Practical Consideration for Optimal MAPF$_R$ Solvers

Andy Li[1], **Zhe Chen**[1], **Daniel Harabor**[1]

[1]Department of Data Science and Artificial Intelligence, Monash University, Melbourne, Australia
{andy.li, zhe.chen, daniel.harabor}@monash.edu

Abstract

Multi-Agent Path Finding in the Real domain (MAPF$_R$) extends classical MAPF to more realistic settings by allowing continuous time and real-valued movement costs. However, this extension introduces significant practical challenges. This paper examines fundamental issues in implementing optimal MAPF$_R$ algorithms, including potential ambiguities in collision detection, complications arising from real number representation, precision errors, and even inconsistencies within single algorithm implementation. These challenges affect the reproducibility of the published work, they complicate comparisons between competing implementations and undermine theoretical claims. We propose guiding principles to minimise inconsistencies and outline the necessary information that should be reported in academic papers to demonstrate meaningful advances in the field. Finally, we argue that computing truly optimal solutions for MAPF$_R$ is currently infeasible with existing technology.

Introduction

The multi-agent path finding (MAPF) problem is used to plan collision-free paths for a team of moving agents, given their starting and goal locations. The classical MAPF problem assumes a grid-based environment where all agent's actions are synchronized perfectly with all movements taking the same time to complete. Under these assumptions, optimal solutions for classical MAPF are able to regularly solve problems with up to hundreds of moving agents (Shen et al. 2023). Unfortunately, these plans are seldom applicable in practice because physical robots (e.g., in warehouse settings (Wurman, D'Andrea, and Mountz 2008)) and virtual agents (e.g., in-game settings (Harabor, Hechenberger, and Jahn 2022)) often operate in continuous environments, where actions have different, non-unit durations and action executions are not synchronised.

Walker, Sturtevant, and Felner (2018) proposed a method to handle these differences, by extending the problem model, such that the movement costs in the problem has a real-value instead of unit-cost, however, potential wait duration remains at unit-cost. Later Andreychuk et al. (2022) further extends the model by allowing arbitrary wait duration for agents. In this setup, known as MAPF$_R$, agents operate on a graph-based model of the environment, where actions can begin at any time instant and aegnts are allowed to have arbitrary (real-positive-valued) durations.

Continuous-time Conflict Based Search (CCBS) (**?**) is the first optimal algorithm to solve this problem. Many improvements has been added to the CCBS frame work to increase the efficiency of the algorithm (Andreychuk et al. 2021; Walker et al. 2021; Walker, Sturtevant, and Felner 2024). CCBS starts by finding each agent's individual shortest path without considering other agents, then looks for any conflict/collisions among agents. If a collision between two agents is found, CCBS will compute a constraint for each agent, which forbids the agent to perform the colliding move and allow the other agent to perform the move. Each constraint will become a search node for high-level tree search, and the tree is named constraint tree. The high-level of CCBS performs a best first search of over the constraint tree. The low-level of CCBS find a path for single agent, and follows the constraints generated by high-level.

The theoretical work itself already presents significant complexity, compared to discrete optimal MAPF, state-of-the-art optimal solvers can only solve up to tens agents, and only found out to be they are incomplete and suboptimal due to some overlooked detail in the proof(Li, Chen, and Harabor 2025). additional challenges arise when attempting to implement these theoretical algorithms for practical use or empirical experiment evaluation for theoretical study, as compared to implementing classical discrete MAPF. Many challenges that current researchers are not even aware of.

The challenges come from a lack of detail regarding the implementation of the theoretical algorithm proposed in the published paper. The missing details include ambiguity on the solution validator, which real-number representation is chosen, and how to handle the precision error. This ambiguity may lead to different implementations of the algorithm producing conflicting results. Different implementations might produce different optimal solution costs, or even disagree on the feasibility of a solution. This paper reveals these challenges in detail and their associated consequences. We also provide general guidance to mitigate the problem as much as possible, and discussed how to describe and compare the algorithmic improvements for future publications.

