Multi-Agent Path Finding with Deadlines [*]

Hang Ma[1], **Glenn Wagner**[2], **Ariel Felner**[3], **Jiaoyang Li**[1], **T. K. Satish Kumar**[1], **Sven Koenig**[1]
[1] University of Southern California, {hangma,jiaoyanl,skoenig}@usc.edu, tkskwork@gmail.com
[2] CSIRO, glenn.wagner@data61.csiro.au
[3] Ben-Gurion University, felner@bgu.ac.il

Abstract

We formalize Multi-Agent Path Finding with Deadlines (MAPF-DL). The objective is to maximize the number of agents that can reach their given goal vertices from their given start vertices within the deadline, without colliding with each other. We first show that MAPF-DL is NP-hard to solve optimally. We then present two classes of optimal algorithms, one based on a reduction of MAPF-DL to a flow problem and a subsequent compact integer linear programming formulation of the resulting reduced abstracted multi-commodity flow network and the other one based on novel combinatorial search algorithms. Our empirical results demonstrate that these MAPF-DL solvers scale well and each one dominates the other ones in different scenarios.

1 Introduction

In many robotics applications, for example, aircraft-towing vehicles [Morris *et al.*, 2016], warehouse and office robots [Wurman *et al.*, 2008; Veloso *et al.*, 2015], game characters [Ma *et al.*, 2017d], and other multi-robot systems [Ma *et al.*, 2017a], robots need to finish tasks that have deadlines. For example, in applications that require long-term autonomy for a team of robots, it is important for the team to move as many robots as possible from a dangerous area to reach a shelter area before a disaster occurs in inclement or adversarial conditions.

One aspect of the problem, namely Multi-Agent Path Finding (MAPF), is to plan collision-free paths for multiple agents in known environments from their given start vertices to their given goal vertices [Ma and Koenig, 2017]. The objective is to minimize the sum of the arrival times of the agents or the makespan. MAPF is NP-hard to solve optimally [Yu and LaValle, 2013b] and even to approximate within a small constant factor for makespan minimization [Ma *et al.*, 2016b]. It can be solved with reductions to other well-studied combinatorial problems [Surynek, 2015; Surynek *et al.*, 2016; Yu and LaValle, 2013a; Erdem *et al.*, 2013] and dedicated optimal [Standley and Korf, 2011; Goldenberg *et al.*, 2014; Sharon *et al.*, 2013; Wagner and Choset, 2015; Sharon *et al.*, 2015; Boyarski *et al.*, 2015; Felner *et al.*, 2018], bounded-suboptimal [Barer *et al.*, 2014; Cohen *et al.*, 2016], and suboptimal MAPF algorithms [Silver, 2005; Sturtevant and Buro, 2006; Wang and Botea, 2011; Luna and Bekris, 2011; de Wilde *et al.*, 2013], including combinatorial search algorithms, as described in several surveys [Ma *et al.*, 2016a; Felner *et al.*, 2017]. MAPF has recently been generalized in different directions [Ma and Koenig, 2016; Hönig *et al.*, 2016a; Ma *et al.*, 2016a; Hönig *et al.*, 2016b; Ma *et al.*, 2017b; 2017c] but none of them capture an important characteristic of many applications, namely the ability to meet deadlines. A MAPF variant, G-TAPF, assigns tasks with deadlines to agents but does not directly maximize the number of agents that can finish the tasks by the deadlines [Nguyen *et al.*, 2017].

We thus formalize Multi-Agent Path Finding with Deadlines (MAPF-DL). The objective is to maximize the number of agents that can reach their given goal vertices from their given start vertices within a given deadline, without colliding with each other. Since none of the existing results directly transfers to MAPF-DL, we first show that MAPF-DL is NP-hard to solve optimally. We then present two families of algorithms to solve MAPF-DL. The first family is based on a reduction of MAPF-DL to a flow problem and a subsequent compact integer linear programming formulation of the resulting reduced abstracted multi-commodity flow network. The second family is based on novel combinatorial search algorithms. We introduce three search-based MAPF-DL algorithms and conduct systematic experiments to compare them on a number of MAPF-DL instances. The results show that all algorithms scale well to large problem instances but each one dominates the other ones in different scenarios. We study their pros and cons and provide a set of guidelines for identifying when each one should be used.

[*]The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1724392, 1409987, and 1319966 as well as a gift from Amazon. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government. The research at Ben-Gurion University was supported by Israel Science Foundation under grant number 844/17 and the Israel Ministry of Science.

2 Multi-Agent Path Finding with Deadlines

In this section, we define MAPF-DL formally and prove its computational hardness. We then present an optimal MAPF-DL algorithm based on integer linear programming (ILP).

