DisCoF: Cooperative Pathfinding in Distributed Systems with Limited Sensing and Communication Range

Yu Zhang, Kangjin Kim and Georgios Fainekos

Abstract Cooperative pathfinding is often addressed in one of two ways in the literature. In fully coupled approaches, robots are considered as a single entity and the paths for all robots are constructed simultaneously. In decoupled approaches, the paths are constructed only for a subset of robots at a time. While decoupled approaches are much faster than fully coupled approaches, they are often suboptimal and incomplete. Although there exists a few decoupled approaches that achieve completeness, global information is assumed, which may not be accessible in distributed robotic systems. In this paper, we provide a window-based approach, called DisCoF, for cooperative pathfinding with limited sensing and communication range in distributed systems. In DisCoF, robots are assumed to be fully decoupled initially, and may gradually increase the level of coupling in an online fashion. DisCoF can naturally transition robots to be fully coupled when global information is actually needed to solve the current problem instance. DisCoF is an online approach since robots in most cases are only aware of a subset of other robots in the environment at a given time, and hence do not have enough information to determine non-conflicting paths with all the other robots. Completeness analysis of DisCoF is performed.

1 INTRODUCTION

Cooperative pathfinding for multi-robot systems has many applications. However, the associated problem is fundamentally hard in general (i.e., PSPACE-hard [6]). Previous approaches often address this problem in one of two ways. In fully coupled approaches, robots are considered as a single entity and the paths for all robots are constructed simultaneously. However, given the complexity of the problem, these approaches can easily become impractical. As a result, previous research more often

Yu Zhang, Kangjin Kim and Georgios Fainekos
School of Computing, Informatics and Decision Systems Engineering, Arizona State University
e-mail: {yzhan442, Kangjin.Kim, fainekos}@asu.edu

concentrates on decoupled approaches. In decoupled approaches, the paths are constructed only for a single robot or a subset of robots at a time; the remaining robots must then take these previously constructed paths into account when constructing their own paths. While decoupled approaches are often suboptimal and incomplete, they typically run much faster than fully coupled approaches, since only a small number of robots needs to be coupled at a time.

There are decoupled approaches that achieve optimality and completeness. However, they assume global information. Global information may not be accessible in distributed robotic systems, since such systems are often subject to limited sensing and communication range. As a result, these approaches cannot be implemented on many distributed systems.

In this paper, we introduce a window-based approach for cooperative pathfinding in distributed systems, with the window size corresponding to the limited sensing and communication range. This approach, called DisCoF, is inherently online, since a robot may not be aware of all the other robots in the environment at a given time, let alone determining a non-conflicting path with them. In DisCoF, a robot can only communicate to coordinate with other robots within its *local window*; however, two robots can coordinate indirectly through other robots using a message relay protocol. All robots are assumed to be fully decoupled initially: they plan and execute their paths independently and simultaneously. Robots can gradually increase the level of coupling in an online fashion.

To reduce coupling in order to reduce computation, we need to determine when to couple robots and only couple them when necessary. We follow an intuitive idea to achieve this: coupling robots only when they have *predictable conflicts*. Furthermore, to reduce the likelihood of future coupling based on local knowledge, instead of planning full paths to their final goals, robots in each coupling only plan to *local goals* that minimize conflicts within a pre-specified horizon. This planning process also ensures that these robots make progress to their final goals.

However, given the localized nature of this approach, it is subject to live-locks. We identify live-locks when robots in a coupling cannot make further progress within a finite horizon, which is a necessary (but insufficient) condition to detect live-locks. Note that detecting live-locks requires global information in general. DisCoF allows multiple "live-locks" to be detected, which are resolved by the corresponding couplings simultaneously. When a live-lock is detected, robots in the coupling use a technique to keep them within each other's sensing and communication range, while progressing to their goals one at a time. Using this technique, DisCoF achieves completeness and can often find a solution before transitioning robots to be fully coupled, which is needed when global information is actually needed to solve the current problem instance. To the best of our knowledge, this is the first work that guarantees completeness for cooperative pathfinding with limited sensing and communication range in distributed systems.

The remainder of this paper is organized as follows. After a brief review of related literature in Sec. 2, we introduce DisCoF in Sec. 3. The live-lock resolution technique is discussed separately in Sec. 4. Preliminary results are provided in Sec. 5. Conclusions and discussions of future work are presented afterwards.

2 RELATED WORK

The most convenient way to address cooperative pathfinding is to consider robots as fully coupled, since then many existing state-space search algorithms (e.g., A*) can be applied. While this fully coupled search is intractable, approaches have been provided to reduce the branching factors to improve the performance, e.g., [15]. There are also approaches that compile cooperative pathfinding problems into related problem formulations [8, 1, 5, 19], and then apply the corresponding algorithms to solve them. However, these approaches are unscalable due to the large state spaces. Methods for spatial abstraction to reduce the state space have also been discussed [17, 13], but they often suffer optimality or even completeness. By restricting the underlying graphs of problem instances to have certain topologies, optimal solutions can be found fast [18, 11, 12].

Meanwhile, much of the previous research has been concentrated on decoupled approaches, due to its better scalability. One commonly used approach is the hierarchical cooperative A* (HCA* [14]), which represents the prioritized planning method. HCA* chooses fixed priorities for robots and plans a path for a single robot at a time based on its priority, while respecting the computed paths for robots of higher priorities. This process is performed through the use of a reservation table that all robots can access. To reduce the influence of the computed paths for robots of higher priorities (on robots of lower priorities), a windowed HCA* approach (WHCA*) is also introduced. In WHCA*, robots only send the portions of their paths within a fixed window size (from their current locations) to the reservation table, which has been shown to enable WHCA* to solve more problem instances. More recently, an extension of WHCA* (CO-WHCA* [2]) is proposed, which improves over WHCA* by only reserving paths when there are conflicts. Another common decoupled approach is to create traffic laws for the robots to follow, e.g., [7], thus reducing the possibility of conflicts. Although these decoupled approaches can often find solutions fast, they are incomplete.

