DisCoF$^+$: Asynchronous DisCoF with Flexible Decoupling for Cooperative Pathfinding in Distributed Systems

Kangjin Kim, Joe Campbell, William Duong, Yu Zhang and Georgios Fainekos

***Abstract*— In our prior work, we outlined an approach, named DisCoF, for cooperative pathfinding in distributed systems with limited sensing and communication range. Contrasting to prior works on cooperative pathfinding with completeness guarantees which assume access to global communication and coordination, DisCoF does not make this assumption. The implication is that at any given time in DisCoF, the robots may not all be aware of each other which is often the case in distributed systems. As a result, DisCoF represents an inherently online approach since coordination can only be realized in an opportunistic manner between robots that are within each other's sensing and communication range. However, there are a few assumptions made in DisCoF to facilitate a formal analysis which must be removed to work with distributed multi-robot platforms. In this paper, we present DisCoF$^+$ which extends DisCoF by enabling an asynchronous solution, as well as providing flexible decoupling between robots for performance improvement. Furthermore, we evaluate our implementation of DisCoF$^+$ by implementing our distributed multi-robot algorithm in the Webots simulator. Finally, we compare DisCoF$^+$ with DisCoF in terms of plan quality and planning performance.**

I. INTRODUCTION

While cooperative pathfinding in multi-robot systems has many applications, it is also fundamentally hard to solve (i.e., PSPACE-hard [6]). The difficulty lies in the potential *coupling* between robots: when robots are completely decoupled (e.g., when robots do not impose constraints on each other's plan to the goal), cooperative pathfinding becomes polynomial-time solvable.[1] As a result, most recent approaches (e.g., [15], [16], [17], [18]) for pathfinding concentrate on how to identify the dependencies between robots in order to couple robots only when necessary.

In these approaches, the solution is constructed for a subset of the robots which are coupled to each other and which are decoupled from the remaining robots. The computational complexity is exponential only in the maximum number of robots in these subsets. While optimistic decoupling can lose optimality and even completeness (e.g., [16]), pessimistic decoupling can only handle situations in which robots are loosely coupled (e.g., [17]).

Meanwhile, to ensure completeness, these approaches often assume access to global communication and coordination which implies that all robots have access to the current positions of the other robots, their individual plans and goals. With this information, any robot can consider all other robots when creating its own plan. While this assumption can be made in many common applications of cooperative pathfinding where planning can be centralized and performed offline (e.g., cooperative pathfinding in computer games), it does not hold in distributed systems with limited sensing and communication range.

In our prior work [23], we introduced a window-based approach, called DisCoF, for cooperative pathfinding in distributed systems with limited sensing and communication range. In DisCoF, the window size corresponds to the sensing range of the robots. Robots can communicate with each other either directly if in range or indirectly if out of range. In the latter case, it is still possible to communicate indirectly through other robots using a communication relay protocol. This allows for coordination beyond a single robot's sensor range.

To ensure completeness, DisCoF uses a flexible approach to decoupling robots such that they can transition from optimistic to pessimistic decoupling when necessary. Robots are assumed to be fully decoupled initially. During the online pathfinding process, robots only couple together when necessary (i.e., when there are *predictable conflicts* [23]). Since access to global communication and coordination is not assumed, the creation of local couplings (i.e., subsets of robots) may not be sufficient to avoid live-locks. In such cases, a mechanism (called *push and pull*) is introduced in which robots in a local coupling can form a *coupling group* [23] in order to coordinate more closely. Robots in a coupling group move to their goals sequentially in a certain order while keeping others (i.e., those that have not yet reached their goals) within communication range. Coupling groups may increase in size (e.g., when previously undetected robots come within sensing range of a robot in the coupling group) and decrease in size (e.g., when robots reach their goals). This mechanism can potentially lead to a global coupling.

Contributions: In this paper, first, we introduce an asynchronous variant of DisCoF, refereed to as DisCoF$^+$, in order to remove DisCoF's assumption that time steps are synchronized.[2] That is, we provide an asynchronous algorithm and its communication strategy. Then, we introduce a new decoupling strategy in DisCoF$^+$ with the goal of

This work has been partially supported by NSF award CNS-1116136 and CNS-1446730, the ARO grant W911NF-13-1-0023, the ONR grants N00014-13-1- 0176, N00014-13-1-0519 and N00014-15-1-2027.

K. Kim, J. Campbell, W. Duong, Y. Zhang and G. Fainekos are with the School of Computing, Informatics and Decision Systems Engineering, Arizona State University, Tempe, AZ 85281, USA {Kangjin.Kim, jacampb1, tbduong, Yu.Zhang.442, fainekos}@asu.edu

[1] A single robot pathfinding problem is polynomial-time solvable.

[2] This is achieved in DisCoF for all robots by 1) maintaining a synchronized clock at the beginning of the task and 2) limiting each robot to have the same planning and execution time at each step. However, this is generally too strong an assumption to make in a distributed system.

improving efficiency. In DisCoF, only a directional transition from optimistic to pessimistic decoupling is allowed. On the other hand, DisCoF$^+$ allows bi-directional transitions between optimistic and pessimistic decoupling.

Furthermore, for evaluation, we first demonstrate a simulation of DisCoF$^+$ in a distributed multirobot environment modeled in Webots. Then, we compare the performance of DisCoF and DisCoF$^+$ in terms of computation time and length of plans on randomly generated environments.

II. RELATED WORK

To address the cooperative pathfinding problem, researchers have used a complilation approach [9], [1], [5], [22], in which the problem is first transformed into other related problems, and then the existing solutions or algorithms for these problems can be applied. Abstraction methods to reduce the search space have also been used [19], [14]. However, due to the inherent complexity of the problem, these approaches do not scale. While approaches that constrain the topologies of the environment [21], [12], [13] can significantly reduce the complexity, they cannot be applied to general problem instances.