Problem Definition

We formalize MAPF-DL as follows: We are given a *deadline*, denoted by a time step T_{end}, a finite undirected graph $G = (V, E)$, and M agents $a_1, a_2 \dots a_M$. Each agent a_i has a start vertex s_i and a goal vertex g_i. In each time step, each agent either moves to an adjacent vertex or stays at the same vertex. Each agent can reach its goal vertex in T_{end} time steps in the absence of other agents (without loss of generality). Let $l_i(t)$ be the vertex occupied by agent a_i at time step $t \in \{0 \dots T_{end}\}$. We call an agent a_i *successful* iff it occupies its goal vertex at the deadline T_{end}, that is, $l_i(T_{end}) = g_i$. A *plan* consists of a path l_i assigned to each successful agent a_i that satisfies the following conditions: (1) $l_i(0) = s_i$ [each successful agent starts at its start vertex]. (2) $(l_i(t-1), l_i(t)) \in E$ or $l_i(t-1) = l_i(t)$ [each successful agent always either moves to an adjacent vertex or does not move]. Each unsuccessful agent a_i is removed at time step zero, and the plan thus contains no path assigned to it, that is, $l_i = \emptyset$.[1] We define a *collision* between two different successful agents a_i and a_j to be either a *vertex collision* (a_i, a_j, v, t) iff $v = l_i(t) = l_j(t)$ [both successful agents occupy the same vertex simultaneously] or an *edge collision* (a_i, a_j, u, v, t) iff $u = l_i(t) = l_j(t+1)$ and $v = l_j(t) = l_i(t+1)$ [both successful agents traverse the same edge simultaneously in opposite directions]. A solution is a plan without collisions.

The objective of MAPF-DL is to maximize the number of successful agents $M_{succ} = |\{a_i | l_i(T_{end}) = g_i\}|$, that is, the number of paths in the solution, or, equivalently, minimize the number of unsuccessful agents $M_{unsucc} = M - M_{succ}$. The cost of a plan is thus the number of unsuccessful agents M_{unsucc}. It can also be defined as the sum of the costs of all agents since a Boolean cost can be defined for each agent where each successful agent incurs cost 0 and each unsuccessful agent incurs cost 1. Obviously, every MAPF-DL instance has a trivial solution where all agents are unsuccessful, namely with cost $M_{unsucc} = M$.

Intractability

Theorem 1. *It is NP-hard to compute a MAPF-DL solution with the maximum number of successful agents.*

The proof of the theorem reduces the **$\leq$3,=3-SAT** problem [Tovey, 1984], an NP-complete version of the Boolean satisfiability problem, to MAPF-DL. The reduction is similar to the one used for proving the NP-hardness of approximating the optimal makespan for MAPF [Ma *et al.*, 2016b]. It constructs a MAPF-DL instance with deadline $T_{end} = 3$ that has a zero-cost solution iff the given **$\leq$3,=3-SAT** instance is satisfiable. Also see our preliminary work [Ma *et al.*, 2018].

[1]Depending on the application, the unsuccessful agents can be removed at time step zero, wait at their start vertices, or move out of the way of the successful agents. We choose the first option in this paper. If the unsuccessful agents are not removed, they can obstruct other agents. However, our proof of NP-hardness does not depend on this assumption, and our MAPF-DL algorithms can be adapted to other assumptions.

Algorithm 1: High Level of CBS-DL (and MA-DBS)

```
   Input: MAPF-DL instance
1  $Root.constraints \leftarrow \emptyset$
2  $Root.plan \leftarrow$ path for each agent found by a low-level search
3  $Root.cost \leftarrow 0$
4  OPEN $\leftarrow \{Root\}$
5  while $true$ do
6      $N \leftarrow \arg\min_{N' \in \text{OPEN}} N'.cost$
7      OPEN $\leftarrow$ OPEN $\setminus \{N\}$
8      Try to find a collision in $N.plan$
9      if $N.plan$ has no collision then
10         return $N.plan$
11     C $\leftarrow$ first vertex or edge collision $(a_i, a_j, \dots)$ in $N.plan$
12     // begin: below for MA-DBS only
13     if $shouldMerge(a_i, a_j)$ then
14         $a_{\{i,j\}} \leftarrow merge(a_i, a_j)$
15         Update $N.constraints$ (external constraints of $a_{\{i,j\}}$)
16         Update $N.plan$ by invoking a low-level search (with DBS) for $a_{\{i,j\}}$
17         $N.cost \leftarrow M_{unsucc}$ in $N.plan$
18         OPEN $\leftarrow$ OPEN $\cup \{N\}$
19         continue // go to [Line 5]
20     // end: above for MA-DBS only
21     foreach $a_i$ involved in $C$ do
22         $N' \leftarrow$ new node
23         $N'.plan \leftarrow N.plan$
24         $N'.constraints \leftarrow N.constraints \cup \{(a_i, \dots)\}$
25         Update $N'.plan$ by invoking a low-level search (with A* or DBS) for $a_i$
26         $N'.cost \leftarrow M_{unsucc}$ in $N'.plan$
27         OPEN $\leftarrow$ OPEN $\cup \{N'\}$
```

ILP-Based MAPF-DL Algorithm

Our ILP-based MAPF-DL algorithm first reduces MAPF-DL to the maximum (integer) multi-commodity flow problem, which is similar to the reductions of MAPF and a MAPF variant, TAPF, to multi-commodity flow problems [Yu and LaValle, 2013a; Ma and Koenig, 2016]. It then encodes the latter problem using a compact integer linear programming (ILP) formulation on a reduced abstracted multi-commodity flow network. See our preliminary work [Ma *et al.*, 2018] for more details on this algorithm.

3 Search-Based MAPF-DL Algorithms

In this section, we present a spectrum of optimal combinatorial search algorithms for solving MAPF-DL: Conflict-Based Search with Deadlines (CBS-DL), an adapted version of Conflict-Based Search (CBS) [Sharon *et al.*, 2015]; Death-Based Search (DBS), which reasons about sets of successful agents; and Meta-Agent DBS (MA-DBS), which incorporates the advantages of CBS-DL and DBS.

