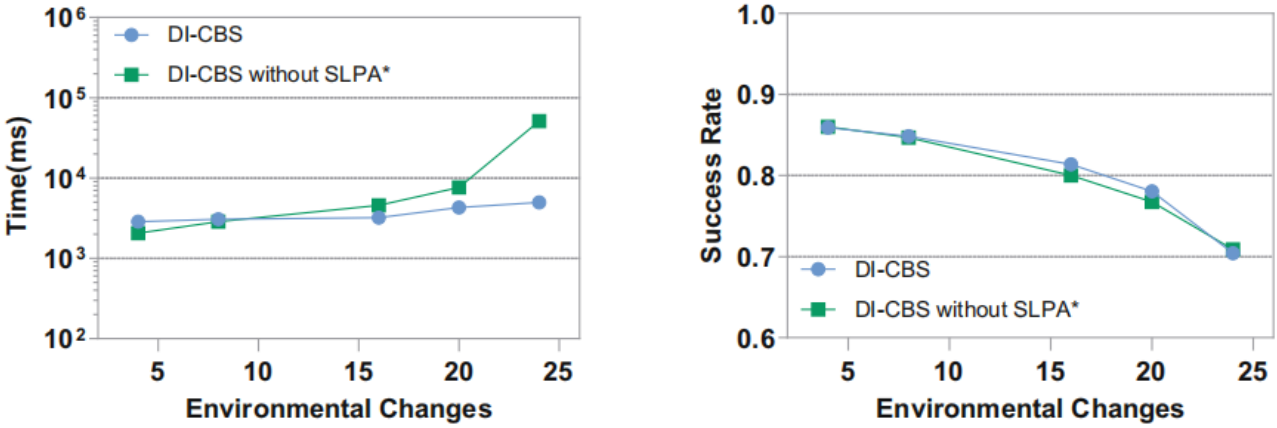


Figure 10: Experimental results of DI-CBS and DI-CBS without SLPA* in map brc-202d [8].

5. Multi-Agent Pathfinding With Real-Time Heuristic Search

Non-real-time algorithms first determine complete paths for all agents before they begin to move. However, their searches are often too costly, and when the environment changes (such as obstacles moving or being added), path replanning may be necessary. Moreover, congestion may occur as multiple agents share the same segment. The paper [9] is dedicated to the development of real-time algorithms that generate the next actions for all agents, eliminating the necessity to find complete paths for them, thereby enabling real-time movement of the agents. The authors introduce a real-time heuristic search algorithm for multi-agent pathfinding, named Bounded Multi-Agent A* (BMAA*) [9]. This algorithm performs an independent real-time heuristic search for each agent, where heuristic values are dynamically updated based on the distance or cost from the agent's current location to its goal. These heuristic values help guide each agent towards its destination while accounting for changes in the environment and the positions of other agents. The functions also by considering other agents as dynamic obstacles, all without mutual coordination or path sharing. Furthermore, the authors conduct

DOI: 10.54254/2755-2721/134/2025.22268

an experimental comparison of four versions of BMAA* against FAR and A*-Replan across multiple maps to assess the overall completion rate.

5.1. The Improved Methods

5.1.1. BMAA*

The BMAA* algorithm, a multi-agent pathfinding (MAPF) method, improves upon previous approaches by allowing each agent to execute its own Real-Time Adaptive A* (RTAA*) search independently. Unlike Windowed Hierarchical Cooperative A* (WHCA*), which requires a reservation table to share paths and adds a time dimension to the search space, or FAR, which relies on flow annotations but can still result in agents getting stuck, BMAA* operates in real-time without needing to compute complete paths beforehand.

BMAA* reduces search costs by treating other agents as moving obstacles and does not require explicit coordination or preprocessing. It is particularly effective in dynamic environments, where the map can change or players might indirectly affect the movement of NPCs (non-player characters), minimizing the impact of such changes on the overall search process. This flexibility and adaptability make BMAA* more suitable for real-time applications than WHCA* and FAR.

5.1.2. RTTA*

The main idea behind the well-known algorithm Real-Time Adaptive A* (RTAA*) is that the A* algorithm operates by maintaining a priority queue, known as the ”open list,” initially containing only the start state, s_{curr}. A* selects a state, s, with the smallest f-value(he sum of the g-value) from the priority queue. If state s is a goal state, the algorithm stops. Otherwise, it expands the state, updating the g-value(the cost from the start state to s) of each successor state(each state reachable directly from s) and adding those successors whose g-value has decreased to the open list. This process is repeated until termination.Upon completion of each A* search iteration, the g-values of all expanded states are recorded to inform subsequent searches. RTAA* then updates the heuristic function using these values to expedite future searches, effectively making the algorithm ”adaptive” by improving the heuristic accuracy based on prior explorations After each A* search, we can make the heuristic function more informed by computing acceptable estimates of the goal distance of a state, to expedite future A* searches. Let s be a state expanded during the A* search. Authors obtain an acceptable estimate of its goal distance $gd[s]$ as follows: the distance from any goal state to state s is equal to the distance from the start state s_{curr} to s plus the goal distance $gd[s]$. Therefore, the goal distance $gd[s]$ of state s is not less than the goal distance $gd[s_{curr}]$ of the start state s_{curr} minus the distance from s_{curr} to s.

