Parallel Processing Letters

Beyond Rings: Gathering in 1-Interval Connected Graphs *

Othon Michail
Department of Computer Science, University of Liverpool, UK

Paul G. Spirakis
Department of Computer Science, University of Liverpool, UK
Computer Engineering and Informatics Department, University of Patras, Greece

Michail Theofilatos
Department of Computer Science, University of Liverpool, UK

ABSTRACT

We examine the problem of gathering $k \geq 2$ agents (or multi-agent rendezvous) in dynamic graphs which may change in every round. We consider a variant of the 1-interval connectivity model [Kuhn et al, STOC 2010] in which all instances (snapshots) are always connected spanning subgraphs of an underlying graph, not necessarily a clique. The agents are identical and not equipped with explicit communication capabilities, and are initially arbitrarily positioned on the graph. The problem is for the agents to gather at the same node, not fixed in advance. We first show that the problem becomes impossible to solve if the underlying graph has a cycle. In light of this, we study a relaxed version of this problem, called *weak gathering*, where the agents are allowed to gather either at the same node, or at two adjacent nodes. Our goal is to characterize the class of 1-interval connected graphs and initial configurations in which the problem is solvable, both with and without *homebases*. On the negative side we show that when the underlying graph contains a spanning bicyclic subgraph and satisfies an additional connectivity property, *weak gathering* is unsolvable, thus we concentrate mainly on unicyclic graphs. As we show, in most instances of initial agent configurations, the agents must meet on the cycle. This adds an additional difficulty to the problem, as they need to explore the graph and recognize the nodes that form the cycle. We provide a deterministic algorithm for the solvable cases of this problem that runs in $O(n^2 + nk)$ number of rounds.

Keywords: gathering; weak gathering; dynamic graphs; unicyclic graphs; mobile agents.

1. Introduction

*All authors were supported by the EEE/CS initiative NeST. The last author was also supported by the Leverhulme Research Centre for Functional Materials Design. This work was partially supported by the EPSRC Grant EP/P02002X/1 on Algorithmic Aspects of Temporal Graphs.

1.1. *Previous Work*

The problem of *gathering* on graphs requires a set of k identical mobile agents that operate in Look-Compute-Move cycles, to end up in the same node. In each cycle, an agent takes a snapshot of its immediate neighborhood (Look), performs some computations in order to decide whether to move to one of its adjacent nodes, or remain idle (Compute), and in the former case makes an instantaneous move to that neighbor (Move).

The feasibility of *gathering* has been extensively studied in the static setting, and under various assumptions. A very common assumption that makes the problem solvable in ring graphs is for the agents to have distinct identities [1, 2, 3]. Alternatively, another assumption which pertains to the communication capabilities of the agents, is either to supply each node with a *whiteboard* where the agents can leave notes as they travel [4], mark the nodes that the agents are initially placed, or provide the agents with a constant number of movable tokens that can be placed on nodes, picked up, and carried while moving [5]. Under the first communication assumption the problem becomes solvable even in the presence of some faults [6, 7].

In [8], the authors study the feasibility of gathering a set of identical and without explicit communication capabilities agents in dynamic rings (1-interval connectivity, [9]). As *strict gathering* becomes infeasible in this setting, they focus on a variation of gathering, called *weak gathering*, where the agents are allowed to gather either at the same node, or at two adjacent nodes. They investigate the impact that *chirality* (i.e., common sense of orientation on the cycle) and *cross detection* (i.e., the ability to detect whether some other agent is traversing the same edge in the same round) have on the solvability of the problem. In order to drop the latter assumption, they later construct a mechanism which avoids agents crossing each other (i.e., no agents traverse the same edge at the same round and in opposite directions), called *Logic Ring*. To enable feasibility of *weak gathering*, they empower the agents with some minimal form of implicit communication, called *homebases* (the nodes that the agents are initially placed are identified by an identical mark, visible to any agent passing by it). They provide a complete characterization of the classes of initial configurations from which *weak gathering* is solvable in the presence or absence of cross detection and chirality, by providing polynomial time distributed algorithms. They prove that without chirality, knowledge of the ring size is strictly more powerful than knowledge of the number of agents. Finally, with chirality, they show that knowledge of the ring size can be substituted by knowledge of the number of agents, yielding the same classes of feasible initial configurations.

In [10] the authors investigate the feasibility of the decentralized (or *live*) exploration problem in 1-interval connected rings by a set of mobile agents. They consider both the fully synchronous and semi-synchronous cases and study the impact that *anonymity* and knowledge of the ring size has on the solvability of the problem. Other recent work [11] has considered the broadcasting problem in dynamic, constantly connected, networks, where the agents can communicate when they meet at

a node, and they have global visibility allowing them to see the location of other agents in the graph. Finally, exploration by $O(n)$ agents of dynamic tori graphs has been investigated in [12], and, in a very recent work, exploration in time-varying graphs (including 1-interval connectivity) of arbitrary topology in [13].

1.2. *Contribution*

The existing literature on the gathering and rendezvous problems is extensive and has been examined under various assumptions for both the environment that the agents navigate and the capabilities of the agents (for surveys see [14, 15]). Despite their differences, they investigate these problems in the case where the topological structure does not change over time (i.e., they only consider static graphs). Recently, there has been a growing interest in studying these problems in dynamic settings, with [8] being the first work that examines the problem of *weak gathering* in 1-interval connected ring graphs. Embarking from this work, we investigate if and under what assumptions we can go beyond rings and how far, in the presence or absence of *homebases*. Our main result is a distributed algorithm that solves *weak gathering* in unicyclic graphs. A *unicyclic* graph is a connected graph containing exactly one cycle. Observe that ring graphs are special cases of unicyclic graphs.

We use a traditional model in the literature of autonomous mobile agents on graphs (see, e.g., [16]), and we consider it in a dynamic synchronous setting. In particular, in this model the edges are locally labeled in each node, and at any round some edges can be missing (i.e., being disabled), provided that the resulting graph is connected. This means that the snapshots of the dynamic graph are always spanning and connected subgraphs of some given underlying graph. This notion of dynamicity includes the classic 1-interval connectivity as a sub-case when the underlying graph is a clique.

We start a characterization of the class of solvable graphs in the aforementioned generalized 1-interval connectivity setting, and we study the effect that port labels have on the solvability of *weak gathering*. We show that *weak gathering* is unsolvable when the underlying graph contains a spanning bicyclic subgraph and satisfies an additional connectivity property, regardless of any other additional assumptions (i.e., communication or knowledge of graph properties). In light of this, we then focus on unicyclic graphs, and we study the classes of initial configurations on which *weak gathering* is feasible in the presence or absence of *homebases*. In particular, we characterize the classes of unicyclic graphs in which certain symmetries occur and would render impossible the problem of symmetry breaking. We show that if neither the size of the graph nor the number of agents is known, then the agents are not able to distinguish between symmetric and asymmetric configurations. The additional difficulty in unicyclic graphs comes from the fact that in most instances of initial agent configurations, the agents must necessarily gather on the unique cycle. This requires them to perform some sort of exploration in order to reach the cycle, while the scheduler can choose some agent, or agents, and delay them.

Graph class	Assumptions	Feasibility of *weak gathering*
Graphs in $\mathcal{F}$ Definition 3.1	Any	Infeasible Proposition 3.2
Symmetric unicyclic graphs in $S_s \cap S_a$	Any	Infeasible Lemma 3.2
Unicyclic graphs	No knowledge of n and k	Infeasible Proposition 3.3
Unicyclic graphs not in S_a	*Homebases*, knowledge of n and k	Feasible Theorem 4.1
Unicyclic graphs not in S_s	Knowledge of n and k	Feasible Theorem 4.1

Table 1: Summary of our results for the *weak gathering* problem. Assumptions include the existence of *homebases*, knowledge of the graph size n, and knowledge of the number of agents k

In [8] the authors utilized *homebases* in order to break the symmetry on the ring, however, in our setting we can exploit the topological assymetries of the graph and the port labeling to solve the problem. We then show that *homebases* can be used in order to expand the class of feasible configurations, that is, the initial placement of the agents might create some additional topological assymetries. We also assume that the agents have cross-detection, and in Section 5 we discuss about how the mechanism of Di Luna et al. [8], that avoids agents crossing each other, could be used in order to drop this assumption, and we leave it as an open problem.

We then provide a deterministic algorithm that solves *weak gathering* in all asymmetric unicyclic graphs, and runs in a polynomial number of synchronous rounds. For the cases of symmetric unicyclic graphs with symmetric initial agent placement, we show that the problem becomes impossible to solve, and we leave as an open problem the case of symmetric unicyclic graphs and asymmetric initial agent placement. We carefully design a non-trivial mechanism that utilizes the graph topology and after $O(n^2+nk)$ rounds all agents reach and forever stay on the cycle. Given this, the second part of the algorithm guarantees that the agents (weakly) gather on the cycle, in $O(n)$ rounds.

In summary, we show that in a large class of graphs F *weak gathering* is unsolvable. Our paper establishes that *weak gathering* is solvable in unicyclic 1-interval connected graphs in $O(n^2+nk)$ time, and we leave a small gap for graphs $G \notin (F \cup \mathit{Unicyclic})$. For a summary of our results see Table 1.

1.2.1. *Organization of the paper*

In Section 2, we formally describe the model and we provide all necessary definitions. In Section 3 we provide impossibility results for (strict) *gathering*, and we describe the class of graphs where *weak gathering* is impossible to solve. In Sections 3.1 and

3.2 we provide a characterization of the feasible initial configurations in unicyclic graphs and the basic limitations of *weak gathering*. Finally, in Section 4 we provide our deterministic *weak gathering* algorithm and its analysis.

2. Model and Definitions

Dynamic Network Model. A network is modeled as an undirected connected graph $G_U = (V, E)$, referred to hereafter as an *underlying graph*. The number of nodes $n = |V|$ of the graph is called its *size*. Every node $u \in G_U$ has $\delta(u)$ incident edges, where $\delta(u)$ is its degree. For each of them, it associates a port and the ports are arbitrarily labeled with unique labels from the set $\{0, \ldots, \delta(u) - 1\}$. We call these labels the *port numbers*.