3.1 CBS-DL

(Standard) CBS is a two-level MAPF algorithm that minimizes the sum of the arrival times of all agents at their goal vertices. CBS-DL is an adaptation of CBS for MAPF-DL. Algorithm 1 shows its high-level search. Lines in red are used in MA-DBS (presented in Section 3.3) only. CBS-DL uses the same framework as CBS but uses M_{unsucc} as cost. On the high level, CBS-DL performs a best-first search to resolve collisions among the agents and thus builds a constraint tree (CT). Each CT node contains a set of constraints and a plan that obeys these constraints. CBS-DL always expands the CT node with the smallest cost M_{unsucc} of its plan. The root CT node has no constraints [Line 1]. CBS-DL performs a low-level search to find a path for each agent (without any constraints). The plan of the root CT node thus contains paths for all agents [Line 2], and its cost is zero [Line 3]. When CBS-DL expands a CT node N, it checks whether the CT node contains a plan that has no collisions [Line 8]. If this is the case, N is a goal node and CBS-DL terminates successfully [Line 10]. Otherwise, CBS-DL chooses a collision to resolve [Line 11] and generates two child nodes N_1 and N_2 that inherit all constraints and the plan from N [Line 21-23]. If the collision to resolve is a vertex collision (a_i, a_j, v, t), CBS-DL adds the vertex constraint (a_i, v, t) to N_1 to prohibit agent a_i from occupying v at time step t and similarly adds the vertex constraint (a_j, v, t) to N_2. If the collision to resolve is an edge collision (a_i, a_j, u, v, t), CBS-DL adds the edge constraint (a_i, u, v, t) to N_1 to prohibit agent a_i from moving from u to v at time step t and similarly adds the edge constraint (a_j, v, u, t) to N_2 [Line 24]. For each child CT node, say N_1, CBS-DL performs a low-level search for agent a_i to compute a new path from its start vertex to its goal vertex within deadline T_{end} that obeys the constraints of N_1 relevant to agent a_i and replaces the old path of agent a_i in N_1.*plan* with the new path returned by the low-level search (it deletes the old path if no path is returned) [Line 25]. CBS-DL thus updates the cost of N_1 accordingly and inserts it into OPEN [Lines 26-27].

On the low level, CBS-DL performs an A* search to find a path for the agent from its start vertex to its goal vertex by pruning all nodes with time step $> T_{end}$. If it finds a path from the start vertex to the goal vertex of length exactly T_{end} time steps that obeys the constraints imposed by the high level, it returns the path for the agent and cost 0. Otherwise, it returns no path and cost 1.

Theoretical Analysis

We now prove that CBS-DL is complete and optimal.

Lemma 1. *CBS-DL generates only finitely many CT nodes.*

Proof. The constraint added on Line 24 to a child CT node is different from the constraints of its parent CT node since the paths of its parent CT node do not obey it. The depth of the (binary) CT is finite because all paths are not longer than T_{end} and only finitely many different vertex and edge constraints exist. □

Lemma 2. *Whenever CBS-DL chooses a CT node on Line 6 and the plan of the node has no collisions, then CBS-DL terminates with a solution with finite cost.*

Proof. The cost of the CT node is M_{unsucc} of its plan, which is bounded by M. □

Lemma 3. *The plan of a CT node has the largest possible number of paths (one for each successful agent) that obey its constraints.*

Proof (by induction). The statement holds for the root CT node because its plan contains one path for each agent (since each agent can reach its goal vertex in T_{end} time steps in the absence of other agents). Assume that the statement holds for the parent CT node N of any child CT node N'. When CBS-DL updates the plan of N' on Line 25, it changes the path for one agent only, say agent a_i, by performing a low-level search with the constraints of N' (including the newly added constraint $\langle a_i, \dots \rangle$). Therefore, CBS-DL correctly updates the path for agent a_i, and the statement holds also for N' due to the induction assumption and the fact that N'.*plan* inherits the paths of all agents different from agent a_i from N.*plan* on Line 23. □

Lemma 4. *CBS-DL chooses CT nodes on Line 6 in non-decreasing order of their costs.*

Proof. CBS-DL performs a best-first search and the cost of a parent CT node N is at most the cost of any of its child CT nodes N' since N'.*plan* contains at most as many paths as N.*plan* contains because (1) the plan of a CT node contains the largest possible number of paths (one for each successful agent) that obey its constraints according to Lemma 3, and thus (2) the set of all plans that obey N'.*constraints* is a subset of the set of all plans that obey N.*constraints* (since N.*constraints* $\subset N'$.*constraints* due to Line 24). □

Lemma 5. *The cost of a CT node is at most the cost of any solution that obeys its constraints.*

Proof. The cost of the CT node is the cost M_{unsucc} of its plan, which in turn is the minimum among the costs of all plans that obey its constraints according to Lemma 3, which in turn is at most the cost of any solution that obeys its constraints since every solution that obeys its constraints is also a plan that obeys its constraints. □