Validator

Collision Checking

To solve the problem of finding collision-free paths, it's essential to have a clear definition of what constitutes a collision, along with a concrete implementation of its validator (i.e., collision detection). This is fundamental, as it directly influences the solution space and determines the feasibility of a solution. Validating the plan is trivial in classical discrete MAPF theoretical and practical.When advancing from MAPF to MAPF$_R$, people are aware there are different ways to detect a collision, however, people have been assuming they are fundamentally equivalent just with some efficiency or other minor differences. There are two different general approaches to detect collision in real domain: analytically, and the numerically. The analytical method uses a derived formula to solve an equation, which is very efficient ($O(1)$), however, it is only capable of detecting collision for agents with certain simple interactions(Walker and Sturtevant 2019). Due to the fact that analytical solution does not exists for all equation(Pierpont 1896). The numerical methods use an iterative approach to identify the closest distance among agents and terminate at a given tolerance of precision error. This allows collision detection for agents with arbitrary interactions with a sacrifice on accuracy and efficiency. These two approaches are fundamentally different on solving the same problem. Therefore, they will provide different results in some cases. Given the importance of the validator, the difference is not negligible. We believe all publicly accessible MAPF$_R$ implementations uses analytical methods for collision checking, potentially because of assumptions for MAPF$_R$ problems are constant velocity in all existing publications.

Constraint Generation

Constraint generation is also a key component of CCBS. Although it is distinct from the collision-checking function, the two are closely interrelated. A properly designed constraint generation function should guarantee that, once the appropriate constraints are applied, a previously colliding move becomes collision-free. Similar to the collision-checking function, constraints can be generated either analytically or numerically. While all MAPF$_R$ implementations adopt analytical methods for collision checking, some implementations compute constraints numerically, whereas others rely on analytical formulations. Such a situation could lead to different implementations producing different "optimal solution cost" due to the size of the generated constraint.

Number representation

If all researchers agree on the same validator, more challenges presents when these method are implemented on computer. Since modern computers are inherently based on binary hardware and operate within a discrete framework, therefore storing and operating with irrational number is not trivial under such system. The research on trying to better represent irrational number in computer has been well-studied in many areas, such as numerical calculation, computational geometry and many others. Many customized number representations have been developed to tackle different types of problems, and all of them can be categorized into two types: exact representation (expressions, that can be evaluated up to ∞ precision), and approximate representation (floating, fix point).

Exact Representation stores an expression for a real number instead of number itself. So that the precision of the real number can be evaluated at any precision given enough time, however, it often suffers from performance problems, even very highly optimized implementation(GmbH 2025) is still 2-order of magnitude slower than standard floating-point operations. More fundamentally, the algorithms for some atomic operations are even incomplete, including comparison between two numbers(Ménissier-Morain 2005), which means it might take an infinite amount of steps to perform a comparison. The implementation in modern libraries can guarantee termination by introduce error tolerance. Therefore, exact real-number representation is not a meaningful option for MAPF$_R$ implementations.

Approximation Representation was first introduced very early with the computer itself in the 1940s, fix-point and floating-point methods. They were considered basic, yet are the most common representation of real numbers nowadays. Floating-point representation is the default representation for all implementation (including all the MAPF$_R$ solvers), and built-in at hardware level in modern computers. FP provides a wide range of values within a relatively small bit size, but the error magnitude depends on the exponent. In contrast, fixed-point representation ensures a uniformly distributed error across the entire range, but its representable range is limited compared to floating-point formats with the same bit size. More advanced approximate representation was formalized later in the 1990s, such as *multiple-digit* and *multiple-term* representations. They provide higher precision by using more memory; however, they still suffers from the core problem in as former two, i.e. rounding error.

Rounding Error is unavoidable for approximate representations, therefore error bounding methods were developed to provide a guarantee on the size of potential errors and help users understand how rounding error behaves in a chain of operations. Such bound is often expressed in an interval of lower bound and upper bound, and the true value will lie within the interval. *Interval math* was developed which simply use such bounds to represent the number and perform all the operations directly on the intervals(Hickey, Ju, and Van Emden 2001). The idea of error bounding is not applicable to MAPF$_R$ algorithms, as it is only possible in a *sequence* of *bounded* operations, such as arithmetic and square root. Most of the MAPF$_R$ solver is a tree-based search algorithm, which has diverging branches based on some conditions.Solution cost difference introduced by the precision error in the validator is not boundable. If a collision is falsely detected due to precision errors, i.e. a collision is reported when none actually exists, a detour could be introduced to avoid the perceived collision. Such detour could be arbitrarily long depends on the map, which cause the final solution cost be much larger then true optimal cost.