There are decoupled approaches that achieve completeness and optimality, e.g., [15]. However, they are still intractable for many problem instances. Hence, more recent approaches often relax the optimality to achieve practicality [9, 4, 16]. In [9], a Push and a Swap operation are introduced, which are used to move robots to their goals one at a time; the resulting individual plans are then optimized for all robots. The authors in [4] further introduce a new operation, Rotate, to complement Push and Swap, in order to ensure completeness in more general problem instances. These approaches, however, assume global information (e.g., the plans of all robots at any time). Global information may not be accessible in distributed robotic systems, in which each robot must often create its individual path based on its local knowledge and the (limited) communicated information.

There are research works proposed for decentralized multi-robot systems, but the emphasis has been on distributed computation (e.g., sharing the collision check [10]). In this paper, we introduce an approach that achieves completeness without assuming global information in distributed systems. While cooperative pathfinding with limited sensing and communication range has been investigated before, e.g.,

[3], to the best of our knowledge, guarantee of completeness has never been provided. The difficulty lies partly in the existence of live-locks, as global information is required to detect them in general. Note that cooperative pathfinding with only local information can be considered as a special case of pathfinding with dynamic obstacles. To achieve completeness, our approach allows robots to gradually increase the level of coupling when potential live-locks are detected.

3 DisCoF

In this section, we first provide the problem formulation. The concept of *local window* and the coupling mechanism in DisCoF are then discussed. All proofs are omitted and can be found in the technical report online.[1]

3.1 Problem Formulation

Given a graph $G(V,E)$ and a set of robots $\mathcal{R}$, the initial locations of the robots are denoted as $\mathcal{I} \subseteq V$ and the goals are denoted as $\mathcal{G} \subseteq V$. Edges in E are undirected. Any robot can move to any adjacent vertex in one time step or remain where they are. A plan $\mathcal{P}$ is a set of individual plans of robots, and $\mathcal{P}[i]$ denotes the individual plan for robot $i \in \mathcal{R}$. Each individual plan is composed of a sequence of actions. In this paper, for simplicity in the presentation, each action is simply the next vertex to be visited. We denote by $\mathcal{P}_k[i]$ $(k \geq 1)$ the action to be taken at time step $k-1$ (or the vertex to be visited at k) for robot i, and by $\mathcal{P}_{k,l}[i]$ $(k \leq l)$ the subplan that results by considering only the actions $\mathcal{P}_k[i]$ up to $\mathcal{P}_l[i]$. The goal of cooperative pathfinding is to find a plan $\mathcal{P}$ such that robots start in $\mathcal{I}$ and end in $\mathcal{G}$ after executing it, without any conflicts. The state of robots (i.e., locations) at time step k is denoted by $\mathcal{S}_k$, and the state of robots after executing plan $\mathcal{P}$ from a state $\mathcal{S}$ is denoted by $\mathcal{S}(\mathcal{P})$. Thus, we have $\mathcal{S}_0 = \mathcal{I}$, $\mathcal{S}_0(\mathcal{P}) = \mathcal{G}$ and $\mathcal{S}_k = \mathcal{S}_0(\mathcal{P}_{1,k})$. A conflict happens at time step k, if two robots are in the same location, or their locations at $k-1$ are exchanged. Formally,

$$\mathcal{S}_k[i] = \mathcal{S}_k[j] \vee (\mathcal{S}_k[i] = \mathcal{S}_{k-1}[j] \wedge \mathcal{S}_{k-1}[j] = \mathcal{S}_k[i]) \tag{1}$$

in which $i \in \mathcal{R}$, $j \in \mathcal{R}$ and $i \neq j$.

Each robot can independently compute a plan (without considering other robots) to its goal using the same local planner, which computes a shortest path. The plan (i.e., a sequence of actions) output by this planner is denoted as $P(u,v)$, for a plan that moves a robot from vertex u to v. The length of $P(u,v)$ is denoted as $\mathcal{C}(u,v)$, i.e., $\mathcal{C}(u,v) = |P(u,v)|$. Furthermore, we make the following assumptions:

[1] www.public.asu.edu/%7eyzhan442/DisCoF/report.pdf

1. Robots are homogeneous and have the same sensing and communication range (this assumption just simplifies the presentation and it can be relaxed).
2. Robots are equipped with a communication protocol that allows them to efficiently relay messages.
3. Time steps are synchronized (asynchronous time steps are to be investigated in future work).
4. Only distance (i.e., length of the shortest path while ignoring obstacles) influences a robot's observability of other robots.
5. Each robot has full knowledge of the environment, i.e., graph G, but it is only aware of robots that it can sense or communicate with.

The individual plans are constructed and updated in an online fashion. Initially, for each robot i, the individual plan is constructed as $\mathcal{P}[i] = P(\mathcal{I}[i], \mathcal{G}[i])$. Robots then start execute their individual plans until conflicts can be predicted (discussed later). In such cases, the individual plans of robots that are involved are updated from $\mathcal{P}_{k+1}$ to avoid these conflicts, given that the current time step is k.

3.2 Local Window

While the window size in WHCA* [14] is a parameter to determine the number of plan steps (from the current state) to be sent by each robot to the reservation table, the window size in DisCoF represents the sensing and communication range of the robot. To reduce communication, we only allow a robot to communicates with other robots when it can sense them. Robots need to coordinate with others within their *local windows* to predict and resolve potential conflicts through communication.

Definition 1 (Local Window). At time step k, the local window of robot $i \in \mathcal{R}$, denoted by $\mathcal{W}_k[i]$, is defined as $\mathcal{W}_k[i] = \{v \in V \mid v$ can be reached from $\mathcal{S}_k[i]$ in λ steps$\}$, in which λ is the window size, which is a positive integer and $\lambda > 1$.

