Delft University of Technology, Bachelor Seminar of Computer Science and Engineering

Multi-Agent Pathfinding with Matching using Increasing Cost Tree Search

Thom van der Woude[1]

Supervisors: Jesse Mulderij[1] Mathijs de Weerdt[1]

[1]TU Delft
t.b.vanderwoude@student.tudelft.nl

Abstract

Both the assignment problem and the multi-agent pathfinding problem are common problems in the fields of robotics and transportation. The joint problem of multi-agent pathfinding extended with the assignment of goals to agents, matching, is something that has not been studied much; few methods exist today that solve it. In this work, two types of algorithms based on the Increasing Cost Tree Search (ICTS) algorithm for multi-agent pathfinding are presented that can optimally solve this joint problem: exhaustive algorithms that reduce the problem to solving many multi-agent pathfinding problems using regular ICTS, and algorithms that search a generalized increasing cost tree. These are compared to each other experimentally on a set of grid maps, and it is shown that exhaustive methods typically outperform the generalized ICTS. Lastly, an exhaustive ICTS algorithm is compared to alternative algorithms based on other multi-agent pathfinding approaches to put its performance into the broader context of algorithms for multi-agent pathfinding with matching.

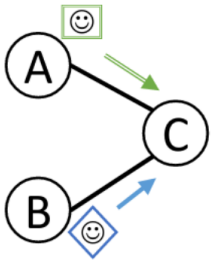
1 Introduction

Finding a shortest path between two vertices on a graph is a fundamental problem that appears in many contexts like GPS navigation [1] and network routing [2]. In some applications like autonomous aircraft towing [3] and train unit routing in shunting yards [4], planning agents separately can result in conflicts between agents as they follow their paths. To prevent such conflicts, agents have to be planned together to find a combination of non-conflicting paths. The problem of finding such a combination is the Multi-Agent Pathfinding (MAPF) problem [5]. In finding optimal solutions, one of two objectives is typically optimized: the makespan which is the cost of the longest path, or the Sum of Individual path Costs (SIC). Algorithms for optimally solving MAPF include Increasing Cost Tree Search [6], Conflict-Based Search [7] and Branch-and-Cut-and-Price [8].

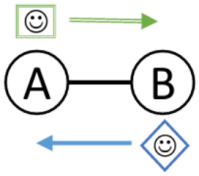
As observed by [9], in real-world applications of MAPF such as warehouse robotics [10] goals are often assigned to teams of agents instead of single agents as in standard MAPF. Using MAPF in such applications requires pre-assignment of goals to agents using the Hungarian Algorithm [11] or similar methods, but such approaches are suboptimal as the true cost of a given matching can only be found by solving the corresponding MAPF instance.

Ma and Koenig [12] formulate the joint problem of finding the optimal makespan solution across all possible matchings as the combined target-assignment and pathfinding (TAPF) problem. They also present the Conflict-Based Min-Cost-Flow (CBM) algorithm for solving TAPF. In CBM, the Meta-Agent variant of CBS [7] is used as a high-level search framework; in the low-level search, single teams are planned in polynomial time as constrained anonymous MAPF instances using a max-flow approach [13]. The experimental results reported in [12] indicate that CBM scales well as the number of agents and the sizes of teams increase; although in addition to grid instances with 10% of tiles being blocked, Kiva [10] instances are also solved, it remains to be seen how CBM performs in scenarios with more obstacles and more maze-like scenarios such as the Moving AI maps [14].

Nonetheless, CBS successfully being used to solve TAPF raises two questions: Can MAPF algorithms other than CBS be used to solve TAPF? Can CBS or other MAPF algorithms be used to solve TAPF minimizing the SIC instead of the makespan? For this latter problem that will be referred to as Multi-agent Pathfinding with Matching (MAPFM), no methods are described in the literature.



(a) Vertex conflict



(b) Edge conflict

Figure 1: Two types of conflicts between agents from [5].

This work presents two types of algorithms based on ICTS [6] that optimally solve MAPFM, and is part of a study in which other algorithms in addition to ICTS were taken as starting points for MAPFM algorithms. Those of the first type exhaustively enumerate matchings and use a variant of ICTS as a subroutine; those of the second type are based on an extension of ICTS itself to support matching, ICTS-m. The properties of both classes of algorithms as well as ICTS more generally are discussed, followed by an experiment in which a selection of algorithm variants is compared. Additionally, one exhaustive ICTS algorithm is compared to MAPFM algorithms based on CBM [12], M* [15], A*+OD+ID [16] and EPEA* [17].

2 Multi-agent Pathfinding with Matching

A MAPF instance can be defined as follows. Let $G = (V, E)$ be an undirected graph, with each $v \in V$ representing an obstacle-free tile on a 4-connected grid and each $e = (u, v) \in E$ representing a legal move between two such tiles. Let there be k agents $a_1, \ldots, a_k$ with starting locations $s_1, \ldots, s_k$ and goals $g_1, \ldots, g_k$. For each agent a_i, a path π_i from s_i to g_i is to be found such that all agents' paths π_i taken together are non-conflicting and therefore are a solution. In this work, non-conflicting means, in the conflict-terminology of [5], that there are no vertex conflicts and no edge conflicts, as shown in Figure 1. Letting π_i^t denote the t'th node of path π_i, this means that for $i \neq j$, for all t, $\pi_i^t \neq \pi_j^t$ and $(\pi_i^t \neq \pi_j^{t+1}) \vee (\pi_i^{t+1} \neq \pi_j^t)$. To achieve this, each agent a_i may wait during any time step t so that $\pi_i^t = \pi_i^{t+1}$ instead of moving to a grid neighbor. Given a solution $(\pi_1, \ldots, \pi_k)$, a vector of per-agent costs $(c_1, \ldots, c_k)$ can be found. For π_i, c_i is defined as the time at which a_i reaches g_i for the last time, meaning that for $t \geq c_i$, $\pi_i^t = g_i$. For each c_i, $c_i \geq c_i^*$ where c_i^* is defined as the cost of a shortest path from s_i to g_i.

2.1 Objectives for MAPF

Using the established terminology, the two common objectives for MAPF can be restated as follows:

- Makespan: $\max_i c_i$
- Sum of individual costs (SIC): $\sum_i c_i$

In this work, an optimal solution is defined as having minimal SIC.

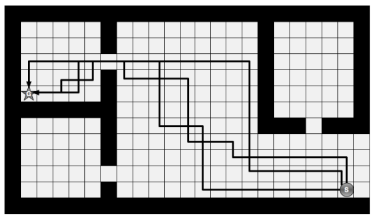


Figure 2: Symmetric paths on a 4-connected grid [18].