Error Handling

Given a commonly agreed validator model, and numerical representation, error handling method is still critical for a consistent result. It's well know that error tolerance ϵ value is needed for equality check, i.e. $|a - b| < \epsilon$. However, the use of ϵ in other comparisons ($<, >$) is not commonly agreed, sometimes even within a single implementation. Further than that, ϵ value itself is also not commonly agreed. This not only cause different implementation is essentially solving different problem, but also breaks the consistency within one implementation.

Consistency within One Implementation

Despite using the same validator on same numerical representation with same error handling method, inconsistencies may still arise within a single implementation due to the rounding error. For instance, consider a model in which $a > b \Rightarrow a > b + \epsilon$ and $a < b \Rightarrow a < b - \epsilon$. In the case of single precision FP numbers with $a = 0.49999899, b = 0.5$ and $\epsilon = 1\text{e}{-6}$, the following can be derived:

- $|a - b| = 1.01\text{e}{-6} > \epsilon$, which implies $a \neq b$,
- $b + \epsilon = 0.50000101 > a$, therefore $a \not> b$, and
- $b - \epsilon = 0.49999899 = a$, therefore $a \not< b$

Existing implementation rarely check all three condition together for a relationship comparison, instead, a common approach is simply checking one condition and assuming its one of other two conditions will be the opposite. A macro-level consequence of this issue is the potential for the search process to enter an infinite loop failure.

Continuous time Conflict Based Search(CCBS) (**?**) is the foundational framework for all state-of-the-art optimal MAPF$_R$ solvers. Thus we will use CCBS as an case study example to explain the *Mutual Agreement* principle. The propsed principle applies to any of the two-level tree search solvers that suffers from the precision error loss during the search. CCBS is a two-level tree search algorithm that begins by computing the shortest path for each agent individually. During each iteration, the algorithm checks for collisions among the agents' current paths. If no collisions are found, the solution is found. However, if a collision is detected, one is selected for resolution. A constraint is then generated to avoid the collision, and the single-agent path planner is invoked to replan the agent's path while considering the newly introduced constraint along with all previously generated constraints.

Mutual Agreements in MAPF$_R$

To address the aforementioned issue, we introduce a guiding principle termed Mutual Agreements, designed to mitigate the effects of precision errors and ensure consistency in FP computations. We acknowledge that precision loss arises solely during computation. Accordingly, the principle of Mutual Agreements stipulates that once a value has been computed, its exact result should be mutually accepted by all components within the implementation. This consistency is primarily achieved by reusing previously computed results rather than re-executing equivalent operations, even for seemingly trivial cases such as simple additions.

Standardized Collision Definition Similar to collision detection, constraint generation is also a core component in many optimal MAPF solvers. In discrete MAPF this component can be implemented without any ambiguity. However, this is not the case for MAPF$_R$. A correctly generated constraint implies that the collision will necessarily be avoided by following the constraint. Constraints can be computed either analytically by derived equation, or numerically such as binary search. We notice that an analytical solution might cause *inf loop* failure due to the precision loss in computing the equations. It might not resolve the collision defined by collision detector, as analytical solution does not explicitly check the result from collision detector. This will result into an *inf loop* failure by fail to generate a collision resolving constraint, and the search will keep resolving such collision. Numerical method performs a binary search by calling collision detector explicitly at each iteration. Therefore there will be no disagreement between collision detector and constraint generation.