When a robot j satisfies $\mathcal{S}_k[j] \in \mathcal{W}_k[i]$, we write $i \rhd_k j$ to indicate that robot i can send messages to robot j. Given our assumptions, $\rhd_k$ is symmetric, i.e., i and j can communicate with each other. We indicate this symmetric relation as $i \lhd\rhd_k j$. Furthermore, given the communication relay protocol, $\lhd\rhd_k$ also defines a transitive relation. Namely, if $i \lhd\rhd_k r$ and $r \lhd\rhd_k j$, we also have that $i \lhd\rhd_k j$.

Definition 2 (Coordination Graph). At time step k, the coordination graph $G^*_k(V^*_k, E^*_k)$ of the robots is constructed as follows:

- $V^*_k = \mathcal{R}$.
- $(i, j) \in E^*_k$ if and only if $i \lhd\rhd_k j$.

Note that coordination graph is only a structure introduced to facilitate our following discussions. Robots are not required to obtain all the information about this graph in DisCoF.

Definition 3 (Outer Closure (OC)). At time step k, the coordination graph G_k^* is partitioned into disjoint connected subgraphs. Denote Φ_k as the set of vertex sets of these subgraphs. Then, for any $(\phi^x, \phi^y) \in \{\Phi_k, \Phi_k\}$, the following is satisfied:

$\forall (i, j) \in \{\mathcal{R}, \mathcal{R}\}$, $i \neq j$, $i \in \phi^x$, $j \in \phi^y$, we have $i \lhd\rhd_k j$ if and only if $x = y$.

Each $\phi \in \Phi_k$ defines an outer closure.

Two robots in different outer closures are unaware of each other and hence cannot communicate each other's plan or current state. Hence, we aim to coordinate robots within each OC.

Definition 4 (Predictable Conflicts). At time step k, given $\phi \in \Phi_k$, we define that a robot $i \in \phi$ has a *predictable conflict* with parameter δ, if it would involve in a conflict at $k+\delta$ ($\delta \leq \beta$, in which β is a pre-specified finite horizon) with another robot in ϕ, and that i would not involve in any conflicts with robots in ϕ at any time step earlier than $k+\delta$, assuming that robots in ϕ continue their current individual plans.

The reason for imposing a finite horizon is due to the partial observability of the environment from the robots, since resolution for potential conflicts in the far future is likely to only waste computation resource and time. Note that parameters λ and β do not have to be related.

At time step k, if a robot i has a predictable conflict with parameter δ, we denote it as $\Delta_k^i(\delta)$. We also use Δ_k^i when the parameter does not need to be identified. Predictable conflicts are associated with the notion of inner closure.

Definition 5 (Inner Closure (IC)). At time step k, the IC ψ of an OC $\phi \in \Phi_k$ is the set of robots that satisfy: $\psi = \{i \mid \Delta_k^i \wedge i \in \phi\}$.

Similarly, we denote Ψ_k as the set of ICs for OCs at time step k. Note that the IC of an OC may be empty. We provide an example of OC and IC below.

Example 1 *Fig. 1 visualizes such a scenario for $\lambda = 2$ and $\beta = 2$. Robots are shown at their initial locations at time step 0. The arrows indicate their respective individual plans for the next few steps and the highlighted gray areas their local windows. We have $r_2 \lhd\rhd_0 r_3$ and $r_3 \lhd\rhd_0 r_4$ and hence $r_2 \lhd\rhd_0 r_4$ through r_3. Thus, the OC at time 0 are $\phi^1 = \{r_2, r_3, r_4\}$, and $\phi^2 = \{r_1\}$. Even though r_2, r_3, r_4 are in the same OC, only r_3 and r_4 belong to the IC since there is a predictable conflict with parameter 1. Hence, $\psi^1 = \{r_3, r_4\}$ and $\psi^2 = \emptyset$.*

3.3 Coupling in OC

Given an OC with predictable conflicts, the goal of coupling is to update the individual plans of robots to proactively resolve these conflicts while avoiding introducing new conflicts in the finite horizon (i.e., specified by β).

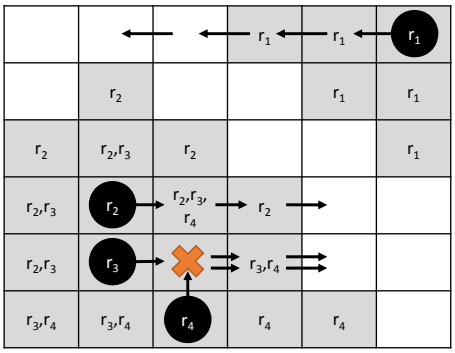


Fig. 1 Scenario that illustrates OC and IC. Two OCs are present, out of which one has a predictable conflict.

At time step k, suppose that conflicts are predicted in an OC $\phi \in \Phi_k$, robots in ϕ need to update their individual plans from $\mathcal{P}_{k+1}$. Note that robots may join and leave different couplings during the online planning process. To make sure that robots make progress to their final goals as a team, we associate a *contribution value* to each robot, which is initially zero. For robot i, denote this value as $\gamma_{k-}[i]$ before the coupling at k; $\gamma_{k+\delta}[i]$, when δ is 0, represents the updated value after the coupling at k, which is discussed below.

First, robots in ϕ compute a plan $\mathcal{Q}$ ($|\mathcal{Q}| \leq \beta$) to *local goals*, which satisfies the following conditions:

$$\sum_{i\in\phi} \mathcal{C}(\mathcal{S}_k[i], \mathcal{G}[i]) + \gamma_{k-}[i] > \sum_{i\in\phi} \mathcal{C}(\mathcal{S}_k[i](\mathcal{Q}[i]), \mathcal{G}[i]) \tag{2}$$

and

$$\forall i \in \phi, \neg\Delta_k^i \tag{3}$$

in which Δ_k^i is computed based on the updated individual plans that are constructed as follows: the new individual plan $\mathcal{P}[i]$ for robot $i \in \phi$ is constructed by replacing actions starting from $\mathcal{P}_{k+1}[i]$ by $\mathcal{Q}[i] + P(\mathcal{S}_k[i](\mathcal{Q}[i]), \mathcal{G}[i])$. $\mathcal{S}_k[i](\mathcal{Q}[i])$ is the *local goal* for i, which is the location of i (currently at $\mathcal{S}_k[i]$) after executing $\mathcal{Q}[i]$. The $+$ symbol is used here to denote concatenation.