Given an underlying graph $G_U = (V, E)$ on n vertices, a *dynamic graph* on G_U is a sequence $G_D = \{G_t = (V, E_t) : t \in \mathbb{N}\}$ of graphs such that $E_t \subseteq E$ for all $t \in \mathbb{N}$. Every G_t is the snapshot of G_D at time-step t. We assume that the sequence G_D is controlled by an adversarial scheduler, subject to the constraint that the resulting dynamic graph should be 1-*interval connected*. The definition of 1-interval connectivity of [9] considers the case where the underlying graph is a complete clique. In our work, we generalize this to any underlying graph, meaning that $G'_D = (V, \bigcup_t E_t) \subseteq G_U$.

Definition 2.1 (Generalized 1-interval connectivity) *A dynamic graph G_D is generalized* 1*-interval connected if for every integer $t \geq 0$, the snapshot $G_t = (V, E_t)$ is a connected and spanning subgraph of a given underlying graph G_U.*

Agents. There is a set $A = \{\alpha_1, \ldots, \alpha_k\}$ of k *anonymous* computational entities (also called *agents*), each provided with memory and computational capabilities, that execute the same protocol and can move on the graph. During the execution of the protocol, an agent learns the local port number by which it enters a node and the degree of the node. The agents are initially arbitrarily placed on some nodes of the graph, and they are not aware of the other agents' positions.

More than one agent can be in the same node and may move through the same port number (i.e., the same edge) in the same round. We say that an agent α is *blocked* if the edge that α decided to cross in the current round is disabled by the scheduler. We consider the *strong multiplicity detection* model, in which each agent can count the number of agents in its current node. Based on that information, the port labeling, and the contents of its memory, it determines whether or not to move, and through which port number. In addition, the agents do not have any visibility around them, meaning that we do not allow them to see agents on their adjacent nodes.

We say that the system has *cross detection* when the agents have the ability to detect whether some other agent is traversing the same edge in opposite direction during the same round. We assume that the system has *cross detection*, and in Section 5 we discuss about how this assumption could be dropped.

We assume that the nodes of G do not have unique identifiers, and the agents do not have *explicit* communication capabilities. We do this in order to capture the limitations and the basic assumptions that make gathering in dynamic networks feasible. Finally, we assume that the agents do not have knowledge of any graph properties, other that its size.

Definition 2.2 (Homebases) *We call* homebases *the nodes that the agents are initially placed. Each node u is specially marked by a bit b_u, such that if $b_u = 0$ no agent was initially placed on u, while $b_u = 1$ means then at least one agent was initially placed on u (i.e., its a homebase). In addition, each agent can determine whether its current node is a homebase or not.*

Definition 2.3 (Gathering problem) *The* gathering *problem requires a set of k agents, initially arbitrarily placed on the graph, to gather within finite time at the same node of the underlying graph, not known to them in advance, and terminate.*

Definition 2.4 (Weak gathering problem) *The relaxed version of the gathering problem, called* weak gathering*, requires all agents to gather within finite time at the same node, or on two neighboring nodes of the underlying graph, and terminate.*

The above definition means that all agents must terminate in at most two nodes of the graph that are adjacent in the underlying graph. Finally, throughout the paper, we call *unicyclic* graphs the connected graphs that have exactly one cycle, and *bicyclic* graphs the connected graphs that have exactly two cycles, with possibly a single common vertex.

Definition 2.5 (Branch) *Let G be a unicyclic graph. Then G consists of a unique cycle C of n_c nodes and b trees, where $0 \leq b \leq n_c$. Each tree B_{w_i} is rooted at a node $w_i \in C$, such that the only node of the intersection of B_{w_i} with C is w_i. We call these trees the* branches *of G. See Figure 1 for an example.*

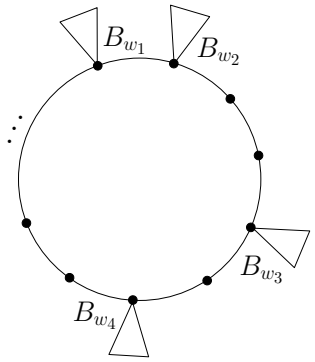


Fig. 1: Example of a graph G with a unique cycle C. Each B_{w_i} is a tree (or branch) of G rooted at $w_i \in C$.

3. Impossibility Results

In this section we start by showing that strict *gathering* is unsolvable in 1-interval connected graphs that have at least one cycle. In the model that we consider, the case of connected acyclic graphs is equivalent to static trees. We then focus on the *weak gathering* problem, and we show that in a large class of graphs *weak gathering* is unsolvable.

Proposition 3.1 *For any generalized* 1*-interval connected graph with at least one cycle, there exists an initial agent placement such that* gathering *is unsolvable, regardless of any communication assumptions and knowledge of graph properties (e.g., its size).*

Proof. Consider an underlying graph $G_U = (V, E)$, where there exists at least one cycle of size $c > 3$. Let C be an arbitrary such cycle of G_U. Consider an execution such that for each cycle $C' \neq C$, the scheduler disables an arbitrary edge in C' that does not belong to C. Call the resulting graph G'_U. G'_U can now be represented as a unicyclic graph, where each node $w \in C$ is the root of a connected tree, or branch, G_w.

Consider an initial agent placement where there are at least two agents α and α' that are placed on branches G_w and G_u, such that $w \neq u$. Then, all paths between α and α' contain at least one edge $e \in C$. Since our graph is 1-interval connected, the scheduler can additionally remove an edge of the cycle C in order to block the agents from reaching the same node, without violating the connectivity constraints. Therefore, they cannot achieve *gathering*.

Definition 3.1 *[Class F of graphs with blocking edges] Let G be a graph that has a spanning bicyclic subgraph with cycles C_1 and C_2. For any node u that belongs to at least one cycle, let G_u be the maximal connected subgraph, such that the intersection of the node set of G_u with C_1 and C_2 contains only u. We say that G belongs to class $\mathcal{F}$ if there are two edges $e_1 \in C_1$ and $e_2 \in C_2$, with endpoints u_1, w_1 and u_2, w_2 respectively, such that no node of G_{u_1} and G_{w_1} is adjacent with any node of G_{u_2} and G_{w_2} in G. Call these edges* blocking.

In other words, we say that a graph G belongs to $\mathcal{F}$ if G has a spanning bicyclic subgraph G_B with two cycles C_1 and C_2, such that all paths that contain two edges $e_1 \in C_1$ and $e_2 \in C_2$ also contain at least one more edge from each cycle. The fact that no node of G_{u_1} and G_{w_1} is adjacent with any node of G_{u_2} and G_{w_2} in G, and because each G_v contains all nodes of the corresponding connected component (i.e., it is maximal), it means that there is no path from any node of G_{u_1} and G_{w_1} to any node of G_{u_2} and G_{w_2} that does not contain at least one more edge from each cycle of G_B. An example is shown in Figure 2.

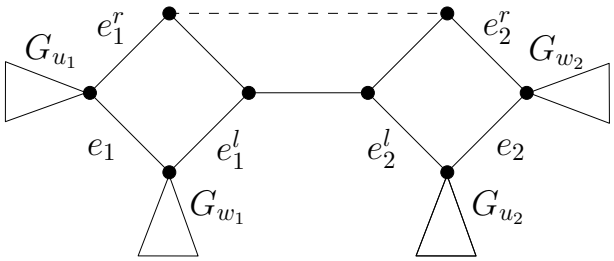


Fig. 2: Example of a graph $G \in \mathcal{F}$. Dashed lines do not belong to the spanning bicyclic subgraph. e_1 and e_2 are called *blocking* edges.

Proposition 3.2 *For any generalized* 1*-interval connected graph that belongs to* $\mathcal{F}$*, there exists an initial agent placement such that* weak gathering *is unsolvable, regardless of any communication assumptions and knowledge of graph properties.*

Proof. Take any underlying graph $G_U \in \mathcal{F}$, and let α and α' be two agents that are initially placed on some endpoint of two *blocking* edges $e_1 \in C_1$ and $e_2 \in C_2$, respectively. Let p_t and p'_t be the positions of these agents at round t.

Consider an execution such that the scheduler disables all edges that are not contained in the (spanning) bicyclic subgraph, and also disables the neighboring edges $e_t \neq e_1$ and $e'_t \neq e_2$ of p_t and p'_t on the corresponding cycles, when $p_t \in C_1$ and $p'_t \in C_2$, respectively. Observe that this does not violate the connectivity constraints, as these edges belong to different cycles. Then, in each round, an agent can either decide to wait at its current node, move on the other endpoint of e_1 (and e_2 for α'), or move towards a tree rooted at p_t (p'_t, respectively). Observe that in the last case, the distance in G_U between the agents is increased. This is because, by the definition of $\mathcal{F}$, the nodes of the trees starting from the endpoints of e_1 are not adjacent with any node of the corresponding trees of e_2 in G. Let w be a node that α moves to. All paths between w and α' pass through some endpoint of e_1. However, when α is on C_1, the scheduler blocks it from making any progress towards α', thus remain in distance at least two from α'. Symmetrically, the same holds also for α'. Therefore, the agents will never reach the same or two neighboring nodes, thus fail to solve *weak gathering*.

As an example, consider that graph of Figure 2 which belongs to $\mathcal{F}$, and two agents α and α' that are initially placed on u_1 and u_2, respectively. When α is on u_1, r_1^r is disabled and when it is on w_1, e_1^l is disabled. Similarly for α'. Observe that in any of these cases, the connectivity of the graph is maintained, while α is always blocked on some node of G_{u_1} or G_{w_1} and α' on some node of G_{u_2} or G_{w_2}. □

3.1. *Symmetric Initial Configurations in Unicyclic Graphs*

The main difficulty in solving gathering is symmetry which occurs in several ways, such as the topology of the graph, the port labeling, and the initial positions of the agents. Given that the agents are identical and there is no means of explicit communication, in case that the configuration is highly symmetric, the problem is clearly impossible to solve by deterministic means. The problem of deterministically breaking the symmetry is translated into the problem of distinguishing a node, or an edge for the *weak gathering* problem, where the agents should meet.

