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

Nir Greshler, Ofir Gordon, Oren Salzman, and Nahum Shimkin

***Abstract*— We introduce the *Cooperative Multi-Agent Path Finding (Co-MAPF)* problem, an extension to the classical MAPF problem, where cooperative behavior is incorporated. In this setting, a group of autonomous agents operate in a shared environment and have to complete *cooperative tasks* while avoiding collisions with each other. This extension naturally models many real-world applications, where groups of agents must work together to complete a given task. To this end, we formalize the Co-MAPF problem and introduce *Cooperative Conflict-Based Search (Co-CBS)*, a CBS-based algorithm for solving the problem optimally for a wide set of Co-MAPF problems. Co-CBS uses a cooperation-planning module integrated into CBS such that cooperation planning is decoupled from path planning, while ensuring that paths obtained are optimal. Finally, we present empirical results on several MAPF benchmarks demonstrating our algorithm's properties.**

I. Introduction and Related Work

The Multi-Agent Path-Finding (MAPF) problem is a special and important type of the more general Multi-Agent Planning (MAP) problem [1]. In MAPF [2], the task is to find paths for each agent in a group, from a start to a goal location, where interactions between agents are restricted to collision avoidance, as agents move in a shared environment. While relevant to many real-world applications, such as warehouse automation [3], autonomous vehicles [4], [5] and robotics [6], recent research in the field has focused on expanding the classical MAPF framework to fit more real-world applications [7]–[9].

A main research direction towards the real-world applicability of MAPF problems is the problem of lifelong MAPF, also known as the Multi-Agent Pickup and Delivery (MAPD) problem. In this problem, a group of autonomous agents operate in a shared environment to complete a stream of incoming tasks, each with start and goal locations, while avoiding collisions with each others [10], [11]. A similar problem, studied by Ma et al. [12] is the package-exchange robot-routing problem (PERR) where payload exchanges and transfers are allowed thus enabling the modelling of more general transportation problems.

In this work, we introduce the *Cooperative-MAPF (Co-MAPF)* framework, a MAPF extension, in which a group of agents collaborate towards completing a *cooperative task*. The classical MAPF problem is inherently cooperative, since each agent has to arrive at its goal, without colliding with other agents. However, in many real-world applications, agents that operate in a shared environment are often *heterogeneous* [13] and may have a different set of abilities and restrictions. Therefore, in the Co-MAPF framework, achieving goals and completing tasks may not depend only on avoiding collisions between agents, but also on actively coordinating their actions. Simply put, we may want agents not just to "not interrupt" each other, but also help each other achieve their goals. We term this a *truly cooperative* setting.

Our motivating problem is taken from the warehouse-automation domain [3]. In this problem, storage locations host inventory pods that hold goods of different kinds. Robots operate autonomously in the warehouse, picking up and carrying inventory pods to designated drop-off locations, where goods are manually taken off the pods for packaging. In this scenario, the robot's main task is to transport the pods around the warehouse, and we refer to robots executing such tasks as *transfer units*. Research in a different, yet closely-related area, has studied the problem of autonomous robotic arms capable of picking-up a specific item from an inventory pod [14]. We refer to a moving robot with such arm as a *grasp unit*. This motivates the investigation of an improved warehouse scenario, where robots of two types, grasp and transfer units, can work together in coordination (for example, by scheduling a meeting between them) to improve some optimization objective. For instance, the number of completed tasks for a given time period. This motivating example is depicted in Fig. 1.

We incorporate a truly cooperative behavior to classical MAPF by assigning cooperative tasks (rather than goals) to agents, similar to (non-cooperative) tasks defined in the MAPD literature [10], [11]. Agents cooperate in the context of these cooperative tasks, and are only able to complete tasks by coordinating their actions and goals with each other.

We suggest a formulation to the Co-MAPF problem which is derived from the classical MAPF formulation [2]. In addition, we discuss differences and further extensions to the Co-MAPF framework which can be used towards achieving more cooperative capabilities in a MAPF problem. In the suggested formulation, presented in Section II, there is more than one set of agents, possibly representing heterogeneous real-world agents, and we specifically focus on the case of two sets of agents. The cooperation between agents is restricted to the form of *meetings*, where agents have to schedule a meeting location and time to complete a task. We also discuss other forms of agent interactions, and generalizations to the suggested formulation. Besides the aforementioned warehouse problem, more real-world problems can be modeled using the Co-MAPF framework, such

Nir Greshler and Nahum Shimkin are with Viterbi Faculty of Electrical & Computer Engineering, Technion, Haifa, Israel. Ofir Gordon and Oren Salzman are with Department of Computer Science, Technion, Haifa, Israel. Emails: {nirgreshler,ofirgo}@campus.technion.ac.il, osalzman@cs.technion.ac.il, shimkin@ee.technion.ac.il.

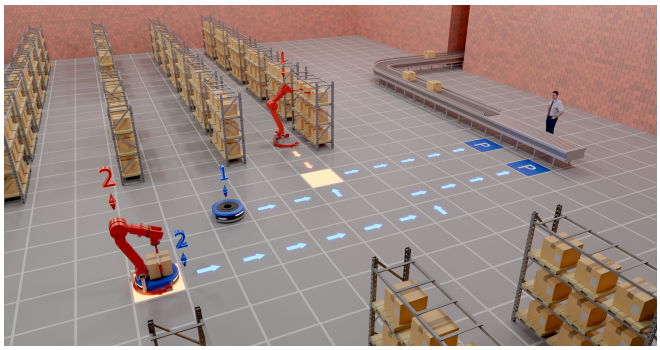


Fig. 1: Two pairs of robots operate in a warehouse–two grasp units and two transfer units. Grasp unit #1 arrived at the task start location, i.e., next to the shelf. It will pick up the box and then drive to the meeting location (marked with a yellow square) to transfer the box to transfer unit #1. The transfer unit has a path (marked with blue arrows) to the meeting point, and from there to the task goal (the P square), where the box will be picked by a human employee. The second pair of robots (#2) are at their meeting location.

as the involvement of aerial robots in fulfilment centers [15], the truck-and-drone "last-mile" delivery problem [16] and multi-drone delivery using transit networks [17].

Based on the suggested formulation, we introduce (in Section III) *Cooperative Conflict-Based Search (Co-CBS)*, an optimal three-level algorithm that is heavily based on two previously-suggested optimal algorithms: the well-known Conflict-Based Search (CBS) [18] for solving a classical MAPF problem and the Conflict-Based Search with Optimal Task Assignment (CBS-TA) [19] for solving the anonymous MAPF problem, where we also need to assign goals (or tasks) to each agent. Finally, we introduce two improvements to the basic version of Co-CBS.