At time step k, and after executing each action in $\mathcal{Q}[i]$, the contribution value for i is updated as follows, until a conflict is predicted or this value becomes 0:

$$\gamma_{k+\delta}[i] = \mathcal{C}(\mathcal{S}_k[i](\mathcal{Q}[i]), \mathcal{G}[i]) - \mathcal{C}(\mathcal{S}_{k+\delta}[i], \mathcal{G}[i]) \tag{4}$$

in which $0 \leq \delta \leq |\mathcal{Q}|$, is the number of actions in $\mathcal{Q}$ that are executed. Note that $\mathcal{S}_{k+\delta}[i] = \mathcal{S}_0[i](\mathcal{P}_{1,k+\delta}[i])$ is the location of robot i at time step $k+\delta$ under the updated individual plan $\mathcal{P}[i]$ at time step k.

Lemma 1. *Planning (i.e., the computation of $\mathcal{Q}$) in the coupling process converges $\mathcal{R}$ to their final goals as k grows, if Eq.* (2) *can always be satisfied.*

Proof. From Eq. (2) and (4), we have the following holds:

$$\sum_{i\in\phi} \mathcal{C}(\mathcal{S}_k[i],\mathcal{G}[i]) + \gamma_{k-}[i] > \sum_{i\in\phi} \mathcal{C}(\mathcal{S}_{k+\delta}[i],\mathcal{G}[i]) + \gamma_{k+\delta}[i] \tag{5}$$

First, Eq. (5) holds for all robots that are still executing the coupled plans (i.e., $\mathcal{Q}$) to move to their local goals; furthermore, Eq. (5) also holds for robots that have already reached their local goals or robots that have not engaged in any coupling yet. As a result, Eq. (5) holds for $\mathcal{R}$. As k grows, we know that $\sum_{i\in\mathcal{R}} \mathcal{C}(\mathcal{S}_{k+\delta}[i],\mathcal{G}[i]) + \gamma_{k+\delta}[i]$ would gradually decrease. This also means that $\sum_{i\in\mathcal{R}} \mathcal{C}(\mathcal{S}_k[i](\mathcal{Q}[i]),\mathcal{G}[i])$ would gradually decrease from Eq. (2) and (4). Given $|\mathcal{Q}| \leq \beta$, the conclusion holds.

Assuming that the condition in Eq. (2) holds, Lemma 1 shows that planning converges to final goals for $\mathcal{R}$. This condition requires robots in a coupling to always make progress jointly within the finite horizon. However, this assumption does not hold in the presence of live-locks. In such cases, robots in a coupling would eventually be unable to find a $\mathcal{Q}$ that satisfies both Eq. (2) and (3).[2] Furthermore, the limited horizon can also cause the search of $\mathcal{Q}$ to fail. However, we realize that when live-locks are present in distributed systems with only local information, the search is bound to fail eventually even with unlimited horizon. Hence, we do not distinguish the two causes, and consider it as a "live-lock" being detected when when Eq. (2) becomes unsatisfiable.

3.4 Computing $\mathcal{Q}$

Before discussing how "live-locks" are addressed in DisCoF, we provide details on how $\mathcal{Q}$ is computed. Given that coupled search is expensive, we aim to minimize $|\mathcal{Q}|$ as well as the number of robots that need to be coupled.

To achieve this, we try to construct $\mathcal{Q}$ that satisfies Eq. (2) and (3) for ρ ($\rho \subseteq \phi$), which is initially set to be the corresponding IC for ϕ, while forcing robots in ρ to respect the plans (i.e., avoiding predictable conflicts) of robots in $\phi \setminus \rho$ in the next β steps. Note that having ρ instead of ϕ satisfy Eq. (2) does not influence the planning convergence.

The search first checks $\mathcal{Q}$ for ρ with $\theta = 1$, in which $\theta = |\mathcal{Q}|$, and gradually increases θ until $\theta = \beta$. If a valid $\mathcal{Q}$ is found for the current θ, the $\mathcal{Q}$ is returned. Otherwise, if $\phi \setminus \rho \neq \emptyset$, ρ is expanded to include robots in $\phi \setminus \rho$ that are also within the combined region of local windows of robots in ρ, and the current θ is re-checked; else, θ is incremented or unsatisfiability is returned when $\theta = \beta$.

[2] Note that Eq. (3) can always be satisfied by forcing all the robots in a coupling to stay, which may cause deadlocks. Eq. (2) prevents deadlocks.

4 Push and Pull

In DisCoF, when Eq. (2) becomes unsatisfiable for an OC ϕ, we consider it as a "live-lock" (i.e., robots in ϕ may have contributed in creating a live-lock situation) being detected. To resolve it, information of all robots in ϕ must be accessible. In distributed systems, this requires the robots in ϕ to maintain within each other's sensing and communication range (thus remain coupled). Furthermore, note that a live-lock may not involve all robots in $\mathcal{R}$ and there may be multiple live-locks in the environment. When a live-lock is detected, robots in ϕ form a *coupling group*, ω, which executes a live-lock resolution process described next. This process also allows a coupling group to merge with other groups and robots, thus gradually increasing the level of coupling. In this way, DisCoF can naturally transition robots to be fully coupled, e.g., when a global live-lock is present.