In light of the above impossibilities, we hereafter consider unicyclic graphs and we describe the class of symmetric initial configurations in such graphs, with and without *homebases*. We show that *weak gathering* is unsolvable in symmetric unicyclic graphs. Note that in this section we do not consider any graph dynamics, as we are only interested in identifying the graph classes that even in a static setting, the problem of symmetry breaking is impossible.

An initial configuration is defined by the graph, the port labeling, and the (initial) positions of the agents.

3.1.1. *Branch classes*

Call C the nodes of the unique cycle and B_i the branch rooted on $u_i \in C$, $\forall u_i \in C$, possibly consisting only of the root node u_i. We define a class I of indistinguishable branches, the class where all of the following hold:

1) Any two branches B and B' in I, rooted at u and u', respectively, are isomorphic.

2) For each pair of branches B and B' in I, the branches are label-preserving, meaning that vertices with equivalent port labels (i.e., the same) are mapped onto the vertices with equivalent port labels and vice versa. This means that for each pair of connected nodes $(u, w) \in B$ which is mapped onto the (connected) pair of nodes $(u', w') \in B'$ in that order, the port label of u leading to w is the same as the port label of u' leading to w' and vice versa.

3) For any two branches B and B' in I with root nodes u and u', respectively, the port labels of u and u' leading to their clockwise neighbor of the ring are the same. Similarly, the port labels of u and u' that lead to their counter-clockwise neighbors of the ring are the same.

3.1.2. *Symmetric configurations*

Let $\mathcal{I} = \{I_0, I_1, \ldots, I_l\}$ be the set of all distinct branch classes of a given unicyclic graph G. Let u_i denote the i-th node in the clockwise direction of the cycle, starting from an arbitrary node $u_0 \in C$, and $s_i \in \mathcal{I}$ be the class that the branch starting from u_i belongs to. We call a graph *symmetric* if the following holds: for each $I_j \in \mathcal{I}$ there exists a set of periods $P_j = \{p_0^j, p_1^j, \ldots, p_{m_j}^j\}$, where $p_z^j < |C|$ and $0 \leq z \leq m_j$, such that for each node u_i with $s_i = I_j$, there exists p_z^j such that $s_i = s_{(i+p_z^j) \mod |C|}$,

$\forall i < |C|$ (see examples in Fig. 3).

We call $\mathcal{S}$ the set of all possible configurations with k agents in a unicyclic graph and $\mathcal{S}_s$ the set of all symmetric unicyclic graphs.

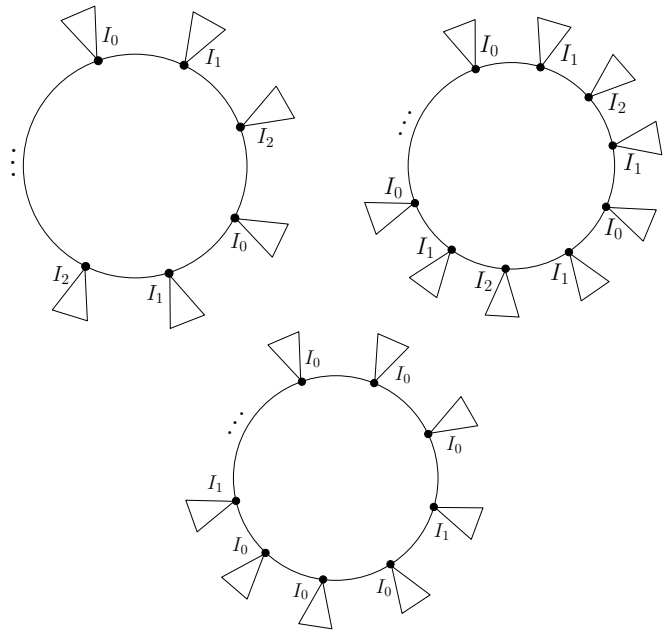


Fig. 3: Examples of symmetric configurations with periodic branches and port labels on the cycle. In the top left figure all unique branches have the same periodic appearance (i.e., $P_i = \{3\}$, $\forall i \in \{0, 1, 2\}$). In the top right and bottom figures, the branches have different periodic appearances (i.e., in the top right figure the sets of periods are $P_0 = \{4\}, P_1 = \{2\}$, and $P_2 = \{4\}$, while in the bottom figure the sets of periods are $P_0 = \{2, 4\}$ and $P_1 = \{4\}$).

3.1.3. *Agent position symmetries*

In a similar way we define the symmetries that are induced by the initial placement of the agents on the graph. These symmetries can only be defined in configurations of $\mathcal{S}_s$. Assume that each vertex is initially labeled with a bit b, indicating whether an agent is initially placed on that vertex, or not; call them agent labels. Then, we define the set $\mathcal{S}_a$ of such configurations as follows. For any pair of branches B and B' that belong to the same branch class I and have the same periodic appearance, the branches are label-preserving, meaning that vertices with the same agent labels are mapped onto the vertices with the same agent labels and vice versa.

We say that a configuration S is symmetric if $S \in \mathcal{S}_s$. If, additionally, the communication model allows homebases*, we call S symmetric if $S \in (\mathcal{S}_s \cap \mathcal{S}_a)$.*

Lemma 3.1 *A unique (leader) node can be elected in all non-symmetric configurations. We call these configurations* feasible*. This holds regardless of any communication assumptions and knowledge of graph properties.*

Proof. We prove this lemma by construction: we provide a deterministic algorithm that given a configuration S (the graph topology, the port labeling, and, if available, the *homebases*), such that $S \in \mathcal{S} \backslash \mathcal{S}_s$ or $S \in \mathcal{S} \backslash (\mathcal{S}_s \cap \mathcal{S}_a)$, elects a unique node as a leader.

Consider a rooted tree B with port labels. Starting from the root node u, a sequence that uniquely represents B can be constructed as follows. Consider a sequence $\mathcal{T}_B =< T_1, T_2 \ldots, T_\ell >$, $T_i = (t_0^i, t_1^i, \ldots, t_{d_i-2}^i)$, where $\ell = |B|$, each tuple T_i corresponds to a node of the tree, and d_i is its degree. Each t_j^i is the tree size rooted at each child node of T_i (i.e., we exclude T_i's parent node from T_i). Then, the size of the subtree rooted at T_i is $1 + \sum_{j=0}^{d_i-2} t_j^i$. Their order in T_i is defined by selecting port labels in ascending order. If we consider the case where identical labels exist on the initial positions of the agents (i.e., *homebases*), then the above sequences can be modified by having an H symbol in the beginning of the tuple T of each node where a *homebase* exists. Finally, the tuples $T_i \in \mathcal{T}_B$ are ordered in a DFS way, where we visit the children of each node by selecting the port labels in an ascending order. Observe that there is a simple algorithmic strategy that can be used by an agent to traverse the tree in a DFS way: when the agent arrives at node w through a port p, it leaves w through port $(p + 1)$ mod $\delta(w)$ in the next step. Initially, the agent starts by leaving the port 0 of the root node u.

Given an arbitrary orientation on the cycle, same for all branches, construct a set $\mathcal{C}_B = (\mathcal{T}_B, p_0, p_1)$ for each branch of the tree B, where p_0 is the port label of the root node of B that leads to its neighboring node on the arbitrarily chosen orientation and p_1 the port label that leads to its second neighbor of the cycle. For each distinct set $\mathcal{C}_B$, assign a unique label $a \in \mathbb{Z}$; call them *branch labels*. Observe that this yields an assignment of unique labels on branches that do not belong to the same branch class. The above construction can be then used to distinguish a node on C as follows. Let P_c^u and P_{ccl}^u be the sequences of branch labels as constructed above by following the clockwise and counter-clockwise directions respectively, starting from node $u \in C$. Let δ_{w_1} and δ'_{w_2} denote the two lexicographically minimum sequences of P_c^u and P_{ccl}^u of size $|C|$, $\forall u \in C$.

If there is a unique lexicographically minimum sequence, let that be δ_w, we elect w as the leader node. In case that the lexicographically minimum sequences δ_{w_1} and δ'_{w_2} are identical, and $w_1 \neq w_2$, it means that there is a unique axis of symmetry, equidistant from w_1 and w_2 (otherwise, the configuration would be symmetric). If that axis passes through one node w and one edge, we elect w as the leader node. If the axis passes through two nodes u_1 and u_2, there exist two lexicographically

minimum sequences of size $|C|/2$ starting from w_1 and w_2 that both contain either u_1 or u_2. Then, we elect u_1 or u_2 as the leader node, respectively. Observe that it is not possible that one sequence contains u_1 and the other u_2, as in that case the configuration would clearly be symmetric (i.e., the configuration would have two axes of symmetry). Similarly, if the unique axis of symmetry passes through two edges e_1 and e_2, the two lexicographically minimum sequences contain both either e_1 or e_2; let that edge be e_1. Observe that in this case the trees starting from the endpoints of e_1 have both been assigned the same label α. This means that the ports of e_1's endpoints are different (if they were the same, the labels would be different). Then, we elect as leader the endpoint of e_1 with the minimum label.

Lemma 3.2 *If the graph and the initial agent placement are symmetric,* weak gathering *cannot be solved. This holds regardless of* homebases *and knowledge of graph properties.*

Proof. Let S be a symmetric configuration with k agents, and B_u the branch starting from a node u of the cycle (containing u). Consider an execution in which no edge of the underlying graph ever becomes disabled.

Consider a symmetric initial agent placement. Then, call a group the agents that are mapped onto vertices of the same branch class with the same periodic appearance. The agents of each group will then perform exactly the same actions, based on the same observations. This means that for each group of agents, they will always be on branches that belong to the same branch class with the same periodic appearance, and their current positions will be always mapped onto the same vertices. If they move on the cycle, they will again perform the same actions (i.e, they will move either clockwise, or counter-clockwise), thus, the distance between consecutive agents of the same group will never change. Observe that this holds regardless of the existence of *homebases*.

The only case that we haven't examined is the case without *homebases*, where the graph is symmetric, and the initial agent placement is asymmetric. Consider a symmetric ring graph, where all port labels that lead to the clockwise neighbors are the same, and the port labels that lead to the counter-clockwise neighbors are also the same, and no edge is ever missing. Independently of the initial agent placement, all agents operate with the same observations in each round, therefore they all move either clockwise, or counter-clockwise (i.e., the distance between consecutive agents of the ring remain always the same). Therefore, a more precise characterization of the feasible graph configurations is necessary for this case. We believe that without a way to elect a leader, same for all agents, the problem of *weak gathering* becomes impossible, and we leave this as an open problem.