The mapping from $(\pi_1, \ldots, \pi_k)$ to a SIC is surjective: many cost vectors $(c'_1, \ldots, c'_k)$ might add up to the same SIC as $(c_1, \ldots, c_k)$ and for a given agent a_i, there might be many equivalent paths π'_i to the goal with cost c_i. On uniform-cost 4-grids in particular, there are often many equivalent and symmetrical paths [18] (see Figure 2), which is why in single-agent pathfinding, symmetry-breaking methods such as Jump Point Search [19] are used to speed up the search. In multi-agent pathfinding, having multiple paths per agent of the same cost can, in contrast, be a boon: it means that there are potentially more non-conflicting ways to combine paths of different agents.

2.2 Extending MAPF with matching

With the multi-agent pathfinding problem defined, the MAPFM problem that is the subject of this work can be described as an extension of MAPF.

Let there be K teams $t_1, \ldots, t_K$, where t_i consists of k_i agents with start positions $s^i_1, \ldots, s^i_{k_i}$ together with an equal number of team goals $g^i_1, \ldots, g^i_{k_i}$. Within each team t_i, agents a^i_j are to be *matched* to a unique goal g^i_k, so that exactly one agent is assigned to each goal within team i; once all agents are matched with goals, for each agent a^i_j matched with g^i_k, a path from s^i_j to g^i_k is to be found such that all agents' paths taken together are non-conflicting and therefore make up a solution, similarly defined as for MAPF.

MAPFM, being TAPF with a SIC objective, generalizes both MAPF and anonymous MAPF [12], which is MAPF allowing an agent to move to any goal, a problem that can be solved in polynomial time.

3 Solving MAPF with Matching using ICTS

In this section, taking ICTS for MAPF as a starting point, two methods to use ICTS to solve MAPFM are described. The first method relies on a reduction from MAPFM to repeated MAPF: an optimal solution to MAPFM corresponds to a matching of agents to goals, so by exhaustively enumerating all matchings and solving these as MAPF instances, an optimal solution to the MAPFM instance can be found. In the second method, the ICT search itself is modified to allow an agent a^i_j to be matched to any g^i_k.

3.1 Increasing Cost Tree Search

In Increasing Cost Tree Search [6], the two-step mapping from path combination to SIC described in Section 2 is reversed to search for a solution with minimal SIC. This is done by searching for cost vectors corresponding to increasing SIC at the top level and searching for non-conflicting path combinations (solutions) for each cost vector at the bottom level.

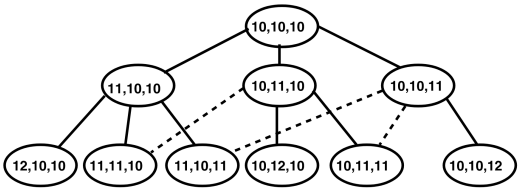


Figure 3: Increasing Cost Tree for three agents [6].

3.1.1 Top-level search

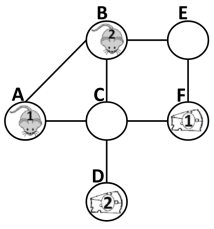
In the top-level search, all possible cost vectors for a given SIC are evaluated by searching an Increasing Cost Tree as depicted in Figure 3, starting from $C^* = \sum_i c_i^*$ corresponding to a single root cost vector $(c_1^*, c_2^*, \ldots, c_k^*)$. This root has k children $(c_1^*+1, c_2^*, \ldots, c_k^*), (c_1^*, c_2^*+1, \ldots, c_k^*), \ldots, (c_1^*, c_2^*, \ldots, c_k^*+1)$ all of cost $C = C^* + 1$. Searching this ICT breadth-first corresponds to evaluating all possible cost vectors corresponding to increasing cost C starting with C^*, guaranteeing optimality of a found solution. When searching up to cost $C^* + \Delta$, $\mathcal{O}(k^\Delta)$ cost vectors are evaluated. Node evaluation corresponds to performing a low-level search for a solution $(\pi_1, \ldots, \pi_k)$ corresponding to the node cost vector $(c_1, \ldots, c_k)$.

3.1.2 Bottom-level search

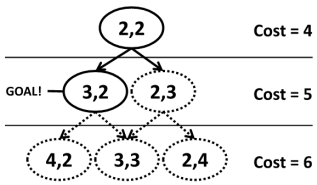
Explicitly generating all paths to the goal for agent a_i of cost c_i is expensive: the number of paths is exponential in c_i. This is why in the low-level search of ICTS, multi-valued decision diagrams (MDDs) are used to compactly represent all paths per agent. For agent a_i with a target cost c_i (from the cost vector), $MDD_i^{c_i}$ can be generated by a breadth-first search on the c_i-steps time-expanded graph starting at s_i, followed by a process similar to the standard path reconstruction method used in A* but allowing multiple node parents. Storing the resulting paths in an MDD, which has at most $|V|$ nodes at each timestep or depth, avoids the cost of storing all paths explicitly and facilitates the efficient search of the k-agent space of path combinations [6].

In Figure 4, a problem with its corresponding ICT is shown, as well as the low-level search of the (solution) node $(3, 2)$ using MDDs. The second agent staying at node D after completing the path is also modeled in its MDD to ensure that the node remains blocked.

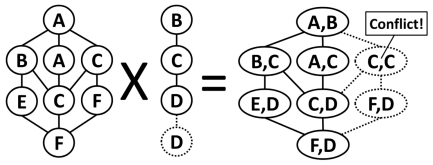
Figure 4c illustrates how two or more MDDs can be combined: first, the root nodes are joined; next, the product of the children of both roots is taken and checked for conflicts. In 4c, a vertex conflict occurs and therefore this joint child can be removed from the MDD. If



(a) An example problem.



(b) The corresponding ICT with solution node $(3, 2)$.



(c) MDDs for the agents are found and combined.

Figure 4: Example of using ICTS to solve MAPF from [6].

a (B, A) node would be generated by the same root, (A, B), this would be an edge conflict and hence (B, A) would also be removed from the joint MDD $MDD_{1,2}^{c_1,c_2}$. In the next layer, the process repeats with multiple parent nodes.

In practice, joint MDDs are not explicitly constructed but instead, a depth-first search is used to search joint MDDs. This has the benefit of finding a path to the bottom of the joint MDD fast if one exists, while not constructing more of the joint MDD than is necessary for finding a solution, which can get large.