Given that pathfinding for a single robot is polynomial-time solvable, the complexity of multi-robot pathfinding is a result of coupling between robots. Then, researchers have studied on various ways to decouple robots. For approaches that perform optimistic decoupling, robots are considered as coupled only when necessary. One of the representative approaches is hierarchical cooperative A* (HCA* [16]) in which robots plan one at a time while respecting plans that have already been computed. To limit the influence of the previous robots on the following robots, a windowed HCA* is introduced to restrict this influence based on a pre-specified window size [16]. Recently, an extension of WHCA* (CO-WHCA* [2]) was introduced to further reduce this influence based on the notion of *conflicts*. Although many problem instances can be solved efficiently, optimistic decoupling leads to loss of optimality and completeness.

One of the earlier approaches that performs decoupling while maintaining optimality and completeness relies on pessimistic decoupling [17] and [20]. That work couples robots when conflicts are detected in the individual robot plans. As a result, the approach tends to over-couple and, hence, remains intractable for many problem instances. More recent approaches relax optimality to achieve better efficiency [10], [4], [18]. However, to maintain completeness, these approaches assume access to global communication and coordination and therefore are inapplicable to distributed systems in which robots have limited sensing and communication range.

While there are extensible approaches to distributed systems (e.g., [7]) and approaches that are designed for distributed systems (e.g., [11], [3]), they do not provide completeness guarantees. The difficulty lies in planning without access to global communication and coordination. This is further discussed in [23].

III. DISCOF

In this section, we provide the problem formulation and we review DisCoF [23]. Extensions to DisCoF, DisCoF$^+$, are discussed in Section IV.

A. Problem Formulation

We assume that the workspace is represented by a undirected graph $G(V, E)$. We assume the existence of a set of robots $\mathcal{R}$ with initial positions $\mathcal{I} \subseteq V$ and goal positions $\mathcal{G} \subseteq V$. Any robot can move to any adjacent vertex in one time step or remain where it is. 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 finite sequence of actions. For simplicity, in this paper, each action is represented by the next vertex to be visited. For example, $\mathcal{P}_k[i]$ $(k \geq 1)$ denotes the action to be taken at time step $k-1$ (or the vertex to be visited at k) for robot i. $\mathcal{P}_{k,l}[i]$ $(k \leq l)$ denotes the subplan that contains the actions from $\mathcal{P}_k[i]$ 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 $\mathcal{G}$ without any collisions.

A set of locations of robots $\mathcal{R}$ at time step k is denoted by $\mathcal{S}_k$ and a location of each robot $i \in \mathcal{R}$ at time step k is denoted by $\mathcal{S}_k[i]$ for convenience. If robots $\mathcal{R}$ execute plan $\mathcal{P}$ from a set of locations $\mathcal{S}$ to another set of locations $\mathcal{S}'$, it is denoted by $\mathcal{S} \overset{\mathcal{P}}{\rightsquigarrow} \mathcal{S}'$. Likewise, if a robot $i \in \mathcal{R}$ executes its plan $\mathcal{P}[i]$ from a location $\mathcal{S}[i]$ to another location $\mathcal{S}'[i]$, it is denoted by $\mathcal{S}[i] \overset{\mathcal{P}[i]}{\rightsquigarrow} \mathcal{S}'[i]$. In addition, given $\mathcal{S} \overset{\mathcal{P}}{\rightsquigarrow} \mathcal{S}'$, we denote $\mathcal{S}'$ as $\mathcal{S}(\mathcal{P})$, and given $\mathcal{S}[i] \overset{\mathcal{P}[i]}{\rightsquigarrow} \mathcal{S}'[i]$, we denote $\mathcal{S}'[i]$ as $\mathcal{S}[i](\mathcal{P}[i])$. Hence, $\mathcal{S}_0 = \mathcal{I}$, $\mathcal{S}_0 \overset{\mathcal{P}}{\rightsquigarrow} \mathcal{G}$, $\mathcal{G} = \mathcal{S}_0(\mathcal{P})$, $\mathcal{S}_0 \overset{\mathcal{P}_{1,k}}{\rightsquigarrow} \mathcal{S}_k$ and $\mathcal{S}_k = \mathcal{S}_0(\mathcal{P}_{1,k})$, and for some robot $i \in \mathcal{R}$, $\mathcal{S}_0[i] = \mathcal{I}[i]$, $\mathcal{S}_0[i] \overset{\mathcal{P}[i]}{\rightsquigarrow} \mathcal{G}[i]$, $\mathcal{G}[i] = \mathcal{S}_0[i](\mathcal{P}[i])$, $\mathcal{S}_0[i] \overset{\mathcal{P}_{1,k}[i]}{\rightsquigarrow} \mathcal{S}_k[i]$ and $\mathcal{S}_k[i] = \mathcal{S}_0[i](\mathcal{P}_{1,k}[i])$.

A *conflict* happens at time step k, if the following is satisfied:

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

in which $i, j \in \mathcal{R}$ and $i \neq j$. In other words, if two robots move to the same place at time k or two robots switch their locations in one consecutive time step (from $k-1$ to k), then we have a conflict. This definition of conflict can be generalized to capture other conditions.

Each robot has a planner that can compute a shortest path, $P(u, v)$ that moves a robot from vertex u to v. The length of $P(u, v)$ is denoted as $\mathcal{C}(u, v)$, i.e., $\mathcal{C}(u, v) = |P(u, v)|$. The following simplifying assumptions are also made in DisCoF:

1) Robots are homogeneous and equipped with a communication protocol for message relay.
2) Robots know G and are synchronized at every time step.

Initially, for each robot i, the individual plan is constructed as $\mathcal{P}[i] = P(\mathcal{I}[i], \mathcal{G}[i])$. Robots then start executing their individual plans until conflicts can be predicted (i.e., *predictable conflicts* in [23]) at time step k. In such cases, the individual plans of the robots which will lead to conflicts are updated in $\mathcal{P}_{k+1}$ to avoid these conflicts.

