Conflict-Based Search with Optimal Task Assignment*

Wolfgang Hönig
University of Southern California
whoenig@usc.edu

Scott Kiesel
Amazon Robotics
skkiesel@amazon.com

Andrew Tinka
Amazon Robotics
atinka@amazon.com

Joseph W. Durham
Amazon Robotics
josepdur@amazon.com

Nora Ayanian
University of Southern California
ayanian@usc.edu

ABSTRACT

We consider a variant of the Multi-Agent Path-Finding problem that seeks both task assignments and collision-free paths for a set of agents navigating on a graph, while minimizing the sum of costs of all agents. Our approach extends Conflict-Based Search (CBS), a framework that has been previously used to find collision-free paths for a given fixed task assignment. Our approach is based on two key ideas: (i) we operate on a search forest rather than a search tree; and (ii) we create the forest on demand, avoiding a factorial explosion of all possible task assignments. We show that our new algorithm, CBS-TA, is complete and optimal. The CBS framework allows us to extend our method to ECBS-TA, a bounded suboptimal version. We provide extensive empirical results comparing CBS-TA to task assignment followed by CBS, Conflict-Based Min-Cost-Flow (CBM), and an integer linear program (ILP) solution, demonstrating the advantages of our algorithm. Our results highlight a significant advantage in jointly optimizing the task assignment and path planning for very dense cases compared to the traditional method of solving those two problems independently. For large environments with many robots we show that the traditional approach is reasonable, but that we can achieve similar results with the same runtime but stronger suboptimality guarantees.

ACM Reference Format:
Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph W. Durham, and Nora Ayanian. 2018. Conflict-Based Search with Optimal Task Assignment. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 9 pages.

1 INTRODUCTION

In the Multi-Agent Path-Finding (MAPF) problem, a set of agents is tasked to move from their start locations to specified goal locations in a known environment without collisions. The MAPF problem and its variants have many applications, including warehouse automation, improving traffic at intersections, and search and rescue; see [25] for detailed references.

In this paper we are motivated by warehouse automation, where robots might be used to deliver shelves to pack stations [22]. In this domain, robots can initially choose which shelf to pick, but then the shelf must be moved to a specified station. Thus, a robot's task is partially anonymous (any shelf can be picked) and partially non-anonymous (the shelf needs to be delivered to a specified goal) at the same time. The objective in this domain is to minimize the idle time of human workers.

*Part of this research was completed during the first author's internship at Amazon Robotics. We would like to thank the research team at Amazon Robotics for their helpful comments and Hang Ma for providing his implementation of CBM. Our research was also partially supported by NSF under grant IIS-1724392.

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden. © 2018 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

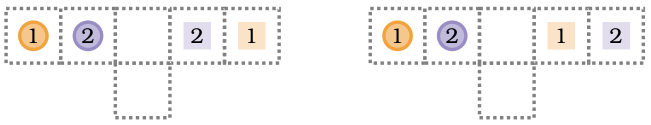


Figure 1: Example where task assignment and path planning cannot be decoupled for the optimal solution. Start positions are circles and goals are squares. Left: Robot 2 must move into the alley temporarily, resulting in a total cost of 8. Right: Both robots can move along their direct path without collision, resulting in a total cost of 6.

In this work, we extend Conflict-Based Search (CBS) [17] (and some of its variants) to simultaneously assign tasks and find paths for agents even for hybrid MAPF problem instances. Another hybrid solver is the Conflict-Based Min-Cost-Flow (CBM) algorithm [12] that minimizes the makespan (time until the last agent reaches its goal), which does not map well to minimizing idle time, where the sum of all costs is a better metric. We provide representative experimental results that demonstrate that our method, called CBS-TA, outperforms the naive (yet frequently used) approach of solving task assignment and path planning independently in dense environments. Finally, the CBS framework allows us to extend our approach to ECBS, a bounded suboptimal MAPF solver. We introduce our bounded suboptimal algorithm, ECBS-TA, and compare its solution quality and runtime to existing solutions.

2 RELATED WORK

MAPF is a well-studied problem in AI. Given an undirected graph of the environment with uniform edge weights and start and goal locations for the agents, we must construct a collision-free path for each agent. Starting at their start location, agents must eventually reach their goal location, and can, at each time step, either wait at a vertex or traverse an edge.

There are two objective functions frequently used in the literature: (1) the sum of the path costs for all agents and (2) the time elapsed until the last agent reaches its goal, i.e. the *makespan.* MAPF has been shown to be NP-hard [14, 18]; however, in the special case where the makespan is minimized and goals can freely be assigned to the agents, it can be reduced to a maximum-flow problem and solved in polynomial time [23].

Most work on MAPF targets the labeled case, where the goal for each agent is preassigned. Labeled MAPF can be solved optimally using, amongst others, CBS [17], M* [19], combinatorial auctions [2], or by applying an ILP-solver [25]. Bounded suboptimal solvers include ECBS [3] and M* variants [20]. A complete but unbounded suboptimal solver is Push-and-Rotate [6]. Furthermore, it is possible to use a polynomial post-processing step to execute plans on real robots [9]. We base our extension on CBS, because the framework is easily extendable and it has been shown that some variants can find solutions for problem instances with hundreds of agents within a few minutes [5].

In the Target Assignment and Path Finding (TAPF) problem agents are split into groups and a set of goals is given to each group [12]. The Conflict-Based Min-Cost-Flow (CBM) algorithm can be used to find makespan-optimal solutions to TAPF instances. Our work provides a bounded suboptimal solver (with respect to sum of path cost) for this and other hybrid MAPF problem instances.

Distributed approaches based on auctions [11] or tokens [13] can find solutions in the MAPF domain, but, unlike our work, cannot compute solutions within a user-specified suboptimality bound.

Our approach is conceptually similar to a method which combines task reassignment and path planning in the M* framework [21]. However, our method works with an arbitrary assignment matrix not requiring that the number of tasks and robots is identical, and shows better scalability to large teams in particular when using our bounded suboptimal solver ECBS-TA.