$$gd[s] \geq f[s] - g[s] \tag{1}$$

Thus, $f[s] - g[s]$ provides an acceptable estimate of the goal distance $gd[s]$ and can be computed rapidly. By calculating this difference and assigning it to each state expanded during the A* search, a more informed heuristic function can be obtained, called Real-Time Adaptive A* (RTAA*).

5.2. Three Extended Modules of BMAA*

In addition, three modules about BMAA* were mentioned in this paper: Procedure NPC-Controller, Procedure Search-Phase and Procedure Search.

DOI: 10.54254/2755-2721/134/2025.22268

5.2.1. Procedure NPC-Controller

Firstly, in Procedure NPC-Controller, for each agent in the system, the following operations are performed: Firstly, the Search-Phase() method of the agent is called to execute the search phase. Subsequently, if the next movement node for the agent has been determined, it attempts to move. If it is possible to push other agents (push = true)and the node is blocked by another agent, it pushes the obstructing agent away.If space is available for the obstructing agent to move into, it is displaced; otherwise, the push operation is halted, and the moving agent waits or recalculates its path. Finally, if the node n is not blocked by other agents, the agent moves to the node. The time step is then incremented by 1.

5.2.2. Procedure Search-Phase

Next, In the search phase of BMAA*, the conditions for path search are initially evaluated. In the event that the path from the current node to the target node is undefined or if the current time exceeds the specified time limit, the search is executed. In the event that the open list of the search is not empty, the first node in the open list is selected and its composite cost f is calculated. Then, the Update-Heuristic-Values procedure is employed to update the heuristic values of the nodes in the closed list throughout the search process, with the new time limit set to the current time plus the number of moves.

5.2.3. Procedure Search

At last, in procedure search, the path P and the closed set is first initialized to be empty, the number of expanded nodes exp is set to 0 and the open set contains the current node, with the cost g from the current node to the start node set to 0. Then, if the open set is not empty, a loop is executed. If the first node in the open list is the goal node, or the number of expanded nodes reaches the expansion limit, the path P is computed and the loop is terminated. Otherwise, the first node is removed from the open list and added to the closed set. Next, for each neighbor node of the current node, the neighbor node is determined based on the flow annotations. If the neighboring node is blocked by another agent and the distance is less than the field of view, then the neighboring node is skipped. If the neighboring node is not in the closed set, its cost is updated and it is added to the open set. Finally, the number of expanded nodes exp is updated.

5.3. Experiment

The author conduct an experimental comparison of four versions of BMAA* which are BMAA*, BMAA*-c, BMAA*-f and BMAA*-f-c against FAR and A*-Replan. BMAA* cannot push other agents away from the goal locations temporarily and no flow annotations were used, BMAA*-c push other agents away from the goal locations temporarily, BMAA*-f uses flow annotations, and BMAA*-f-c combines both features. The experiment assess the overall completion rate across 10 maps from 3 games(Dragon Age: Origins, WarCraft III and Baldur's Gate II) and ten MAPF instances for each map with the number of agents ranging from 25 to 400 in increments of 25 and from 400 to 2000 in increments of 200.

Fig. 11 illustrates the completion rates of various MAPF algorithms across all instances and maps. As the number of agents increases, completion rates decrease due to heightened congestion and search efforts. It turns out that BMAA* versions exhibit notably higher completion rates compared to FAR and A*-Replan for over 200 agents. This is attributed to BMAA* versions' ability to temporarily dis-

DOI: 10.54254/2755-2721/134/2025.22268

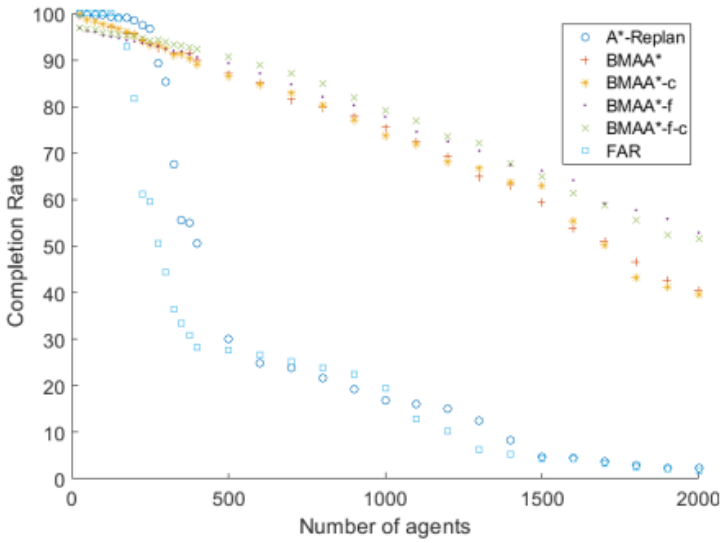


Figure 11: The average completion rates of FAR, A*-Replan and 4 versions of BMAA* algorithm over all MAPF instances [9].

place other agents from their goal locations, reducing congestion at choke points and involve larger travel distances. However, FAR and A*-Replan pre-determine complete paths, leading to shared segments and resulting in congestion.