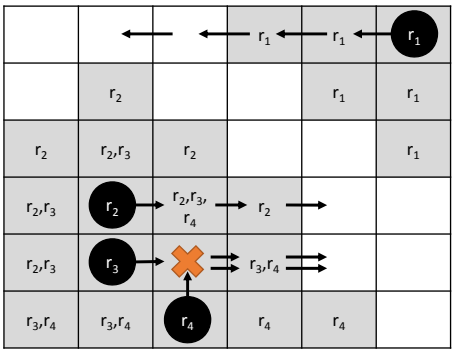

Fig. 1. [23]: Scenario that illustrates OC and IC. Two OCs are present $\{r_1\}$ and $\{r_2, r_3, r_4\}$, out of which one has an IC $\{r_3, r_4\}$ with a predictable conflict. The sensing ranges of the robots are shown in gray. The arrows show the next few steps in the individual plans.

B. Optimistic Decoupling

In DisCoF, the window size corresponds to the sensing range of the robot. To reduce communication overhead, a robot is only allowed to communicate with other robots when it can sense them. However, robots that cannot sense each other can communicate using the message relay protocol through other robots. A closure of the set of robots that can communicate (directly or via message relay) in order to coordinate is called an *outer closure* (OC). In an OC, there can be multiple predictable conflicts. A closure that contains agents with potential conflicts is the *inner closure* (IC) of the OC. Figure 1 shows an example of OC and IC. For details, refer to [23].

In DisCoF, decoupling is optimistic initially, and gradually becomes more pessimistic when necessary. Given an OC with predicted conflicts, in *optimistic decoupling* DisCoF updates the individual plans of robots to proactively resolve these conflicts, while avoiding introducing new conflicts within a finite horizon (which is specified by a parameter in DisCoF). The finite horizon is key to efficiency since the resolution for conflicts in the far future is likely to waste computation efforts given the incomplete information (e.g., the positions of other robots in the environment). Note that the window size, i.e., sensing range, in DisCoF represents a horizon for detecting conflicts.

To ensure that robots are jointly making progress towards their goals, DisCoF uses the notion of *contribution value*. In order to resolve conflicts, plans are updated in a process known as conflict resolution. In this process, each robot is associated with a contribution value when using optimistic decoupling. If this process is successful, robots continue as fully decoupled. The contribution value is also used to determine cases when optimistic decoupling is insufficient. That is, when the resolution process would fail due to potential live-locks. When there are no potential live-locks, it is shown that optimistic decoupling is sufficient for robots to converge to their goals. Otherwise, robots within the OC use the following *pessimistic decoupling* process.

C. Pessimistic Decoupling

In DisCoF, when there are potential live-locks, robots within an OC transition to *pessimistic decoupling* by remaining within each other's communication range (whether direct or indirect). These robots are referred to as a *coupling group*, and this coupling group executes a process known as push and pull which allows it to merge with other groups and robots. Thus, the level of coupling gradually increases. In this way, DisCoF can naturally transition robots to be fully coupled when necessary.

In push and pull, robots move to goals one at a time according to the priorities of subproblems (first introduced in [4]). However, due to the incompleteness of information in distributed systems, the priorities will not be fully known. As a result, DisCoF employs the following process. At time step k, for each coupling group that has been formed, DisCoF will:

1) Maintain robots in the group within each other's communication range.
2) Move robots to goals one at a time based on a relaxed version of the priority ordering which is consistent to that in [4].
3) Add other robots or merge with other groups that introduce potential conflicts with robots in the current group as they move to their goals.

Unless there are potential conflicts, each coupling group progresses independently of other robots and coupling groups.

In [23], we proved that the combination of optimistic and pessimistic decoupling in DisCoF guarantees completeness.[3]

IV. DisCoF$^+$

In this section, we discuss the extensions to DisCoF that are made in the new approach named DisCoF$^+$. First, we relax the assumption that robots synchronize at every time step (or plan step). Note that even though robots in different OCs cannot communicate in DisCoF, it is assumed that robots act in synchronized time steps. That is, robots are given a fixed amount of time to finish planning and execute a single action at every time step. The relaxation of this synchronization is necessary for implementation in a real distributed system because we cannot always assume the existence of a global clock and a fixed amount of time for each time step (e.g., the time required for planning for each robot may be arbitrarily different).

We remark that each robot can still access the entire map. We can assume that this information is static such that it is initially given and does not change.[4] However, each robot cannot recognize where other robots are if they are out of (indirect) communication and sensing range. This information is dynamic such that it changes arbitrarily.

[3] DisCoF is complete for the class of cooperative pathfinding problems in which there are two or more unoccupied vertices in each connected component which is an extension of results in [4].

[4] Our replanning framework can be extended to partially known environments with unknown static obstacles.

Furthermore, we introduce a new decoupling strategy such that robots are also allowed to decouple after they form a coupling group (i.e., executing push and pull); thus, transitioning back to optimistic decoupling from pessimistic decoupling. This strategy makes DisCoF$^+$ more computationally efficient while achieving higher quality plans that require fewer steps.[5]

A. Asynchronous Time Steps

Unlike DisCoF, DisCoF$^+$ allows robots in different OCs to proceed independently and asynchronously. However, robots within the same OC are assumed to still have synchronized plan steps. This is a reasonable assumption because these robots communicate to coordinate with each other. As a result of this assumption, robots who finish their current plan step must wait until all others in the OC also finish theirs. Afterwards, all members of the group start the next plan step at the same time in order to avoid unnecessary conflicts. We remark that since we assume homogeneous robots, the waiting time at each time step is not significant.[6]