Theorem 2. *CBS-DL is complete and optimal.*

Proof. A solution always exists, for example, where all agents are unsuccessful. Now assume that the cost of an optimal solution is x and, for a proof by contradiction, that CBS-DL does not terminate with a solution of cost x. Therefore, whenever CBS-DL chooses a CT node with cost x on Line 6, its plan has collisions (because otherwise CBS-DL would correctly terminate with a solution of cost x according to Lemma 2 since it chooses CT nodes on Line 6 in non-decreasing order of their costs according to Lemma 4). Pick an arbitrary optimal solution. A CT node whose constraints the optimal solution obeys has cost $\leq x$ according to Lemma 5. The root CT node is such a node since the optimal solution trivially obeys its (empty) constraints. Whenever CBS-DL chooses such a CT node on Line 6, its plan has collisions (as shown directly above since its cost is $\leq x$). CBS-DL thus generates the child CT nodes of this parent CT node, the constraints of at least one of which the optimal solution obeys and which CBS-DL thus inserts into OPEN with cost $\leq x$. Since CBS-DL chooses CT nodes on Line 6 in non-decreasing order of their costs according to Lemma 4, it chooses infinitely many CT nodes on Line 6 with costs $\leq x$, which contradicts Lemma 1. □

3.2 Death-Based Search

Death-Based Search (DBS) is also a two-level algorithm. Conceptually, instead of imposing vertex or edge constraints on agents, DBS marks individual agents as unsuccessful and then searches for the minimal set of unsuccessful agents necessary to produce a solution.

We define a group γ of agents to be *consistent* iff all agents in it can simultaneously be successful, that is, the sub-MAPF-DL instance with the agents in γ has a zero-cost solution (an empty group is consistent). This condition is verified by a special call to CBS-DL with deadline T_{end}, which reports that the condition holds if all agents in γ are successful or reports that the condition does not hold once CBS-DL expands a CT

Algorithm 2: High Level of DBS

```
    Input: MAPF-DL instance
1   $Root.live \leftarrow \{\{a_i\}|i = 1 \ldots M\}$
2   $Root.cost \leftarrow 0$
3   $\text{OPEN} \leftarrow \{Root\}$
4   while true do
5       $N \leftarrow \arg\min_{N' \in \text{OPEN}} N'.cost$
6       $\text{OPEN} \leftarrow \text{OPEN} \setminus \{N\}$
7       Check whether all groups in $N.live$ are consistent by calling CBS-DL
8       if all groups in $N.live$ are consistent then
9           if $|N.live| = 1$ then
10              return the zero-cost solution for the single group $\gamma$ in $N.live$
11          else
12              $N' \leftarrow$ new node
13              $\gamma \leftarrow \gamma_1 \cup \gamma_2$, where $\gamma_1$ and $\gamma_2$ are the smallest groups in $N.live$
14              $N'.live \leftarrow (N.live \setminus \{\gamma_1, \gamma_2\}) \cup \{\gamma\}$
15              $N'.cost \leftarrow N.cost$
16              $\text{OPEN} \leftarrow \text{OPEN} \cup \{N'\}$
17      else
18          $\gamma \leftarrow$ first group in $N.live$ that does not have a zero-cost solution
19          foreach $a_i \in \gamma$ do
20              $N' \leftarrow$ new node
21              $N'.live \leftarrow (N.live \setminus \{\gamma\}) \cup \{\gamma \setminus \{a_i\}\}$
22              $N'.cost \leftarrow N.cost + 1$
23              $\text{OPEN} \leftarrow \text{OPEN} \cup \{N'\}$
```

node with non-zero cost (that is, at least one agent in γ is not successful).

On the high level, DBS performs a best-first search on the *death tree* (DT). Each DT node N contains a set N*.live* of disjoint groups of live agents (agents that have not been declared unsuccessful) and a cost N*.cost* equal to the number of agents that have been declared unsuccessful. Algorithm 2 shows the high-level search of DBS. The root DT node contains a set of M groups of live agents, each group containing a single unique agent [Lines 1] and its cost is zero [Line 2]. DBS chooses the DT node N with the smallest cost N*.cost* and checks if all groups in its set N*.live* are consistent [Line 5-7]. If N*.live* contains a single consistent group γ, the DT node N is a goal node, and DBS returns the zero-cost solution for γ [Line 10]. If all (more than one) groups in N*.live* are consistent, DBS merges the two smallest groups γ_1 and γ_2 in N*.live* to form a new group γ and adds a child DT node whose set contains all the groups in N*.live* but replaces γ_1 and γ_2 with γ [Lines 12-16]. Otherwise, there is an inconsistent group γ in N*.live* [Line 18]. We know that at least one agent in γ must be declared unsuccessful, forcing a split. In this case, DBS adds $|\gamma|$ child nodes, one for each agent $a_i \in \gamma$, to DT, where each of these nodes declares its own unique agent $a_i \in \gamma$ unsuccessful, and its cost is thus one larger than that of its parent [Lines 20-23].

Other Versions of DBS

DBS could have started with a root DT node [Line 1] whose set contains only a single group of all M agents, which does not require merging groups of live agents but results in a larger branching factor for the root DT node. DBS could have chosen different groups to merge [Line 14], which might result in an inconsistent group of larger size. Whenever DBS splits a parent DT node [Lines 20-23], it could have generated child DT nodes whose sets contain only consistent additional groups (and thus possibly declare more than one additional agent unsuccessful for the child DT nodes), which requires a procedure that can determine all consistent subgroups of the (inconsistent) group γ of agents efficiently and might result in a larger branching factor. In this paper, we chose to present the version that is the easiest to understand and analyze.

Theoretical Analysis

We now prove that DBS is complete and optimal.