3.2. *Additional Limitations on the Solvability of* Weak Gathering

In this section we examine the impact that some additional limitations have on the solvability of *weak gathering*. First, observe that in generalized 1-interval connected

unicyclic graphs, the scheduler can completely block an agent from reaching some part of the graph. This implies that if some other agent moves only on that part, then these agents would never meet or end up in neighboring nodes. As we show in the following lemma, this problem can be overcome only if all agents explore the graph, identify the cycle, and solve the problem there.

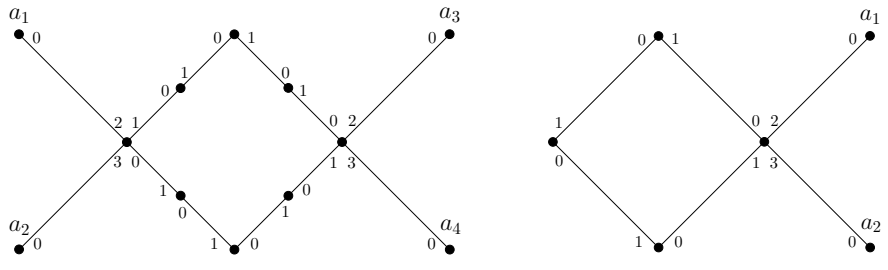
Lemma 3.3 *For any generalized* 1*-interval connected unicyclic graph with cycle* C *of size* $|C| > 3$*, there exists an initial agent placement such that* weak gathering *can only be achieved on the cycle. This holds regardless of any communication assumptions and knowledge of graph properties.*

Proof. We call a branch empty if it only consists of the root node. Consider any unicyclic graph with a cycle C, $|C| > 3$, and at least one non-empty branch (if all nodes have empty branches, then the lemma trivially follows). Let B_w be the branch rooted on node $w \in C$. Let α be an agent that is initially placed on some node of $B_w \backslash w$, and an agent α' that is initially placed on a (possibly empty) branch B_u, such that u is in distance at least two from w. Then, consider an execution in which the scheduler selects α' and blocks it in distance two from B_w (i.e., whenever it is at distance two from w, it disables the corresponding edge). Therefore, in order to solve *weak gathering*, agent α must first reach the cycle. Additionally, the scheduler can always block them from reaching the same branch, as it can keep them at distance at least one. If some agent decides to move towards a branch, then the scheduler can still block the other agent from reaching that branch.

The above lemma means that the agents must first explore the graph in order to identify the nodes of the cycle C and gather on some node $v \in C$, otherwise, the scheduler can always block some agent from reaching the rest of them.

Proposition 3.3 *If neither* n *nor* k *are known, then the agents cannot distinguish periodic from aperiodic graphs. This holds regardless of* homebases.

Proof. Let G_1 be the graph of Figure 4a and G_2 be the graph of Figure 4b. The numbers represent the local port labels of each node, and a_i are the initial positions of the agents.



(a) A symmetric graph (b) A non symmetric graph

Fig. 4: Indistinguishable unicyclic graphs

Consider an execution where no edge is missing at any round. If neither k nor n are known to the agents, these two graphs are indistinguishable between each other, even when the initial positions of the agents are identically marked (*homebases*). If the agents are able to recognize infeasibility in G_1 (because of being symmetric), then this would also wrongly happen in G_2. Otherwise, if the agents solved the problem in G_2 and terminated, then the agents in G_1 would also terminate in two nodes that are not neighbors.

Given any unicyclic graph G_u with k agents, we can construct periodic unicyclic graphs that the agents cannot distinguish from G_u. Let $C_u = \{u_1, u_2, \ldots, u_c\}$ be the set of nodes of the cycle in G_u. Then, to construct a periodic graph G_p, construct a cycle $C_p = \{u_1^1, u_2^1, \ldots, u_c^1, u_1^2, u_2^2, \ldots, u_c^2, \ldots, u_1^p, u_2^p, \ldots, u_c^p\}$, and for each i, all nodes $u_i^j, \forall j$, have the same port numbers between them, and with the corresponding nodes of C_u. In addition, for each i, the trees starting from nodes $u_i^j \in C_p, \forall j$ are exact copies of the corresponding trees of nodes $u_i \in C_u$. Finally, all agents of G_u are also mapped onto all copies of the corresponding nodes of G_p (i.e., we have pk agents in G_p). Then, similarly to Figure 4, the agents cannot distinguish G_u from any G_p.

4. An Algorithm for *Weak Gathering* in Dynamic Unicyclic Graphs

In light of the impossibilities of Section 3, we hereafter consider generalized 1-interval connected unicyclic graphs, and we provide a deterministic algorithm that solves *weak gathering* for all non-symmetric configurations. We assume that the agents have knowledge of n, knowledge of the number of agents k, and the system has *cross detection*. Finally, our algorithm solves *weak gathering* for both cases, with and without *homebases*, provided that the configuration is not symmetric (as defined in Definition 3.2). If the configuration is symmetric, then the agents agree on unsolvability and terminate. A very significant aspect of mobile agent systems is the memory requirements of the agents. In order to achieve symmetry breaking by exploiting the topological asymmetries of the graph itself and the port labeling, our algorithm constructs a map of the graph in the local memory of each agent. Therefore, we assume that the agents have non-constant memory.

4.1. Weak Gathering *Algorithm*

Our deterministic algorithm is divided into two phases, and the overall idea is the following: During the first phase all agents explore the graph using a DFS approach, try to identify the nodes that belong to the cycle, and at the same time independently build a map of the graph. The latter is necessary in order to break the symmetry on the cycle and agree on a unique target node.

The problem of graph mapping becomes impossible if neither of n or k are known and without *whiteboards* on the nodes [17]. Most of the algorithms rely on either the usage of *whiteboards* [18, 19], or assume that the agents can observe the memory

contents of each other when they meet at the same node [20]. In the latter approach, the agents maintain multiple hypotheses when ambiguity about the graph topology occurs, and they resolve that ambiguity when they meet. Interestingly, in the special case of unicyclic graphs we show that knowledge of n alone (i.e., knowledge of k and *whiteboards* are not necessary) can lead to the construction of graph maps that are consistent (i.e., the same) among all agents.

When all agents have completed the first phase, the executed process (second phase) ensures that they will eventually gather on the cycle. Note that the agents may move to the second phase asynchronously. In this case, gathering might be unsuccessful; the agents recognize it and they start the second phase again. In order to make the description of the algorithm more clear, we first introduce a number of variables that are stored in the local memory of each agent.

- *rounds*: A counter that is used in order to count the number of rounds in several cases of the second phase. It is increased by one in each round.
- *inPort*, *outPort*: Port labels that the agent enters and leaves a node, respectively.
- *Graph (or $\mathcal{G}$)*: Contains lists that represent the nodes visited by an agent. A specific node of the underlying graph might correspond to multiple nodes in $\mathcal{G}$. We refer to the *Graph* of an agent α as $\mathcal{G}_\alpha$.
- *currendNode*: A pointer to the current node in $\mathcal{G}_\alpha$.
- *depth*: The distance between the agent and its initial position in $\mathcal{G}_\alpha$.
- *roundsBlocked*: The number of consecutive rounds that the agent remains blocked.
- *numAgents*: The number of agents in the current node of the agent. The considered model allows the agents to count the number of agents in their current node.
- *numAgentsPrev*: The number of agents in the node of the agent during the previous round (at the end of each round, the value of *numAgents* is copied to *numAgents*).
- *numAgentsTemp*: A temporary variable that is used to store the number of agents in several cases of the second phase.
- *orientation*: This variable is used during the second phase of the algorithm and indicates the direction in which the agent traverses the cycle (i.e., clockwise or counter-clockwise).
- *orientationTemp*: A temporary variable that is used to store the orientation.
- *crossed*: A bit which becomes one when the agent crosses some other(s) agent(s). At the end of each round it is reset to zero.

Phase 1. This phase is responsible for traversing the graph (exploration) and identifying the nodes that form the cycle. We now present all procedures that take place during this phase.

Graph exploration. Each agent α stores in its local memory (in $\mathcal{G}$) a list of the

neighbors of each vertex visited and the port numbers that led to those nodes. Let u be the initial node of an agent α. Then, α constructs a list $L(u)$ which represents u. Assume that it traverses an edge through port number i and arrives at a node w at port number j. It then constructs a new list $L(w)$, and in $L(u)$ it stores a tuple which consists of the port number i and a pointer to $L(w)$. At the same time, it stores in $L(w)$ a tuple with the port number j and a pointer to $L(u)$. If α is in a node v of $\mathcal{G}$ and moves through a port that is contained in a tuple of $L(v)$, it does not update $\mathcal{G}$. Finally, for each node visited (or for each list of $\mathcal{G}$), it also stores its degree. We call these lists the *Graph* of α or $\mathcal{G}_\alpha$. The initial node of each agent in $\mathcal{G}_\alpha$ (i.e., the first node that was added in $\mathcal{G}_\alpha$) is marked with a special character $\mathcal{I}$. In each round, the agent α stores in *currentNode* a pointer to the node (or list) of $\mathcal{G}_\alpha$ that it is at. In addition, in each round it calculates the (shortest) distance in $\mathcal{G}_\alpha$ between its current node and its initial node and stores it to its *depth* variable.

We use a traditional technique which makes each agent traverse a tree in a DFS way. In a round, when the agent arrives at node u through a port i, it leaves u through port $(i+1) \bmod \delta(u)$ in the next round (if the edge is available). Initially, each agent starts by leaving the port 0, and when $depth = n$, the agent moves through the port that it arrived from. The exploration ends when the agent has explored all paths of length n from its initial position. This can be achieved by checking if there is a node in $\mathcal{G}_\alpha$ in distance less than n that have at least one unexplored neighbor. In particular, if the number of tuples stored in a node list is less than its degree, then the exploration is not complete. At any point, if the edge that an agent tries to traverse is missing, it waits until it becomes enabled. As we show later, all agents either make progress towards the exploration of the graph and eventually complete the first phase of the algorithm, or if the scheduler blocks an agent indefinitely, our algorithm guarantees that the rest of the agents will reach some endpoint of the missing edge, and terminate. This means that if some agent(s) fail to explore the graph, all agents will still solve *weak gathering*.