3.1.3 ICTS with Independence Detection

Like other MAPF algorithms, ICTS can be embedded in the Independence Detection (ID) framework introduced by Standley [16]. ID starts by assuming each agent is a group that can be assigned an optimal path independent of other groups. If group paths do conflict, they become one group that is planned together using a MAPF algorithm, in this case ICTS, until no conflicts between groups occur. The goal of ID is to minimize k', the size of the largest group that has to be planned. Optionally, a conflict avoidance table (CAT) can be maintained for use during the planning of groups. This records the paths planned by other groups and can facilitate tie-breaking in search that minimizes the number of conflicts; the CAT is omitted from Algorithm 1 for brevity but was implemented and used in experiments.

Algorithm 1 Simple Independence Detection [16]

```
1: Assign each agent to a singleton group
2: plan a path for each group
3: repeat
4:     simulate execution of all paths until a conflict occurs
5:     merge two conflicting groups into a single group
6:     cooperatively plan new group
7: until no conflicts occur
8: solution ← paths of all groups combined
9: return solution
```

3.1.4 Pruning in ICTS

In addition to ID, three pruning methods can be used to speed up ICTS [6], two of which are used in this work. These methods rely on the fact that if no solution exists for any combination of j out of k agents with $j < k$, no solution exists for all k agents. If any such combination is found for a node, there is no need to search the k-agent MDD so the next ICT node can be considered. The three pruning methods are as follows.

1. Simple pruning: for all combinations of j out of k agents, the j-agent MDD is searched using DFS for a solution.

2. Enhanced pruning: for all combinations of j out of k agents, the j-agent MDD is searched using BFS and when children are generated, for each agent, all children that do not appear in any unified node are removed from the MDD. In 4c, the C node of MDD_1 would be removed as it cannot be unified with any node of MDD_2 at that level.

3. Repeated enhanced pruning: the enhanced pruning procedure is repeated until a fixed point is reached, which is when no child node is removed from any of the agent MDDs in an iteration. This was not used in this work.

The full ICTS algorithm including a pruning step as described above is given by Algorithm 2. In this work, triple-pruning ($j = 3$) was used and enhanced triple-pruning was taken as a baseline, as this often outperforms other variants [6].

Algorithm 2 Increasing Cost Tree Search

```
1: procedure ICT-SEARCH(MAPF instance)
2:     Construct ICT root
3:     for all ICT nodes (c_1, ..., c_k) in breadth-first order do
4:         for all agents a_i do Construct MDD_i^{c_i}
5:         for all j-agent combinations (i_1, ..., i_j) do
6:             Run node-pruning procedure with MDDs MDD_{i_1}^{c_{i_1}}, ..., MDD_{i_j}^{c_{i_j}}
7:             if no solution was found for a combination in pruning then
8:                 Break and continue with next ICT node
9:         Search k-agent MDD space
10:        if solution was found then
11:            return solution
```

3.2 Exhaustive ICTS

For K teams $t_1, \ldots, t_K$ of size k_i, there are in total $M = \prod_i k_i!$ matchings from team agents to team goals. These matchings can be enumerated by representing each permutation p_i of goals for each team t_i as an integer $n \in \{1, \ldots, k_i!\}$. Using this representation, each $e \in \{1, \ldots, k_1!\} \times \cdots \times \{1, \ldots, k_K!\}$ corresponds to a unique combination of permutations and can be mapped to a matching represented as $m \in \{1, \ldots, M\}$. Assuming some fixed ordering of agents within each team, each matching m can be translated to a MAPF instance with an optimal SIC cost C_m, with $C_m = \infty$ for m corresponding to an unfeasible MAPF instance. In order to find the optimal solution, m with lowest C_m has to be found, which requires all M matchings to be solved or found to be infeasible, meaning that $\mathcal{O}(M)$ MAPF instances have to be solved.

3.2.1 Bounded search

Once a solution is found for a matching m with cost C_m, this can be used in the next search as an (exclusive) upper bound B for the SIC of nodes that have to be searched. Formally, let $B_0 = \infty$ represent the initial SIC bound of the ICT search. After solving each m with solution cost C_m, let $B_{m+1} = \min(B_m, C_m)$. By successively updating B in this manner, as more matchings are solved, the effort needed to solve the remaining matchings can be expected to be reduced significantly. With C_m denoting the optimal SIC for matching m, this effort can be expressed as

$$\Delta_{m,\text{bounded}} = \min(B_m, C_m) - C_m^* = B_{m+1} - C_m^*$$

This is proportional to the effort due to the complexity of the ICTS search for matching m, $\mathcal{O}(k^{\Delta_{m,\text{bounded}}})$. If $B_m \leq C_m^*$, only the root node will be generated to determine

this so $\Delta_{m,\text{bounded}} = 0$. Using this, the number of ICT nodes searched in total becomes $\mathcal{O}(\sum_m k^{\Delta_{m,\text{bounded}}})$.

3.2.2 Ordered enumeration

To reduce the total effort, the matchings m can instead be enumerated and solved in increasing order of C^*_m. In this case, the effort per matching is monotonically non-increasing by Property 1. This induces a case in which this enumeration minimizes the number of nodes searched in total (Property 2). However, cases in which Δ_m of the first m's searched is much higher could also be constructed, in which case this approach might cause more nodes to be processed in total than alternative methods. Therefore, this ordered enumeration remains a heuristic improvement that works well in practice, as shown in Section 4.

Property 1 (Monotonicity)**.** *In bounded exhaustive search using ICTS, solving the matchings in increasing order of root SIC makes the effort $\Delta_{m,bounded}$ monotonically non-increasing as a function of m.*

Proof. Due to the order of matchings for successive matchings $m, m+t$ for some $t \geq 1$, $C^*_m \leq C^*_{m+t}$. By definition, $B_{m+2} = \min(B_{m+1}, C_{m+1})$ meaning that $B_{m+2} \leq B_{m+1}$ so transitively $B_{m+t+1} \leq B_{m+1}$. Therefore

$$B_{m+t+1} - C^*_{m+t} \leq B_{m+1} - C^*_m \implies \Delta_{m+t,\text{bounded}} \leq \Delta_{m,\text{bounded}}$$

□

Property 2 (Conditional optimality)**.** *Assuming that Δ_m, the unbounded effort, is monotonically non-decreasing as a function of m (with m in increasing order of C^*_m), ordered enumeration minimizes the total number of nodes searched across matchings.*

3.2.3 Approximate ordered enumeration

In the implementation of exhaustive ICTS, matchings are enumerated by taking the product of goal permutation indices as described above so they are not enumerated in increasing order of root SIC. To achieve this ordering, matchings would have to be exhaustively enumerated and stored in memory before being sorted. Storing all matchings requires $\mathcal{O}(M)$ memory which was found to be more problematic in practice than the time complexity of the sorting, $\mathcal{O}(M \log(M))$. This is why in implementation, ordered enumeration is approximated using a priority queue that maintains a subset of matchings of a fixed size M_{pq}.[1] Algorithm 3 outlines the procedure.