We now explain Alg. 1. First, all variables including a set of current locations $\mathcal{S}$ and a set of current local window $\mathcal{W}$ are initialized and updated until line 6. Then, while progressing its own plan $\mathcal{P}$, it senses a conflict at line 8. If a conflict is not detected, then it progresses the next step at line 12. If a conflict is detected, then it resolves the conflict and updates the current plan $\mathcal{P}$ with the new plan $\mathcal{P}'$ from line 16 to line 30. If a conflict is detected such that the IC ψ is not empty, robot i tries to resolve the conflict after checking if it is already involved in any conflicts at line 16. If ω is not empty at line 16, it means that from the previous iterations, ω has been already assigned. Then, at the current iteration, another conflict is detected. That is, a coupling group meets another coupling group while resolving its conflict. Then, it merges the ω with an current OC ϕ and begins PUSHANDPULL in order to resolve it through pessimistic decoupling process.[7] If ω is empty, then it means that it hasn't involved any conflict yet. That is, robot $i \in \mathcal{R}$ was executing its plan independently. Then, it forms a local coupling ω. It first tries to decouple optimistically through CONVERGENCE. If it cannot find a plan $\mathcal{P}'$, then it decouples pessimistically through PUSHANDPULL. After finding a plan $\mathcal{P}'$, it continues to the next iteration to sense if there are new conflicts. In this way, the above process continues until it reaches its goal.[8]

Algorithm 1 DisCoF$^+$ with asynchronous time steps for a robot $i \in \mathcal{R}$, given the environment $G = (V, E, \{i\}, \mathcal{I}[i], \mathcal{G}[i])$, its initial location $\mathcal{I}[i]$, final destination $\mathcal{G}[i]$ and initial plan $\mathcal{P}[i]$ from $\mathcal{I}[i]$ to $\mathcal{G}[i]$ and local window $\mathcal{W}$; γ denotes the contribution values.

1: $\langle \psi, \phi, \omega, \mathcal{S}[:], \mathcal{W}, \gamma[:], k \rangle \leftarrow \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, 0, 0 \rangle$
2: $S[i] \leftarrow \mathcal{I}[i]$ ▷ Update the current location to $\mathcal{I}[i]$
3: $\mathcal{G}[: i-1] \cup \mathcal{G}[i+1 :] \leftarrow \emptyset$ ▷ Initialize goals for others
4: $\mathcal{P}[: i-1] \cup \mathcal{P}[i+1 :] \leftarrow \emptyset$ ▷ Initizliae plans for others
5: $G' \leftarrow (V, E, \emptyset, \mathcal{S}, \mathcal{G})$
6: $\langle \mathcal{S}, \mathcal{W} \rangle \leftarrow$ PROCEEDONESTEP$(G', \mathcal{P}, i, k)$
7: **while** True **do**
8: $\quad \langle \psi, \phi \rangle \leftarrow$ SENSECONFLICT$(\mathcal{P}, i, \mathcal{S}, k, \mathcal{W})$
9: $\quad$ **if** $\psi = \emptyset$ **then**
10: $\quad\quad k \leftarrow k + 1$ ▷ Increase the time step k by 1
11: $\quad\quad G' \leftarrow (V, E, \omega, \mathcal{S}, \mathcal{G})$
12: $\quad\quad \langle \mathcal{S}, \mathcal{W} \rangle \leftarrow$ PROCEEDONESTEP$(G', \mathcal{P}, i, k)$
13: $\quad\quad G' \leftarrow (V, E, \omega, \mathcal{S}, \mathcal{G})$ ▷ Update G' with new $\mathcal{S}$
14: $\quad\quad \langle \gamma, \omega, \mathcal{P} \rangle \leftarrow$ RECOMPUTECONT$(G', \mathcal{P}, i, k, \gamma)$
15: $\quad$ **else**
16: $\quad\quad$ **if** $\omega \neq \emptyset$ **then** ▷ It meets another group.
17: $\quad\quad\quad \omega \leftarrow \omega \cup \phi$ ▷ Merge ω with OC ϕ
18: $\quad\quad\quad G' \leftarrow (V, E, \omega, \mathcal{S}, \mathcal{G})$
19: $\quad\quad\quad \mathcal{P}' \leftarrow$ PUSHANDPULL(G', i, γ)
20: $\quad\quad$ **else**
21: $\quad\quad\quad \omega \leftarrow \psi$ ▷ Set ω to IC ψ
22: $\quad\quad\quad G' \leftarrow (V, E, \omega, \mathcal{S}, \mathcal{G})$
23: $\quad\quad\quad \mathcal{P}' \leftarrow$ CONVERGENCE$(G', i, k, \phi, \mathcal{P}, \mathcal{W}, \gamma)$
24: $\quad\quad\quad$ **if** $|\mathcal{P}'| = 0$ **then**
25: $\quad\quad\quad\quad \omega \leftarrow \phi$ ▷ Set ω to OC ϕ
26: $\quad\quad\quad\quad G' \leftarrow (V, E, \omega, \mathcal{S}, \mathcal{G})$
27: $\quad\quad\quad\quad \mathcal{P}' \leftarrow$ PUSHANDPULL(G', i, γ)
28: $\quad\quad$ **if** $\mathcal{P}' = \emptyset$ **then**
29: $\quad\quad\quad$ return False
30: $\quad\quad \mathcal{P}[\omega] \leftarrow \mathcal{P}_{1,k}[\omega] + \mathcal{P}'[\omega]$
31: return True

We need to explain some codes and procedures in details. First, in order to simplify each procedure, at line 5, 11, 13, 18, 22 and 26, we use G' as a tuple of V, E, ω, $\mathcal{S}$ and $\mathcal{G}$. Here, V and E are from the workspace $G = (V, E)$, ω is a set of robots which represents a coupling group, $\mathcal{S}$ is a set of current locations, and $\mathcal{G}$ is a set of goal locations. Second, given a tuple G', a set of plans $\mathcal{P}$, a robot $i \in \mathcal{R}$, and i's local time step k, PROCEEDONESTEP returns a set of current locations $\mathcal{S}$ and a current location window $\mathcal{W}$. We remark that PROCEEDONESTEP does not increase the time step variable k. If k is not increased before calling PROCEEDONESTEP, like line 6, then it does not update the

[5] How efficient this strategy is depends on the problem instance. Robots in a denser environment may need frequent coupling and decoupling, thus increasing the computation overhead. This is discussed in Sec. V through simulation experiments.