4.1 Overview

To achieve completeness, DisCoF uses a technique that is similar to Push and Rotate [4], which we call Push and Pull. To ensure completeness in Push and Rotate, robots must move to goals one at a time according to the priorities of subproblems to which they belong. Robots that have already reached their goals are respected (i.e., considered as obstacles) by the following Push operations; when Push fails, Push and Rotate uses a Swap operation to ensure that these robots move back in their goals as the remaining robots move. Such a priority ordering must also be respected in DisCoF. At time step k, for all coupling groups that have been formed, the basic idea is to: 1) maintain robots in these groups within each other's sensing and communication range; 2) for each group, move robots to goals one at a time based on a relaxed version of the priority ordering, which is consistent to that in Push and Rotate; 3) add robots that introduce predictable conflicts with a coupling group as robots in the group move to their goals. Each coupling group progresses independently of other robots and coupling groups unless there are predictable conflicts. The main process is described in Alg. 1:

In Alg. 1, the coupling of robots in ω is maintained by the Push and Pull technique. As a result, predictable conflicts can only be introduced by other robots. When a coupling group detects predictable conflicts with another group, two groups are merged. Furthermore, when a robot that has already reached its goal is added to a coupling group in the live-lock resolution process, if the robot's priority is not the highest among all robots that have not reached their goals after recomputing the priorities, this robot is not considered as having reached its goal in Push and Pull. This means that the Push and Pull operations can move these robots. Also, the priority ordering (i.e., the $\succ$ relations in [4]) is maintained and aggregated by the robots whenever new relations are identified (in Line 5); given that the relaxed priority ordering is consistent with that in Push and Rotate, robots can gradually achieve a consensus of this ordering.

Algorithm 1 Live-lock Resolution Process in DisCoF for a Coupling Group ω

1: *Current time step is k.*
2: **while** $\exists i \in \omega$, $\mathcal{S}_k[i] \neq \mathcal{G}[i]$ **do**
3: **if** predictable conflicts detected with other robots **then**
4: Add other robots with predictable conflicts to, or merge their groups with ω.
5: Recompute the priorities of subproblems.
6: **end if**
7: **if** r is not selected $\vee$ robots with a higher priority than r is found $\vee$ r reaches $\mathcal{G}[r]$ **then**
8: $r \leftarrow$ the robot with (equal) highest priority in ω.
9: **end if**
10: Push and Pull r to $\mathcal{G}[r]$.
11: **end while**

4.2 Assigning Priorities

To ensure completeness, the priorities of subproblems in Push and Rotate [4] must be respected. However, given the partial observability of the environment, this priority ordering can only be partially computed for each coupling group. This partially computed ordering in DisCoF is kept consistent with the priority ordering in Push and Rotate.

To compute the priorities, first, Push and Rotate identifies the subproblems. Since this computation is only dependent on the graph structure, robots in DisCoF can individually identify the set of subproblems.

Next, Push and Rotate assigns robots to subproblems. DisCoF computes a relaxed version of this assignment to ensure that assignments are only made when they are consistent with those in Push and Rotate. Denote the set of subproblems as $\mathcal{D}$. Alg. 2 presents the algorithm to compute the assignment in a coupling group ω.

Algorithm 2 Algorithm for Assigning Robots to Subproblems in ω

1: **for all** $\mathcal{D}_h \in \mathcal{D}$ **do**
2: **for all** $v \in \mathcal{D}_h$ **do**
3: **for all** $u \notin \mathcal{D}_h$ for which $(u,v) \in E^\omega$ **do**
4: $m' \leftarrow$ number of unoccupied vertices reachable from v in $G^\omega \setminus \{u\}$.
5: $m'' \leftarrow$ number of unoccupied vertices reachable from $\mathcal{D}_h$ in $G^\omega \setminus \{v\}$.
6: $\neg m \leftarrow$ number of unoccupied vertices unreachable from v in $G^\omega \setminus \{u\}$.
7: **if** $(m' \geq 1 \wedge \neg m \geq 1) \vee m'' \geq 1$ **then**
8: Assign robot in v to $\mathcal{D}_h$.
9: **end if**
10: Follow path from u away from v and assign the first $m'-1$ (all if less than $m'-1$) on this path to $\mathcal{D}_h$ in G^ω.
11: **end for**
12: **end for**
13: **end for**

The differences of Alg. 2 from that in Push and Rotate lie in Line 4, 5, 6, 7 and 10. While the computation for these lines is performed based on the global graph

(i.e., G) in Push and Rotate, the computation in DisCoF is based on $G^{\omega}(V^{\omega}, E^{\omega})$, which represents the combined region of the local windows of robots in ω. Note that not every robot may be assigned to a subproblem and the unassigned robots are assumed to have the lowest priorities.

Lemma 2. *The assignment of robots to subproblems in DisCoF is consistent to that in Push and Rotate [4]: if a robot r is assigned to subproblem $\mathcal{D}_h$ in Alg. 2, it is also assigned to $\mathcal{D}_h$ in Push and Rotate.*

Proof. We only need to prove that: 1) when the condition in Line 7 is satisfied, the corresponding condition in Push and Rotate is also satisfied; 2) m' in Alg. 2 is smaller than that in Push and Rotate. These directly follow from how they are computed.

In the third step, Push and Rotate assigns priorities to the subproblems. Robots within the same subproblems receive the same priorities. Similarly, the reference of global graph is changed to G^{ω}; otherwise, the process is unchanged.

Lemma 3. *The assignment of priorities to subproblems in DisCoF is consistent to that in Push and Rotate [4]: if two subproblems $\mathcal{D}_{h1}$ and $\mathcal{D}_{h2}$ satisfy $\mathcal{D}_{h1} \prec \mathcal{D}_{h2}$, they must also satisfy $\mathcal{D}_{h1} \prec \mathcal{D}_{h2}$ in Push and Rotate.*

Proof. This conclusion follows almost directly from Lemma 2 and the process for assigning priorities to subproblems.

Note that this assignment process is executed by each coupling group in DisCoF instead of all robots in Push and Rotate. This means that while the assignments are consistent with that in Push and Rotate, they are computed for different (and disjoint) sets of robots in DisCoF.