For clarity of exposition, the description of our Co-CBS algorithm is based on the original CBS algorithm which has numerous extensions and improvements. Many of these improvements can be immediately applied to Co-CBS, as we discuss in Section VII.

A theoretical analysis of Co-CBS is presented in Section IV where we discuss the conditions under which Co-CBS is complete and prove that it is optimal. We present empirical results of running Co-CBS on several MAPF benchmarks and show that it solves nontrivial problem instances (detailed in Section VI). We show that our two suggested Co-CBS improvements significantly improve the algorithm's performance.

Finally, in Section VII we discuss some extensions and research directions, specifically for Co-CBS, but more importantly, general for the Co-MAPF framework.

II. Background and Setting

We first describe and formulate the classical MAPF problem followed by a formulation of our proposed Cooperative-MAPF (Co-MAPF) framework. Then, we define the objective function used in Co-MAPF.

A. Classical MAPF

In the classical MAPF problem [2], we are given an undirected graph $G = (V, E)$ whose vertices V correspond to locations and whose edges E correspond to connections between the locations that the agents can move along. $A = \{a_1, \ldots, a_k\}$ is a set of k agents, each is provided with a start and goal location, (s_i, g_i) s.t. $s_i, g_i \in V$.

Time is discretized and at each time step, each agent can either *move* on the graph or *wait* at its current vertex. A feasible MAPF solution is a set of paths $\mathcal{P} = \{p_1, \ldots, p_k\}$ such that p_i is a path for agent a_i from vertex s_i to vertex g_i and there are no conflicts between any two paths in $\mathcal{P}$. We consider two types of conflicts—a *vertex conflict*, in which two agents occupy the same vertex at the same time step, and an *edge conflict* (or *swapping conflict*), in which two agents traverse the same edge from opposite sides ("swap positions") at the same time step. An optimal solution is a feasible set of paths $\mathcal{P}$ which optimizes some objective function (specifically defined in Section II-C).

B. Cooperative-MAPF (Co-MAPF)

We wish to incorporate cooperative behavior into the classical MAPF problem. This is done by replacing agent goals with a set of *cooperative tasks*, i.e., tasks that require the cooperation and coordination of a group of agents in order to be completed. Specifically, here we limit ourselves to cooperative tasks (simply referred to as tasks in the rest of this paper) that require pre-defined pairs of agents to work together. We discuss possible extensions in Section VII.

In the Co-MAPF problem we are given an undirected graph $G = (V, E)$. The set of agents A consists of two distinguishable sets, i.e., $A = \mathcal{A} \cup \mathcal{B}$. Each set includes k agents of a specific type, namely $\mathcal{A} = \{\alpha_1, \ldots, \alpha_k\}$ and $\mathcal{B} = \{\beta_1, \ldots, \beta_k\}$ ($2k$ agents in total). The two types of agents may differ in their traversal capabilities or possible actions (for instance, picking up an object). We are also given a set of tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_k\}$ s.t. each task τ_i is assigned to a pair of agents (α_i, β_i). We refer to α_i and β_i as the *initiator* and *executor* agents, respectively. Each task $\tau_i \in \mathcal{T}$ is defined by a start location s_i and a goal location g_i.

Each agent has a unique start location given by a function $\mathcal{V}_0 : A \to V$ s.t. $\mathcal{V}_0(a)$ is the location of agent a at time step 0. An agent's goal is not directly given but rather derived from its assigned task. In our setting, a task $\tau_i = (s_i, g_i)$ for agents (α_i, β_i) is composed of the following steps: (i) moving the initiator agent α_i to the task's start location s_i, (ii) moving both agents to a so-called *meeting* $m_i = (v_i^m, t_i^m)$ where $v_i^m \in V$ is the meeting location and t_i^m is the meeting time step, both of which are computed by the algorithm (and not specified by the task[1]), (iii) moving the executor agent to the task's goal location g_i. For a visualization, see Fig. 1. Note that a meeting (v_i^m, t_i^m) is defined using a single vertex v_i^m, which means that both agents arrive at the same location at the same time, and this

[1]Note that a meeting m_i is defined by its location and time. Thus, when referring to a meeting, we mean both.

is not considered a collision in our setting. This represents a scenario where the two robots meet following some special *rendezvous* protocol (see robots #2 in Fig. 1), allowing them to be at the same location without colliding. We may also define a meeting using a pair of adjacent vertices, i.e. $v^m_{i,\alpha}$ and $v^m_{i,\beta}$ where each agent has its own meeting location, similar to [20]. This extension is discussed in Section VII.

Formally, a solution to a Co-MAPF instance is a set of paths pairs $\mathcal{P} = \left\{(p^\alpha_1, p^\beta_1), \ldots, (p^\alpha_k, p^\beta_k)\right\}$ s.t. for each pair $1 \leq i \leq k$, p^α_i, p^β_i start in $\mathcal{V}_0(\alpha_i)$ and $\mathcal{V}_0(\beta_i)$, respectively. Path p^α_i goes through s_i at some time step t_i, and both paths contain a meeting at vertex v^m_i at the same time t^m_i s.t. $t_i \leq t^m_i$. Finally, p^α_i ends in vertex v^m_i at time t^m_i and p^β_i ends in vertex g_i. Similarly to classical MAPF, in order for a solution to be feasible, there should be no conflicts between the paths in $\mathcal{P}$, with the exception that the paths of agents sharing a task intersect at their meeting point.

C. Objective functions for Cooperative MAPF

Arguably, the most common objective functions used in classical MAPF to evaluate solutions are *makespan (MKSP)* and *sum-of-costs (SOC)* [2], both to be minimized. MKSP is defined as the number of time steps required for all agents to reach their target, while SOC is the sum of time steps required by each agent to complete all tasks. In this paper we focus on the SOC objective, which is, arguably, more natural for our setting—it implicitly minimizes both the time it takes to complete a task, and the time the initiator finishes its part in the task. We note that all results presented can be applied to the MKSP objective as well. The sum of costs of $\mathcal{P}$ is defined as $\sum_{1 \leq i \leq k} |p^\alpha_i| + |p^\beta_i|$. Wait actions are counted until an agent finishes its plan (i.e., after the meeting for α_i and after arriving at g_i for β_i).