Cycle detection. During this step, the agents check a number of predicates that help them to detect the nodes that belong to the cycle. In particular, whenever an agent α reaches a node u with degree $\delta(u) = 1$, it marks the corresponding node in $\mathcal{G}_\alpha$ with a special character $\not\oslash$, indicating that it does not belong to the cycle, and never moves to that node (of $\mathcal{G}_\alpha$) again. In addition, if a node $u \in \mathcal{G}_\alpha$ with degree $\delta(u)$ has $\delta(u) - 1$ marked neighbors in $\mathcal{G}_\alpha$, the agent also marks u.

In addition, each agent marks in $\mathcal{G}_\alpha$ its initial node with a different special character $\mathcal{T}$. Let u be the $\mathcal{T}$-marked node in $\mathcal{G}_\alpha$. If at any point the agent α marks u in $\mathcal{G}_\alpha$ with $\not\oslash$, it moves the $\mathcal{T}$ mark to the unique neighbor of u that is not marked with $\not\oslash$ in $\mathcal{G}_\alpha$.

As we show in Lemma 4.1, this procedure guarantees that by the end of the *graph exploration* step, the $\mathcal{T}$ marks of all agents will correspond to nodes of the cycle. Each agent moves on that node by following the shortest path of $\mathcal{G}$, and performs the following computations.

Let u_0 be a node that an agent α is located after the end of the *graph exploration* procedure. Let B_{u_0} be the branch that starts from node u_0. The agent α needs to resolve the ambiguity that occurs in $\mathcal{G}_\alpha$ in order to identify the nodes that belong to the cycle. At this point, α arbitrarily chooses one of the two directions of the cycle (e.g., the one with the lowest port number p_{u_0}), and deletes all nodes of $\mathcal{G}_\alpha$ on the opposite direction. To distinguish B_{u_0} from the nodes of the cycle, observe that at that point all nodes of B_{u_0} are $\not\mathcal{C}$-marked in $\mathcal{G}$, while the neighbors of u on the cycle are not marked in $\mathcal{G}$. This holds even if the initial position of an agent α is on the cycle. Then, because of the fact that the agent explored all paths up to distance n from its initial node, it means that it has traversed at least once the whole cycle. In addition, as we show in Lemma 4.2, during the first cycle traversal, all branches have been explored and marked with $\not\mathcal{C}$. Then, observe that $\mathcal{G}$ is a tree that has a line path $L = (u_0, u_1, u_2, \ldots, u_c, u_0, u_1, u_2 \ldots)$ that all nodes are unmarked. Then, starting from the first node of L, it counts and keeps all nodes of the corresponding branches, until the total number of nodes becomes n. It removes the rest of the nodes, and constructs the cycle by setting the corresponding port of the last node of L that it kept to lead to the first node of L (i.e., u_0), and vice versa.

$\mathcal{G}_\alpha$ is now a correct map of the graph that can be used to break the symmetry on the cycle. Note that α can now traverse the cycle both clockwise and counter-clockwise, though, the orientation between two agents might be different. In particular, each agent has a private orientation o_i, $0 \leq i \leq k$, where *clockwise* is initially the orientation defined by traversing the nodes in the order $u_0, u_1, \ldots, u_c$ of L. We say that there is *chirality* if there are no agents α_i and α_j such that $o_i \neq o_j$, $i \neq j$. We later explain how to obtain chirality in our model.

An overview of the steps of the first phase of the algorithm is the following.

(1) Initialization of variables.
(2) Add the initial node in the local graph map $\mathcal{G}$, and mark it with $\mathcal{T}$.
(3) Explore the graph up to distance n, and construct the map of the graph. Initially, leave through port 0.
(4) Mark all nodes with $\not\mathcal{C}$ in $\mathcal{G}$ that have exactly one unmarked neighbor (i.e., initially the leaves). When you mark the node where the $\mathcal{T}$ mark is, move $\mathcal{T}$ on the unique unmarked neighbor.
(5) Upon exploring all paths of length n from the initial position, move on the node which is marked with $\mathcal{T}$, and construct the cycle C.
(6) If the map $\mathcal{G}$ is symmetric, terminate. Otherwise, elect a leader and move to the *Second phase*.

Finally, the pseudocode of the first phase can be found in Algorithm 1.

Phase 2. When an agent α enters this phase, it means that is has constructed a correct map $\mathcal{G}_\alpha$ of the graph in its local memory. Then, a unique node can be elected as a leader, as described in Lemma 3.1. If the configuration is symmetric, then the agents recognize it and terminate. In any other case a unique node will be elected and will be the same for all agents. Let that leader be a node ℓ. At this

Algorithm 1 First phase of *weak gathering* algorithm

Result: Identifies the nodes that form the cycle.
state ← Phase 1, *statePrev* ← ∅
rounds, *depth*, *roundsBlocked*, *outPort* ← 0
$\mathcal{G} \leftarrow \emptyset$ # create a new empty list
procedure FirstPhase
  $L \leftarrow \emptyset$
  mark(L, $\mathcal{T}$)
  mark(L, $\mathcal{I}$)
  append($\mathcal{G}$, (L, *degree*)) # add L in $\mathcal{G}$.
  currentNode ← $\mathcal{G}(L)$ # the current node is a pointer to the L list of $\mathcal{G}$
  while $\exists u \in \mathcal{G}$: distance($u$,$\mathcal{I}$) $< n$ and $|L(u)| <$ degree(u) AND *state* $\neq$ terminate **do**
    if *state* = terminating **then**
      TerminationCondition() # See Algorithm 2
      Continue to next iteration of the loop (next round).
    if *degree* = 1 OR markedNeighbors($\mathcal{G}$, *currentNode*, $\not\mathcal{O}$) = *degree* - 1 **then**
      mark(*currentNode*, $\not\mathcal{O}$)
      if *currentNode* is marked with $\mathcal{T}$ **then**
        unmark(*currentNode*, $\mathcal{T}$)
        neighbor ← unmarkedNeighbor(*currentNode*, $\not\mathcal{O}$)
        mark(*neighbor*, $\mathcal{T}$)
    nextNode ← getNode(*currentNode*, *outPort*)
    if nextNode = ∅ **then** *depth* ← *depth* + 1
      if *depth* = $n + 1$ **then**
        inPort ← *outPort*
      else
        Move(*outPort*) # See Algorithm 3
        $L \leftarrow$ [(*inPort*, pointer(*currentNode*))]
        append(*currentNode*, (*outPort*, L))
        append($\mathcal{G}$, (L, *degree*))
        currentNode ← $\mathcal{G}(L)$
    else
      depth ← distance($\mathcal{G}$, *currentNode*)
      if *nextNode* is marked with $\not\mathcal{O}$ OR *depth* = $n + 1$ **then**
        inPort ← *outPort*
      else
        Move(*outPort*)
        currentNode ← *nextNode*
    if *numAgents* = k **then**
      state = *terminate*
    outPort ← (*inPort* + 1) mod *degree*
  if *state* $\neq$ terminate **then**
    state ← Phase 2
    cycle ← detectCycle($\mathcal{G}$) # As described in Phase 1 of Section 4.1
    leader ← electLeader($\mathcal{G}$, *cycle*) # Algorithm of Lemma 3.1

Algorithm 2 Termination condition of *weak gathering* algorithm

Result: Achieves *weak gathering* in case that the agent is blocked long enough for the rest of the agents to reach some endpoint of the missing edge.

procedure TERMINATIONCONDITION
  if *state* $\neq$ terminating AND *roundsBlocked* $\geq 4n + k$ **then**
    statePrev $\leftarrow$ state
    state $\leftarrow$ terminating
  else if *state* = terminating AND numAgents $\neq$ numAgentsPrev **then**
    state $\leftarrow$ *statePrev*
    roundsBlocked $\leftarrow 0$
  else if *roundsBlocked* $\geq 8n + 2k$ **then**
    state $\leftarrow$ terminate

Algorithm 3 Move step of an agent

procedure MOVE(*outPort*)
  while edge through port *outPort* is disabled **do**
    roundsBlocked $\leftarrow$ *roundsBlocked* $+1$
    TerminationCondition() # See Algorithm 2
  roundsBlocked $\leftarrow 0$
  Move through port *outPort*

point, they can also obtain *chirality* by utilizing the port numbers of ℓ. Let p_1 and p_2 be the ports that lead to its neighboring nodes in the cycle. Assume, without loss of generality, that $p_1 < p_2$. Then, α sets as *clockwise* the orientation that is defined by traversing p_1 and *counter-clockwise* the one defined by traversing p_2.

In this phase, an agent can either be in state *walking* or *gathering*, and initially it is in state *gathering*. Each agent α in this phase assumes that all agents have entered the second phase, and it performs some actions that would solve weak gathering in case that this assumption is true. Otherwise (i.e., there exists some other agent that has not entered the second phase), *weak gathering* will (temporarily) fail, and updates its state to *walking*. In *grouping* we explain how the agents form groups when certain predicates are satisfied. We call a set of agents a *group* if they are on the same node and move in the same direction.

Walking state. An agent in this state traverses the cycle counter-clockwise, and after $|C|$ rounds, where C is the cycle, it changes its state to *gathering*. To achieve this, when it enters to this state, it resets the value of its *rounds* variable to zero and in each round it increases its value by one.

Gathering state. The agents in this state perform the following actions, and if they fail to solve *weak gathering* they change their state to *walking*. We divide this process

into two steps. During the *first step*, each agent initially resets its *rounds* variable to zero, and moves for $2|C|$ rounds towards the elected node ℓ, by following the shortest path. The orientation that is followed is stored in both *orientation* and *orientationTemp* variables. After $2|C|$ rounds, the agents enter the *second step*. We distinguish the following cases for an agent α, depending on whether α reached ℓ, or not, by the end of the *first step*. If α arrived at node ℓ, it checks whether all agents are there (i.e., $numAgents = k$). If yes, it terminates. Otherwise, it resets *rounds* to zero, sets *orientation* and *orientationTemp* to *clockwise*, and starts moving clockwise on the cycle for $|C|$ rounds. The agents that due to missing edges did not reach the elected node ℓ, reset *rounds* to zero, set *orientation* and *orientationTemp* to *counter-clockwise* and start moving counter-clockwise for $|C|$ rounds. As we show in Lemma 4.5, by the end of round $2|C|$ all agents that entered state *gathering* during a time window of length $|C|$ are divided into at most two groups.