Lemma 6. *DBS generates only finitely many DT nodes.*

Proof. The branching factor of a DT node is bounded by M due to Line 19. Due to Lines 14 and 21, when we consider each DT node in a downward traversal of any branch of DT from the root DT node, its set contains either one less group (when merging two groups) or one less agent (when declaring an unsuccessful agent) than that of its parent CT node. Its set is thus different from the sets of all its ancestor DT nodes. Therefore, the depth of DT is also finite since there are finitely many possible sets of disjoints groups of the M agents. □

Lemma 7. *Whenever DBS chooses a DT node on Line 5 whose set contains one single consistent group of live agents, then DBS correctly terminates with a solution of finite cost.*

Proof. Its cost is the number of agents that have been declared unsuccessful, which is bounded by M. □

Lemma 8. *DBS chooses DT nodes on Line 5 in non-decreasing order of their costs.*

Proof. DBS performs a best-first search, and the cost of a parent DT node is at most the cost of any of its child DT nodes due to Lines 15 and 22. □

Theorem 3. *DBS is complete and optimal.*

Proof. A solution always exists, for example, where all agents are unsuccessful. Now assume that the cost of an optimal solution is x and, for a proof by contradiction, that DBS does not terminate with a solution of cost x. Therefore, whenever DBS chooses a DT node with cost x on Line 5, its set does not contain one single consistent group (because otherwise DBS would correctly terminate with a solution of cost x according to Lemma 7 since it chooses DT nodes on Line 5 in non-decreasing order of their costs according to Lemma 8). Pick an arbitrary optimal solution with the set γ_{unsucc} of x unsuccessful agents. Trivially, a DT node that has declared the agents in a subset of γ_{unsucc} unsuccessful has cost $\leq x$. The root DT node is such a node since it has not declared any agents unsuccessful. Whenever DBS chooses such a DT node on Line 5, its set does not contain one single consistent group (as shown directly above since its cost is $\leq x$). Its set thus contains (1) more than one consistent group or (2) an inconsistent group (in which case the DT node has declared the agents in a *strict* subset of γ_{unsucc} unsuccessful). In case (1), DBS thus generates the only child DT node of this parent DT node, which has declared the same agents unsuccessful as the parent DT node and which DBS thus inserts into OPEN with cost $\leq x$. In case (2), DBS thus generates the child DT nodes of this parent DT node, at least one of which has still declared the agents (including one additional agent) in a subset of γ_{unsucc} unsuccessful and which DBS thus inserts into OPEN with cost $\leq x$. Since DBS chooses DT nodes on Line 5 in non-decreasing order of their costs according to Lemma 8, it chooses infinitely many DT nodes on Line 5 with costs $\leq x$, which contradicts Lemma 6. □

3.3 Meta-Agent DBS

CBS may perform poorly when an environment contains many possible, but colliding, paths for the agents since the size of CT is exponential in the number of collisions resolved. On the other hand, DBS may perform poorly for MAPF-DL if the conflicting agents are not added to the same group early in the search. We thus combine the power of CBS for weakly coupled agents and the power of DBS for identifying unsuccessful agents in a tightly coupled subset of agents using the Meta-Agent CBS [Sharon *et al.*, 2015] framework, which results in a new optimal MAPF-DL algorithm, called Meta-Agent DBS (MA-DBS).

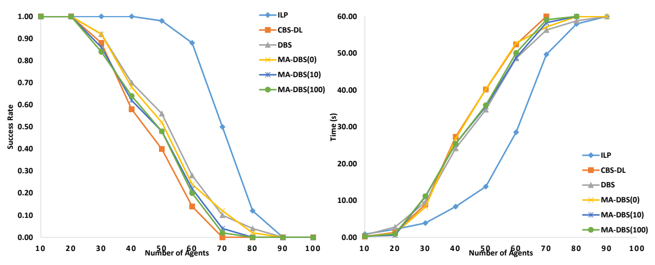
MA-DBS is a two-level algorithm: It uses the high-level search of CBS-DL on the high level and DBS on the low level. Algorithm 1 shows its high-level search. MA-DBS is similar to CBS-DL but also keeps track of the number of times collisions between every pair of (simple) agents that it has considered thus far during the search in a conflict matrix CM. Before MA-DBS expands a CT node N for the colliding agents a_i and a_j, if the number of collisions between the two agents exceeds a user-defined *merge threshold* B, MA-DBS merges them into a composite *meta agent* $a_{\{i,j\}}$. To do so, whenever MA-DBS considers a collision between (meta) agents a_i and a_j [Line 11], because two simple agents $a_k \in a_i$ and $a_{k'} \in a_j$ collide, it increases the value of $CM[\{k,k'\}]$ by one. Function *shouldMerge*(a_i, a_j) returns *true* iff $\sum_{a_k \in a_i, a_{k'} \in a_j} CM[\{k,k'\}] > B$ [Line 13]. Since DBS uses a low-level search that finds a plan for a meta agent without any internal collisions between all (simple) agents in the meta agent, it only needs to store external constraints resulting from (external) collisions between any two (simple) agents in different meta agents. Therefore, if MA-DBS decides to merge a_i and a_j into $a_{\{i,j\}}$, it updates the constraints N.*constraints* of the CT node N accordingly [Line 15]. It then calls DBS to find new paths (without internal collisions) for all agents in $a_{\{i,j\}}$ subject to the constraints in N.*constraints* relevant to $a_{\{i,j\}}$ (by solving a MAPF-DL instance with agents in $a_{\{i,j\}}$) and updates the plan N.*plan* and cost N.*cost* of the CT node N according to the new paths returned by DBS [Lines 16-17]. Then, instead of expanding N, MA-DBS inserts N back into OPEN [Line 18]. When MA-DBS generates a new child CT node, it also calls DBS to find an optimal solution for a meta agent that obeys the constraints of the child CT node [Line 25].