One-way calculation: *Only perform one-way calculations and avoid redundant calculations by reusing results.* For example, a move constraint may forbid a movement from occurring within a specific time interval, such as $[t_1, t_2)$. As a result, the move is only allowed to take place starting at time t_2. The g value of the successor node is then calculated as $t_2 + d(move)$, where $d(move)$ is the travel distance of the move. In the successor node, it is common to use $g - d(move)$ to determine the departure time of the move.However, we avoid using this approach here, since t_2 is a verified number (one that does not violate constraints) and $g = t + d(move) \rightarrow t = g - d(move)$ is not true when dealing with FP calculations. Fail to do so will also result in *inf loop* failure, as generated constraints are not interpreted properly at single agent planner, hence, not resolving the collision. To mitigate this, we only use addition when checking successors, even though the two equations are mathematically equivalent.

Calculation in Comparison: Due to the structure of FP representation, the error tends to be larger when performing calculations on two numbers with large differences in magnitude. This issue, known as *catastrophic cancellation*, occurs when the precision of the smaller number is insufficient to maintain accuracy after the operation.

The broken comparison shown above is an example of this effect. Therefore, when performing the comparison with a calculated result, such as $a - b < c - d$ where $d << a, b, c$, the comparison should be performed as $a - b - c < -d$. In ϵ comparison model, generally speaking, the compared value tends to be much larger than ϵ, therefore, we propose using a comparison model that use $|a - b| <= \epsilon$ for equality check,

Alg.	ϵ	Sol.	$\overline{\delta^+}$	$\max_{\delta+}$	$\lvert\delta^+\rvert$	$\overline{\delta^-}$	$\max_{\delta-}$	$\lvert\delta^-\rvert$	Inf
V.	8	655	-	-	-	-	-	-	21
m.	∞	638	0	0	0	5.33e-7	1.48e-6	111	0
m.	8	653	0	0	0	5.45e-7	1.47e-6	119	0
m.	6	653	4.30e-7	9.63e-7	38	8.38e-7	1.73e-6	63	0
m.	4	654	4.54e-5	1.03e-4	40	4.07e-5	1.05e-4	80	0
m.	1	764	3.77e-1	1.069	61	7.905e-1	11.75	195	0
V.DS	8	1038	-	-	-	-	-	-	105
m.DS	∞	973	1.01e-5	1.01e-5	1	5.92e-7	2.46e-6	274	0
m.DS	8	985	0	0	0	6.39e-7	2.45e-6	284	0
m.DS	6	1014	4.96e-7	1.92e-6	50	4.1e-6	7.43e-5	217	0
m.DS	4	1005	7.40e-5	2.48e-4	89	1.16e-4	1.57e-3	210	0
m.DS	1	1150	5.64e-1	2.637	113	1.623	87.08	459	0
V.DSH	8	1038	-	-	-	-	-	-	105
m.DSH	∞	975	1.01e-5	1.01e-5	1	5.92e-7	2.46e-6	275	0
m.DSH	8	985	0	0	0	6.39e-7	2.45e-6	284	0
m.DSH	6	1014	4.96e-7	1.92e-6	50	4.1e-6	7.43e-5	217	0
m.DSH	4	1005	7.40e-5	2.48e-4	89	1.16e-4	1.57e-3	210	0
m.DSH	1	1151	5.64e-1	2.637	113	1.620	87.08	460	0
V.DSHP	8	1163	-	-	-	-	-	-	4
m.DSHP	∞	1117	2.26e-6	1.01e-5	22	7.57e-7	4.98e-6	405	0
m.DSHP	8	1117	0	0	0	7.81e-7	5.02e-6	424	0
m.DSHP	6	1135	5.19e-7	1.99e-6	63	1.31e-5	1.80e-4	320	0
m.DSHP	4	1128	8.15e-5	2.55e-4	149	1.54e-4	1.57e-3	287	0
m.DSHP	1	1301	9.89e-1	5.649	186	1.977	87.08	502	0

Table 1: *V.* for existing CCBS implementation, *m.* for our modified implementation, *Disjoint Splitting* (DS), *High-Level Heuristics* (H), *Conflict Prioritisation* (P). ϵ is showed in 10^{-x} format, and ∞ stands for $\epsilon = 0$. $\overline{\delta}$ is the average cost difference compared to baseline, $\max_\delta$ is the max difference, $|\delta|$ for the number of different instances. δ^- for modified version with lower costs, and δ^+ for higher costs.