3.3 ICTS-m

ICTS with Matching (ICTS-m) generalizes ICTS to optimally solve MAPFM instances. Few changes to ICTS were necessary to accommodate for matching in this increasing cost tree scheme. Specifically, the ICT root generation and the MDD generation as described in Subsection 3.1 had to be changed.

[1] $M_{pq} = 10000$ was used in implementation but this could be much increased at the cost of memory.

Algorithm 3 Approximate ordered exhaustive ICTS

```
Fill PQ with min(M, M_pq) matchings
B ← ∞
solution ← none
repeat
    m ← Take matching with lowest SIC from PQ
    solution_m with cost C_m ←Solve m with upper bound B
    if C_m < B then
        B ← C_m
        solution ← solution_m
    B ← min(B, C_m)
    if there is an unsolved matching m_new not in PQ then
        Compute root SIC and insert m_new into PQ
until PQ is empty return solution
```

3.3.1 Root generation for matching

Given a team t_i of size k_i, for any agent a_j^i there are k_i costs $c_j(1), c_j(2), \ldots, c_j(k_i)$ for shortest paths from s_j^i to each goal g_k^i. Making no assumptions about what goals other agents in t_i are assigned to, it could be that in the optimal solution a_j^i is assigned to a goal g_m^i with path cost $c_j^* = c_j(m)$ such that $c_j(m) = \min_{n \in \{1,\ldots,k_i\}} c_j(n)$. Therefore, to guarantee optimality, the minimal shortest path cost to any matching goal has to take the place of the shortest path cost from the original ICTS root. Intuitively, the sum of individual costs of this root is highly optimistic, particularly if the teams are large. This is because often, agents will not move to their closest matching goal in an optimal solution, even without considering conflicts (see 3.3.3).

3.3.2 MDD generation for matching

To generate MDDs for MAPFM, each agent has to consider not one goal but all matching goals. To do this, the MDD construction following the breadth-first search has to be changed: each goal becomes an endpoint from which paths to the start node are traced, resulting in an MDD with one start node and one or more terminal nodes.

3.3.3 Pathological case

As a consequence of the min operation in the root definition, a pathological case can be derived. Consider a team of 4 agents a_1^1 through a_4^1. Let them be arranged in a configuration as depicted in Figure 5. All four agents match the color of the center goal and hence for each i, $c_i^* = 1$, so that the root is $(1, 1, 1, 1)$ with SIC 4. The optimal solution however is $(1, 3, 3, 3)$ with SIC 10 numbering the agents clockwise, meaning that $\Delta = 6$. It can be seen that instances of this form with arbitrarily large Δ can be constructed. As ICTS-m being a generalization of ICTS is $\mathcal{O}(k^\Delta)$, even just extending the limbs of the instance to yield an ICT root of $(1, 9, 9, 9)$ with $\Delta = 24$ will result in an unfathomably large number of nodes being evaluated assuming no pruning or other modifications to the base algorithm.

3.4 Optimizations

To improve both types of algorithms, some optimizations were found that can be used to speed up the search.

3.4.1 Bounds in ID

The information gained about solution costs of independent groups can be used to establish a lower bound on solution SIC as per Property 3. In bounded search, the upper bound can be further constrained by Property 4.

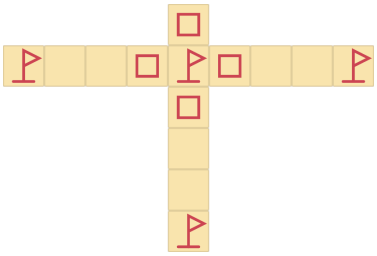


Figure 5: Pathological case for ICTS-m. The squares represent agents and the flags represent goals.

Property 3 (Lower bound)**.** *When merging groups g_i, g_j with solution costs C_i, C_j to yield group g_k, for any solution for this group with cost C_k*

$$C_k \geq C_i + C_j$$

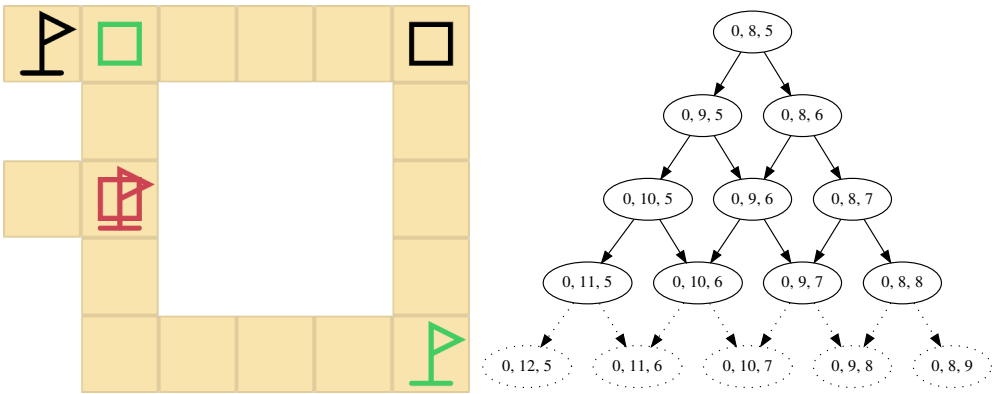
Proof. Assume there would be a solution with $C_k < C_i + C_j$. Taking C_k^i to denote the SIC of agents that used to make up g_i in the solution to g_k, without loss of generality $C_k^i < C_i$, which contradicts the optimality of the solution with cost C_i for g_i independent of g_j. □

Property 4 (Reduced upper bound)**.** *In bounded exhaustive search, the sum of costs of the groups not being merged can be subtracted from bound B without losing optimality.*

3.4.2 Child pruning

In the original ICTS formulation all k-children are generated in every case because pruning only controls skipping the low-level search, assumed to be the most expensive operation. However, in simple pruning, when there is a conflict between two or more agents at a given ICT node $(c_1, \ldots, c_k)$, in any solution, one or more of the path costs corresponding to these conflicting agents will have to be incremented. This can be exploited by pruning the node's children that do not correspond to the conflicting agents, of which there are often just 2 or 3, drastically improving the (average) branching factor as k increases. By Property 5, child pruning is not compatible with enhanced pruning.