[6] Heterogeneous robots may have different speed, sensing & communication range, size and etc. Considering these issues and resolving them are beyond the scope of this paper.

[7] We remark that our description of PUSHANDPULL in Alg. 1 is simplified to show the overall process. Once PUSHANDPULL returns a new plan $\mathcal{P}'$ in Alg. 1, it contains the individual plans for the coupling group ω to move from their locations at the time step k to their goals.

[8] Due to lack of global communication and coordination, our algorithm (running on each robot) would not be able to determine whether all other robots have reached their goals, thus we cannot compute a termination condition. In our simulation, we stop the programs (on all robots) when they have reached their goals.

current locations $\mathcal{S}$ with the set of plan $\mathcal{P}$. However, it is required to be called because the current local window $\mathcal{W}$ should be updated before sensing a predictable conflict at line 8. Third, given a set of plans $\mathcal{P}$, a robot $i \in \mathcal{R}$, a current set of locations $\mathcal{S}$, i's local time step k and the current local window $\mathcal{W}$, SENSECONFLICT returns a tuple of an IC ψ and an OC ϕ. If no conflict is detected, the IC ψ is empty. Regardless of the existence of conflicts, SENSECONFLICT also returns an OC ϕ. This may require to communicate with other agents (we will explain in the next subsection). Fourth, the contribution value γ is used in RECOMPUTECONT, CONVERGENCE and CONVERGENCE. In the next subsection, we will explain the details about how to update the contribution value γ and how the contribution value γ affects the set of plans $\mathcal{P}$.

Correctness: For Alg. 1, we need to show two conditions. First, if a given problem instance is valid (solvable), robot $i \in \mathcal{R}$ eventually reaches its goal location. If there is no conflict from the initial location $\mathcal{I}[i]$ to its goal location $\mathcal{G}[i]$, it can progress through its plan while sensing conflicts at line 8 and proceeding one step at line 12 until it reaches its goal. Whenever there is a conflict, it always computes a valid plan. At line 16, robot i checks if it is already involved in a conflict (with ω). If $\omega \neq \emptyset$ (i.e., it is already involved in a conflict), it merges the OC (i.e., ϕ in Alg. 1) with ω, and then call PUSHANDPULL for i. In line 24, if $\mathcal{P}'$ is not empty, it means that CONVERGENCE returns a new plan $\mathcal{P}'$. If $\mathcal{P}'$ is empty, then robot i calls PUSHANDPULL. In both cases, the returned plan $\mathcal{P}'$ is either from CONVERGENCE or PUSHANDPULL. We have shown that CONVERGENCE or PUSHANDPULL always returns a valid plan in [23] if a valid solution exists.

Second, if a given problem instance is invalid (unsolvable), Alg. 1 returns False. In order to resolve a conflict, Alg. 1 first calls CONVERGENCE at line 23 which is for optimistic decoupling in DisCoF. Then, if it cannot compute its new plan, it calls PUSHANDPULL at line 27 which is for pessimistic decoupling. In [23], we showed DisCoF guarantees the completeness, and DisCoF uses these two conflict resolution processes in order to resolve its conflict. Hence, if a solution exists, the combination of these two processes returns a solution. However, if a solution does not exists, it returns False. At line 28, it can check whether it returns a solution or not. If not, it returns false.

We remark that PROCEEDONESTEP in line 12 always results in the robot proceeding one step forward in its plan. If robot i has already reached its final goal (while there are robots that still need to reach their goals), proceeding one step in this case simply adds a step for robot i to stay.

B. Communication and Leader Selection

There are two major cases in which robots communicate with each other in DisCoF$^+$. The first case is to detect predictable conflicts. For detecting conflicts, given a robot $i \in \mathcal{R}$, SENSECONFLICT requires the following steps:

1) Check nearby environment (i.e., $\mathcal{W}$) through a sensor for other robots (e.g., a laser sensor).
2) Compute the OC ϕ of robot i.
3) Communicate with robots in ϕ to obtain their plans, then check if predictable conflicts exist among them.

In the above process, the first step does not require any communication between robots; it only depends on sensors. Since robots know the environment (i.e., G), they can easily detect when there are moving robots nearby using range sensors.

The second step requires the use of a message relay protocol to compute the OC ϕ. This is because OC ϕ includes robots which cannot directly communicate with the robot i which originally tried to determine its OC ϕ. Even though it computed an OC ϕ in its previous time step, the OC ϕ can be changed whenever SENSECONFLICT is called. This is because each robot in the OC has its own asynchronous time if it is not involved in any conflicts and it can update its OC without considering other members. In this way, if one of the members in the OC moves out of its neighbors' sensing range before communicating with its neighbors, other members cannot update their own OC. In addition, if each member in an OC is involved in a conflict, all the members have a synchronized time step until reaching their local goals. In this case, computing a new OC is still required because each member of the OC can meet another group and each member can update their own OC propagating new information to each other. Hence, whenever each agent calls SENSECONFLICT, it should communicate with others so that it can update its OC.

In the third step, once robot i obtains all the plans of the robots in ϕ, it can check these plans against its own plan for predictable conflicts (from its current time step to the next β steps [23]). In this case, after electing a leader of the IC ψ, the leader computes the new plan for the IC ψ and communicates the new plan back to the others in the IC.[9] In order to compute a new plan, the leader tries CONVERGENCE. If CONVERGENCE returns a valid set of plans $\mathcal{P}'$, then the leader can pass $\mathcal{P}'$ to others. If not, the leader begins PUSHANDPULL. However, in PUSHANDPULL a new leader is selected which is based on the priorities of subproblems. Then, the new leader will send the new plan $\mathcal{P}'$ (which is computed from PUSHANDPULL) back to others in the OC ϕ.