Theoretical Analysis

Lemmas 1 and 4 hold for MA-DBS without change. Since the low-level search of MA-DBS, namely DBS, returns the maximum number of paths for a meta agent that obey the constraints of a CT node, Lemma 3 also holds for MA-DBS because (1), when it updates the plan of a CT node on Line 16, the resulting plan contains the maximum number of paths for the new meta agent and the original paths of the other agents, and (2), when it updates the plan of a child CT node on Line 25, the resulting plan contains the maximum number of paths for the meta agent and inherits paths of other agents from the plan of the parent CT node, and thus the induction argument for the proof of Lemma 3 holds. Consequently, Lemma 4 also holds for MA-DBS.

Theorem 4. *MA-DBS is complete and optimal.*

Proof. MA-DBS only merges two agents into one agent on Line 14 but never splits any agent. Therefore, MA-DBS does the merge operation [Lines 14-19] finitely many times (bounded by M) for each CT node. The rest of the proof is the same as the proof of Theorem 2. □



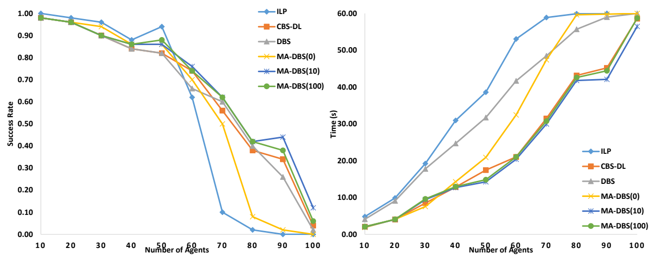
agents	#instances	ILP	CBS-DL	DBS	MA-DBS(0)	MA-DBS(10)	MA-DBS(100)
10	50	0.86	**0.24**	0.65	0.25	0.30	0.25
20	50	2.18	1.31	2.80	1.07	**0.58**	0.91
30	42	3.76	**1.57**	4.24	3.23	1.76	1.85
40	26	6.74	3.02	7.16	8.54	2.82	**2.35**
50	14	11.88	9.23	11.76	19.43	**4.82**	5.46
60	5	29.26	4.82	10.82	27.07	**4.77**	5.88

Figure 1: Success rates (top left), averaged running times over **all** instances (top right), and averaged running times over the instances **solved by all six algorithms** (bottom table) for different numbers of agents.

4 Experiments

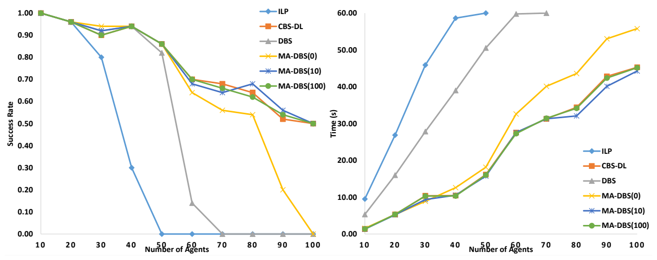
In this section, we describe our experimental results on a 2.50 GHz Intel Core i5-2450M laptop with 6 GB RAM. We tested six optimal MAPF-DL algorithms: the ILP-based algorithm, CBS-DL, DBS, and MA-DBS with merge thresholds 0, 10, and 100 (labeled as MA-DBS(0), MA-DBS(10), and MA-DBS(100), respectively). The ILP-based algorithm uses CPLEX V12.7.1 [IBM, 2011] as the ILP solver. We experimented on instances where the start and goal vertices of each agent are placed randomly so that the distance between them is close to the deadline. An instance becomes much easier to solve if this distance is much smaller than the deadline (since there is more leeway to plan a path for the agent). Specifically, we use three sets of randomly generated MAPF-DL instances with different numbers of agents (varied from 10 to 100 in increments of 10) labeled as SMALL, MEDIUM, and LARGE on 40×40, 80×80, and 120×120 4-neighbor 2-D grids with deadlines $T_{end} = 50$, 100, and 150, respectively. The cells in each grid are blocked independently at random with 20% probability each. We generate 50 MAPF-DL instances for each number of agents for each set. The start and goal vertices of each agent are randomly placed at distance 48, 49, or 50 for SMALL, 98, 99, or 100 for MEDIUM, and 148, 149, or 150 for LARGE. Each algorithm is given a time limit of 60 seconds to solve each instance. We did not run an algorithm for some number of agents if it solved none of the 50 instances for a smaller number of agents.