3 PROBLEM DEFINITION

Consider an undirected graph $G = (V, E)$, where $v \in V$ correspond to locations and $e \in E$ are unit-weight edges connecting two different vertices, indicating a direct passage between those locations. There are N agents at different start locations $s^i \in V, i \in \{1, 2, \ldots, N\}$. The set of M potential goal locations is $\{g^1, \ldots, g^M\}$. The binary $N \times M$ matrix A indicates whether an agent can be assigned to a specified goal: the entry a_{ij} is 1 if agent i is allowed to reach goal g^j and 0 otherwise.

At each time step, an agent can either move to an adjacent vertex (i.e., traverse an edge) or wait at its current vertex. We denote the current vertex of agent i at time step t as v^i_t. A path $p^i = [v^i_0, v^i_1, \ldots, v^i_{T^i}]$ for agent i is feasible if and only if the following conditions hold:

(1) Agent i starts at its start vertex, i.e. $v^i_0 = s^i$.
(2) Agent i ends at one of its potential goal vertices and remains there, i.e. $v^i_{T^i} = g^j$ s.t. $a_{ij} = 1$.[1]
(3) Every action is either moving along an edge or waiting at a vertex, i.e. $\forall t \in \{0, \ldots, T^i - 1\}$: $(v^i_t, v^i_{t+1}) \in E$ or $v^i_t = v^i_{t+1}$.

A collision between agents i and i' can be either a vertex collision (i.e. $\exists t : v^i_t = v^{i'}_t$) or an edge collision (i.e. $\exists t : v^i_t = v^{i'}_{t+1}$ and $v^i_{t+1} = v^{i'}_t$). A solution consists of feasible paths for all N agents such that no collision occurs. A solution is optimal if the sum of individual path costs (SIC) is minimized. The CBS framework requires that agents move in unit time steps. We want to optimize the total time and therefore we minimize $\sum_{i=1}^{N} T^i$.

[1]If there is no assigned goal, we only require agent i to remain at some location eventually.

This problem definition is identical to the traditional MAPF problem with the exception that we introduce the potential assignment matrix A. In case of $N = M$ and A being a permutation matrix (i.e., A contains exactly one 1 in each row and column, and all other elements are 0), our problem is identical to non-anonymous or labeled MAPF. In case of $N = M$ and $A = \mathbf{1}$, the problem is identical to the anonymous or unlabeled MAPF case. Our formulation also allows cases where there are more goals than agents, more agents than goals, or not all agents can reach all goals.

4 CBS-TA

A typical approach for task assignment and path planning is to separate them into two stages. However, both problems are tightly coupled, and certain task assignments may result in fewer collisions during path planning (see Figure 1 for an example). To find an optimal solution, a naive approach would be to generate all possible assignments and solve the path planning for each of those assignments. However, there are $\binom{M}{N}$ assignments for N robots and M goals (assuming $M \geq N$), making this approach infeasible in practice. Instead, we generate an additional assignment on demand once we know that this assignment needs to be considered for the optimal solution, similar to an approach discussed for M* [21].

4.1 Algorithm

We start by briefly describing Conflict-Based Search (CBS), which we extend to incorporate task assignment. CBS is a two-level search. The low-level constructs paths for each individual agent given constraints provided by the high-level. The high-level finds conflicts (in our case, collisions) and resolves them at their earliest start time. Conflict resolution works by adding two successor nodes in the high-level search tree and introducing an additional constraint for each agent participating in the conflict at the lower level. CBS is complete and optimal with respect to the sum of the cost of all agents [17].

For CBS-TA we only need to change the high-level search; see Algorithm 1. Lines that were changed compared to CBS (Algorithm 1 in [16]) are highlighted. In CBS-TA, each high-level node has two additional fields: *root* describes if the current node is a root node and *assignment* describes the current task assignment which is used during the low-level search. CBS builds a search tree with a single root node. In comparison, CBS-TA creates a search forest, but expands new root nodes only on demand. CBS-TA starts with a single root node which uses the best task assignment, while ignoring possible conflicts between agents. Whenever a root node is expanded during the search, we create another root node with the next best assignment.

By design, CBS-TA requires an efficient way of computing the next-best assignment. It is possible to enumerate the K best solutions in various domains, including task assignment [7]. We base our method on existing algorithms [4, 15] but compute new solutions on demand, rather than a set of K solutions. Our notation is closely based on [4]. We compute a lower bound of the cost for agent i to reach goal g^j (if $a_{ij} = 1$) by computing the shortest path, ignoring all other agents. A helper function *assignment*(C) computes an optimal

Algorithm 1: high-level of CBS-TA

```
Input: Graph, start and goal locations, assignment matrix
Result: optimal path for each agent
R.constraints ← ∅
R.assignment ← firstAssignment()
R.root ← True
R.solution ← find individual paths using low-level()
R.cost ← SIC(R.solution)
insert R to OPEN
while OPEN not empty do
    P ← best node from OPEN // lowest solution cost
    Validate the paths in P until a conflict occurs.
    if P has no conflict then
        return P.solution // P is goal
    if P.root is True then
        R ← new node
        R.constraints ← ∅
        R.assignment ← nextAssignment()
        R.root ← True
        R.solution ← find individual paths using low-level()
        R.cost ← SIC(R.solution)
        insert R to OPEN
    $Conflict ← (a_i, a_j, v, t)$ first conflict in P
    for agent $a_i$ in $Conflict$ do
        Q ← new node
        Q.constraints ← P.constraints + $(a_i, s, t)$
        Q.assignment ← P.assignment
        Q.root ← False
        Q.solution ← P.solution
        Update Q.solution by invoking low-level($a_i$)
        Q.cost ← SIC(Q.solution)
        Insert Q to OPEN
```

assignment for a given cost matrix C; this can be achieved, for example, by the Hungarian method [10] or flow-based approaches [1]. We introduce a new function $constrainedAssignment(I, O, C)$, where I is the set of assignments that must be part of the solution, O is the set of assignments that cannot be part of the solution, and C is the cost matrix. This function can be implemented as follows. First, we compute another cost matrix C' such that C' is identical to C, except that we change the cost to 0 for each entry in I and to infinity for each entry in O. Second, we execute any optimal assignment algorithm (e.g., the Hungarian Method) using C'. The pseudo code of our next-best assignment functions are shown in Algorithms 2 and 3.