The second case in which robots communicate is to synchronize planning and execution among robots in an OC. Note that robots in different OCs proceed independently and asynchronously. Since planning and plan execution are synchronized within an OC, it is guaranteed that no collision can occur among robots in the OC. In PROCEEDONESTEP, each robot in the OC executes a single plan step, communicates this to the rest of the robots in the OC (through broadcasting to the local network), and then it halts. Only after all robots in the OC have completed a plan step are they free to execute

[9] The simplest voting mechanism is to elect the robot with the smallest ID in the group.

another, thus achieving synchronization.[10] However, when robots move out of the communication range, they do not synchronize their plan steps anymore.

C. Flexible Decoupling

Flexible decoupling is achieved with the help of contribution values. Contribution values are assigned in DisCoF to each robot in the CONVERGENCE process (in optimistic decoupling) in which the robots must compute an update to the current plan to avoid potential conflicts. Contribution values are introduced in DisCoF to ensure that robots are jointly making progress to their goals. In DisCoF, when the CONVERGENCE process fails, robots are in a coupling group, running on the plan computed by PUSHANDPULL until they reach their goals. In DisCoF$^+$, however, robots that are executing PUSHANDPULL can also decouple by checking whether certain conditions involving the contribution values hold.

Next, we discuss the new decoupling strategy in DisCoF$^+$ which is illustrated in the following example. Suppose that a conflict is predicted between two robots. Then, an IC ψ (initially including only the two robots) is formed and there is an associated OC ϕ for ψ. When the leader of ψ makes a new plan in the CONVERGENCE process, if the leader cannot find a new plan that avoids the conflict with the current set of conflicting robots ψ, then the set of conflicting robots gradually expends (until becoming ϕ). When a new plan is found, DisCoF$^+$ associates each robot with a *contribution value* γ which captures the individual contribution of the robot to the summation of shortest distances from all robots' current locations to their goal locations.

For the remaining part of this section, we will use the cost relation $\mathcal{C} : V \times V \to \mathbb{N}$. For example, $\mathcal{C}(v_1, v_2)$ is the distance of the shortest path from node v_1 to node v_2.

At the very beginning of a problem instance, the contribution value γ is initialized to be 0 for all robots. Given a predictable conflict at time step k, a set of conflicting robots ϕ, the set of current locations S_k for ϕ and the set of goal locations $\mathcal{G}$, the new plan $\mathcal{Q}$ (where $|\mathcal{Q}| < \beta \in \mathbb{N}$)[11] must avoid collisions and satisfy the following:

$$\sum_{i \in \phi} \mathcal{C}(\mathcal{S}_k[i], \mathcal{G}[i]) + \gamma_k^-[i] > \sum_{i \in \phi} \mathcal{C}(\mathcal{S}_k[i](\mathcal{Q}[i]), \mathcal{G}[i]) \quad (2)$$

where $\gamma_k^-[i]$ is the contribution value that is associated with robot i at the time step k and $\mathcal{S}_k[i](\mathcal{Q}[i])$ is a local goal for each $i \in \phi$, i.e., the position reached by each robot i after executing plan $\mathcal{Q}[i]$.

We remark that while k in Eq. (2) is a constant in DisCoF, in DisCoF$^+$, k represents the synchronized current time step for the group of robots within ϕ which may differ between OCs.

[10] We assume that in a fixed amount time, each robot can complete its own movement and within the communication range there is no problem to communicate with each other.

[11] We assume that the length of the plan $\mathcal{Q}$ is bigger than β. If the length of some agent i's plan $\mathcal{Q}[i]$ has shorter than β, then the last state $S_k[i](\mathcal{Q}[i])$ should be appended at the end of $\mathcal{Q}[i]$ until $|\mathcal{Q}[i]| \geq \beta$.

An interesting point of Eq. (2) is that the new plan $\mathcal{Q}$ may not satisfy Eq. (2) during the execution of $\mathcal{Q}$, as long as Eq. (2) is satisfied after $\mathcal{Q}$ has completed. After executing the new local plan $\mathcal{Q}$, each agent reaches its local goal. In this way, they avoid the predicted conflict. Then, each robot $i \in \phi$ can decouple, following its individual plan from the local goal $\mathcal{S}_k[i](Q[i])$ to its goal $\mathcal{G}[i]$. Given a predicted conflict at the current time step and a computed $\mathcal{Q}$, the contribution value γ while executing the actions in $\mathcal{Q}$ is updated for robot i in ϕ as follows:

$$\gamma_{k+\delta}[i] = \mathcal{C}(\mathcal{S}_k[i](\mathcal{Q}[i]), \mathcal{G}[i]) - \mathcal{C}(\mathcal{S}_{k+\delta}[i], \mathcal{G}[i]) \quad (3)$$

where $0 \leq \delta \leq |\mathcal{Q}|$ and $\mathcal{S}_{k+\delta}[i] = \mathcal{S}_k[i](\mathcal{Q}_{1,1+\delta}[i])$. We remark that δ is a relative time step after the robots have formed an OC. For all robots in a group, δ is the same. This update continues until the robot become involved in other conflicts or the value becomes 0.

In DisCoF [23], the contribution value γ is only used for the CONVERGENCE process, and robots do not update their contribution values when a coupling group is formed and robots start PUSHANDPULL. This can lead to inefficient behaviors, e.g., when the leader's goal location is located opposite to where the others' goals are located.

In DisCoF, the only way to reduce the size of a coupling group is to have the current leader reach its goal. Then, a new leader will be selected and the remaining robots will follow the new leader to its goal. This is clearly an inefficient solution. In DisCoF$^+$, we use the contribution values γ also in PUSHANDPULL, such that robots can decouple even before the leader reaches its goal.