After the end of the *first step*, we want the agents of each group to start the *second step* at the same time (the two different groups may start at different rounds). However, observe that the agents might not enter into state *gathering* at the same time, thus start the *second step* asynchronously. In *grouping*, we explain how the agents start walking on the cycle as groups during the *second step*. At this point, there are two groups of agents moving towards each other. In any case, the two groups of agents will either end up on the same node, or they will cross each other, or they will become blocked on the endpoints of the same edge. In *grouping*, we explain how these groups of agents merge after at most $|C|$ rounds, or terminate in neighboring nodes. In the first two cases, if there exists some agent that has not entered the second phase, *weak gathering* will temporarily fail and they all update their states to *walking*. In the later case, we show that all agents will be gathered at the endpoints of the same edge.

Grouping. This subroutine of the algorithm forms groups of agents in the following cases.

(1) *First predicate.* During the *first step* of state *gathering*, the agents reset their *rounds* variable to zero and move towards the elected node for $2|C|$ rounds. However, not all agents start this step at the same time. The first predicate of *grouping* is responsible to synchronize the agents so as to begin the *second step* at the same time, and then continue moving as groups. In particular, when an agent α enters the *second step*, it resets *rounds* to zero and starts moving either clockwise or counter-clockwise depending on whether it reached the elected node or not. Let u be the node where α was at the end of the *first step*. Then, for the next $|C|$ rounds it tries to move to the neighboring node (and wait there). The rest of the agents in u detect that the number of agents in their current node was decreased; this is achieved by calculating the difference between *numAgents* and *numAgentsPrev* in each round. They enter into the *second step* and they try to move towards α. If they successfully reach α, they continue moving as a group until $rounds = n$. Otherwise, the *second*

step is completed (after $|C|$ rounds), therefore they enter into state *walking*.

(2) *Second predicate.* If an agent in state *walking* visits the elected node ℓ and there are some other agents there, it assumes that they are in state *gathering*. In this case, it enters into state *gathering*, resets *rounds* to zero, and waits there at most $2|C|$ rounds, or until the first predicate of *grouping* is satisfied. In other words, the elected node absorbs the agents passing by it.

(3) *Third predicate.* When two agents or groups of agents cross each other or visit the same node, they merge into a single group. To achieve this, when they cross each other, the agents of the group which is closer to the elected node ℓ by following the clockwise path (say G_1), reverse direction and update the value of their *orientation* variable. The agents of the other group G_2 wait until G_1 catches them. The distance to ℓ can be easily calculated using the graph map which is stored in the local memory of each agent. Therefore, each agent knows the distance to ℓ from the nodes of both G_1 and G_2. If the edge between the two groups is missing, they wait until it becomes available again, or until the *termination condition* is satisfied. After a successful merging, the agents that are in state *gathering* continue walking the cycle in their initial direction defined by *orientationTemp*, while the agents in state *walking* reverse their initial direction (i.e., they change the value of their *orientationTemp* variable and set *orientation* = *orientationTemp*). Similarly, if the groups of agents visit the same node (i.e., they do not cross each other), the agents in state *walking* reverse direction, while the group of agents in state *gathering* does not do anything (i.e., they continue walking in their initial direction). Finally, after a successful edge traversal of G_1, if G_2 is missing, it reverses direction again, otherwise the agents in state *walking* enter into the *second step* of *gathering*.

Termination condition. The overall idea is that if an agent is blocked long enough for the rest of the agents to reach some endpoint of the missing edge, then weak gathering is achieved and the agents terminate. To achieve this, in each round, if an agent α is blocked at a node u, it increases *roundsBlocked* by one and waits there until either the edge becomes available again (in which case it resets *roundsBlocked* to zero), or until the termination condition is satisfied. In particular, if $roundsBlocked_\alpha \geq 4n + k$, it enters into state *terminating*. In this state, if during the next $4n + k$ rounds the number of agents remains the same on u, it terminates. Otherwise it moves back to its previous state and resets *roundsBlocked* to zero. Finally, at any time during the execution of the algorithm, if the number of agents on a node is k, they all terminate.

An overview of the steps of the second phase of the algorithm can be found in Algorithm 5.

4.2. *Analysis*

We first show that after the end of the first phase of the algorithm, all agents correctly identify the nodes that form the cycle C, and then they only move on

Algorithm 4 Grouping subroutine of *weak gathering* algorithm

Result: Group formation of agents.

Each agent α performs the following during the second phase of the algorithm.

(1) First predicate:

(a) At the end of the *first step* of state *gathering*, store to *numAgentsTemp* the value of numAgents. After the first edge traversal during the *second step* of state *gathering*, wait until *numAgents* = *numAgentsTemp*.

(b) If α is in the *first step* of *gathering* and has either reached the elected node, or it is blocked, if *numAgents* < *numAgentsPrev*, enter into the *second step* of *gathering*.

(2) Second predicate:
If α is in state *walking*, its current node is the elected node ℓ, and the number of agents on ℓ are more than one, enter into state *gathering*.

(3) Third predicate:
During the *walking* state and during the second step of *gathering*, in each round that this predicate is not satisfied, store to *numAgentsTemp* the value of *numAgents*. If α crossed some agent(s), go to (3a). If *numAgents* > *numAgentsPrev*, go to (3b):

(a) Calculate the distance between the current node and the elected node ℓ in the clockwise path of the cycle. If α is closer to ℓ than the agents that were crossed, reverse direction (i.e., change the value of *orientation*), and after a successful edge traversal (merging) go to (3b). Otherwise wait until the number of agents in the current node is increased (i.e., *numAgents* > *numAgentsPrev*) and then go to (3b).

(b) If α is in state *waking*, reverse the initial direction (i.e., the direction before the crossing/merging which is stored in *orientationTemp*), enter into the *second step* of *gathering*, and go to (3c). If α is in state *gathering*, move towards the initial direction as defined in *orientationTemp*.

(c) After a successful edge traversal, if the number of agents remains the same as before the crossing/merging (i.e., *numAgents* = *numAgentsTemp*), go back to the previous state.

C. In addition, because of the fact that an agent can be blocked on a node of C indefinitely, we show that during the first phase all agents reach some endpoint of the missing edge after at most $2n$ rounds. We then continue and show that in the second phase of the algorithm all agents eventually enter into state *gathering* and they correctly solve *weak gathering*.

Algorithm 5 Second phase of *weak gathering* algorithm

Result: Solves *weak gathering* on some node of the cycle.

(1) State *walking*:
   (a) Reset *rounds* to zero and move counter-clockwise.
   (b) When *rounds* $= |C|$, change to state *gathering*.

(2) State *gathering*:
   (a) *First step.* Reset *rounds* to zero and follow the shortest path towards the elected node ℓ until *rounds* $= 2|C|$.
   (b) *Second step.* If reached ℓ, reset *rounds* to zero and move clockwise for $|C|$ rounds. Otherwise, move counter-clockwise for $|C|$ rounds.

(3) *Grouping* and *termination*:
   (a) In each round, depending on the state and step of the agent, check the appropriate predicates of *grouping* and perform the corresponding actions.
   (b) If at any time *numAgents* $= k$, terminate.
   (c) If *roundsBlocked* $= 4n + k$, change to state *terminating*.
   (d) If during the next $4n+k$ rounds the number of agents in the current node remains the same (i.e., *numAgents* = *numAgentsPrev* in each round), terminate. Otherwise, move back to the previous state.

4.2.1. *First phase of the algorithm*

Lemma 4.1 *Let $d_t(\mathcal{T}_\alpha, \mathcal{C})$ denote the (shortest) distance between the $\mathcal{T}$ mark of an agent α and the closest node u of the cycle C at round t. Then, $\{d_t(\mathcal{T}_\alpha, \mathcal{C})\}$, $t \geq 0$ is a decreasing sequence (i.e., $d_t \geq d_{t+1}$), and the $\mathcal{T}$ mark will eventually correspond to u.*

Proof. Initially, the agents are arbitrarily placed on some nodes of the graph. During the first phase, each agent α constructs in its local memory the graph $\mathcal{G}_\alpha$, and marks with $\mathcal{T}$ its initial node (in $\mathcal{G}_\alpha$). We refer to the $\mathcal{T}$ mark of an agent α as $\mathcal{T}_\alpha$. Then, it starts the exploration of the graph in a DFS way, and up to a maximum depth which depends on the size of the graph (*depth* $= n$). Call C the unique cycle and B_u the branch rooted on $u \in C$ where an agent α is initially placed. In order to mark a node $w \in \mathcal{G}_\alpha$ with $\not{C}$, all its neighbors except one must already be marked in $\mathcal{G}_\alpha$. This can only happen initially on the leaf nodes, then their neighbors, and so on. Now observe that all nodes of the cycle (including u) have two neighbors that belong to the cycle C, thus, α cannot mark any of them. This means that all nodes in the shortest path between the current position of the $\mathcal{T}_\alpha$ mark and u are not marked in $\mathcal{G}_\alpha$, while the rest of the nodes $v \in B_u$ will eventually be marked with $\not{C}$. When α marks with $\not{C}$ the node that its $\mathcal{T}_\alpha$ mark is, it removes it, and marks the unique neighbor that is not marked with $\not{C}$. Similarly, the above argument will be

satisfied for the new position of $\mathcal{T}_\alpha$. Because of this, $\mathcal{T}_\alpha$ can only move closer to the cycle every time the corresponding agent moves it. The exploration of all paths of length n from its initial position guarantees that α will visit all nodes of the branch B_u, thus, mark with $\not\mathcal{C}$ all of its nodes, except u. Consequently, the $\mathcal{T}$ mark will correspond u.