4.3 Maintaining and Expanding ω

Robots in a coupling group can use the operations (i.e., Push, Swap and Rotate) in Push and Rotate to move to their goals one at a time (for details, refer to [4]). To maintain robots within ω in sensing and communication range, we introduce a new operation, called Pull. Denote r as the current robot that is being moved to its final goal in ω. As r moves to its goal, it can use any of the Push, Swap and Rotate operations. Every step that r moves as a result of these operations, it also invokes the Pull operation on the other robots in ω.

The Pull operation computes a shortest path plan p from r to any robot $s \in \omega \setminus r$. A set $\mathcal{U}$ is created, which contains only r initially. If p does not pass through other robots in ω, and the first step in p leads s closer to r, s is added to $\mathcal{U}$. If the first step in p does not introduce conflicts with other robots in ω, this step is added to the individual plan of s; otherwise, an action to stay is added to the individual plan of s. For robots that have been newly added into $\mathcal{U}$, they recursively apply the Pull

operation on robots that are not in $\mathcal{U}$. This process ends until all robots in ω are in $\mathcal{U}$. The Pull operation is presented in Alg. 3. Fig. 2 illustrates the Pull operation in a simple scenario.

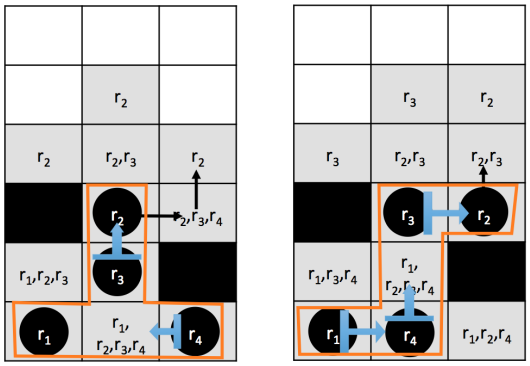


Fig. 2 Scenario that illustrates the Pull operation. Left figure shows that robot r_2 is moving to its goal. Right figure shows the same scenario after one time step. Blue arrows show the actions being added to the individual plans of the corresponding robots at each step by the Pull operation.

Lemma 4. *The Pull operation maintains robots in each coupling group within each other's sensing and communication range.*

Proof. The Pull operation, after execution, ensures that any robot $s \in \omega$ is no further away from one of the robots in ω before its execution. Hence, the conclusion holds.

Similar to Push, Pull may fail (Line 13 in Alg. 3) since it must respect the robots (with equal or higher priorities) that have already reached their goals. In such cases, a similar procedure using Swaps as for the Push operation in [4] can be used; these Swaps can cause robots that are being swapped to recursively invoke Pull.

When there are other robots within the combined region of the local windows of robots in ω, robots must plan to consider predicted conflicts. Each coupling group makes a plan for the next β steps considering only robots in the group. When no conflicts are predicted, robots continue with this plan. When conflicts are predicted, ω is expanded as we previously discussed. The expanded coupling group chooses the robot currently with the (equal) highest priority to move to the goal. Fig. 3 illustrates the merge of two coupling groups.

4.4 Analysis

To prove the completeness of DisCoF, we use a property that is derived directly from Theorem 2 in Push and Rotate [4].

Algorithm 3 Pull operation in ω

$\mathcal{U} \leftarrow \{r\}$; $\mathcal{N} \leftarrow \emptyset$
while $\mathcal{U} \neq \omega$ **do**
  for all $s \in \omega \setminus \mathcal{U}$ **do**
    $p \leftarrow P(\mathcal{S}_k[s], \mathcal{S}_k[r])$, considering robots that have reached goals as obstacles.
    if p does not pass through robots in ω that have not reached goals $\wedge$ p moves s closer to r **then**
      if no conflicts with other robots in ω after executing the first step in p **then**
        Add the first step in p to the individual plan of s.
      else
        Add an action for s to stay in the next step.
      end if
      $\mathcal{U} \leftarrow \mathcal{U} \cup \{s\}$; $\mathcal{N} \leftarrow \mathcal{N} \cup \{s\}$
    else
      return False.
    end if
  end for
  $r \leftarrow Pop(\mathcal{N})$.
end while

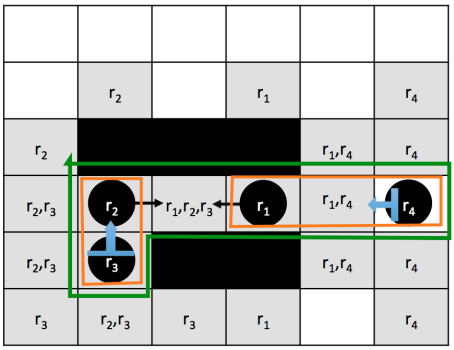


Fig. 3 Scenario that illustrates the expanding process, in which two coupling groups, each with two robots, are merged when a conflict is predicted in the next step.

Corollary 1. *If the cooperative pathfinding problem is solvable, the assignment of robots in a coupling group to subproblems remains unchanged unless the group is expanded.*

Theorem 1. *DisCoF is complete for the class of cooperative pathfinding problems in which there are two or more unoccupied vertices in each connected component.*

Proof. We provide the proof sketch here, which is based on the following observations: (1) When every coupling group is independent of other robots and groups, DisCoF is complete; this is almost a direct result from Push and Rotate, since the Pull operation does not influence the other operations. (2) When a coupling group is expanded, robots in the group are maintained within each other's sensing and communicating range; this is a direct result from Lemma 4. (3) The priority ordering

relations (i.e., $\succ$) are maintained and gradually aggregated (to reach a consensus) as they are identified; this is a result from Lemma 2, Lemma 3 and Corollary 1. (4) Robots with the highest priorities are respected (in Push and Pull operations) by the coupling groups in the Push and Pull process (similar to that in Push and Rotate), which moves robots with the highest priorities to goals first.

Since it has been shown in [4] that robots with the highest priorities must be moved to goals first in order to ensure a solution, these robots must eventually be assigned the highest priorities as the coupling groups move. Hence, these robots would be moved to their final goals. This process then continues to robots with the second highest priorities and so on. Hence, DisCoF is complete.