Next, we discuss how the contribution values can be used in the PUSHANDPULL process. More specifically, we provide a decoupling condition for a coupling group to check which determines when the robots in the group can decouple while executing the PUSHANDPULL process. Suppose that there is a coupling group ω. After ω computes a new plan $\mathcal{P}'$ (in PUSHANDPULL), each robot in ω will progress using the plan. During this execution, robots continue recomputing their contribution values γ as in Eq. (3). At any step, if the following condition holds, then the group can be decoupled:

$$\sum_{i \in \omega} \mathcal{C}(S_k[i], \mathcal{G}[i]) + \gamma_k^-[i] > \sum_{i \in \omega} \mathcal{C}(S_{k+\delta}[i], \mathcal{G}[i]) \quad (4)$$

where k is the time step when PUSHANDPULL starts planning and $k + \delta$ is the current time step such that $0 < \delta \in \mathbb{N}$. $\gamma_k^-[i]$ is the contribution value that robot $i \in \omega$ had before the PUSHANDPULL returned its plan.

Intuitively, Eq. (4) is the condition when the summation of the length of the shortest-path from robots' current locations to their goal locations is less than the summation of the length of the shortest-path from their original coupling locations to their goal locations plus their contribution values just before forming the coupling group.

In Alg. 1, Eq. (4) is checked inside of RECOMPUTECONT at line 14. Given a set of current locations $\mathcal{S}$, a set of goal locations $\mathcal{G}$, and contribution values γ, if the condition holds, then RECOMPUTECONT returns an updated plan $\mathcal{P}$

(i.e., the shortest-path plan from $\mathcal{S}[i]$ to $\mathcal{G}[i]$) with an empty coupling group ω. Then, the coupling group ω becomes decoupled and each robot follows their individual plan. Otherwise, RECOMPUTECONT returns the current plan $\mathcal{P}$ without changing the coupling group ω. Then, the coupling group ω follows the current plan $\mathcal{P}$ which was computed from PUSHANDPULL.

When a coupling group is decoupled and it immediately predicts a conflict in the next iteration, it uses the conflict resolution process through CONVERGENCE, just as when fully decoupled robots have predicted conflicts. Even though we discussed the correctness of DisCoF$^+$ (Alg. 1), we also need to show that this new decoupling strategy is not subject to live-locks (i.e., robots are always making joint progress to the goals).

Theorem 4.1: The decoupling condition in Eq.(4) ensures that robots in the group gradually progress to their final goals.

Proof: For detail, see [8]. ■

V. RESULTS

In this section, we present some experimental results. First, we will show a simulation result on a physics based simulator. Second, we will provide results from numerical experiments on artificial benchmarks.

A. Simulation in Webots

The simulation shown in Fig. 2 was created using Webots 7.3.0 and the included iRobot Create models. A grid environment was modeled which contained 30 iRobots and 40 obstacles placed at random locations. This instance is solvable, i.e., each robot can reach its goal position. Each iRobot was running with a controller which implemented DisCoF+. However, one exception was made: rather than being completely distributed and simulating ad hoc networks and localization, the robots communicated with a central supervisor which provided this information as well as synchronization for robots involved in a conflict, i.e., in the same OC. Robots in different outer closures acted completely asynchronously, but robots in the same outer closure were synchronized if a conflict was detected between any of the member robots.

The target computer for the simulation was a MacBook Pro running Mac OS X 10.10.2 with a 2.3GHz i7 and 16GB of RAM. The simulation was run two times: once with decoupling enabled and once with decoupling disabled. Decoupling enabled yielded a total simulation duration of 3 minutes and 23 seconds. Out of all robots, the maximum number of steps required to reach their destination was 40. When decoupling was disabled, it yielded a total simulation duration of 5 minutes and 1 second. Out of all robots, the maximum number of steps required to reach their destination was 54.

These results are interesting in two aspects: the total running time and the number of maximum steps. First, in terms of the total running time, enabling decoupling performs significantly better than without decoupling. The simulation took only 67% of the time that the other did. Second, in terms of the maximum steps, enabling decoupling took only 74% of the steps than without decoupling. The reason for this discrepancy is that with decoupling enabled there are more stay actions in which a robot's action is to stay where it is. Since robots are asynchronous except for when they are in a conflict, this means robots will take less time to complete a plan with stay actions compared to one that doesn't. It is expected that environments which remains more complex plans will benefit from this fact even more.

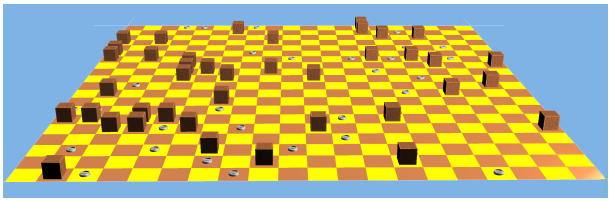

Fig. 2. A simulation environment in Webots modeling a 20×20 grid world with a 10% wooden boxes as obstacles. In this environment, there are 30 iRobot Create finding their path to their goal positions.

We provide the demo video for this simulation (which is also submitted as an attachment to this paper). In addition, you may refer to the videos at the following URL: `https://www.assembla.com/spaces/discof/wiki/DisCoF_Plus`.

B. Simulation Experiments on benchmarks

In order to evaluate the improvement of DisCoF$^+$ over DisCoF, we execute a number of numerical experiments. For these experiments, we used a 3.2GHz i7 and 8GB of RAM in Cygwin environment which runs on Windows 8.1. Our prototype implementation is written in Python 2.7.2.

Since we only want to get the total number of concurrent steps and the computation time for these experiments, instead of using the Webots simulator, we used a simple discrete time simulator which does not simulate the physics of the robots. In addition, we have not computed the overhead of any communication between the robots. Hence, we are comparing the total number of steps and the computation times between DisCoF and DisCoF$^+$.

As a result of this implementation, an approximate running time is calculated for each problem instance by summing the computation time and the movement time, where the movement time is the amount of time required to execute all steps assuming 5 seconds per step.

In order to perform the experimental analysis, instead of scaling up the number of robots, we increase the density of the environment. That is, we increase obstacle rates in the environment. The experiment was performed on a 20×20 grid environment with 30 robots. Obstacles were randomly generated according to their rate which is defined as the percentage of the grid environment that is considered to be an obstacle. Table I shows the results for 100 instances of DisCoF and DisCoF$^+$ as the obstacle rate was varied from 5% to 20%.