The central idea of the algorithm is to partition the solution space such that we forbid some assignments and forcefully include others. It has been shown that such a partitioning covers the complete solution space [15]. If the Hungarian Method is used and $N = M$, the complexity for finding the next solution is $O(N^4)$.

Algorithm 2: firstAssignment

```
Input: cost matrix $C$
Result: best assignment, initial ASG_OPEN
$R$ ← new node
$R.O ← ∅$
$R.I ← ∅$
$R.solution = constrainedAssignment(R.I, R.O, C)$
Insert $R$ to ASG_OPEN
return $R.solution$
```

Algorithm 3: nextAssignment

```
Input: cost matrix $C$, ASG_OPEN
Result: next best assignment, updated ASG_OPEN
$P$ ← best node from ASG_OPEN // lowest solution cost
if $P$ does not exist then
    return No next assignment
for $i ← 1$ to $N$ do
    if $i$ not part of $P.I$ then
        $Q$ ← new node
        $Q.O = P.O ∪ \{P.solution[i]\}$
        $Q.I = P.I ∪ \{P.solution[j] : j < i\}$
        $Q.solution = constrainedAssignment(Q.I, Q.O, C)$
        if $Q.solution$ not empty then
            Insert $Q$ to ASG_OPEN
return solution of best node from ASG_OPEN
```

4.2 Properties of CBS-TA

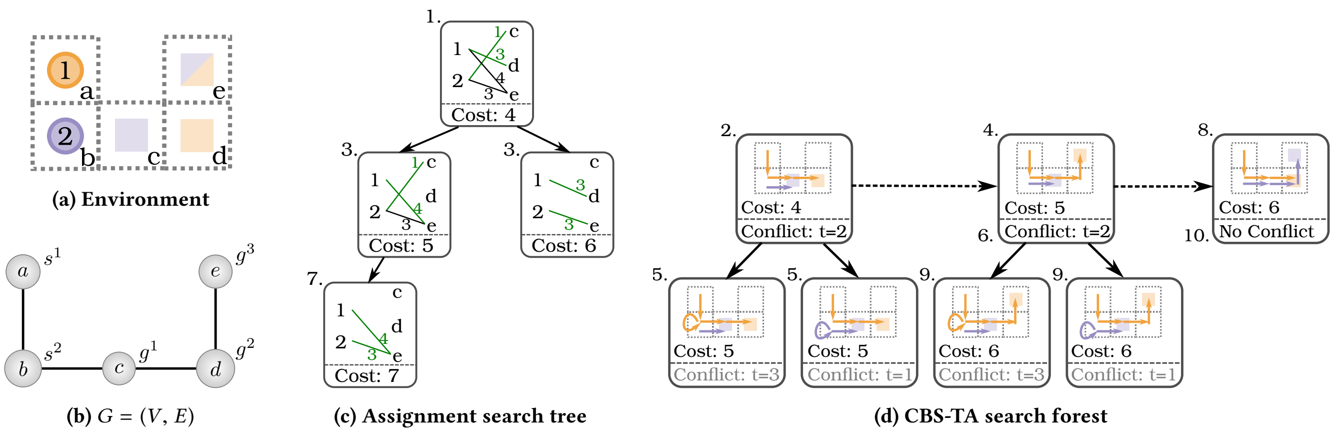
In the following we show that CBS-TA, like CBS, is complete and optimal with respect to sum-of-cost.

Theorem 4.1. *CBS-TA is complete.*

Proof. It has been shown that CBS will return an optimal solution if one exists [17]. CBS-TA performs a CBS search on each root node. Whenever a root node is expanded the next best possible assignment is computed, until all possible assignments have been enumerated. Thus, the search is exhaustive in both task assignment and path planning. □

Theorem 4.2. *CBS-TA computes a solution that minimizes the sum of individual costs of all agents if one exists.*

Proof. If the assignment is fixed, the cost of each root node in the high-level search is a lower bound on the real cost (proof: Lemma 1, [17]). CBS-TA expands assignments in increasing cost order, therefore all expanded high-level nodes are a lower bound on the optimal cost. During each high-level search node expansion, the minimum cost either stays the same or increases because of the best-first expansion order in the high-level search. A different assignment can only be part of the optimal solution if its lower cost bound is identical or smaller than the current minimum cost in the high-level search. However, in this case this assignment was already added as new root node, because a previous root node (as a lower bound for its fixed assignment) must have been expanded. □



(a) Environment

(b) $G = (V, E)$

(c) Assignment search tree

(d) CBS-TA search forest

Figure 2: Example execution of CBS-TA. The environment (a) can be represented as graph (b) with some vertices being start and/or goal vertices. During the search, ASG_OPEN and OPEN are incrementally updated. The former can be visualized as a tree (c), while the latter forms a forest (d). See section 4.3 for additional details.

4.3 Example

The algorithm proceeds as follows. Consider an environment with $N = 2$ agents and $M = 3$ goals, see Figure 2a. The problem can be formulated on a graph (see Figure 2b), with an assignment matrix

$$A = \begin{array}{c c} & \begin{array}{ccc} c & d & e \end{array} \\ \begin{array}{c} 1 \\ 2 \end{array} & \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \end{array}.$$

Based on G and A, we can compute the cost matrix C by considering the shortest path for each agent to the respective goal (ignoring all other agents):

$$C = \begin{array}{c c} & \begin{array}{ccc} c & d & e \end{array} \\ \begin{array}{c} 1 \\ 2 \end{array} & \begin{pmatrix} \infty & 3 & 4 \\ 1 & \infty & 3 \end{pmatrix} \end{array}.$$

We labeled the different steps in their respective orders in Figures 2c and 2d. Using the cost matrix C, the first assignment commits agent 1 to goal d and agent 2 to goal c ($[1 \mapsto d, 2 \mapsto c]$) with cost 4. This creates an entry in the Assignment Open List (ASG_OPEN) (step 1) and a node in OPEN (step 2).