III. Cooperative Conflict-Based Search

We now present the Cooperative Conflict-Based Search (Co-CBS) algorithm, a three-level optimal planning algorithm for solving Co-MAPF problem instances. As our suggested algorithm is based on CBS [18], we start with a brief description of it. CBS is a two-level search algorithm. The high-level performs a best-first search over a so-called *constraint tree* (CT). Each CT node consists of a solution, its cost and a set of constraints. CBS finds conflicts in the solution and resolves them by imposing constraints on agents. A constraint is either a vertex constraint (a, v, t), or an edge constraint (a, u, v, t). The low-level constructs paths for each individual agent while satisfying the imposed constraints. CBS resolves conflicts by splitting a CT node and introducing an additional constraint for each agent participating in the conflict at the lower level.

We now continue with an overview of Co-CBS (depicted in Fig. 2 and outlined in Algorithm 1). We then continue with lower-level details.

1) Algorithm overview: Co-CBS is a search algorithm based on CBS that considers the cooperative aspect of the problem. More specifically, Co-CBS consists of three levels of search in three different spaces (similar to [19] and [21]): (i) the *meetings space*, (ii) the *conflicts space* and (iii) the *paths space*. The meetings space contains all possible combinations of meetings, one for each task. We'll refer to the three levels of search as the meetings level, conflicts level and paths level, respectively.

Algorithm 1 Cooperative Conflict-Based Search (Co-CBS)

Input: $G, \mathcal{A}, \mathcal{B}, \mathcal{V}_0, \mathcal{T}$ ▷ Co-MAPF problem instance
Returns: optimal path for each agent
for all $\tau_i \in \mathcal{T}$ **do** ▷ using Algorithm 2 for each task
  $T_i \leftarrow compute\_meetings\_table(\tau_i, \alpha_i, \beta_i)$
R = new node
$R.constraints \leftarrow \emptyset$
$R.meetings \leftarrow$ get the initial set of optimal meetings
$R.root \leftarrow$ True
$R.solution \leftarrow plan\_paths()$ ▷ to and from meetings
$R.cost \leftarrow compute\_cost(R.solution)$
insert R to OpenRoots
while Open *not empty* or OpenRoots *not empty* **do**
  $N \leftarrow$ lowest cost node from Open∪OpenRoots
  Validate the path in N until a conflict occurs
  if N has no conflicts **then**
    return $N.solution$ ▷ N is goal
  if $N.root$ *is* $True$ **then**
    $expand\_root(N)$ ▷ using Algorithm 3
  $C \leftarrow$ first conflict (a_i, a_j, v, t) in N
  for all *agent* a_i *in* C **do**
    $A \leftarrow$ new node
    $A.constraints \leftarrow N.constraints + (a_i, v, t)$
    $A.meetings \leftarrow N.meetings$
    $A.root \leftarrow$ False
    $A.solution \leftarrow N.solution$
    Update $A.solution$ by invoking $plan\_paths(a_i)$
    $A.cost \leftarrow compute\_cost(A.solution)$
    Insert A to Open

Co-CBS simultaneously searches over all possible meetings and for each meeting, over all possible paths. To perform this search in a systematic and efficient manner, we need to consider an *ordering* of the meetings. Indeed, in Equation 1 we define a meeting's cost which is dependent both on the meeting's location and time. To efficiently traverse the set of possible meetings, we introduce the notion of a *Meetings Table* which stores for each meeting location the currently-best meeting time. As we will see, this table will allow us to iterate over all meetings in a best-first manner.

In contrast to CBS that constructs a single constraint tree (CT), Co-CBS creates a forest of CTs, similar to [19]. Each CT starts in a *root* node and corresponds to a specific set of meetings (a specific meeting for each task). In Co-CBS, each CT node has two additional fields (when compared to CBS): *root* specifies if the node is a root or a *regular* node and *meetings* specifies the current set of meetings (one for each task) which is used during the path-level search.

Co-CBS starts with a single root node, with the minimum-cost set of meetings (see Equation 3), while ignoring possible

conflicts between agents. In each iteration, Co-CBS selects a lowest-cost node from the OPEN list (either a root or regular node), in a best-first approach similar to CBS. Whenever a root node is selected, in addition to splitting the tree due to a conflict, Co-CBS also expands it in the meetings space by generating the next best sets of meetings. Namely, new root nodes are created only on demand. For each expanded node, given its set of meetings and constraints, the paths level computes a solution by planning the different steps a task solution is composed of (Section II-B).

2) Computing the Meetings Table: We denote the cost of a meeting $m_i = (v_i^m, t_i^m)$ as $C_i(v_i^m, t_i^m)$. C_i is given for the SOC objective, by

$$C_i(v,t) = \begin{cases} 2 \cdot t + d\left(v, g_i\right), & t \geq t_i^*\left(v\right) \\ \infty, & \text{otherwise} \end{cases}, \quad (1)$$

where $t_i^*\left(v\right)$ is the earliest possible meeting time at v for task τ_i, i.e., the earliest time both assigned agents can arrive at v. Specifically, $t_i^*\left(v\right)$ is defined as

$$t_i^*\left(v\right) = \max\left\{d\left(\mathcal{V}_0\left(\alpha_i\right), s_i\right) + d\left(s_i, v\right), d\left(\mathcal{V}_0\left(\beta_i\right), v\right)\right\}, \quad (2)$$

where $d(u,v)$ is the length of the single-agent shortest path from u to v. If $d(u,v) = \infty$, no such path exists.

The first step of Co-CBS is to compute T_i, the meetings table for each task τ_i (lines 3-4 in Alg. 1). The meetings table is a function $T_i : V \to \mathbb{R} \cup \{\infty\}$ that returns for each vertex $v \in V$ the cost for completing task τ_i with a meeting in v at the earliest possible time. $T_i\left(v\right)$ is initialized for each $v \in V$ with $T_i\left(v\right) = C_i(v, t_i^*\left(v\right))$. Each meetings table is stored as a heap which allows for $update$ and $getMin$ operations in $\mathcal{O}(\log|V|)$. These operations are used during the root node expansion which will be described shortly.

We compute $T_i\left(v\right)$ for all $v \in V$ in polynomial time using A* and Dijkstra's algorithm as described in Algorithm 2. Computing the meetings table for each task τ_i requires finding paths from every node $v \in V$ to the agents' start locations, as well as tasks' start and goal locations.