In contrast to the literature on exploration of graphs and graph map construction, in our model the agents cannot assign distinct labels on the nodes, thus recognize them when encountered again (cf., e.g., [21]). For this reason, when an agent enters the cycle and completes a tour, the whole graph is again considered as unexplored. However, in the special case of unicyclic graphs, the problem of map construction becomes solvable and our algorithm guarantees that after $O(n^2 + nk)$ rounds all agents construct a correct map of the graph.

Lemma 4.2 (Cycle detection) Cycle detection *correctly identifies the nodes that form the cycle, and $\mathcal{G}_\alpha$ of each agent α is a correct map of the graph.*

Proof. By Lemma 4.1 the $\mathcal{T}$ marks of all agents will eventually correspond to nodes of the cycle. Let α be an agent that after the exploration process is on a node $u_1 \in C$.

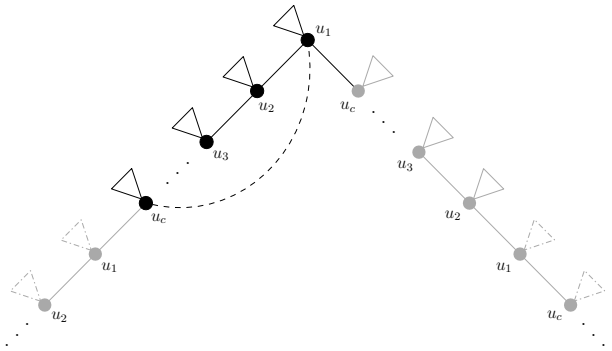


Fig. 5: Locally constructed graph $\mathcal{G}$ by the end of the exploration process

Observe that $\mathcal{G}_\alpha$ is a tree (rooted at u_1), which has two line paths that the nodes are not marked with $\not\mathcal{C}$, that correspond to the clockwise and counter-clockwise directions of the cycle. Figure 5 represents the locally constructed graph $\mathcal{G}_\alpha$. Then, α deletes one of the two paths and their corresponding branches (e.g, the one with the lowest port number of u_1). Observe that the distance between u_1 and all nodes of the branches until node u_c, where c is the size of the cycle, is less than n. This means that α has explored them and marked all nodes of the branches with $\not\mathcal{C}$. Then, by counting the nodes of the cycle and their branches, it can construct a correct map of the graph.

Lemma 4.3 *The number of rounds until all agents complete phase* 1 *is bounded by* $O(n^2 + nk)$.

Proof. Call C the unique cycle and B_u the branch rooted on node $u \in C$, where an agent α is initially placed. The number of rounds until an agent explores all paths of length n depends on the topology of the graph. In particular, when an agent enters the cycle and completes a tour, the whole graph is again seen as unexplored, thus, the agent continues exploring nodes that it has already visited in previous rounds. The number of complete tours of the cycle that can occur are $n/|C|$, and $n - |C|$ nodes are visited during each tour, provided that the depth that the agent has reached is less than n. The number of rounds R needed for the DFS exploration is then:

$$R = 4 \sum_{i=0}^{n/|C|} (n - i|C|) = \frac{4n(n + |C|)}{2|C|} = O(n^2) \tag{1}$$

The worst case is when the cycle has size 3, while the best case is when the cycle has size n. Due to the 1-interval connectivity, the scheduler can block α when it wants to traverse an edge of the cycle. During the DFS exploration, the number of edge traversals on the cycle is $2|C|$ for every complete tour of it. Now observe that if α is blocked for more than $6n + 2k$ rounds, it terminates, and as we show later, *weak gathering* is achieved. This means, that the worst case which does not lead to gathering, the scheduler blocks the agents for $8n + 2k - 1$ rounds for each edge traversal in C. In addition, for each cycle tour, $n - |C|$ nodes (in the worst case) can be explored without being blocked by the scheduler. Therefore, the total number of rounds that an agent can remain blocked during the first phase, considering the worst case choices of the scheduler, can be bounded by:

$$S = 2|C|\frac{n}{|C|}(8n + 2k - 1) = O(n^2 + nk) \tag{2}$$

The total number of rounds until all agents complete the first phase of the algorithm is then $O(n^2 + nk)$.

Observation 4.1 *Each agent in the first phase of the algorithm visits all nodes of the cycle every* $O(n)$ *rounds (at most* $2n$ *rounds), if not blocked by the scheduler.*

The above observation holds because the number of nodes that are explored during each cycle tour is $n - |C|$ nodes on the branches and $|C|$ nodes on the cycle. The DFS exploration then guarantees that the number of rounds needed are $2(n - |C| + |C|) = 2n$.

4.2.2. *Second phase of the algorithm*

We now show that phase 2 of the algorithm successfully gathers all agents either at the same node, or at the endpoints of the same edge.

Lemma 4.4 *Let the variable roundsBlocked$_\alpha$ of an agent α be $4n + k$. Then, all agents are gathered on the endpoints of the missing edge and terminate.*

Proof. Let α be an agent that is blocked on some node u of the cycle. By Observation 4.1 and because of the fact that the scheduler can only remove at most one edge in each round, the rest of the agents in phase 1 perform a block-free execution, thus, after at most $2n$ rounds, they traverse the cycle and they reach u. An agent in phase 2 of the algorithm can either be in state *walking* or *gathering*. In the first case, after at most $4|C|$ rounds, it reaches u. This is because it either reaches u after at most $|C|$ rounds, or it enters into state *gathering* after at most $|C|$ rounds (second predicate of *grouping*) and remains at the elected node for $2|C|$ rounds. Then, after $|C|$ more rounds it reaches u. In the second case, an agent needs $2|C|$ rounds to move towards the elected node and then it walks the cycle (either clockwise or counter-clockwise) for $|C|$ more rounds. In all these cases observe that *grouping* can only delay the agents from reaching some endpoint of the missing edge for at most $k - 2$ rounds. This holds because they perform a block-free execution of the algorithm, and each merging of agent groups takes in the worst case one additional round. Therefore, after at most $4|C| + k - 2 \leq 4n + k$ rounds, all agents reach some endpoint of the missing edge.

After $4n + k$ rounds α enters into state *terminating*. In this state, it remains idle and observes the number of agents on its node. In case that the number of agents changes, as a result of the missing edge being enabled again, it moves back to its previous state and continues the execution of the algorithm. If after $4n + k$ more rounds the number remains the same, it terminates. Observe that when it terminates, it is guaranteed that all agents will be in state *terminating* and (weakly) gathered on the endpoints of the missing edge. This means that independently of the choices of the scheduler, after that round all agents will remain idle and will eventually terminate.

Lemma 4.5 *Consider a set of agents S moving towards a node u in cycle C, following the shortest path. After $|C|$ rounds the agents of S are in at most two nodes of C, and one of them is in u.*

Proof. Consider a set of agents $S_1 \in S$ moving clockwise and a set of agents $S_2 \in S$ moving counter-clockwise. Consider two agents $\alpha_1,\ \alpha_2 \in S_1$ moving towards u. Assume that in the shortest path to u, the distance between α_1 and u is d_1 and the distance between α_2 and u is d_2.

The number of successful edge traversals until they reach u is at most $|C|/2$. Assume that α_1 didn't reach u after $|C|$ rounds. This means that it was blocked for at least $|C|/2+1$ rounds. Since 1-interval connectivity in this setting allows only one edge to be missing in each round, α_2 can be blocked for at most $|C|/2 - 1$ rounds (when not in the same node with α_1). Thus, if $d_1 < d_2$, α_2 reaches α_1 by round $|C|$, and if $d_1 > d_2$, it reaches u by round $|C|$. Now consider an agent $\alpha_3 \in S_2$ moving towards u (different orientation from α_1 and α_2). Since α_1 was blocked for at least

$|C|/2 + 1$ rounds and the agents follow the shortest path to u (they cannot be blocked on the endpoints of the same edge), α_3 can be blocked for at most $|C|/2-1$ rounds. Thus it reaches u by round $|C|$.

Overall, if an agent α is blocked for more than $|C|/2+1$ rounds, then all agents that move in the same orientation towards α reach α by round $|C|$, while the rest of the agents reach u. Otherwise, all agents reach u by round $|C|$.

Theorem 4.1 *Our algorithm solves* weak gathering *in unicyclic graphs in* $O(n^2 + nk)$ *rounds.*

Proof. By Lemma 4.3, in $O(n^2 + nk)$ rounds all agents complete the first phase of the algorithm and by Lemma 4.2 they construct a correct map of the graph. By Lemma 3.1, if the configuration is not symmetric, the agents elect a unique node on the cycle as a leader. If the configuration is symmetric, it means that there are several candidate leaders, thus, by Lemma 3.2, there are agent configurations where *weak gathering* is unsolvable and the agents terminate. Let r' be the round that the last agent enters into the second phase of the algorithm. Let also $R = \{r_1, r_2, \ldots, r_k\}$ be the rounds that the k agents enter into state *gathering* for the first time after r' (i.e., $r_i \geq r'$, $1 \leq i \leq k$).

In the second phase of the algorithm the agents can either be in state *walking* or *gathering*. Consider a set of agents S_1 that are in state *gathering* and $|r_i - r_j| < |C|$, $\forall i, j \in S_1$, and a second set of agents S_2 contains the rest of them. At this point, by Lemma 4.5 all agents that enter phase 2 at the same time, after at most $|C|$ rounds are divided into at most two groups G_1 and G_2, and one of them (say G_1) is on the elected node u. After $|C|$ rounds all agents in S_1 are in state *gathering*, thus, after $2|C|$ rounds all agents of S_1 are divided into two groups. In addition, the first predicate of *grouping* subroutine guarantees that the agents of G_1 and G_2 will continue moving as groups during the second step of phase 2. We now consider two cases.

(1) All agents of S_1 reached u. The agents of S_2 are in state *walking*, and because of the second predicate of *grouping* some of them reach u and enter into state *gathering*, while the rest of them, again by Lemma 4.5, become a group that did not reach u due to missing edges. Observe that this group walks the cycle counterclockwise, while the agents of S_1 walk the cycle clockwise. At this point there are two groups of agents moving towards each other. Therefore, the third predicate of *grouping* guarantees that after at most $|C|$ rounds the two groups will either merge (in this case they terminate), or they will become blocked on the endpoints of the same edge until the *termination condition* will be satisfied.