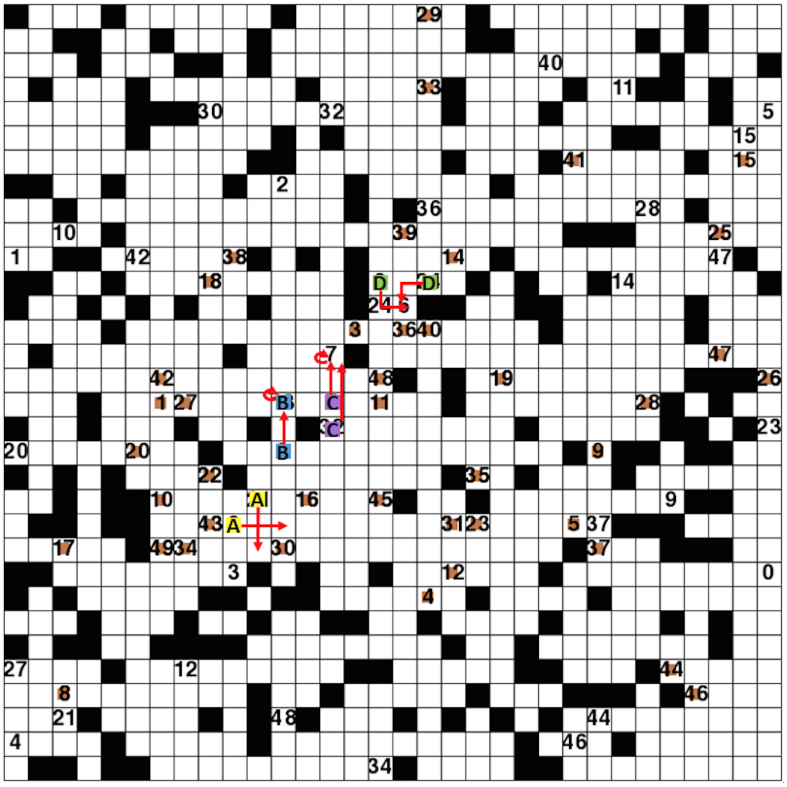
5 RESULTS

In this section, we only provide preliminary results for DisCoF. In particular, we concentrate on evaluating the efficiency of computing $\mathcal{Q}$ (Sec. 3.4) in relatively simple problem instances. The idea is to show that robots in different outer closures (OCs) can simultaneously compute $\mathcal{Q}$ in a distributed manner to resolve conflicts. We plan to provide a full evaluation with the Push and Pull technique in future work.

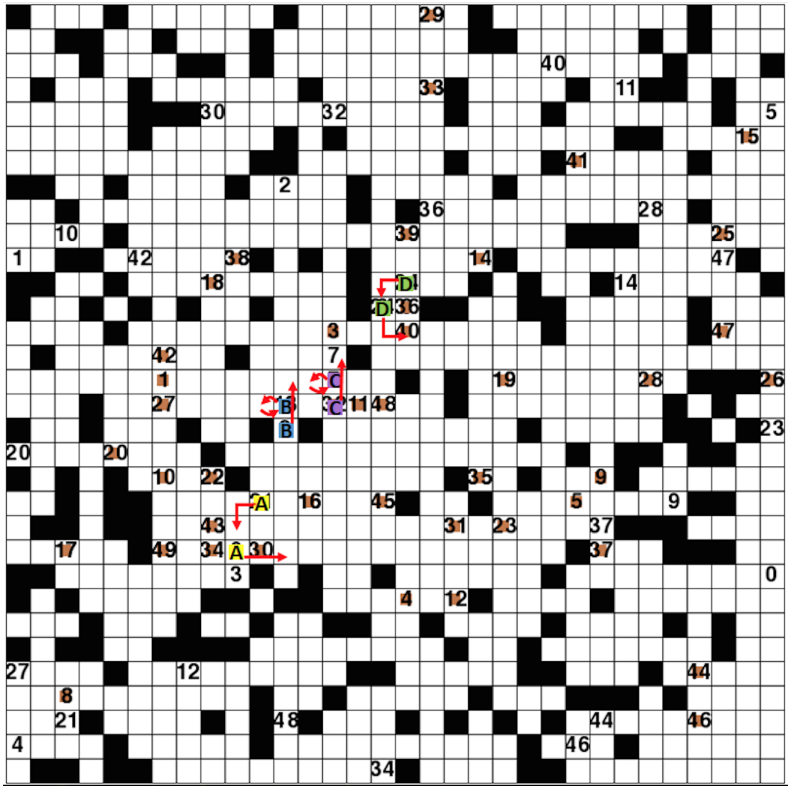
In this evaluation, we build a 32×32 grid environment and randomly generate 20% obstacles for each problem instance. Our evaluation starts from 20 to 60 robots (with an increment of 10); we generate 20 problem instances with 20 robots and with each increment. The initial locations and goals of the robots are distinct cells in the environment, which are randomly generated. λ (local window) is set to be 2 and β (finite horizon) is set to be 3. During the planning phase, if the robots within any OC fail to compute $\mathcal{Q}$ with $|\mathcal{Q}| \leq \beta$, we mark the entire instance as a failure. Also, we do not perform any pruning or sampling in this evaluation and hence the computation of $\mathcal{Q}$ is exponential with the number of robots and β. We expect that the computational performance can be improved. The results are provided as follows.

Fig. 4(a) shows multiple OCs resolving conflicts in one of the problem instances with 50 robots. Robots are marked as small solid squares with numbers (i.e., IDs), and goals are marked correspondingly with the same numbers. At time step 22, there are 4 predicted conflicts and the robots involved in each conflict are labeled by different letters. We also show in the figure the next few actions in the robots' individual plans. Fig. 4(b) shows the robots after the conflicts are resolved at step 23. There are two interesting observations. The first observation is that while group B and group C belong to the same OC, their conflicts can be independently resolved. This can be used to improve the performance for computing $\mathcal{Q}$. The second observation is that, at step 22, the two robots in group C are heading in the same direction, but they have a predicted conflict in 3 steps. This is because one of the robot would reach its goal in two steps and stay, according to its current individual plan.