The path validation finds a conflict between the two agents at time step 2 (line 9). When expanding a root node, we must also compute the next best assignment and add a new root node (lines 12 – 19). For the next best assignment, we compute two possible successors in the assignment tree: the first one disallows the assignments $O = \{1 \mapsto d\}$ while the second one disallows $O = \{2 \mapsto c\}$ and enforces $I = \{1 \mapsto d\}$ (lines 4 – 11 in Algorithm 3; step 3). In general, there might be up to N successors. The function `nextAssignment` returns the lowest cost option ($[1 \mapsto e, 2 \mapsto c]$). We compute the shortest path for each agent individually based on this assignment and add it to the OPEN list (step 4).

We now try to resolve the first conflict $(1, 2, c, 2)$, by adding additional nodes to the OPEN list (lines 21 – 29). Namely, we consider the case where agent 1 is constrained to not be at node c at time step 2 and the case where agent 2 cannot be at node c at time step 2 (step 5).

In the next iteration we pick the second root node from the OPEN list (step 6).[2] We need to compute the next best assignment ($[1 \mapsto d, 2 \mapsto e]$) and add an additional root node because the node being expanded is a root node (steps 7 and 8). The currently selected node from OPEN has a conflict (node c at time step 2) and we need to attempt to resolve it by adding two additional child nodes (step 9). Finally, we select the third root node from OPEN and return its solution because it is conflict free (step 10).

5 EXTENSIONS

We now show how CBS-TA can be extended to solve problem instances within a suboptimality bound and how it can be applied to additional interesting MAPF variants.

5.1 ECBS-TA

Enhanced CBS (ECBS) is a bounded suboptimal solver for MAPF [3]. ECBS uses focal search in both low- and high-level search algorithms. In focal search, a FOCAL list is maintained alongside the OPEN list. The FOCAL list contains a subset of the entries in the OPEN list, such that the cost of the entries in FOCAL are within a constant factor w of the best cost in OPEN. The low-level search of ECBS is changed in the following way compared to CBS: First, focal search (rather than A*) is used with a second inadmissible heuristic that estimates the number of conflicts. This is used to minimize the number of expected conflicts. Second, the lowest f-value of the low-level OPEN list is returned, in addition to the solution path. On the high-level, a lower bound $LB(n)$ is computed for each node n as the sum of the minimum f-values (from the low-level search). The high-level FOCAL list then only contains entries of the high-level OPEN list whose cost is less than or equal to $w \min LB(n)$. As in the lower-level search, an inadmissible heuristic that counts the number of expected conflicts is used for expansion. Keeping track of the lower bound through both search levels allows ECBS to use a single suboptimality factor w.

[2] We assume the FIFO principle as tie breaker in the OPEN list. An implementation could pick any of the nodes with cost 5.

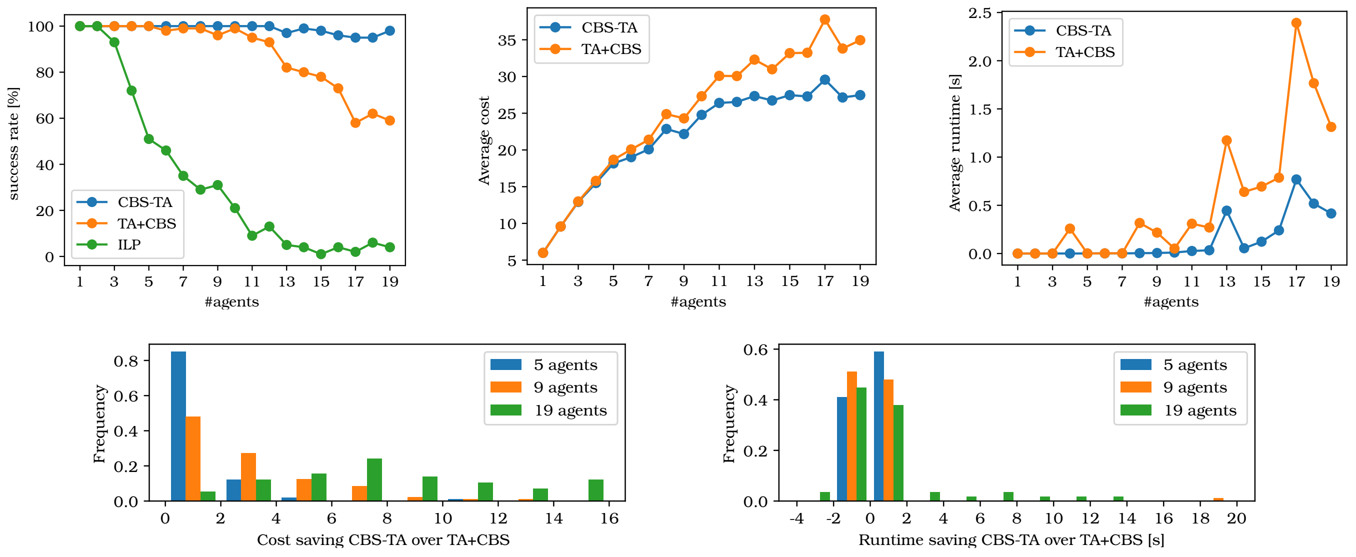


Figure 3: Benchmark results comparing CBS-TA with task assignment followed by CBS (TA+CBS) and ILP. Top: Each data point summarizes 100 randomly generated 8×8 4-connected grids with random start and goal locations. CBS-TA has a higher success rate, while achieving a lower average cost and runtime compared to TA+CBS. Bottom: Histograms comparing cost and runtime savings (i.e., TA+CBS minus CBS-TA) for 5, 9, and 19 agents. For few agents, there is frequently no cost or runtime difference between TA+CBS and CBS-TA, but a few outliers result in better average performance for CBS-TA. For many agents, CBS-TA computes better results in most cases, with large runtime benefits in a few cases.

In order to jointly optimize for task assignment and path planning in the ECBS framework, we can use the same idea as for CBS-TA and generate a search forest with the root nodes referring to different assignments. However, the suboptimality bound w creates slack in the search, allowing us to be more flexible on when to generate additional root nodes. We consider three variants:

- **MaxRoot** Add as many root nodes as possibly useful for the given w. In particular, we keep track of the highest cost of the already expanded root nodes. If that cost is smaller than $w \min LB(n)$, we add all additional root nodes whose cost is no larger than $w \min LB(n)$.
- **CBS-TA-style** Following the same logic as CBS-TA, we add one additional root node each time a root node is expanded.
- **MinRoot** Add as few root nodes as possible, without violating the suboptimality guarantee. In particular, we initially set $r = w \min LB(n)$. We only add an additional root node if the lowest-cost entry in the high-level OPEN list has a cost larger than r. In this case, we compute an additional assignment and update r.