Property 5 (Incompatibility with enhanced pruning)**.** *Enhanced pruning combined with child pruning is suboptimal due to the cascading effect described in [6].*



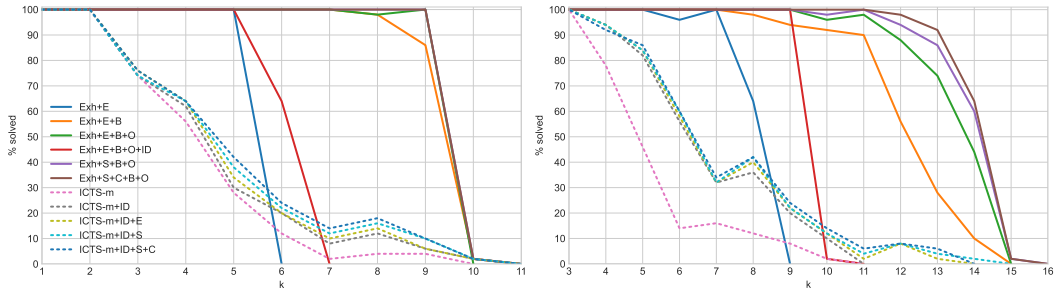
(a) Counter-example instance. (b) Pruned ICT that is traversed.

Figure 6: Counter-example for enhanced pruning combined with child pruning. If the black and green agent pair is considered last, an ICT is traversed while solving the instance that does not contain an optimal solution.

Proof. Consider Figure 6a with a red agent r, green agent g and black agent b. It can be seen that $(3, 8, 5)$ with SIC 16 is an optimal solution for this instance, which corresponds to r moving aside, allowing g to pass. Say enhanced pruning with pairs is applied to these three agents at $(c_r^*, c_g^*, c_b^*) = (0, 8, 5)$. Let the enhanced pruning order be (r, g), (r, b), (g, b) without loss of generality. (r, g) is solvable but will leave for g out of the two paths of length 8 only the clock-wise path. (r, b) will not change either MDD in this node. When (g, b) is evaluated, it is found that the remaining clockwise path for g and b's path are incompatible (edge conflict) so $(0, 9, 5)$ and $(0, 8, 6)$ are generated. By repeating this order of enhanced pruning with child pruning, the ICT shown in Figure 6b is traversed. None of the nodes with SIC 16 in this ICT is a solution, meaning that enhanced pruning combined with child pruning will return a suboptimal solution to the instance depicted in Figure 6a, making this pruning strategy suboptimal. □

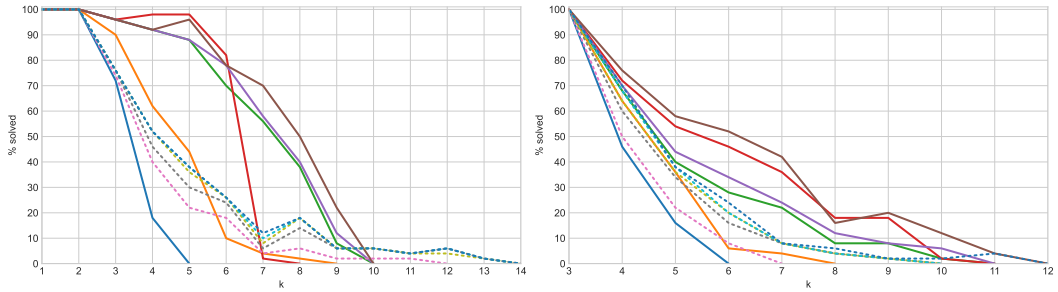
4 Experimental Results

To compare the algorithms and variants described above with each other and with alternative MAPFM algorithms, two types of experiments were performed. In the first, ICTS-based algorithms discussed in this work are compared; in the second, the approximate ordered enumeration variant is compared to other algorithms for MAPFM [20]–[23]. The data for the second experiment originates in [20]. In both experiments, there are three variables: the number of agents, the fraction of blocked grid tiles and the number of teams. The values measured are the fraction of instances solved and the average run time per instance. The recorded run times converge to the 120s timeout used as fewer instances are solved; therefore, while the times are included for completeness at the end of this work (Figure 9 and 10 for experiments 1 and 2), only the fraction of instances solved is used in the analysis of results.



(a) 25% wall, one team.

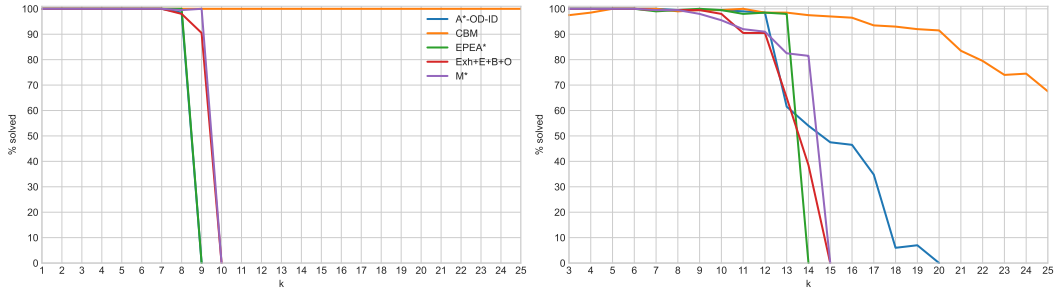
(b) 25% wall, three teams.



(c) 75% wall, one team.

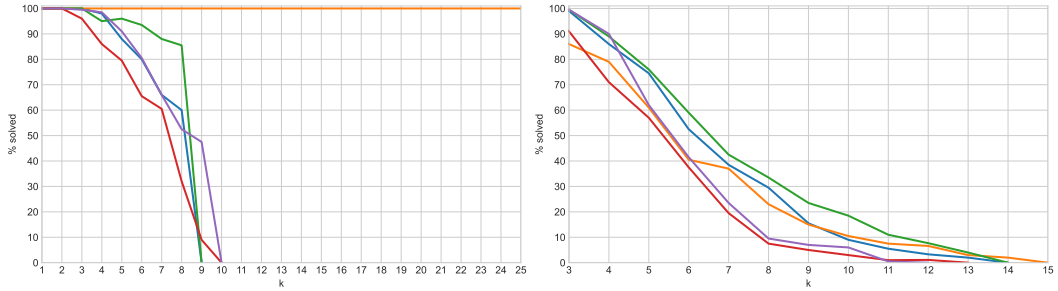
(d) 75% wall, three teams.