3) Root initialization: We define the cost of a set of meetings $\mathcal{M} = \{m_1, \ldots, m_k\}$ as follows: $C\left(\mathcal{M}\right) = \sum_{i=1}^{k} C_i\left(v_i^m, t_i^m\right)$. $\mathcal{M}^*$ is a set of meetings that minimizes the problem objective while ignoring possible conflicts between agents. Namely,

$$\mathcal{M}^* \in \arg\min_{\mathcal{M}} C\left(\mathcal{M}\right). \quad (3)$$

Co-CBS's search starts with creating the initial CT root node with an empty set of constraints, and a minimal-cost set of meetings $\mathcal{M}^*$, by choosing a lowest-cost meeting for each task from the meeting tables (lines 5-8). Given $\mathcal{M}^*$, the paths level is called to compute individual paths for each agent (line 9). This is similar to CBS, except that in the path-level search we plan for each task τ_i in parts: (i) for α_i from $\mathcal{V}_0\left(\alpha_i\right)$ to s_i, and then from s_i to v_i^m at time t_i^m, and (ii) for β_i from $\mathcal{V}_0\left(\beta_i\right)$ to v_i^m at time t_i^m and then to g_i. Note that when planning for a meeting, we should consider both the meeting location and time. The initial CT root node cost is computed and it is inserted to the OPEN list (lines 10-11).

Algorithm 2 Compute Meetings Table

1: **Input:** A Task τ_i and two assigned agents α_i and β_i
2: **Returns:** T_i, the meetings table for task τ_i
3: Compute $d\left(\mathcal{V}_0\left(\alpha_i\right), s_i\right)$ ▷ initiator start to task start
4: Compute $d\left(s_i, v\right), \forall v \in V$ ▷ task start to all
5: Compute $d\left(\mathcal{V}_0\left(\beta_i\right), v\right), \forall v \in V$ ▷ executor start to all
6: Compute $d\left(v, g_i\right), \forall v \in V$ ▷ task goal to all
7: **for all** $v \in V$ **do**
8: Calculate $t_i^*\left(v\right)$ ▷ earliest meeting time in v
9: Calculate $T_i\left(v\right)$ ▷ task cost with meeting at v
10: Store $\left(v, t_i^*\left(v\right), T_i\left(v\right)\right)$ in table

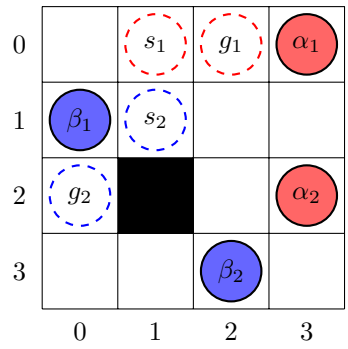
Algorithm 3 Expand root

1: **Input:** Meetings tables of all tasks, a root node P
2: **for all** $\tau_i \in \mathcal{T}$ **do** ▷ loop over all tasks
3: $R =$ new node
4: $R.constraints = \emptyset$
5: $R.meetings = P.meetings$
6: $R.meetings[\tau_i] = get\_next\_meeting(T_i)$
7: $R.root$ = True
8: Update $R.solution$ by invoking $plan\_paths(\alpha_i, \beta_i)$
9: $R.cost = compute\_cost(R.solution)$
10: insert R to OPENROOTS

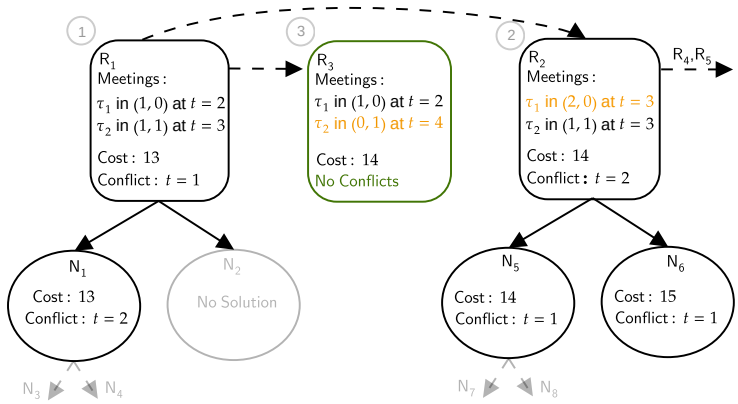
4) Selecting a node for expansion: As long as there are nodes in the OPEN list (line 12), we follow CBS's best-first search approach and select a node with a lowest cost (line 13). If the OPEN list contains both root and regular nodes with the same lowest cost, Co-CBS chooses to expand a regular node (to perform this in practice, Co-CBS keeps root and regular nodes in two separate OPEN lists).

5) Expanding a root node: After selecting a lowest-cost node N from the OPEN list, Co-CBS checks for conflicts in its solution (line 14). If none are found, $N.solution$ is returned as the optimal solution (lines 15-16). Otherwise, if N is a root node, it is expanded to get its successors in the meetings space (lines 17-18). The process of expanding a root node is described in Algorithm 3. Given the current set of meetings (in the expanded root node) $\mathcal{M} = \{m_1, \ldots, m_k\}$, Co-CBS generates up to k new sets of meetings, one for each task. This is done in a non-decreasing manner, by replacing one meeting $m_i \in \mathcal{M}$ at a time, an idea similar to the Increasing Cost Tree Search (ICTS) [22] algorithm, thus creating k new root nodes.

To get the next-best meeting for task τ_i, we have to search both for different locations and time steps in the meetings space. The meetings table T_i of τ_i initially consists of meetings at each possible location, at the earliest time possible. Each time Co-CBS invokes the get-next-meeting procedure for τ_i (line 6 in Algorithm 3), it returns the lowest-cost meeting $m_i = (v_i^m, t_i^m)$ from T_i. The table is then updated so that it holds the next lowest-cost meeting. This is done by updating $T_i\left(v_i^m\right) = C_i\left(v_i^m, t_i^m + 1\right)$. Namely, updating the cost of meeting at v_i^m, but at time $t_i^m + 1$ rather than t_i^m. The next time the get-next-meeting procedure is invoked, the next best meeting will be returned by the table.



(a) Example instance with two tasks: α_1 and β_1 execute τ_1 from s_1 to g_1 and α_2 and β_2 execute τ_2 from s_2 to g_2.



(b) Co-CBS search forest.