The first variant (MaxRoot) can potentially compute low-cost solutions, even if high suboptimality bounds are used. However, the approach is impractical for large M and N, because there are too many potential assignments. Therefore this method is not implemented. We empirically evaluate CBS-TA-style and MinRoot on various instances, as described in Section 6.2.

5.2 Suboptimal TAPF

Our MAPF formulation permits agents to have a set of possible goals. One example of such a problem is the TAPF problem, in which each agent is part of a group and a set of goals is assigned to each group. TAPF can be solved optimally with respect to the makespan using the Conflict-Based Min-Cost-Flow (CBM) algorithm [12], which uses a two-level search like CBS. Compared to CBS, CBM uses a maximum flow algorithm per group to find paths on the low-level rather than using A* per agent. We can model TAPF problem instances by setting $N = M$ and matrix A according to the group assignment. CBS-TA can compute optimal solutions with respect to the sum of costs, which can be more relevant in some scenarios (e.g. minimizing the total energy usage of the team). It has been shown that makespan and sum of cost cannot be simultaneously optimized [24]. Therefore, we need to consider our optimization objective directly. ECBS-TA can be used to find bounded suboptimal solutions to such problem instances. We present empirical results comparing CBM and ECBS-TA in Section 6.3.

6 EXPERIMENTS

We implement CBS-TA and ECBS-TA in C++ using the boost library for fast heap data structures. We use a minimum-cost maximum-flow formulation that is part of the boost graph library to solve unconstrained assignment problems efficiently. All experiments were executed on a laptop (i7-4600U 2.1 GHz and 12 GB RAM).

6.1 CBS-TA

We use a set benchmark instances to compare CBS-TA to other existing methods. We randomly generated 8×8 4-connected grids with 20% obstacles and with random start and goal locations, such that it is guaranteed that there is at least one assignment where all agents can reach their respective goals. We limit the computation

			ECBS			ECBS-TA (CBS-TA style)			ECBS-TA (MinRoot)		
Grid Size	w	Agents	Success	Cost	Runtime	Success	Cost	Runtime	Success	Cost	Runtime
8×8	1.0	5	**1.00**	18.7	**0.00**	**1.00**	**18.2**	**0.00**	**1.00**	**18.2**	**0.00**
		9	0.96	24.3	**0.01**	**1.00**	**22.2**	**0.01**	**1.00**	**22.2**	**0.01**
		19	0.56	34.3	1.30	**0.98**	**27.2**	**0.31**	**0.98**	**27.2**	0.32
	1.1	5	**1.00**	18.7	**0.00**	**1.00**	**18.3**	**0.00**	**1.00**	18.4	**0.00**
		9	0.97	24.5	0.17	**1.00**	**22.7**	**0.00**	**1.00**	22.9	0.01
		19	0.56	34.6	0.93	**0.99**	**28.9**	**0.10**	**0.99**	29.3	0.11
	1.3	5	**1.00**	18.8	**0.00**	**1.00**	**18.7**	**0.00**	**1.00**	18.9	**0.00**
		9	**1.00**	25.6	0.01	**1.00**	**24.5**	**0.00**	**1.00**	25.4	0.01
		19	0.68	37.9	1.16	**1.00**	**32.8**	**0.05**	0.89	34.5	0.44
32×32	1.00	40	**0.92**	248	**0.5**	0.57	**244**	4.3	0.58	**244**	4.2
		70	**0.32**	283	**1.4**	0.01	**271**	3.0	0.01	**271**	2.8
		100	**0.01**	-	-	0.00	-	-	0.00	-	-
	1.05	40	0.95	264	**0.4**	**0.99**	**262**	0.5	0.95	263	0.7
		70	0.45	352	1.7	**0.52**	**350**	5.5	0.40	351	**1.6**
		100	**0.04**	**359**	2.5	0.03	**359**	16.8	**0.04**	**359**	**1.6**
	1.10	40	0.99	268	0.2	**1.00**	**267**	0.3	0.99	269	**0.1**
		70	**0.84**	371	0.6	0.82	**369**	2.6	0.80	371	**0.5**
		100	**0.33**	**417**	2.6	0.27	**417**	15.3	0.29	418	**2.3**

Table 1: Benchmark results comparing task assignment followed by ECBS with ECBS-TA for different suboptimality bounds *w*. Each data point averages 100 randomly generated 4-connected grids with random start and goal locations.

time to 30 s and mark a trial as a failure if no solution was found within the time limit. We vary the number of agents and report the success rate, average cost, and average runtime over 100 randomly created examples per number of agents. For CBS-TA, we use the shortest distance as heuristic in the low-level search.

6.1.1 *TA+CBS versus CBS-TA.*

We compare CBS-TA to task assignment followed by CBS (TA+CBS). To ensure fair runtime comparison, we implement TA+CBS by executing the same CBS-TA implementation with an artificial limit of a single root-node expansion. Our results (see Figure 3) show that the success rate of CBS-TA is higher compared to TA+CBS. For examples that were successful with both algorithms, we compute the average cost and average runtime. CBS-TA achieves a lower average cost in a shorter average time compared to TA+CBS. We analyze the relative frequency of this effect by looking at individual histograms of the cost savings (that is cost(TA+CBS) - cost(CBS-TA)) and runtime savings, comparing only examples that were successful with both algorithms. With only 5 agents, over 80% of the test cases show no cost difference and the runtime is identical in nearly all cases. With 19 agents, however, the cost improvement peaks at an improvement of 7 (over 20% of the examples), while the runtime is identical in over 75% of the cases. This shows, that CBS-TA is in particular beneficial for dense cases, where the task assignment and path planning are more tightly coupled. Additionally, CBS-TA does not seem to require additional runtime, even for sparser examples.