Figure 7: Fraction solved out of 50 instances by different ICTS-based algorithms.



(a) 25% wall, one team.

(b) 25% wall, three teams.



(c) 75% wall, one team.

(d) 75% wall, three teams.

Figure 8: Fraction solved out of 200 instances by MAPFM algorithms.

4.1 Experimental setup

Because MAPFM lies on the fringe of pathfinding research, no standard benchmark exists. For MAPF, the Moving AI maps [14] are often used and potentially a new standard benchmark for MAPFM could be defined based on these maps, similar to an existing standard benchmark for MAPF [5]; however, it was noticed that optimally solving even small MAPFM instances with comparatively few agents and short distances to goals was already a challenge computationally. This observation informed the creation of a map generator by the author of [23] with various parameters.

Using this map generator, two types of 20x20 grids similar to the random maps in [14] were generated:

1. 25% wall: a grid with 25% of tiles being obstacles.
2. 75% wall: a grid with 75% of tiles being obstacles.

For each of these types, instances were generated with between 1 and 25 agents either all in the same team or evenly partitioned into three teams. For each combination of map-type, number of teams and number of agents, 200 instances were generated. In scenarios with three teams, instances with $k < 3$ were not used. When solving the instances, for each combination of map-type and number of teams, the number of agents was increased until 0% of instances could be solved within a timeout of 120s, at which point no more data was recorded for greater numbers of agents. This approach is similar to the method described in [5].

The first experiment was performed on a 2.2GHz Intel i7-8750H hexacore with 16GiB of memory. For each setting, a fixed subset of 50 out of 200 instances was used.

The second experiment comparing different MAPFM algorithms was run on a server with a virtualized Intel Xeon E5-2683 dodecacore operating at 2.0GHz with 8GiB of RAM. All 200 instances were used for each setting.

All algorithms are implemented in Python 3.9 to avoid performance differences as often seen between interpreted versus compiled languages; however, CBM [21] in experiment 2 does use an SSP solver implemented in C++ as a subroutine. The comparison in this experiment is more coarse-grained as performance differences are sometimes subtle and could potentially be explained by slight implementation optimizations rather than algorithmic properties.

4.2 Experiment 1: ICTS-based MAPFM algorithms

To investigate the relative performance of various algorithms discussed in Section 3, a selection of both exhaustive ICTS and ICTS-m variants was made. Their performance in terms of fraction of problems solved within a timeout of 120s is shown in Figure 7. The meaning of the abbreviations used in the legend are as follows:

- S: Simple pruning
- E: Enhanced pruning
- C: Child pruning
- ID: Independence detection
- B: Bounded search
- O: Approximate ordered enumeration

Across scenarios, it is seen that typically exhaustive methods outperform ICTS-m algorithms (Figure 7), with performance being more similar in the maps with 75% wall. In particular on 75% wall maps with three teams (Figure 7d), the performance of ICTS-m+ID (and optional pruning) is seen to outperform bounded exhaustive search as k increases; additionally, some

ICTS-m variants manage to solve a small fraction of 75% maps with one team that exhaustive methods can not within the time-out (Figure 7c). This can be explained by the complexity of ICTS-m not being directly related to the number of matchings in contrast to the complexity of exhaustive ICTS.

Between exhaustive variants, with exception of the ID variant, a type of hierarchy can be distinguished, best illustrated by Figure 7b; almost without exception, Exh+S+C+B+O outperforms Exh+E+B+O which outperforms Exh+E+B which in turn outperforms Exh+E. Bounded search outperforms unbounded search as expected, given that the former does as much work as the latter and often less. Approximate ordering outperforming unordered enumeration is not as obvious from a theoretical standpoint, as discussed in 3.2.2, and simply indicates that ordered enumeration works well in practice. Simple pruning with child pruning outperforms enhanced pruning but this needs to be qualified. Comparing Exh+S+B+O with Exh+E+B+O shows that part of the difference can be explained by the use of simple pruning instead of enhanced pruning. Nonetheless, in instances with 75% wall (Figures 7c,7d) there appears to be a real benefit to searching a smaller ICT. Across ICTS-m variants, similar trends can be observed but the differences across pruning strategies are too slight to draw any conclusions from. As discussed in Section 5, more experiments are needed to adequately compare simple child pruning with other pruning methods for ICTS.

The benefit of using ID together with ordered enumeration is not clear. Ordered enumeration with ID performs much better in a setting with more obstacles and more teams (Figure 7d), but ordered enumeration without ID on the other hand is superior in an open setting with a single team (Figure 7a). The overhead associated with the various planning and merging steps (see Algorithm 1) could be the cause of the slowdown observed once M grows large (Figures 7a,7c); once the upper bound is tightened, ICTs only have to be searched shallowly so there is no point in going through the steps of ID. This explains why no speedup is seen, as between ICTS-m and ICTS-m+ID. Another factor is that if all agents belong to the same team, the benefit of using ID is reduced as agents from the same team planned independently often route to the same goal.

4.3 Experiment 2: Algorithms for MAPFM

From the variants discussed and compared above, the Exh+E+B+O variant was selected for comparison with various other algorithms for MAPFM. This results in the following set of algorithms whose performance is shown in Figure 8.

- ICTS with approximate sorted exhaustive matching (Exh+E+B+O)
- A*+ID+OD with matching ID and sorted exhaustive matching [23]
- EPEA* with matching ID and sorted exhaustive matching [22]
- M* with sorted exhaustive matching [20]
- CBM adapted to use SIC, using the successive shortest path algorithm (SSP) [24] as subroutine for solving teams [21].

With exception of CBM, all methods described use a variant of sorted exhaustive matching. This is because from each family of algorithms one of the best performing algorithms was chosen, which always was such an exhaustive method as opposed to a generalization of the base algorithm to MAPFM (ICTS-m in this work). A*+ID+OD and EPEA* also make use of matching ID, which is ID applied to the MAPFM instance using the respective exhaustive algorithm as MAPFM solver. The initial groups are here the teams, which is why in single team scenarios (Figures 8a,8c) matching ID has no effect. Hypothetically, matching ID

could explain why both A*+ID+OD and EPEA* outperform the methods using just sorted exhaustive matching in maps with 75% wall and three teams (Figure 8d).