(2) In this case, the agents of S_1 are divided into two groups at round $r_1 + 2|C|$. During the first $2|C|$ rounds, some of the agents of S_2 may reach u, thus enter into state *gathering* and continue moving as a group with G_1.

(a) If the agents of G_2 move clockwise towards u, then the rest of the agents of S_2 may cross the agents of G_2 or arrive on the same node. In both cases they will

merge into a single group (third predicate of *grouping*).

(b) If the agents of G_2 move counter-clockwise towards u, then the rest of the agents of S_2 end up on the same node with the agents of G_2. This is because the agents of G_2 remain blocked long enough that at round $r_1 + 2|C|$ they did not reach u. Then, all of the agents in the clockwise path from G_2 to u, after $2|C|$ rounds reach G_2 (by Lemma 4.5). In this case, the agents of S_2 reverse direction to clockwise (due to the third predicate of *grouping*), however G_2 will continue moving counter-clockwise. Then, they reverse to their initial direction (counter-clockwise), and in the next round, if the edge is not missing, they again reach G_2. This procedure continues until some agent in G_2 enters into the *second step* of phase 2. Then, they will cross each other and *grouping* guarantees that they will merge into a single group.

Finally, in both cases (a) and (b), all agents in state *walking* (S_2) are absorbed by the agents in state *gathering* (S_1). Then, the two groups of agents move towards each other and *grouping* guarantees that after $|C|$ rounds they achieve *weak gathering*.

In all these cases, all agents reach either the same node and the *termination condition* is satisfied, or they become blocked at the endpoints of the same missing edge where, by Lemma 4.4, they solve *weak gathering*.

5. Open Problems

In [8] the authors provide a mechanism which avoids agent crossing on the ring. In particular, each agent constructs an edge labeled bidirectional ring, such that the intersection of the labels assigned in the edges of the clockwise direction with the ones of the counter-clockwise direction is empty. Then, the agents move on the actual ring subject to the constraint that at round r they can traverse an edge only if the set of labels of that edge contains r. This guarantees that two agents moving in opposite directions will never cross each other on an edge of the actual ring. An immediate open problem is to examine whether that, or a similar, mechanism could be adapted and used in our algorithm. In our algorithm, *cross detection* is only required during the second phase. All agents after $O(n^2 + nk)$ rounds enter into that phase and elect the same node ℓ as leader, thus, obtain the same sense of orientation. By implementing a counter for the rounds, we could then allow the agents to move clockwise on the cycle only at odd rounds, and counter-clockwise only at even rounds. This guarantees that no two agents could traverse the same edge in opposite directions during the same round. Then, by slightly modifying the second phase of our algorithm (e.g., allow $4|C|$ rounds during the *first step* in state *gathering*, and $2|C|$ during the *second step*) the agents would again solve *weak gathering*, though a formal proof is left as an open problem.

Even though we almost completely characterized the class of 1-interval connected graphs in which gathering can be solved, there is a number of interesting directions emanating from our existing knowledge on the problem. An immediate open problem is whether we can achieve the same results if the class of dynamics is the T-interval

connectivity, for $T > 1$. Other dynamic models that can be considered are *periodic*, that is, each edge is periodically enabled/disabled, *recurrent* [22], meaning that an edge cannot remain disabled indefinitely, and other worst-case dynamic networks in which the topology may change arbitrarily from round to round subject to some constraints (cf., e.g., [23]). The problem of strict *gathering* becomes feasible in these cases, and the goal is to find efficient algorithms for the problem. For example, consider a ring graph and two groups of agents blocked on the endpoints of a missing edge. Then, our algorithm could eventually achieve strict *gathering* by just waiting for the disabled edge to become enabled, rather than terminate after $O(n)$ rounds. However, a more efficient algorithm could decide to change the orientation of the agents and meet on some other node of the cycle. This might be the case for many *strict gathering* algorithms for static graphs. By simulating any such algorithm, while the agents just wait for the missing edges to become enabled, it might be possible to solve *strict gathering* in all the solvable cases of static graphs. However, this technique may not be applied to algorithms that are based on synchronization of agents.

Other interesting related problems for the generalized 1-interval connectivity model are *partial gathering*, and *gathering with waste*. The *partial gathering* problem requires, for a given positive integer g, that each agent should move to a node and terminate so that at least g agents should meet at each of the nodes they terminate at. This is a generalization of the *strict gathering* problem, and for values of $g \leq k/2$ it enables feasibility in a larger class of graphs. It is not clear whether this requirement is weaker of stronger than that of *weak gathering*. For example, in ring graphs with k agents and $g = k/2$, the agents can terminate in any two nodes of the graph, provided that the number of agents in both nodes are $k/2$. However, observe that this problem enables feasibility in a larger class of graphs. Consider two cycles that are connected with a line. *Weak gathering* is unsolvable in this setting, while *partial gathering* might be possible. Consider the case that $k/2 + a$, $a < k/2$ agents are on a cycle C_1 and the rest of them are on the cycle C_2. Then, a agents from C_1 is possible to escape from the cycle and reach C_2, thus achieve *partial gathering*. *Partial gathering with waste* g is the problem of gathering at least g agents on some node (the rest of them being the waste). Similarly to *partial gathering*, at most one agent might remain trapped on a cycle in this dynamic model. Finally, a generalization of *weak gathering*, where the agents are allowed to gather in at most g nodes (*grouping*), might also enable feasibility in a larger class of (dynamic) graphs.

References

[1] J. Czyzowicz, A. Pelc and A. Labourel, How to meet asynchronously (almost) everywhere, *ACM Transactions on Algorithms (TALG)* **8**(4) (2012) 1–14.

[2] G. De Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc and U. Vaccaro, Asynchronous deterministic rendezvous in graphs., *Theor. Comput. Sci.* **355**(3) (2006) 315–326.

[3] A. Dessmark, P. Fraigniaud and A. Pelc, Deterministic rendezvous in graphs, in *European Symposium on Algorithms* (Springer, 2003) 184–195.
[4] M. Shibata, N. Kawata, Y. Sudo, F. Ooshita, H. Kakugawa and T. Masuzawa, Move-optimal partial gathering of mobile agents without identifiers or global knowledge in asynchronous unidirectional rings, *Theoretical Computer Science* **822** (2020) 92–109.
[5] J. Czyzowicz, S. Dobrev, E. Kranakis and D. Krizanc, The power of tokens: rendezvous and symmetry detection for two mobile agents in a ring, in *International Conference on Current Trends in Theory and Practice of Computer Science* (Springer, 2008) 234–246.
[6] L. Barriere, P. Flocchini, P. Fraigniaud and N. Santoro, Rendezvous and election of mobile agents: impact of sense of direction, *Theory of Computing Systems* **40**(2) (2007) 143–162.
[7] J. Chalopin, S. Das and N. Santoro, Rendezvous of mobile agents in unknown graphs with faulty links, in *International Symposium on Distributed Computing* (Springer, 2007) 108–122.
[8] G. A. Di Luna, P. Flocchini, L. Pagli, G. Prencipe, N. Santoro and G. Viglietta, Gathering in dynamic rings, *Theoretical Computer Science* **811** (2020) 79–98.
[9] F. Kuhn, N. Lynch and R. Oshman, Distributed computation in dynamic networks, in *Proceedings of the 42nd ACM symposium on Theory of computing (STOC)* (ACM, 2010) 513–522.
[10] G. A. Di Luna, S. Dobrev, P. Flocchini and N. Santoro, Live exploration of dynamic rings, in *36th International Conference on Distributed Computing Systems (ICDCS)* (IEEE, 2016) 570–579.
[11] S. Das, N. Giachoudis, F. L. Luccio and E. Markou, Broadcasting with mobile agents in dynamic networks, in *24th International Conference on Principles of Distributed Systems (OPODIS 2020)* (Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021)
[12] T. Gotoh, Y. Sudo, F. Ooshita, H. Kakugawa and T. Masuzawa, Group exploration of dynamic tori, in *38th International Conference on Distributed Computing Systems (ICDCS)* (IEEE, 2018) 775–785.
[13] T. Gotoh, P. Flocchini, T. Masuzawa and N. Santoro, Tight bounds on distributed exploration of temporal graphs, in *23rd International Conference on Principles of Distributed Systems (OPODIS 2019)* (Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020)
[14] E. Kranakis, D. Krizanc and E. Markou, The mobile agent rendezvous problem in the ring, *Synthesis Lectures on Distributed Computing Theory* **1**(1) (2010) 1–122.
[15] A. Pelc, Deterministic rendezvous in networks: A comprehensive survey, *Networks* **59**(3) (2012) 331–347.
[16] A. Dessmark, P. Fraigniaud, D. R. Kowalski and A. Pelc, Deterministic rendezvous in graphs, *Algorithmica* **46**(1) (2006) 69–96.
[17] S. Das, Mobile agents in distributed computing: Network exploration, *Bulletin of EATCS* **1**(109) (2013).
[18] S. Das, P. Flocchini, A. Nayak and N. Santoro, Distributed exploration of an unknown graph, in *International Colloquium on Structural Information and Communication Complexity* (Springer, 2005) 99–114.
[19] S. Das, P. Flocchini, S. Kutten, A. Nayak and N. Santoro, Map construction of unknown graphs by multiple agents, *Theoretical Computer Science* **385**(1-3) (2007) 34–48.
[20] C. Gong, S. Tully, G. Kantor and H. Choset, Multi-agent deterministic graph mapping via robot rendezvous, in *International Conference on Robotics and Automation* (IEEE, 2012) 1278–1283.

[21] P. Panaite and A. Pelc, Exploring unknown undirected graphs, in *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms* (Society for Industrial and Applied Mathematics, 1998) 316–322.

[22] A. Casteigts, P. Flocchini, W. Quattrociocchi and N. Santoro, Time-varying graphs and dynamic networks, *International Journal of Parallel, Emergent and Distributed Systems* **27**(5) (2012) 387–408.

[23] O. Michail, I. Chatzigiannakis and P. G. Spirakis, Causality, influence, and computation in possibly disconnected synchronous dynamic networks, *Journal of Parallel and Distributed Computing* **74**(1) (2014) 2016–2026.