$a - b > \epsilon$ for greater than, $a - b < -\epsilon$ for less than

$$|a-b| <= \epsilon \Rightarrow a = b$$
$$a - b > \epsilon \Rightarrow a > b$$
$$a - b < -\epsilon \Rightarrow a < b$$

Emprical evaluation on CCBS as a Case Study

Continuous time Conflict Based Search(CCBS) is the foundation of many optimal solvers. We have used the publicly available implementation of CCBS provided by original authors as a case study in this empirical evaluation. We modified the original implementation by applying the *Mutual Agreements*, the new implementation is publicly accessible[1].

We conduct experiments based on the graph benchmark instances provided in the existing CCBS implementation. The three graphs, sparse, dense and super dense, are sampled from a grid-based benchmark map D520d from movingai.com (Stern et al. 2019). For each instance, we run experiments for each map with the number of agents varying from 3 up to 40. For each map and each number of agents, we run 25 instances. We use the existing implementation (V.), with $\epsilon = e^{-8}$ and $precision = 100 \times \epsilon$, to compare our modified CCBS implementation (*m.*), with different ϵ settings and with *Disjoint Splitting* (DS), *High-Level Heuristics* (H), and *Conflict Priorisation* (P) turned on or off. $precision$ is a variable to control the error tolerance for computing constraint intervals, which is set to $100 \times \epsilon$ by default.

[1]https://link will be provided if accepted for publication.

Comparison between different implementations The results in Table1 show a comparison of the optimal solution costs computed by each algorithm with different implementation. A notable observation is that more than 200 infinite loop failures were found in the unmodified version, due to inconsistencies within the single implementation. For co-solvable instances, the modified implementation always found solutions with lower costs using $\epsilon = e^{-8}$ and $\epsilon = e^{-7}$. Even for larger ϵ values, we observe more solutions with lower costs than those with higher costs. Although the results indicate that the existing implementation finds more solutions, this is often because the original implementation's greedy $precision$ value imposes significantly larger constraints, leading to "greedily" finding solutions with higher costs. We also experimented with our $m.$ version with $\epsilon = 0$, meaning comparisons were done based on raw algebraic comparisons, and constraints were binary searched until machine ϵ level accuracy was reached. For the $m.$ version, smaller ϵ values resulted in more detected collisions, which explains why the $m.$ version includes cases where $\epsilon = 0$ resulted in higher solution costs.

Comparison between algorithms

Improving techniques (such as DS), should function like add-ons that layer on top of existing algorithms, speeding up the solving process without altering the problem's solution cost. However this is not the case in MAPF$_R$, even under the same implementation. We observed that different improvement techniques in CCBS produced different optimal solution costs for the same problem due to precision loss during the search process, despite applying mutual agreement principles. In Fig.1, we used the minimum-cost solution across all algorithms as the baseline and plotted the cost differences of the other algorithm solutions in comparison. The solution cost differences among algorithms in V.CCBS are the around 10^{-7} level, peaking at 8.14e-7 when $\epsilon = 10^{-8}$. Modified CCBS at the 10^{-8} level peaks at 1.20e-8, and when $\epsilon = 0$ in the modified version, the solver consistently returns the exact same solution across all algorithms.

In Fig2, we illustrate the trend of how cost difference changes as we reduce the size of ϵ. The sample variance is calculated over the sum of difference among algorithms for each instance.The maximum cost difference is determined by the largest difference observed for each instance. The maximum difference is at least two orders of magnitude larger than the variance, and can be up to ten orders of magnitude higher. The maximum cost difference typically aligns with the order of magnitude of ϵ, except when ϵ=0.1, which represents 20% of the agent's radius. At this point the definition of collision becomes somewhat ambiguous.