6.1.2 *CBS-TA versus ILP.*

Integer Linear Program (ILP) formulations have been used for the non-anonymous MAPF problem minimizing different objectives including makespan and sum-of-cost [25]. The idea behind such formulations is to construct a time-expanded flow graph and formulate a multi-commodity flow problem. We implement an ILP based on this idea assuming $M = N$ and a fully anonymous assignment. This is challenging for CBS-TA (because $N!$ possible assignments have to be considered), but easier for the ILP formulation because it can be framed as a single commodity flow. Such instances can be solved in polynomial time when optimizing for makespan [14]. In order to be able to minimize for the sum-of-cost instead, we use the following steps. First, we generate the time-expanded flow graph and formulate an ILP, similar to [25], but using a single commodity for all agents, rather than one commodity per agent. Second, we add one additional auxiliary integer variable for each goal capturing the time until an agent reaches and stays at that goal. Third, we set our optimization objective to minimize the sum of all such auxiliary variables [14].

We use Gurobi 7.5 as ILP solver [8]. In order to solve an instance, we need an upper bound of the makespan of the optimal solution. We find an upper bound dynamically, by doubling the makespan on each attempt. Only if the cost between two successive attempts did not change do we report a solution. This avoids solutions where the makespan but not sum-of-cost is minimal.

The ILP solver computes results with the same minimum cost in all solved cases, as expected. However, the runtime is significantly higher compared to CBS-TA. For example, the average runtime for 10 agents is 21 s. This also affects the success rate (see Figure 3), which is significantly lower compared to CBS-TA for larger numbers of agents.

6.2 ECBS-TA

We use benchmarks to compare the two ECBS-TA variants to optimal task assignment followed by ECBS using different environment sizes and suboptimality bounds. Note that even though we use the same suboptimality bound for ECBS-TA and ECBS, they have different semantics. In the ECBS-case we guarantee that the returned result is within a factor of w given that the task assignment is fixed.

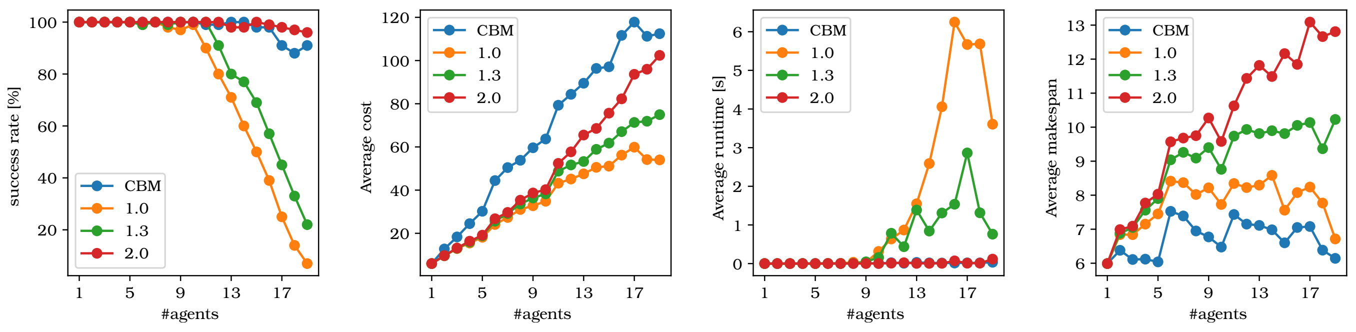


Figure 4: Benchmark results comparing CBM with ECBS-TA for 8×8 4-connected grid environments. ECBS-TA achieves a higher success rate at comparable runtime with $w = 2$. For all used suboptimality bounds the achieved sum of costs is lower when using ECBS-TA when compared to CBM. However, because CBM optimizes makespan, it outperforms ECBS-TA with respect to makespan.

ECBS-TA, on the other hand, guarantees to return a solution that is within a factor of w for the optimal valid task assignment. Thus, the guarantee given by ECBS-TA is stronger. In all cases we numerically verified that the suboptimality bounds are fulfilled.

6.2.1 Small Environments. In the first set of tests, we use the same 8×8 4-connected grids with 20% obstacles as for the CBS-TA analysis in the previous section. We vary the number of agents and report the success rate, average cost, and average runtime over 100 examples per numbers of agents. A subset of our results is shown in Table 1. Both ECBS-TA variants achieve higher success rates, lower or comparable costs, and lower runtime compared to ECBS when used with the same suboptimality bound. When comparing the two different ECBS-TA versions, we notice that the CBS-TA style root-node expansion results in lower costs at higher suboptimality bounds. In case ($w = 1.3$), this version also provides a higher success rate at a lower runtime, compared to the MinRoot expansion policy.

6.2.2 Large Environments. In another set of tests (see Table 1) we used 32×32 4-connected grids, again with 20% obstacles. Instead of up to 20 robots we test with up to 100 robots. This results in longer required paths for each robot, but has a lower robot-to-free-space density of 12% compared to the 38% of the smaller maps.

When computing the optimal solution ($w = 1$), the success rate of ECBS-TA (both variants) is now significantly lower than ECBS. For instances that could be solved by all variants, the solution found by ECBS-TA has a lower cost, but not significantly (less than 5% on average). Higher suboptimality bounds ($w = 1.05$ and $w = 1.1$) improve the success rate of ECBS-TA to a comparable level and the solution quality is nearly identical for ECBS and ECBS-TA. However, the runtime of ECBS-TA using the CBS-TA style expansion is significantly higher and grows quickly with the number of agents. Our MinRoot expansion on the other hand has the same (and sometimes better) runtime than ECBS.

This effect can be explained as follows. The number of possible task assignments grows factorially in the number of robots. Thus, instances with many robots have many possible assignments with identical cost (we noticed several hundred possible assignments with optimal cost for some of our examples). Adding another root node in the ECBS forest is time-consuming, because another assignment needs to be computed and low-level search for each robot for this assignment needs to be executed. Therefore, our CBS-TA style expansion will create many additional root nodes, but those root nodes do not help significantly to find lower cost solutions. The MinRoot expansion delays creating additional root-nodes as long as possible for the given w. High suboptimality bounds might not trigger the creation of any additional root node.