Fig. 2: Example execution of Co-CBS (b) on the instance depicted in (a). Root and regular nodes are denoted with R and N, respectively. The initial root node R_1 contains the optimal set of meetings, has a cost of 13 and a conflict at $t = 1$. It is expanded to root nodes R_2, R_3 (with cost 14) and split on its first conflict, creating N_1, N_2. N_2 has no solution, thus N_1 will be chosen next for expansion, creating N_3, N_4, both with no solution. In the next iterations, Co-CBS will expand R_2 (to R_4, R_5, N_5, N_6), then N_5, and finally R_3. Since R_3 has no conflicts, it will be returned as a feasible optimal solution.

Subsequently, a new path is planned for the pair of agents whose meeting changed, the new CT node cost is computed and it is inserted into the OPEN list.

6) Resolving a conflict: The last part of the algorithm is almost identical to CBS: when expanding a node N (either root or regular) Co-CBS splits its CT and creates a regular node for each agent by the first conflict found (lines 19-28). These nodes has the same set of meetings as N (line 23).

IV. Theoretical Analysis

A. Co-CBS Completeness

In the classical MAPF setting, it is possible to check whether an instance is solvable in polynomial time [23]. In the Co-MAPF setting, given a set of meeting locations, we may decompose the problem into two MAPF instances and check that both are solvable. However, Co-CBS also searches for different meeting locations (and times), which means that agents' (intermediate) goals are determined during the search and not pre-defined. Therefore, to ensure completeness we define *source-connected* instances:

Definition 4.1 (source-connected instances): A source-connected Co-MAPF instance is an instance in which for each task τ_i, the following paths exist: (i) $\mathcal{V}_0(\alpha_i) \to s_i$, (ii) $\mathcal{V}_0(\beta_i) \to s_i$, and (iii) $s_i \to g_i$, and none of them pass via an agent's start location.

Lemma 4.1: A source-connected Co-MAPF instance is solvable.

Proving this lemma is straightforward by sequentially planning each agent. We therefore omit the proof. By using the notion of source-connected Co-MAPF instances, we are able to guarantee Co-CBS's completeness while still solving a very wide and realistic set of problem instances. Investigating Co-CBS's completeness in the more general case is left for future research. For simplicity, we also assume similar to [24], a *disappear-at-target* behavior [2]. More specifically, the initiator agent disappears after the meeting, and the executor agent disappears after completing the task (at the task goal location). Note that the following proofs still work without this assumption. A more interesting scenario is where agents are assigned with new tasks upon finishing their part, commonly known as the *lifelong planning* problem as discussed in Section VII.

We start with the following lemma.

Lemma 4.2: The get-next-meeting procedure performs an exhaustive non-decreasing search in the meetings space.

Proof Sketch: Denote n as the number of times get-next-meeting has been invoked. For $n = 1$, because of how T_i is initialized, get-next-meeting returns the meeting location and time with lowest cost. Assume by induction, that the $n-$th time get-next-meeting is invoked, it returns the $n-$th best meeting in the location-time space. Denote this meeting location and time $\hat{v}_n$ and $\hat{t}_n$, respectively. Denote the total cost of both agents using this meeting $\hat{c}_n$. After the $n-$th invocation, T_i is updated such that the meeting at $\hat{v}_n$ is at time $(\hat{t}_n + 1)$ and costs $(\hat{c}_n + 2)$ (for the SOC objective, since we add a time step for each agent). T_i is then sorted by the meetings costs. This means that in the $(n+1)-$th invocation of get-next-meeting, it will return the $n+1$-th best meeting (with cost $\leq (\hat{c}_n + 2)$). ■

Theorem 4.3: Co-CBS will return a solution for any source-connected Co-MAPF instance.

Proof Sketch: By Lemma 4.1 we know that there exists a solution. Denote the set of meetings in this solution by $\mathcal{M} = \{(v_1^m, t_1^m), (v_2^m, t_2^m), \ldots, (v_k^m, t_k^m)\}$. By Lemma 4.2 we know that during the search, Co-CBS will create a root node, denoted by $R_{\mathcal{M}}$, whose set of meetings is $\mathcal{M}$. There exists a feasible solution such that each pair of agents (α_i, β_i) meet at (v_i^m, t_i^m). By the completeness of CBS it is guaranteed that the search from the CT root node $R_{\mathcal{M}}$ will eventually find the solution. ■

By Theorem 4.3, Co-CBS is complete for source-connected Co-MAPF instances. However, Co-CBS will also solve most instances where this assumption doesn't hold, without a completeness guarantee.

B. Co-CBS Optimality

We show that Co-CBS returns an optimal solution for every *solvable* Co-MAPF instance.

Lemma 4.4: Let $\mathcal{M}$ be a set of meetings with $C(\mathcal{M}) = c$ and let N be a CT node with cost larger than c. Co-CBS will generate a root node corresponding to $\mathcal{M}$ before expanding N.

Proof Sketch: Assume that there exists a set of meetings $\mathcal{M}$ s.t. $C(\mathcal{M}) = c$, that hasn't been generated yet. Assume by contradiction that Co-CBS expands a node N with a solution cost $c' > c$. By definition, the first generated set of meetings $\mathcal{M}^*$ (line 7 in Algorithm 1) induces a solution which minimizes the SOC objective function. This implies that the cost of completing all tasks in the (possibly infeasible) solution induced by $\mathcal{M}^*$ is less than or equal to c. The cost of completing all tasks in the (possibly infeasible) solution induced by $\mathcal{M}$ is equal to c. By Lemma 4.2 we know that each set of meetings generated between $\mathcal{M}^*$ and $\mathcal{M}$ has a cost smaller or equal to c. Furthermore, there must be at least one root node in the OPEN list consisting of one of these meeting sets. Therefore, there exists a root node that hasn't been expanded yet in the OPEN list with a cost smaller than c', in contradiction to the best-first search approach which chose node N with a larger cost for expansion. ■

Theorem 4.5: Co-CBS returns an optimal solution for any solvable Co-MAPF instance.

Proof Sketch: Assume that there exists an optimal solution with some cost c^*. Co-CBS performs a CBS-like search on each generated CT, namely, it searches through a forest of constraint trees. By Lemma 4.4 we get that the cost of each expanded root node of each CT constitutes a lower-bound on c^*. From the optimality guarantees of CBS, we get that any node expanded in each of those CTs (i.e., regular nodes) is also a lower bound on c^*. Due to Co-CBS's best-first approach, it won't expand a node with a cost larger than c^* before completing a search through all possible CT nodes with cost c^* (by expanding neither a root node nor a regular one). Since there exists a solution with such cost, and the number of possible solutions with a specific cost is finite, Co-CBS will eventually expand a node with an optimal and feasible solution and return it. ■

V. Co-CBS IMPROVEMENTS

In Section III, we introduced the basic version of Co-CBS for solving the Co-MAPF problem. Co-CBS creates a forest of constraint trees and runs CBS on each tree. Thus, we can apply previously-suggested CBS improvements to Co-CBS. One such improvement that has been shown to significantly decrease CBS's run-time is *prioritizing conflicts (PC)* [25]. In this section we present in detail the application of PC to Co-CBS. More CBS improvements are discussed in Section VII. In addition, we introduce a unique improvement for Co-CBS called *Lazy Expansion (LE)*, which exploits special characteristics of root nodes. Both improvements keep Co-CBS optimal, while introducing a significant improvement in run time, as shown empirically in Section VI.