Proceeding Research without Ambiguity

Given that all the issues mentioned above are inherent to the problem and cannot be fully eliminated, the question then becomes: *How do we proceed?*

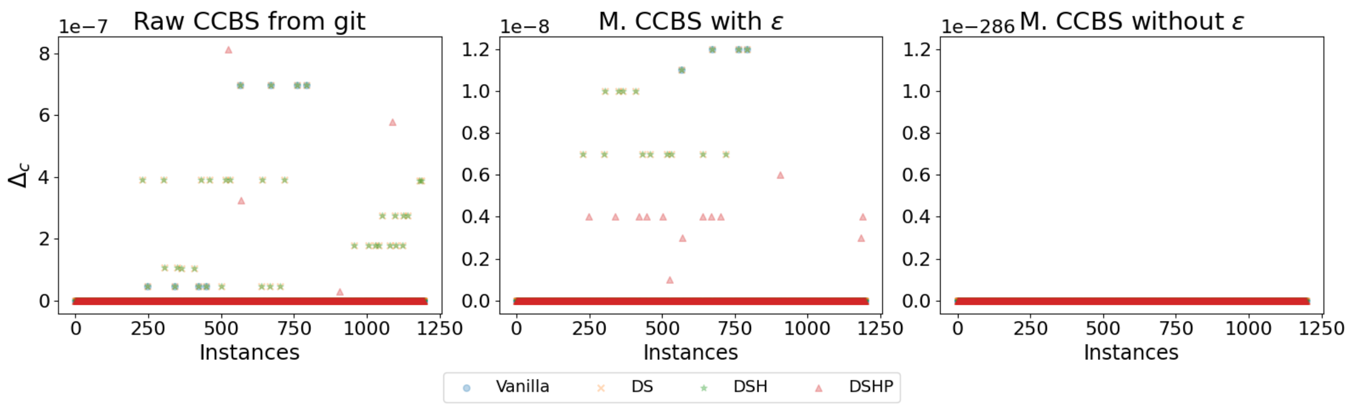

Figure 1: Cost difference compared to the lowest solution cost between all algorithms

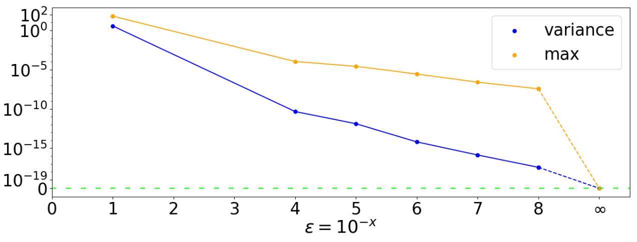

Figure 2: Sample variance and max on the cost difference against varying ϵ on m. implementation.

Reproducible Implementation A reproducible implementation and its experimental results are crucial in research. However, due to insufficient description in the current MAPF$_R$ papers, they are not reproducible. Based on our investigation so far, we believe the following information is essential for ensuring a reproducible implementation:

- Method of detecting collision
- Representation of real number
- How to address representation errors
- Size of tolerance
- Interpretation of the tolerance
- Additional approaches to handling representation errors

Comparable Metric After providing a description of a reproducible implementation, we must also report the results of an empirical experiment. However, due to the inconsistency issues previously mentioned, we believe solvers will still produce different optimal solution cost.

We identify three different solution spaces when trying to implement an optimal MAPF$_R$ algorithm. There are: $S_{\mathbf{R}}$, total solution space over the real number domain. $S_{\mathbf{C}}$, Representable solution space given a numerical representation. $S_{\mathbf{S}}$, Reachable solution space given a numerical representation with a starting point.

Unless proven otherwise, we assume the following relationship between the solution spaces: $S_{\mathbf{R}} \in S_{\mathbf{C}} \in S_{\mathbf{S}}$. We believe achieving optimality in $S_{\mathbf{R}}$ is currently impossible due to technological limitations, and it could only be theoretically feasible if an analytical solution for optimal MAPF$_R$ exists—something that doesn't yet exist for optimal MAPF. Even with an analytical solution, modern computers are still limited by real number representation as the overall algorithm can not be compelete, if each number comparison operation is not complete as mentioned in *Number representation*. While tie-breaking doesn't affect MAPF's theoretical properties, in MAPF$_R$, implementation issues may lead to different results from theoretically equivalent processes. The current default numerical representation used in all MAPF$_R$ implementation, Floating Point representation, for all, additionally suffers from the loss of several fundamental mathematical properties during computation, including Associativity, Distributivity, Reversibility, and transitivity of equality. Their absence can cause the search process to yield different solution costs when starting from equivalent nodes. Hence, we suspect $S_{\mathbf{S}} \neq S_{\mathbf{C}}$. Achieving optimality in $S_{\mathbf{R}}$ might only be possible through brute-force enumeration. Unless $S_{\mathbf{S}} = S_{\mathbf{C}}$ is proven, no reasonable search will exhaust all equivalent solutions without quickly devolving into brute-force enumeration. Thus, for theoretically equivalent algorithms, differences in tie-breaking or starting points can lead to different "optimal" costs. Therefore, empirical results should be treated with caution, comparing not just success rates and computation times but also solution costs.