The 40×40 **SMALL domain** Figure 1 (top left) plots the success rates (numbers of instances solved within the time limit divided by 50) for all algorithms for SMALL. ILP has the highest success rates, and they start to drop only at 50 agents. The success rates for the search-based algorithms start to drop at 30 agents. DBS and MA-DBS(0) have the highest success rates among all search-based algorithms. Figure 1 (top right) plots the average running times over **all** 50 instances. 60 seconds are used for an instance that is not solved. Therefore, the



agents	#instances	ILP	CBS-DL	DBS	MA-DBS(0)	MA-DBS(10)	MA-DBS(100)
10	49	3.80	0.89	2.95	**0.69**	0.82	0.90
20	48	8.87	**1.71**	6.97	1.75	**1.71**	1.75
30	43	17.56	**2.65**	12.19	3.01	2.86	3.02
40	42	26.92	**3.84**	17.96	6.04	4.08	4.24
50	37	37.46	**5.92**	25.57	14.33	6.36	6.12
60	22	48.80	**7.26**	31.89	21.31	7.29	7.33
70	3	47.76	8.70	35.06	26.37	**7.74**	7.97

Figure 2: Results for MEDIUM.



agents	#instances	ILP	CBS-DL	DBS	MA-DBS(0)	MA-DBS(10)	MA-DBS(100)
10	50	9.52	1.38	5.31	1.55	**1.33**	1.36
20	48	25.49	3.04	14.15	2.98	**2.91**	3.03
30	38	42.43	4.84	24.88	5.67	4.86	**4.80**
40	15	55.44	7.20	37.23	8.61	6.88	**6.67**

Figure 3: Results for LARGE.

data points in the chart are lower bounds on the running times in those cases when not all instances are solved. ILP performs the best. Finally, the table in Figure 1 reports the average running times over those "easy" instances that are **solved by all six algorithms** (it also reports the numbers of those instances but does not show the rows where no instance is solved by all the algorithms). The best entry in each row is shown in bold. The search-based algorithms use less time to solve these "easy" instances than ILP. CBS-DL, MA-DBS(10), and MA-DBS(100) seem to use the least times and outperform ILP by up to a factor of 6. In some cases, the running times are smaller for larger numbers of agents because fewer (and "easier") instances are solved by all algorithms.

The 80×80 **MEDIUM domain** Figure 2 reports the same statistics for MEDIUM in the same format as reported for SMALL. Figure 2 (top left) plots the success rates. ILP has the highest success rates for small numbers (≤ 50) of agents but the lowest success rates for large numbers of agents. MA-DBS(10) seems to perform the best for large numbers of agents. Figure 2 (top right) plots the average running times over **all** 50 instances. ILP has the longest running times. MA-DBS(10) seems to perform the best in general. Finally, the table in Figure 2 reports the average running times over instances that are **solved by all six algorithms**. ILP performs the worst. CBS-DL seems to have the smallest running times and outperforms ILP by up to a factor of 7.

The 120×120 **LARGE domain** Figure 3 reports the same statistics for LARGE in the same format as reported for SMALL. Figure 3 (top left) plots the success rates. CBS-DL, MA-DBS(10), and MA-DBS(100) have the best success rates. ILP has the worst success rates. Figure 3 (top right) plots the average running times over **all** 50 instances. MA-DBS(10) seems to perform the best. ILP performs the worst. The table in Figure 3 reports the average running times over instances that are **solved by all six algorithms**. MA-DBS(10) and CBS-DL perform the best (very close to each other) and outperform ILP by up to a factor of 9.

Summary of Experimental Results For the same number of agents, SMALL has higher agent density, more tightly-coupled agents, and shorter planning horizons than MEDIUM and LARGE. ILP outperforms the search-based algorithms for SMALL because the size of the ILP formulation is small. When T_{end} increases, the size of the ILP formulation and the running time required to solve it increase significantly.

On the other hand, among all search-based algorithms, there seems to be a spectrum where DBS and CBS-DL sit at two extremes. DBS has higher success rates than CBS-DL for SMALL. CBS-DL has significantly higher success rates than DBS for MEDIUM and LARGE. CBS-DL uses much less times than DBS for instances that are solved by all algorithms. MA-DBS seems to balance between CBS-DL and DBS: (a) MA-DBS(0) is more similar to DBS because it merges agents into meta agents more frequently, which can result in a large meta agent containing many agents that need to be solved by DBS on the low level; and, on the other hand, (b) MA-DBS(10) and MA-DBS(100) are more similar to CBS-DL because they merge agents less frequently and their searches mostly remain in the CBS-DL framework.

5 Conclusions and Future Work

We formalized MAPF-DL, a new variant of MAPF. Theoretically, we proved that MAPF-DL is NP-hard to solve optimally. We presented two families of optimal MAPF-DL algorithms, one based on an ILP formulation and one based on combinatorial search techniques. Our experimental results show that each of them performs the best in different scenarios. We suggest the following future directions: (1) develop and compare new MAPF-DL algorithms, for example, A*-, ASP-, and SAT-based algorithms; (2) study important generalizations of MAPF-DL (for example, when agents have different priorities) more deeply; (3) study the combinatorial difference between MAPF-DL and MAPF; and (4) explore different merge criteria for MA-DBS.

References

[Barer *et al.*, 2014] M. Barer, G. Sharon, R. Stern, and A. Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *SoCS*, pages 19–27, 2014.

[Boyarski *et al.*, 2015] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and S. E. Shimony. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *IJCAI*, pages 740–746, 2015.

[Cohen *et al.*, 2016] L. Cohen, T. Uras, T. K. S. Kumar, H. Xu, N. Ayanian, and S. Koenig. Improved solvers for bounded-suboptimal multi-agent path finding. In *IJCAI*, pages 3067–3074, 2016.