A. Prioritizing Conflicts (PC) for Co-CBS

The Improved CBS (ICBS) algorithm [25] introduced an enhancement to CBS by defining rules dictating how to split the CT. In particular, conflicts are divided into three types: *cardinal*, *semi-cardinal* and *non-cardinal*. Cardinal conflicts always cause an increase in the solution cost, therefore ICBS chooses to split cardinal conflicts first. Cardinal conflicts are identified by examining the width of a *multi-value decision diagram (MDD)* [22], which is constructed for each low-level path found. The MDD is a directed a-cyclic graph which compactly stores all possible paths of a given cost c for a given agent, from its start vertex to its goal vertex. An MDD of cost c consists of c layers, corresponding to c time steps.

Applying PC to Co-CBS is not straightforward, since an MDD stores paths from a start vertex to a goal vertex, while in Co-MAPF paths are constrained to ensure cooperation between agents. More specifically, in our Co-MAPF setting, each agent has an *intermediate goal*, i.e., the task start location, or the meeting location (at a specific time). We therefore need to modify the way an MDD is constructed, and indeed we suggest a method for efficiently doing so for both agents.

For the initiator agent, we must ensure it passes through the task's start location. In other words, we need to prune MDD nodes that are not part of any of the agent's paths which pass the task's start location. We refer to such nodes as *invalid nodes*. Constructing an MDD efficiently is done using two breadth-first searches–one forward and one backward (start to goal and vise versa) [22]. In order to efficiently prune invalid nodes, we follow the following procedure: during the forward search, we mark MDD nodes corresponding to the task start location and all their descendants as *valid_forward*. Similarly, during the backward search, we mark these nodes and all their ancestors as *valid_backward*. Finally, all MDD nodes that are not marked with either flags are pruned.

For the executor agent, constructing the MDD requires only slight changes. We need to constrain the agent to be at the meeting's location at the meeting's time. We simply do it by eliminating all other nodes from the MDD layer corresponds to the meeting time during the forward pass in the MDD construction.

B. Lazy Expansion (LE) of root nodes

Co-CBS searches the meetings space by creating root nodes, each corresponding to a unique set of meetings. Note that since no constraints are imposed on paths of root nodes, their cost is given as an aggregation of their meeting costs. Furthermore, meeting costs are computed a-priori during the construction of meeting tables (see Section III). This means that when a root node is expanded, and new root nodes are created, they can immediately be inserted into the OPEN list *without* computing their low-level paths. The low-level paths

will be computed only when these root nodes are extracted from the OPEN list. We term this *Lazy Expansion (LE)*.

Each time a root node is expanded, it creates k new root nodes by replacing the meeting of each of the tasks. We emphasize that while generating those nodes is mandatory in order to guarantee optimality, most of them won't be expanded. Thus, the run-time saved by LE can be significant

VI. EXPERIMENTAL EVALUATION

Co-CBS solves the newly introduced Co-MAPF problem. To the best of our knowledge, there does not exist an off-the-shelf optimal solver for MAPF problems involving co-operative behavior. Suggesting a centralized A*-based implementation for solving the Co-MAPF problem is challenging due to constraints imposed on low-level paths to achieve cooperation. Such approach would require to perform a search in the meetings space, resulting in an exponentially-large state space. Moreover, an attempt to solve Co-MAPF using such implementation would yield similar results as solving classical MAPF problem using A* [18], due to their similar search approach and conflict-resolution mechanism. However, we do compare Co-CBS with a baseline prioritized planning algorithm. The baseline algorithm plans for each agent independently, considering the paths of previous agents, and using the optimal meeting location. Thus, it runs very fast, albeit it is sub-optimal.

To measure the quality of Co-CBS, we present the results of an empirical evaluation performed on standard MAPF benchmarks [2], [26] showing the performance of the basic version of Co-CBS, as well as the two suggested improvements (see Section V). Co-CBS is implemented in C++[2] and is based on the implementation of Li et al. [27]. All simulations were performed on an Intel Xeon Platinum 8000 @ 3.1Ghz machine with 32.0 GB RAM.

A. Benchmarks and setup

We evaluated Co-CBS on several 2D grid-based benchmarks. Specifically, we tested Co-CBS on different types of maps—a dense game map (*DAO, den312d*), random map (*random-32-32-20*), a large warehouse (*warehouse-10-20-10-2-1*) and a custom small warehouse (57×27). We ran 25 random queries for each benchmark for the SOC objective with the number of tasks ranging from 6 tasks (12 agents) to 22 tasks (44 agents) and with a timeout of two minutes. On each benchmark, we compare the performance of three different variances of Co-CBS: (i) basic Co-CBS, (ii) Co-CBS with prioritizing conflicts (PC), and (iii) Co-CBS with PC and lazy expansion (LE) of root nodes. As opposed to the classical MAPF, where each agent is provided with start and goal locations, in Co-MAPF, a task's start and goal need to be provided (instead of explicitly providing an agent's goal). Thus, we defined the tasks in each scenario as follows, based on the original benchmark scenario: for each pair of agents, one set of start and goal locations is used for the task, and the other set is used for the agents' start locations.

[2]https://github.com/CRL-Technion/Cooperative-MAPF

B. Results

We first examine the algorithm's success rate (i.e., the ratio of solved instances within the time limit) for all benchmarks. Fig. 3 shows the success rates of Co-CBS on all maps. Co-CBS successfully solves more than 80% of the instances (excluding the den312d benchmark) with ten tasks. The success rate sharply drops below 20% for twelve tasks or more on the dense den312d map. Using PC improves the basic Co-CBS in all cases, achieving up to 30% increase in the success rate. Furthermore, adding LE on top of PC further improves the performance in most cases, and never degrades the performance. This is especially notable with a large number of tasks, where many root nodes are created.