Consequently, instances with many robots should use ECBS-TA with the MinRoot expansion. This provides stronger suboptimality guarantees compared to ECBS and better results with low suboptimality bounds. At the same time ECBS-TA achieves the same results in terms of runtime and cost as ECBS for high suboptimality bounds.

6.3 TAPF

Thus far our experiments have only considered the unlabeled case. We now evaluate cases where the target assignment is more constrained. To be able to compare to a baseline we set $N = M$ and arrange agents into groups, such that agents within the same group are interchangeable. We solve the same problem instance using the Conflict-Based Min-Cost-Flow (CBM) algorithm [12], and compare it with ECBS-TA (MinRoot expansion) with varying suboptimality bounds. Both algorithm use different objective functions: CBM finds solutions with minimal makespan, while ECBS-TA minimizes the sum of cost up to a given suboptimality factor. The CBM implementation is written in C++ and uses boost graph as well.

6.3.1 Small Environments. In the first set of tests we run both algorithms on the same 8×8 maps with varying number of agents, but fixed group size of 5 agents per group.[3] We report the success rate, average cost, runtime, and makespan for CBM and ECBS-TA with different suboptimality bounds ($w = 1.0, 1.3, 2.0$), see Figure 4. We notice that the achieved average cost (that is, the sum of the individual agent costs) is smaller for ECBS-TA in all cases, while the makespan (as the metric being optimized in CBM) is lowest for CBM. When we choose $w = 2.0$, ECBS-TA achieves a higher success rate at comparable runtimes compared to CBM.

[3]The last group may have less than 5 agents.

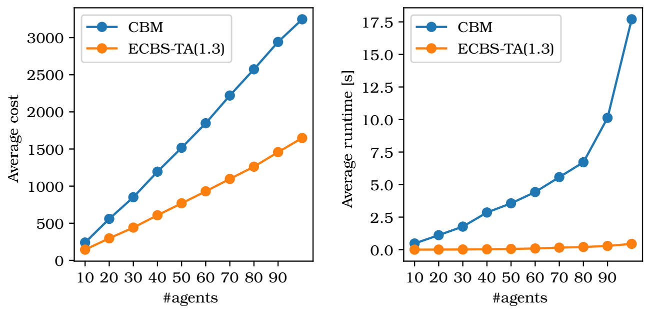


Figure 5: Benchmark results comparing CBM with ECBS-TA for 32×32 4-connected grid environments. ECBS-TA achieves lower cost and runtime compared to CBM.

Using lower suboptimality bounds results in lower cost solutions, but finding solutions is significantly less successful and requires longer runtimes compared to CBM.

6.3.2 Large Environments. We use the same fixed group size of 5 agents per group on the larger 32×32 maps that were also used in the ECBS-TA experiments. Using ECBS-TA to compute cost-optimal solutions ($w = 1.0$) leads to a low success rate (no instance with 100 agents can be solved within the timeout), while bounds with $w \geq 1.3$ can solve all instances. In comparison, CBM solved 995 out of the 1000 test instances in the given time limit. For brevity, we report the results for $w = 1.3$ in Figure 5 on instances that were solved by both algorithms. The achieved cost of CBM is more than twice as high compared to ECBS-TA in the 100 agent case. It is surprising that the difference is that large considering the relatively high suboptimality bound used for ECBS-TA. Not surprisingly, CBM achieves a lower makespan (the metric it is minimizing): in the 100 agent case CBM has an average makespan of 33 while ECBS-TA achieves an average makespan of 47. The runtime of ECBS-TA is significantly better compared to CBM, especially when more agents are considered.

6.3.3 Varying Group Size. To evaluate the influence of the group size, we use the 32×32 maps with 100 agents while varying the group size. The results (using $w = 1.3$ for ECBS-TA) are shown in Figure 6. As before, we limit the computation time to 30 s. ECBS-TA was able to compute solutions for all cases within the given time limit. If using 100 groups (group size of 1), CBM was not able to compute a single solution, but it was successful for all other group sizes and instances. If the group size is 1, we get the labeled case where each agent has an assigned goal. The other extreme is the unlabeled case, which we achieve with a group size of 100. CBM uses a maximum flow algorithm in the low-level search that is executed per robot group. Thus, it performs best if there is just a single group (of size 100) and worst if there are 100 groups (of size 1). ECBS-TA, on the other hand, considers a single possible assignment if there are 100 groups, and 100! assignments if there is a single group. The runtime results reflect this behavior: ECBS-TA can find solutions much faster for small group sizes compared to CBM. The runtime requirements slowly increase for larger group sizes. The runtime of CBM decreases with the group size. As in the previous results, the cost of the solution is lower for ECBS-TA in all cases. Similarly, as expected, the makespan of CBM solutions is always better than for ECBS-TA solutions.

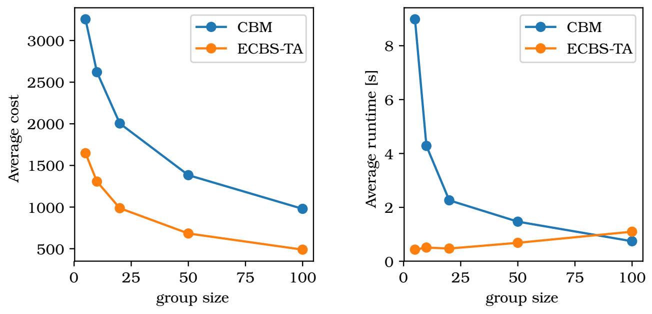


Figure 6: Benchmark results comparing CBM with ECBS-TA for varying group sizes. The runtime of ECBS-TA slowly grows, while the runtime of CBM rapidly decays with the group size. The average cost of the solution is always smaller when using ECBS-TA, independent of the group size.

7 CONCLUSION

In this work, we extended Conflict-Based Search to simultaneously assign tasks and plan paths for multiple agents. The key insight is the extension of the high-level search to operate on a search forest rather than a search tree, where each root node represents a fixed assignment. The forest can be efficiently constructed on demand, avoiding the need to consider all irrelevant possible task assignments. The use of the CBS framework provides two significant advantages. First, other extensions of CBS, such as the bounded suboptimal ECBS, are directly applicable. Second, we can optimize for the sum of individual costs, which is more appropriate in some domains than makespan.