CBM solving all problems with one team (Figures 8a,8c) is at first surprising but follows directly from the definition of the CBM method. CBM solves single teams as anonymous MAPF instances in polynomial time, meaning that solving scenarios with a single team reduces to a polynomial-time procedure, namely SSP. Following the results reported in [12], it is expected that as the number of teams approaches k, CBM performance will deteriorate and thus become more similar to the results for exhaustive methods here reported.

5 Conclusions and Future work

The goal of this research was to seek methods for solving MAPFM based on ICTS. In this, the research has succeeded, but the scalability of the algorithms leaves something to be desired. Exhaustive methods, like those discussed in detail in Subsection 3.2, cannot be expected to scale well as teams grow larger due to their complexity, no matter what optimizations are put in place. Nonetheless, as was shown in Section 4, the more advanced exhaustive ICTS algorithms consistently outperform the ICTS-m based algorithms. Compared to the SIC-adaptation of the CBM method [21], it was seen that exhaustive methods perform relatively poorly if there are fewer teams and fewer obstacles, while being competitive once teams grow smaller and maps become more constrained. Delineating under which conditions exhaustive methods outperform CBM and vice versa is something for further research.

A key idea for future research to look into is how to exploit the lower bound on solution SIC that can be found using the Hungarian Algorithm [11] or similar methods like the JVC algorithm [25]. This lower bound could be used in a branch-and-bound [26] search of matchings. An algorithm for MAPF could be used at leaf nodes as in exhaustive ICTS; however, solving (small) branches as MAPFM instances using approaches other than branch-and-bound is also a direction to explore. To retain the benefits of ordered enumeration and to potentially allow for a higher degree of bounding, an optimized version [27] of Murty's algorithm [28] for ranking all matchings in increasing order of cost could be taken as a starting point for such a branch-and-bound approach.

Another possibility would be to use the lower bound on solution SIC to improve ICTS-m by only executing the k-agent low-level search for nodes that satisfy this bound. The lower bounding in ICTS-m could then be further improved by finding the optimal SIC for the anonymous MAPF instance corresponding to each team using an approach similar to those described in [21]. As long as there is a team that does not meet its bound, branching only on agents within this team is possible without losing optimality.

A related observation is that ICTS-m in its current form does not rely on the property of agents and goals having one color (team). It could easily be adapted to support one or more colors for each agent and goal while still guaranteeing optimality, in contrast to CBM [12] and its SIC adaptation [21] which specifically use this property to plan single teams in polynomial time. A multi-color generalization of exhaustive ICTS would require a different matching enumeration method. Future research could study the multi-color MAPFM generalization and the application of methods described in this work to that problem.

Lastly, more experimentation is needed to reach definite conclusions about how simple child pruning compares to existing pruning techniques for ICTS, which could be done by reproducing the ICTS pruning experiments reported in [6] extended with simple child pruning variants.

6 Reproducibility

The implementation of the ICTS-based algorithms described in this work is publicly available online in the form of a Github repository.[2] Included is also code for benchmarking different configurations of the ICTS solver and the instances that were used in the benchmarks reported above. The benchmarks and instances used in the relative comparison of MAPFM algorithms are also available in the code and data repository of [20].[3]

References

[1] N. Sturtevant and R. Geisberger, "A comparison of high-level approaches for speeding up pathfinding", *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 6, no. 1, pp. 76–82, Oct. 2010.

[2] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols", in *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, ser. MobiCom '98, New York, NY, USA: Association for Computing Machinery, 1998, pp. 85–97, ISBN: 158113035X. DOI: 10.1145/288235.288256.

[3] R. Morris, C. Pasareanu, K. Luckow, W. Malik, H. Ma, T. K. S. Kumar, and S. Koenig, "Planning, scheduling and monitoring for airport surface operations", in *30th AAAI Conference on Artificial Intelligence*, 2016, pp. 608–614.

[4] J. Mulderij, B. Huisman, D. Tönissen, K. van der Linden, and M. M. de Weerdt, *Train unit shunting and servicing: A real-life application of multi-agent path finding*, 2020. arXiv: 2006.10422 [cs.MA].

[5] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Barták, "Multi-agent pathfinding: Definitions, variants, and benchmarks", *Symposium on Combinatorial Search (SoCS)*, pp. 151–158, 2019.

[6] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding", *Artificial Intelligence*, vol. 195, pp. 470–495, 2013.

[7] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding", *Artificial Intelligence*, vol. 219, pp. 40–66, 2015, ISSN: 0004-3702. DOI: 10.1016/j.artint.2014.11.006.

[8] E. Lam, P. Le Bodic, D. D. Harabor, and P. J. Stuckey, "Branch-and-cut-and-price for multi-agent pathfinding", in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2019, pp. 1289–1296. DOI: 10.24963/ijcai.2019/179.

[9] H. Ma, S. Koenig, N. Ayanian, L. Cohen, W. Hönig, T. K. S. Kumar, T. Uras, H. Xu, C. Tovey, and G. Sharon, "Overview: Generalizations of multi-agent path finding to real-world scenarios", in *IJCAI-16 Workshop on Multi-Agent Path Finding*, 2016.

[2] https://github.com/tbvanderwoude/icts-m

[3] https://github.com/jonay2000/research-project

[10] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses", in *Proceedings of the 19th National Conference on Innovative Applications of Artificial Intelligence - Volume 2*, ser. IAAI'07, AAAI Press, 2007, pp. 1752–1759, ISBN: 9781577353232.

[11] H. W. Kuhn, "The Hungarian method for the assignment problem", *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955. DOI: 10.1002/nav.3800020109.

[12] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents", in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, ser. AAMAS '16, Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2016, pp. 1144–1152, ISBN: 9781450342391.

[13] J. Yu and S. M. LaValle, "Multi-agent path planning and network flow", in *Algorithmic Foundations of Robotics X*, E. Frazzoli, T. Lozano-Perez, N. Roy, and D. Rus, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 157–173, ISBN: 978-3-642-36279-8.

[14] N. Sturtevant, "Benchmarks for grid-based pathfinding", *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144–148, 2012.

[15] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds", in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 3260–3267. DOI: 10.1109/IROS.2011.6095022.

[16] T. Standley, "Finding optimal solutions to cooperative pathfinding problems", in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 24, 2010, pp. 173–178.

[17] M. Goldenberg, A. Felner, R. Stern, G. Sharon, N. R. Sturtevant, R. Holte, and J. Schaeffer, "Enhanced partial expansion A*", *Journal of Artificial Intelligence Research*, vol. 50, pp. 141–187, 2014.