5.4. Conclusion

The paper focuses on the significant role of Multi-Agent Pathfinding (MAPF) in many video games and introduces a new real-time MAPF algorithm, BMAA*. This algorithm employs a modular design and can be enhanced through the latest flow annotation techniques. Overall, BMAA* demonstrates the potential of real-time heuristic search for MAPF. However, it has some notable drawbacks: longer path lengths, susceptibility to congestion, and the possibility of deadlock situations.

If the initial heuristic values are poorly set, leading to depressions in the heuristic surface, agents may be guided into dead ends. Although recent RTHS techniques have made improvements to shorten travel distances, in certain cases, agents exploring new areas imperfectly can still be considered reasonable. When other agents cannot leave their goal locations, even the BMAA* versions that can temporarily push other agents away may still fail to allow all agents to reach their goal positions. This limitation becomes more pronounced in crowded or complex environments.

In the future, the limitations of the BMAA* algorithm regarding path lengths and congestion issues can be further optimized through techniques such as search space reduction, precomputation, and heuristic value initialization. These approaches can help reduce path lengths and lower the probability of congestion, thereby enhancing its applicability in multi-agent pathfinding scenarios.

6. Conclusion

This paper reviews several studies addressing the Multi-Agent Pathfinding (MAPF) problem, highlighting their applications in online gaming and a variety of real-world scenarios, including automated warehouse logistics and customer parking systems. In the literature presented, research on the application of cooperative mechanisms in multi-agent pathfinding has been proposed. The paper presents

DOI: 10.54254/2755-2721/134/2025.22268

an improved solution based on CBS, Co-CBS, which achieves agent cooperation in solving pathfinding problems by operating a meeting level search. Moreover, the paper proposes two improvement schemes based on Co-CBS to meet the efficiency requirements. Another paper studied the pathfinding problem of intelligent agents in dynamic environments. This paper adopts ECT to enable intelligent agents to ”perceive” changes in the environment and avoid newly emerging obstacles. At the same time, the paper also introduces SLPA*, which meets the efficiency of intelligent agent pathfinding when the environment frequently changes. The last paper made a certain contribution to real-time intelligent agent path planning. The paper focuses on studying the pathfinding mechanisms of intelligent agents in many well-known online games and proposes an implementation planning algorithm called BMAA*. This algorithm includes three programs to improve the path planning of NPCs in the game and enhance game performance. In summary, this review paper investigates the practical utility of different algorithms in MAPF, and through a series of experiments, the total research results will be beneficial for improving the success rate and efficiency of multi-agent pathfinding.

Acknowledgment

Thanks to Professor Bill Nace and two teaching assistants, Fang Chen and Ellen Xv, for their guidance.

References

[1] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial intelligence*, 219:40–66, 2015.

[2] David Silver. Cooperative pathfinding. In *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*, volume 1, pages 117–122, 2005.

[3] Ko-Hsin Cindy Wang, Adi Botea, et al. Fast and memory-efficient multi-agent pathfinding. In *ICAPS*, volume 8, pages 380–387, 2008.

[4] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, pages 151–158, 2019.

[5] Nir Greshler, Ofir Gordon, Oren Salzman, and Nahum Shimkin. Cooperative multi-agent path finding: Beyond path planning and collision avoidance. In *2021 International symposium on multi-robot and multi-agent systems (MRS)*, pages 20–28. IEEE, 2021.

[6] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, Oded Betzalel, David Tolpin, and Eyal Shimony. Icbs: The improved conflict-based search algorithm for multi-agent pathfinding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 6, pages 223–225, 2015.

[7] Fatih Semiz and Faruk Polat. Incremental multi-agent path finding. In *Future Generation Computer Systems*, pages 220–233, 2021.

[8] Yu Wang, Yuhong Shi, Jianyi Liu, and Xinhu Zheng. Incremental conflict-based search for multi-agent path finding in dynamic environment. In *Artificial Intelligence Applications and Innovations*, 2024.

[9] Devon Sigurdson, Vadim Bulitko, William Yeoh, Carlos Hernández, and Sven Koenig. Multi-agent pathfinding with real-time heuristic search. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2018.