We evaluated our algorithm extensively, with the following important results:

(1) We can compute solutions significantly faster than an ILP-based solver while providing the same optimality guarantees.
(2) The traditional method of independently assigning tasks followed by path planning can be improved in terms of solution quality and runtime by using (E)CBS-TA in dense environments with few agents. However, larger environments with many agents do not benefit significantly from a joint optimization. Nevertheless, ECBS-TA (MinRoot expansion) provides stronger suboptimality guarantees than before with negligible additional runtime overhead.
(3) ECBS-TA produces lower cost solutions than CBM (which optimizes for a different objective) even when high suboptimality bounds are used. For small group sizes, ECBS-TA can produce such a solution in significantly shorter time compared to CBM.

We believe that (E)CBS-TA can be used in all cases where task assignment and path planning might be optimized jointly with respect to the sum of costs of the individual agents' plans.

In future work, we would like to apply (E)CBS-TA on more realistic scenarios, such as planning for robots in warehouses.

REFERENCES

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[2] Ofra Amir, Guni Sharon, and Roni Stern. 2015. Multi-Agent Pathfinding as a Combinatorial Auction. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 2003–2009. http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9572

[3] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. 2014. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *Annual Symposium on Combinatorial Search (SOCS)*. AAAI Press. http://www.aaai.org/ocs/index.php/SOCS/SOCS14/paper/view/8911

[4] Chandra R. Chegireddy and Horst W. Hamacher. 1987. Algorithms for Finding K-best Perfect Matchings. *Discrete Applied Mathematics* 18, 2 (1987), 155–165. https://doi.org/10.1016/0166-218X(87)90017-5

[5] Liron Cohen, Tansel Uras, and Sven Koenig. 2015. Feasibility Study: Using Highways for Bounded-Suboptimal Multi-Agent Path Finding. In *Annual Symposium on Combinatorial Search (SOCS)*. AAAI Press, 2–8. http://www.aaai.org/ocs/index.php/SOCS/SOCS15/paper/view/11301

[6] Boris de Wilde, Adriaan ter Mors, and Cees Witteveen. 2014. Push and Rotate: a Complete Multi-agent Pathfinding Algorithm. *Journal of Artificial Intelligence Research (JAIR)* 51 (2014), 443–492. https://doi.org/10.1613/jair.4447

[7] David Eppstein. 2016. *k-Best Enumeration*. Springer, 1003–1006. https://doi.org/10.1007/978-1-4939-2864-4_733

[8] Inc. Gurobi Optimization. 2016. Gurobi Optimizer Reference Manual. (2016). http://www.gurobi.com

[9] Wolfgang Hönig, T. K. Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. 2016. Multi-Agent Path Finding with Kinematic Constraints. In *International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 477–485. http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13183

[10] Harold W. Kuhn. 1955. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly* 2, 1-2 (1955), 83–97. https://doi.org/10.1002/nav.3800020109

[11] Michail G. Lagoudakis, Evangelos Markakis, David Kempe, Pinar Keskinocak, Anton J. Kleywegt, Sven Koenig, Craig A. Tovey, Adam Meyerson, and Sonal Jain. 2005. Auction-Based Multi-Robot Routing. In *Robotics: Science and Systems (RSS)*. The MIT Press, 343–350. http://www.roboticsproceedings.org/rss01/p45.html

[12] Hang Ma and Sven Koenig. 2016. Optimal Target Assignment and Path Finding for Teams of Agents. In *International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*. ACM, 1144–1152. http://dl.acm.org/citation.cfm?id=2937092

[13] Hang Ma, Jiaoyang Li, T. K. Satish Kumar, and Sven Koenig. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*. ACM, 837–845. http://dl.acm.org/citation.cfm?id=3091243

[14] Hang Ma, Craig A. Tovey, Guni Sharon, T. K. Satish Kumar, and Sven Koenig. 2016. Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 3166–3173. http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12437

[15] Katta G. Murty. 1968. Letter to the Editor – An Algorithm for Ranking all the Assignments in Order of Increasing Cost. *Operations Research* 16, 3 (1968), 682–687. https://doi.org/10.1287/opre.16.3.682

[16] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. 2012. Conflict-Based Search For Optimal Multi-Agent Path Finding. In *AAAI Conference on Artificial Intelligence*. AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5062

[17] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219 (2015), 40–66. https://doi.org/10.1016/j.artint.2014.11.006

[18] Pavel Surynek. 2010. An Optimization Variant of Multi-Robot Path Planning Is Intractable. In *AAAI Conference on Artificial Intelligence*. AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1768

[19] Glenn Wagner and Howie Choset. 2011. M*: A Complete Multirobot Path Planning Algorithm with Performance Bounds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 3260–3267. https://doi.org/10.1109/IROS.2011.6095022

[20] Glenn Wagner and Howie Choset. 2015. Subdimensional expansion for multirobot path planning. *Artificial Intelligence* 219 (2015), 1–24. https://doi.org/10.1016/j.artint.2014.11.001

[21] Glenn Wagner, Howie Choset, and Nora Ayanian. 2012. Subdimensional Expansion and Optimal Task Reassignment. In *Annual Symposium on Combinatorial Search (SOCS)*. AAAI Press. http://www.aaai.org/ocs/index.php/SOCS/SOCS12/paper/view/5390

[22] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine* 29, 1 (2008), 9–20. http://www.aaai.org/ojs/index.php/aimagazine/article/view/2082

[23] Jingjin Yu and Steven M. LaValle. 2012. Multi-agent Path Planning and Network Flow. In *Workshop on the Algorithmic Foundations of Robotics (WAFR) (Springer Tracts in Advanced Robotics)*, Vol. 86. Springer, 157–173. https://doi.org/10.1007/978-3-642-36279-8_10

[24] Jingjin Yu and Steven M. LaValle. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI Conference on Artificial Intelligence*. AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6111

[25] Jingjin Yu and Steven M. LaValle. 2016. Optimal Multirobot Path Planning on Graphs: Complete Algorithms and Effective Heuristics. *IEEE Transactions on Robotics* 32, 5 (2016), 1163–1177. https://doi.org/10.1109/TRO.2016.2593448