Fig. 4a shows the average number of generated meeting sets. Fig. 4b shows the ratio η between the number of instances where the first set of meetings is used to obtain the solution and the total number of instances. Both warehouse environments are typically sparser, causing fewer conflicts. Thus, a feasible solution is usually found quickly using the first set of meetings. This is especially noticeable in the large warehouse, when η is close to one. The search in this case is equivalent to running CBS with the first set of meetings. For the same reason, PC does not improve the performance in this environment. Applying LE as well, however, does improve the success rate for the majority of tasks. On the other hand, in other smaller and denser environments, most solutions are not obtained using the first generated set of meetings. A more exhaustive meeting-space search is therefore required to find an optimal solution, as shown in Fig. 4a.

Fig. 5 presents the results achieved using the baseline planner. More specifically, the baseline algorithm achieves 100% success rate on all benchmarks. However, it performs very poorly in terms of cost-optimality. In dense maps and a large number of tasks, the cost is increased by roughly 40%.

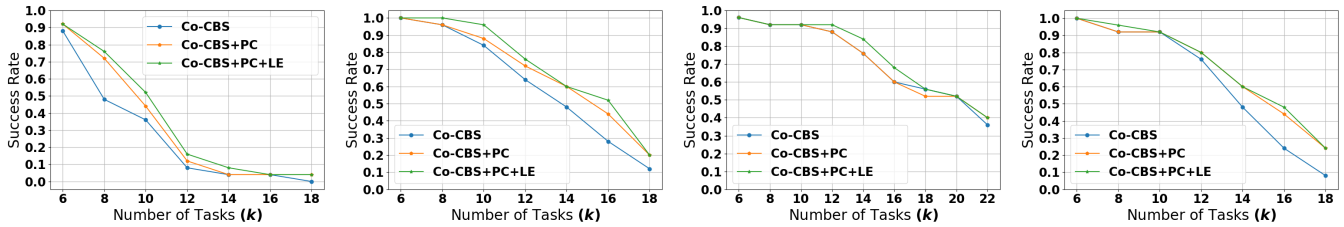
VII. DISCUSSION AND FUTURE WORK

In this paper, we introduced the Cooperative Multi-Agent Path Finding (Co-MAPF) problem, an extension to classical MAPF that incorporates cooperative behavior. We introduced Co-CBS, a three-level search algorithm that optimally solves Co-MAPF instances, as well as two improvements, Prioritizing Conflict (PC) and Lazy Expansion (LE).

In this section, we provide a comprehensive discussion regarding the suggested model and algorithm. Specifically, we discuss further possible improvements that can be applied to Co-CBS and suggest possible extensions to the Co-MAPF model. We argue that Co-CBS forms a basic framework that may serve as a natural starting point for future extensions.

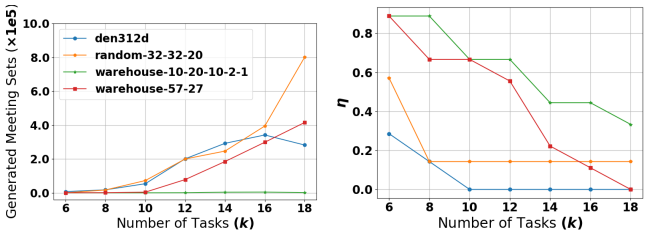
A. Co-CBS's Extensions and Improvements

1) Information reusing between constraint trees: Co-CBS expands root nodes by only changing one meeting in the newly-created node. Moreover, the next selected meeting is usually very close to the current meeting, both in location and time. This implies that Co-CBS searches over multiple trees that potentially have very similar solutions. We may exploit this for more efficient computation.



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

Fig. 3: Success rates.



(a) (b)

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

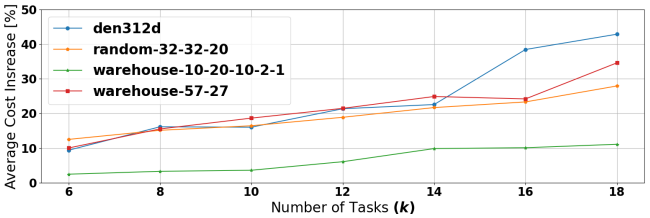


Fig. 5: The average cost increase rate over baseline planner.

2) Meetings-level search: Co-CBS uses a simple-yet-effective method for finding an optimal meeting for each task. For large problem instances, this method may become memory and run-time expensive, due to the maintenance of large meeting tables. We may consider incorporating an algorithm such as the recently-proposed CF-MM* algorithm [28], for the Multi-Agent Meeting problem. Furthermore, we may couple meetings and paths planning, and handle conflicts during the search for a meeting. This may be advantageous as meetings and conflicts may be tightly coupled.

3) Existing CBS improvements: In addition to the PC improvement presented in Section V, more CBS improvements exist. Some of these include adding heuristics [29], disjoint splitting [30], bypassing a conflict [31], symmetry breaking [32] and exploiting similarities between nodes in a single constraint tree [33]. We can also apply to Co-CBS (bounded) sub-optimal variants of CBS [34].

4) Meetings in adjacent locations: Co-CBS solves the Co-MAPF problem where a agents are required to meet in a single location. As discussed in Section II-B, this definition of a meeting can be modified by requiring the agents to meet in adjacent locations (namely two vertices connected by an edge). Co-CBS can also solve these scenarios, with only slight modifications, as planning for each agent is done independently. More specifically, when computing a meetings table, for each vertex we need to account for all its neighbors and create a meeting for each one. Moreover, when planning paths, each agent would have to arrive at its corresponding meeting location (and time), rather than both agents arriving at a single location.

B. Extensions to the Co-MAPF Framework

1) Number and types of collaborating agents: A rather straightforward generalization of Co-MAPF is to require more than two agents to collaborate on a task. The problem introduced in Section I motivates this extension: several grasp units may pickup several items for a single transfer unit. Co-CBS can solve this problem with a few minor changes. However, if the number of agents per task isn't fixed, additional work is required. Moreover, we may consider agents with different traversal capabilities (e.g., different velocities [6]), by possibly changing the single-agent planner.

2) Other forms of cooperative interaction: We introduced a definition for the Co-MAPF problem, where interaction between agents is expressed via meetings between two types of agents. While this interaction is very intuitive, more forms of cooperative interaction can be modeled (for example, temporal constraints). We may generalize the formulation to include a finite set of possible agent types, and define more complex tasks where each agent type has its dedicated role.

The framework provided by Co-CBS might allow to address such general definitions by only adjusting the *cooperation-level* search (the meetings level in our case). Any cooperative planning, which results in inducing goals for an agent, can be easily plugged in into Co-CBS.