In all cases, DisCoF$^+$ needed 24% to 42% less steps than DisCoF's result and DisCoF$^+$ took 25% to 43% less than

	DisCoF						DisCoF$^+$ (DisCoF$^+$/DisCoF)					
	COMP. TIME		STEPS		APPROX. RUN TIME		COMP. TIME		STEPS		APPROX. RUN TIME	
OBSTACLES	AVG	STD	AVG	STD	AVG	STD	AVG	STD	AVG	STD	AVG	STD
5%	10.064	8.405	352.35	356.207	1771.815	1788.861	10.733 (1.0086)	22.068 (0.931)	63.95 (0.4266)	80.632 (0.356)	330.483 (0.43)	423.885 (0.3555)
10%	13.19	10.372	521.1	521.24	2618.69	2615.82	14.37 (1.061)	36.52 (1.538)	73.51 (0.344)	108.93 (0.346)	381.92 (0.348)	579.065 (0.348)
15%	17.6318	13.296	653.67	580.01	3285.982	2911.463	23.92 (1.217)	49.768 (1.3)	99.18 (0.294)	157.356 (0.312)	519.82 (0.3)	831.07 (0.314)
20%	26.39	14.009	954.46	620.08	4798.691	3111.208	52.391 (1.942)	75.8 (2.3989)	175.9192 (0.2427)	218.859 (0.3132)	931.987 (0.2535)	1161.61 (0.3242)

TABLE I

SIMULATION EXPERIMENTS: COMP. TIME REPRESENTS THE TOTAL COMPUTATION TIME IN SEC, STEPS REPRESENTS CONCURRENT TIME STEPS FOR ENTIRE ROBOTS' PLAN, AND APPROX. RUN TIME REPRESENTS APPROXIMATE RUNNING TIME IN SEC. AVG STANDS FOR AVERAGE AND STD FOR STANDARD DEVIATION. THE RATIO INSIDE THE PARENTHESIS IS DISCOF$^+$/DISCOF.

DisCoF's approximate run time.

The time ratio in Table I indicates that if the environment is less populated, then decoupling makes better quality plans in terms of the total number of concurrent steps and the total computation time of plans.

Despite the fact that DisCoF$^+$ consistently outperforms DisCoF in the approximate run time, it is important to comment on the computation time. When the environment is dense, it takes more computation time. This is because in dense environments groups that decouple may have to re-couple with a higher frequency. That is, when it recouples, a group should make a new plan which requires extra computation time.

VI. CONCLUSIONS

In this paper, we introduced DisCoF$^+$ which is an asynchronous extension of our previous work. We also introduced a strategy of decoupling in DisCoF$^+$. Through simulations, we showed how DisCoF$^+$ works in a simulated grid environment to resolve predictable conflicts in a distributed fashion. We also provided simulation experiments to compare DisCoF with DisCoF$^+$. In moderately populated environments, the decoupling approach shows better results than DisCoF. In future work, we plan to devise different approaches for decoupling and, also, heuristics for ordering robots while performing PUSHANDPULL so that when at any point in time a decoupling occurs, the conflicts are minimized.

REFERENCES

[1] N. Ayanian, D. Rus, and V. 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] Z. Bnaya and A. Felner. Conflict-oriented windowed hierarchical cooperative A*. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation*, 2014.

[3] C. Clark, S. 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] B. de Wilde, A. W. ter Mors, and C. Witteveen. Push and rotate: Co-operative multi-agent path planning. In *12th International Conference on Autonomous Agents and Multiagent Systems*, 2013.

[5] V. R. Desaraju and J. P. How. Decentralized path planning for multi-agent teams with complex constraints. *Autonomous Robots*, 32(4):385–403, 2012.

[6] J. Hopcroft, J. 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] R. Jansen and N. 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] K. Kim, J. Campbell, W. Duong, Y. Zhang, and G. Fainekos. DisCoF$^+$: Asynchronous DisCoF with flexible decoupling for co-operative pathfinding in distributed systems. Technical report, http://arxiv.org/abs/1506.03540.

[9] L. Liu and D. 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.

[10] R. Luna and K. Bekris. Efficient and complete centralized multirobot path planning. In *IEEE/RSJ Int. Conf. on IROS*, 2011.

[11] M. Otte, J. Bialkowski, and E. Frazzoli. Any-com collision checking: Sharing certificates in decentralized multi-robot teams. In *Proceedings of the 2014 IEEE ICRA*.

[12] L. E. Parker. *Encyclopedia of Complexity and System Science*, chapter Path Planning and Motion Coordination in Multiple Mobile Robot Teams. Springer, 2009.

[13] M. Peasgood, C. 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.

[14] M. 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.

[15] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219(0):40 – 66, 2015.

[16] D. Silver. Cooperative pathfinding. In *Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005.

[17] T. Standley. Finding optimal solutions to cooperative pathfinding problems. In *AAAI Conference on Artificial Intelligence*, 2010.

[18] T. Standley and R. Korf. Complete algorithms for cooperative pathfinding problems. In *Proceedings of the 22nd International Joint Conference on Artifical Intelligence*, 2011.

[19] N. Sturtevant and M. Buro. Improving collaborative pathfinding using map abstraction. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 80–85, 2006.

[20] G. Wagner, M. Kang, and H. Choset. Probabilistic path planning for multiple robots with subdimensional expansion. In *IEEE International Conference on Robotics and Automation, ICRA*, 2012.

[21] K. C. Wang and A. Botea. Fast and memory-efficient multi-agent pathfinding. In *International Conference on Automated Planning and Scheduling*, pages 380–387, 2008.

[22] 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.

[23] Y. Zhang, K. Kim, and G. Fainekos. Discof: Cooperative pathfinding in distributed systems with limited sensing and communication range. In *to appear in International Symposium on Distributed Autonomous Robotic Systems*, 2014.