[18] D. Harabor and A. Botea, "Breaking path symmetries on 4-connected grid maps", in *Proceedings of the 6th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, ser. AIIDE 2010, AAAI Press, 2010, pp. 33–38.

[19] D. Harabor and A. Grastien, "Online graph pruning for pathfinding on grid maps", in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, ser. AAAI'11, AAAI Press, 2011, pp. 1114–1119.

[20] J. Dönszelmann, "Matching in multi-agent pathfinding using M*", Bachelor's thesis, Delft University of Technology, 2021.

[21] R. Baauw, "Adapting CBM to optimize the sum of costs", Bachelor's thesis, Delft University of Technology, 2021.

[22] J. de Jong, "Multi-agent pathfinding with matching using Enhanced Partial Expansion A*", Bachelor's thesis, Delft University of Technology, 2021.

[23] I. de Bruin, "Solving multi-agent pathfinding with matching using A*+ID+OD", Bachelor's thesis, Delft University of Technology, 2021.

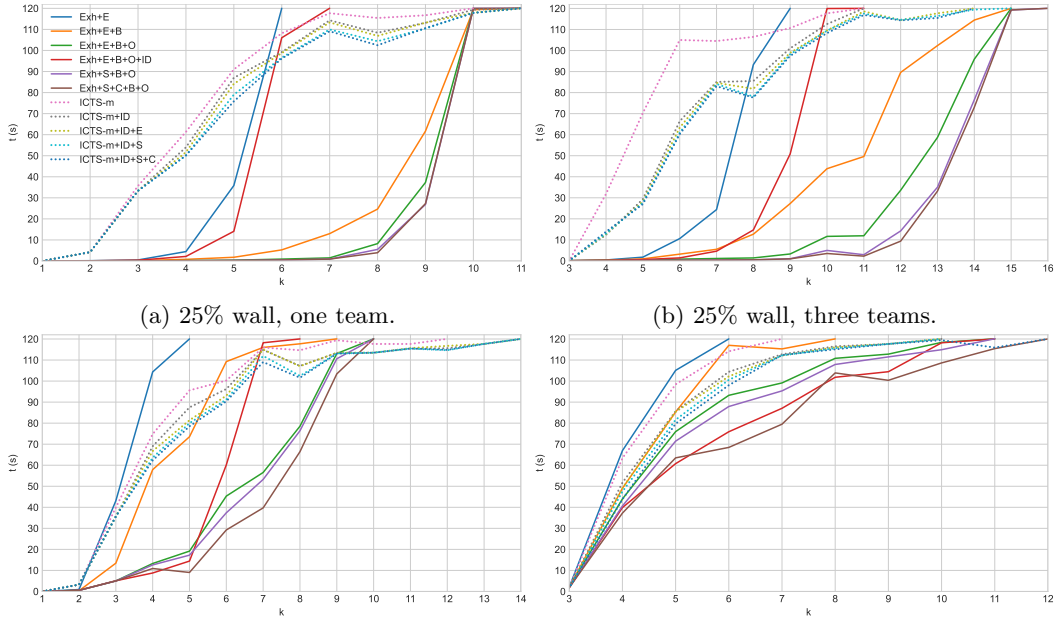
[24] A. Goldberg and R. Tarjan, "Solving minimum-cost flow problems by successive approximation", in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '87, New York, NY, USA: Association for Computing Machinery, 1987, pp. 7–18, ISBN: 0897912217. DOI: 10.1145/28395.28397.

[25] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems", *Computing*, vol. 38, no. 4, pp. 325–340, 1987.

[26] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems", *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960, ISSN: 00129682, 14680262.

[27] M. Miller, H. Stone, and I. Cox, "Optimizing Murty's ranked assignment method", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 33, no. 3, pp. 851–862, 1997. DOI: 10.1109/7.599256.

[28] K. G. Murty, "An algorithm for ranking all the assignments in order of increasing cost", *Operations Research*, vol. 16, no. 3, pp. 682–687, 1968, ISSN: 0030364X, 15265463.



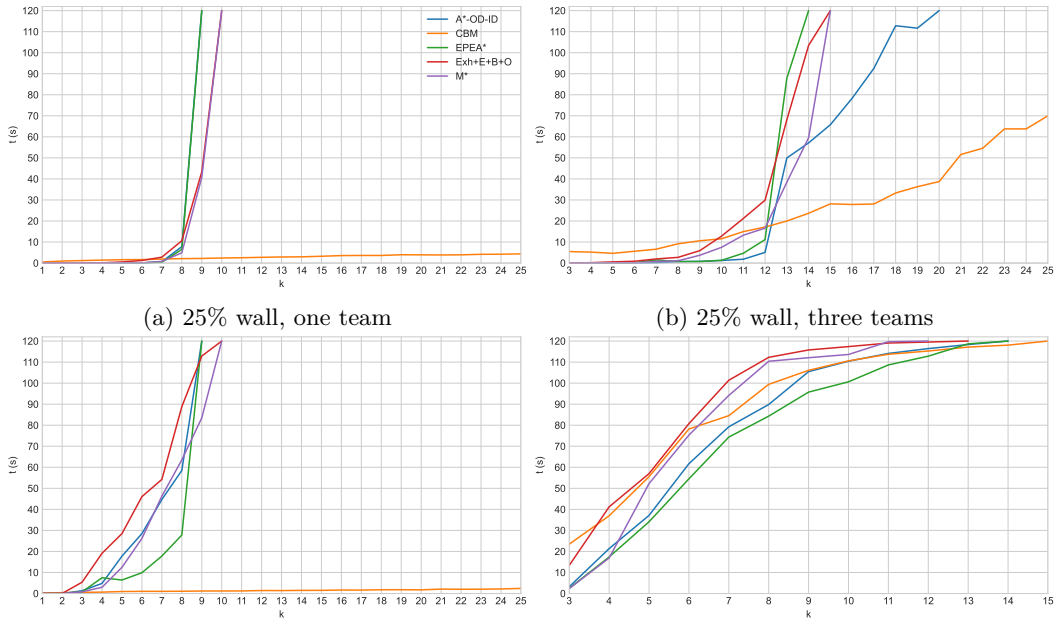
(a) 25% wall, one team.

(b) 25% wall, three teams.

(c) 75% wall, one team.

(d) 75% wall, three teams.

Figure 9: Average solving time of different ICTS-based algorithms with a timeout of 120s.



(a) 25% wall, one team

(b) 25% wall, three teams

(c) 75% wall, one team

(d) 75% wall, three teams

Figure 10: Average solving time of MAPFM algorithms with a timeout of 120s.