3) Task assignment and lifelong planning: In this problem we assume cooperative tasks are pre-assigned to collaborating agents. However, optimizing the task assignment as well may significantly affect solution quality (as in classical MAPF). This is extremely relevant for lifelong-planning problems, where agents have to attend to a stream of incoming tasks. Such Generalization of the Co-MAPF framework will bring the formulation closer to real-world problems.

ACKNOWLEDGMENTS

This research was partially supported by grants No. 102583, 2028142 from the Isaeli Ministry of Science & Technology (MOST), and by grant No. 1018193 from the United States-Israel Binational Science Foundation (BSF).

REFERENCES

[1] A. Torreño, E. Onaindia, A. Komenda, and M. Stolba, "Cooperative multi-agent planning: A survey," *Computing Research Repository (CoRR)*, vol. abs/1711.09057, 2017.

[2] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, R. Barták, and E. Boyarski, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Int. Symp. on Combinatorial Search (SOCS)*, 2019, pp. 151–159.

[3] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *Artificial Intelligence*, vol. 29, no. 1, pp. 9–20, 2008.

[4] K. Dresner and P. Stone, "A multiagent approach to autonomous intersection management," *Journal of Artificial Intelligence Research (JAIR)*, vol. 31, pp. 591–656, 2008.

[5] J. Švancara, M. Vlk, R. Stern, D. Atzmon, and R. Barták, "Online multi-agent pathfinding," in *AAAI Conf. on Artificial Intelligence*, vol. 33, 2019, pp. 7732–7739.

[6] 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 *Int. Conf. Automated Planning and Scheduling (ICAPS)*, 2016, pp. 477–485.

[7] H. Ma, S. Koenig, N. Ayanian, L. Cohen, W. Hönig, T. K. S. Kumar, T. Uras, H. Xu, C. A. Tovey, and G. Sharon, "Overview: Generalizations of multi-agent path finding to real-world scenarios," *Computing Research Repository (CoRR)*, vol. abs/1702.05515, 2017.

[8] 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 *Int. Symp. on Combinatorial Search (SOCS)*, 2017, pp. 29–37.

[9] O. Salzman and R. Stern, "Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems," in *Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2020, pp. 1711–1715.

[10] H. Ma, J. Li, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," in *Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2017, pp. 837–845.

[11] M. Liu, H. Ma, J. Li, and S. Koenig, "Task and path planning for multi-agent pickup and delivery," in *Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2019, pp. 1152–1160.

[12] H. Ma, C. A. 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 Conf. on Artificial Intelligence*, 2016, pp. 3166–3173.

[13] D. Atzmon, Y. Zax, E. Kivity, L. Avitan, J. Morag, and A. Felner, "Generalizing multi-agent path finding for heterogeneous agents," in *Int. Symp. on Combinatorial Search (SOCS)*, 2020, pp. 101–105.

[14] N. Correll, K. E. Bekris, D. Berenson, O. Brock, A. Causo, K. Hauser, K. Okada, A. Rodriguez, J. M. Romano, and P. R. Wurman, "Analysis and observations from the first Amazon picking challenge," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 1, pp. 172–188, 2016.

[15] R. Shome, "Roadmaps for robot motion planning with groups of robots," *Current Robotics Reports*, pp. 1–10, 2021.

[16] C. C. Murray and R. Raj, "The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones," *Transportation Research Part C: Emerging Technologies*, vol. 110, pp. 368–398, 2020.

[17] S. Choudhury, K. Solovey, M. J. Kochenderfer, and M. Pavone, "Efficient large-scale multi-drone delivery using transit networks," in *IEEE Int. Conf. Robotics and Automation (ICRA)*, 2020, pp. 4543–4550.

[18] 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.

[19] W. Hönig, S. Kiesel, A. Tinka, J. W. Durham, and N. Ayanian, "Conflict-based search with optimal task assignment," in *Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2018, pp. 757–765.

[20] G. Revach, N. Greshler, and N. Shimkin, "Planning for cooperative multiple agents with sparse interaction constraints," in *The online Proceedings of the 6th Workshop on Distributed and Multi-Agent Planning (DMAP) at ICAPS 2020*, 2020, pp. 48–56.

[21] P. Surynek, "Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering," *Computing Research Repository (CoRR)*, vol. abs/2009.05161, 2020.

[22] 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.

[23] J. Yu and D. Rus, "Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms," in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, ser. Springer Tracts in Advanced Robotics, vol. 107, 2014, pp. 729–746.

[24] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *AAAI Conf. on Artificial Intelligence*, 2019, pp. 7643–7650.

[25] 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 *Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2015, pp. 740–746.

[26] N. Sturtevant, "Benchmarks for grid-based pathfinding," *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144 – 148, 2012. [Online]. Available: http://web.cs.du.edu/ sturtevant/papers/benchmarks.pdf

[27] J. Li, D. Harabor, P. J. Stuckey, and S. Koenig, "Pairwise symmetry reasoning for multi-agent path finding search," *Computing Research Repository (CoRR)*, vol. abs/2103.07116, 2021.

[28] D. Atzmon, S. I. Freiman, O. Epshtein, O. Shichman, and A. Felner, "Conflict-free multi-agent meeting," in *Int. Conf. Automated Planning and Scheduling (ICAPS)*, 2021, pp. 16–24.

[29] 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 *Int. Conf. Automated Planning and Scheduling (ICAPS)*, 2018, pp. 83–87.

[30] J. Li, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig, "Disjoint splitting for multi-agent path finding with conflict-based search," in *Int. Conf. Automated Planning and Scheduling (ICAPS)*, 2019, pp. 279–283.

[31] E. Boyarski, A. Felner, G. Sharon, and R. Stern, "Don't split, try to work it out: Bypassing conflicts in multi-agent pathfinding," in *Int. Conf. Automated Planning and Scheduling (ICAPS)*, 2015, pp. 47–51.

[32] J. Li, G. Gange, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig, "New techniques for pairwise symmetry breaking in multi-agent path finding," in *Int. Conf. Automated Planning and Scheduling (ICAPS)*, 2020, pp. 193–201.

[33] E. Boyarski, A. Felner, D. Harabor, P. J. Stuckey, L. Cohen, J. Li, and S. Koenig, "Iterative-deepening conflict-based search," in *Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2020, pp. 4084–4090.

[34] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Int. Symp. on Combinatorial Search (SOCS)*, 2014.