[de Wilde *et al.*, 2013] B. de Wilde, A. W. ter Mors, and C. Witteveen. Push and rotate: Cooperative multi-agent path planning. In *AAMAS*, pages 87–94, 2013.

[Erdem *et al.*, 2013] E. Erdem, D. G. Kisa, U. Oztok, and P. Schueller. A general formal framework for pathfinding problems with multiple agents. In *AAAI*, pages 290–296, 2013.

[Felner *et al.*, 2017] A. Felner, R. Stern, S. E. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. R. Sturtevant, G. Wagner, and P. Surynek. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *SoCS*, pages 29–37, 2017.

[Felner *et al.*, 2018] A. Felner, J. Li, E. Boyarski, H. Ma, L. Cohen, T. K. S. Kumar, and S. Koenig. Adding heuristics to conflict-based search for multi-agent path finding. In *ICAPS*, 2018.

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

[Hönig *et al.*, 2016a] W. Hönig, T. K. S. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, and S. Koenig. Multi-agent path finding with kinematic constraints. In *ICAPS*, pages 477–485, 2016.

[Hönig *et al.*, 2016b] W. Hönig, T. K. S. Kumar, H. Ma, N. Ayanian, and S. Koenig. Formation change for robot groups in occluded environments. In *IROS*, pages 4836–4842, 2016.

[IBM, 2011] IBM. *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual*, 2011.

[Luna and Bekris, 2011] R. Luna and K. E. Bekris. Push and Swap: Fast cooperative path-finding with completeness guarantees. In *IJCAI*, pages 294–300, 2011.

[Ma and Koenig, 2016] H. Ma and S. Koenig. Optimal target assignment and path finding for teams of agents. In *AAMAS*, pages 1144–1152, 2016.

[Ma and Koenig, 2017] H. Ma and S. Koenig. AI buzzwords explained: Multi-agent path finding (MAPF). *AI Matters*, 3(3):15–19, 2017.

[Ma *et al.*, 2016a] 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.

[Ma *et al.*, 2016b] H. Ma, C. Tovey, G. Sharon, T. K. S. Kumar, and S. Koenig. Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In *AAAI*, pages 3166–3173, 2016.

[Ma *et al.*, 2017a] H. Ma, W. Hönig, L. Cohen, T. Uras, H. Xu, T. K. S. Kumar, N. Ayanian, and S. Koenig. Overview: A hierarchical framework for plan generation and execution in multi-robot systems. *IEEE Intelligent Systems*, 32(6):6–12, 2017.

[Ma *et al.*, 2017b] H. Ma, T. K. S. Kumar, and S. Koenig. Multi-agent path finding with delay probabilities. In *AAAI*, pages 3605–3612, 2017.

[Ma *et al.*, 2017c] H. Ma, J. Li, T. K. S. Kumar, and S. Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *AAMAS*, pages 837–845, 2017.

[Ma *et al.*, 2017d] H. Ma, J. Yang, L. Cohen, T. K. S. Kumar, and S. Koenig. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *AIIDE*, pages 270–272, 2017.

[Ma *et al.*, 2018] H. Ma, G. Wagner, A. Felner, J. Li, T. K. S. Kumar, and S. Koenig. Multi-agent path finding with deadlines: Preliminary results. In *AAMAS*, 2018.

[Morris *et al.*, 2016] R. Morris, C. Pasareanu, K. Luckow, W. Malik, H. Ma, S. Kumar, and S. Koenig. Planning, scheduling and monitoring for airport surface operations. In *AAAI-16 Workshop on Planning for Hybrid Systems*, 2016.

[Nguyen *et al.*, 2017] V. Nguyen, P. Obermeier, T. C. Son, T. Schaub, and W. Yeoh. Generalized target assignment and path finding using answer set programming. In *IJCAI*, pages 1216–1223, 2017.

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

[Sharon *et al.*, 2015] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[Silver, 2005] D. Silver. Cooperative pathfinding. In *AIIDE*, pages 117–122, 2005.

[Standley and Korf, 2011] T. S. Standley and R. E. Korf. Complete algorithms for cooperative pathfinding problems. In *IJCAI*, pages 668–673, 2011.

[Sturtevant and Buro, 2006] N. R. Sturtevant and M. Buro. Improving collaborative pathfinding using map abstraction. In *AIIDE*, pages 80–85, 2006.

[Surynek *et al.*, 2016] P. Surynek, A. Felner, R. Stern, and E. Boyarski. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *ECAI*, pages 810–818, 2016.

[Surynek, 2015] P. Surynek. Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally. In *IJCAI*, pages 1916–1922, 2015.

[Tovey, 1984] C. A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8:85–90, 1984.

[Veloso *et al.*, 2015] M. Veloso, J. Biswas, B. Coltin, and S. Rosenthal. CoBots: Robust symbiotic autonomous mobile service robots. In *IJCAI*, pages 4423–4429, 2015.

[Wagner and Choset, 2015] G. Wagner and H. Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

[Wang and Botea, 2011] K. Wang and A. Botea. MAPP: A scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research*, 42:55–90, 2011.

[Wurman *et al.*, 2008] P. R. Wurman, R. D'Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–20, 2008.

[Yu and LaValle, 2013a] J. Yu and S. M. LaValle. Planning optimal paths for multiple robots on graphs. In *ICRA*, pages 3612–3617, 2013.

[Yu and LaValle, 2013b] J. Yu and S. M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*, pages 1444–1449, 2013.