Despite the fact that all robots are executing on continuous environment, the planner will inevitably be in discretion domain because of the digital computer. We propose that a model where minimal unit size can be considered. In such model, instead of planning on graph with arbitrary real value edge costs, a very fine grid can be considered as an underlying map. The collision detection, and other critical components can be computed as classical grid MAPF by checking the occupancy of the cell. We believe search on this model can provide a comparable and consistent result to the field.

Future Work

As discussed in the paper, the current implementation searches in a *Reachable Solution Space*, which is significantly smaller than the *Continuous Solution Space*, and yet it cannot guarantee optimality within the *Reachable Solu-*

tion Space. Future work aims to explore approaches that can provide guarantees for the solutions produced by the implementation. These guarantees focus on several key aspects.

The first is the feasibility guarantee. While error analysis for MAPF$_R$ is not possible at the algorithmic level, error bounds can still be provided within the collision detection process, guaranteeing a truly collision-free plan by considering any ambiguous situation as collisions.

For completeness and optimality, it's possible to guarantee optimality within *Reachable Solution Space* and *Representable Solution Space* by using Fixed-point Numbers. This approach avoids rounding operations and their associated errors by only operating on representable numbers.

References

Andreychuk, A.; Yakovlev, K.; Surynek, P.; Atzmon, D.; and Stern, R. 2022. Multi-agent pathfinding with continuous time. *Artificial Intelligence*, 305: 103662.

Andreychuk, A.; Yakovlev, K. S.; Boyarski, E.; and Stern, R. 2021. Improving Continuous-time Conflict Based Search. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, 11220–11227. AAAI Press.

GmbH, A. S. S. 2025. LEDA: A Library of Efficient Data Types and Algorithms. Accessed: 2025-03-27.

Harabor, D.; Hechenberger, R.; and Jahn, T. 2022. Benchmarks for pathfinding search: Iron harvest. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, 218–222.

Hickey, T.; Ju, Q.; and Van Emden, M. H. 2001. Interval arithmetic: From principles to implementation. *J. ACM*, 48(5): 1038–1068.

Li, A.; Chen, Z.; and Harabor, D. 2025. CBS with Continuous-Time Revisit. *arXiv preprint arXiv:2501.07744*.

Ménissier-Morain, V. 2005. Arbitrary precision real arithmetic: design and algorithms. *The Journal of Logic and Algebraic Programming*, 64(1): 13–39. Practical development of exact real number computation.

Pierpont, J. 1896. On the Ruffini–Abelian theorem.

Shen, B.; Chen, Z.; Cheema, M. A.; Harabor, D. D.; and Stuckey, P. J. 2023. Tracking Progress in Multi-Agent Path Finding. *CoRR*, abs/2305.08446.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, 151–158.

Walker, T. T.; and Sturtevant, N. R. 2019. Collision Detection for Agents in Multi-Agent Pathfinding. arXiv:1908.09707.

Walker, T. T.; Sturtevant, N. R.; and Felner, A. 2018. Extended Increasing Cost Tree Search for Non-Unit Cost Domains. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 534–540. International Joint Conferences on Artificial Intelligence Organization.

Walker, T. T.; Sturtevant, N. R.; and Felner, A. 2024. Clique Analysis and Bypassing in Continuous-Time Conflict-Based Search. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '24, 2540–2542. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9798400704864.

Walker, T. T.; Sturtevant, N. R.; Felner, A.; Zhang, H.; Li, J.; and Kumar, T. K. S. 2021. Conflict-Based Increasing Cost Search. *Proceedings of the International Conference on Automated Planning and Scheduling*, 31(1): 385–395.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, 29(1): 9–20.