Fig. 5 presents an analysis of the runs as the number of robots increases. In this figure, we show the number of robots along the X axis. The planning time (the moving time of the robots is ignored) for all robots is aggregated throughout the online



(a) Time step 22



(b) Time step 23

Fig. 4 One problem instance in our evaluation. Arrows from the robots show the next few steps in the robots' individual plans. (a) There are 4 predicted conflicts. (b) All conflicts are resolved.

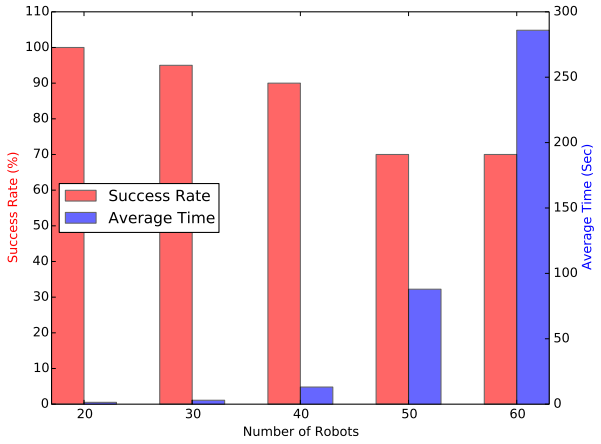


Fig. 5 X axis shows the number of robots; success rates and averaged planning times are shown along the Y axis.

planning process for each instance. The averaged planning times (over 20 instances), and the success rates, are shown along the Y axis. As expected, the computation of $\mathcal{Q}$ works well in simple scenarios and the number of failures increases as the number of robots increases. Note that the OCs that fail to compute a $\mathcal{Q}$ in the failure cases are to execute the Push and Pull technique as we discussed in Sec 4. Evaluation of DisCoF with the Push and Pull technique is to be presented in our future work.

6 CONCLUSIONS

In this paper, we introduce a window-based approach for cooperative pathfinding in distributed systems, with the window size corresponding to the limited sensing and communication range in such systems. This approach, called DisCoF, is an online approach. To limit coupling in order to reduce computation, we introduce a formulation that allows robots to avoid future conflicts while still making joint progress to their final goals. This formulation also allows "live-locks" to be detected; in such cases, we use a Push and Pull technique. We show that DisCoF is complete and present some preliminary results. To the best of our knowledge, this is the first work that guarantees completeness for cooperative pathfinding with limited sensing and communication range in distributed systems.

In future work, we plan to provide a full evaluation of DisCoF and compare it with other related approaches. We also plan to extend the formulations to consider more complex environment and goal specifications (e.g., using LTL formulas).

References

1. Nora Ayanian, Daniela Rus, and Vijay Kumar. Decentralized multirobot control in partially known environments with dynamic task reassignment. In *3rd IFAC Workshop on Distributed Estimation and Control in Networked Systems*, 2012.
2. Zahy Bnaya and Ariel Felner. Conflict-oriented windowed hierarchical cooperative A*. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation*, 2014.
3. C.M. Clark, S.M. Rock, and J.-C. Latombe. Motion planning for multiple mobile robots using dynamic networks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 4222–4227, Sep. 2003.
4. Boris de Wilde, Adriaan W. ter Mors, and Cees Witteveen. Push and rotate: Cooperative multi-agent path planning. In *12th International Conference on Autonomous Agents and Multiagent Systems*, 2013.
5. Vishnu R. Desaraju and Jonathan P. How. Decentralized path planning for multi-agent teams with complex constraints. *Autonomous Robots*, 32(4):385–403, 2012.
6. J.E. Hopcroft, J.T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; pspace- hardness of the "warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88, 1984.
7. Renee Jansen and Nathan Sturtevant. A new approach to cooperative pathfinding. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pages 1401–1404, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.
8. Lantao Liu and Dylan A Shell. Physically routing robots in a multi-robot network: Flexibility through a three-dimensional matching graph. *The International Journal of Robotics Research*, 32(12):1475–1494, 2013.
9. R. Luna and K. Bekris. Efficient and complete centralized multirobot path planning. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2011.
10. Michael Otte, Joshua Bialkowski, and Emilio Frazzoli. Any-com collision checking: Sharing certificates in decentralized multi-robot teams. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation*, 2014.
11. Lynne E. Parker. *Encyclopedia of Complexity and System Science*, chapter Path Planning and Motion Coordination in Multiple Mobile Robot Teams. Springer, 2009.
12. M. Peasgood, C.M. Clark, and J. McPhee. A complete and scalable strategy for coordinating multiple robots within roadmaps. *IEEE Transactions on Robotics*, 24(2):283–292, April 2008.
13. Malcolm Ryan. Graph decomposition for efficient multi-robot path planning. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 2003–2008, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.
14. D. Silver. Cooperative pathfinding. In *Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005.
15. Trevor Standley. Finding optimal solutions to cooperative pathfinding problems. In *AAAI Conference on Artificial Intelligence*, 2010.
16. Trevor Standley and Richard Korf. Complete algorithms for cooperative pathfinding problems. In *Proceedings of the 22nd International Joint Conference on Artifical Intelligence*, 2011.
17. N. Sturtevant and M. Buro. Improving collaborative pathfinding using map abstraction. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 80–85, 2006.
18. Ko-hsin Cindy Wang and Adi Botea. Fast and memory-efficient multi-agent pathfinding. In *International Conference on Automated Planning and Scheduling*, pages 380–387, 2008.
19. J. Yu and S. M. LaValle. Multi-agent path planning and network flow. In *Algorithmic Foundations of Robotics X*, volume 86, pages 157–173. Springer, 2013.