DISTRIBUTED TREE REARRANGEMENTS FOR REACHABILITY AND ROBUST CONNECTIVITY*

MICHAEL SCHURESKO† AND JORGE CORTÉS‡

Abstract. This paper studies connectivity maintenance in robotic networks. We propose a distributed coordination algorithm that can be combined with the individual motion control strategies of the robotic agents to maintain the overall network connectivity. The coordination algorithm is based on the idea of maintaining the edges of an evolving spanning tree, termed the constraint tree, of the communication graph. The evolution of this tree is determined by the preferences that each agent possesses as to which other agents it would like to maintain connectivity with, the network configuration, and the allowed set of re-arrangement operations. We analyze the reachability and repair properties of the proposed algorithm. Regarding reachability, we show that the constraint tree can transition between any two trees which are subgraphs of the current communication graph. Regarding repair, we show that the algorithm is robust against link drops in the constraint tree and can repair an initially disconnected constraint tree. We provide simulations of the performance of the algorithm in deployment scenarios.

1. Introduction. Given a group of robots with processing, motion, and communication capabilities executing a motion coordination algorithm to achieve some task, we set out to address the following problem: how can we guarantee that the graph induced by the inter-agent communication remains connected?

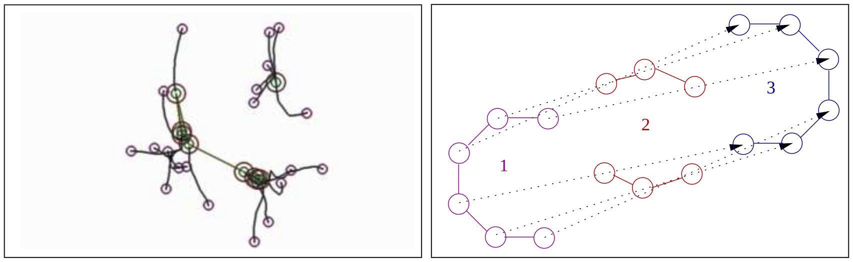
Surprisingly, many coordination algorithms for robotic networks fail to maintain connectivity under simple communication models. Tasks such as "deploy over a region of interest" or "explore an area" are examples where the control objective naturally conflicts with the goal of staying close enough to maintain global network connectivity. Even tasks which appear not to conflict with the concept of network cohesiveness can cause connectivity to fail. Consider a robotic network where any pair of robots within a certain distance can communicate. For such networks, it turns out that simple averaging algorithms that achieve flocking [28] and rendezvous [24] can easily fail to maintain connectivity, see Figure 1.1(a). Another example where connectivity maintenance is not guaranteed is the motion planning problem between two network configurations in physical space. A simple linear interpolation between the initial and final configurations does not guarantee that connectivity is preserved during the evolution, even if the network at the two configurations is connected, see Figure 1.1(b).

A strategy to solve the connectivity problem is to make custom modifications to each motion control algorithm to enforce inter-agent connectivity constraints. It is desirable, however, to synthesize a general methodology for maintaining connectivity that goes beyond a case by case study, and can be used in conjunction with any motion coordination algorithm. In this paper we take on this aim and propose an approach based on the preservation of a spanning tree of the underlying communication graph. The idea is to synthesize a distributed algorithm to agree upon "safe" re-arrangements of the constraint tree (i.e., re-arrangements that do not break con-

*Submitted to the SIAM Journal on Control and Optimization on March 2009, revised version on August 2010. This work was supported in part by NSF CAREER award ECS-0546871 and NSF award CMMI-0908508. Preliminary versions of this manuscript were presented as [31] at the 2007 IEEE Conference on Decision and Control, New Orleans, LA and as [33] at the 2009 International Conference on Hybrid Systems: Computation and Control, San Francisco, CA.

†Department of Applied Mathematics and Statistics, University of California, Santa Cruz, mds@soe.ucsc.edu

‡Department of Mechanical and Aerospace Engineering, University of California, San Diego, cortes@ucsd.edu



(a) (b)

Fig. 1.1. *Failure to maintain connectivity in rendezvous (a) and while transitioning between connected configurations (b). In (b), (1) denotes the initial positions, (2) denotes an intermediate position and (3) denotes the final configuration. Paths shown in black. In (a), the agents within each group that stays connected will correctly rendezvous to a point.*

nectivity or cause cycles) based on preferences specified by the motion coordination algorithm. Conversely, we show that our proposed strategy satisfies a distributed reachability property, whereby given a desired target spanning tree, a suitable preference function exists that drives the constraint tree to it. Because of the fundamental role that spanning trees play in graph theory and computer science, a distributed algorithm for agreeing on a spanning tree is of interest beyond the area of distributed control of robotic networks.

Literature review. The fundamental importance of spanning trees to distributed algorithms has motivated a vast collection of literature which explores their properties and designs algorithms to construct them, see e.g., [6, 22, 25]. A series of works [2, 12, 14, 16] improve upon the running time of distributed algorithmic solutions to the problem of finding a minimum spanning tree of a network. The work [26] gives a lower bound on this task, which is reasonably close to the running time of the solution proposed in [16]. The work [3] provides an algorithm for distributed repair and construction of a minimum spanning tree. Such algorithms are unsuitable for our purposes as we would like the spanning tree to dynamically change based on robot positions while robot motion is inducing links which are not part of the tree to disappear.

The same reasons that make spanning trees important for distributed computation are even more critical in ad hoc wireless networking, where link failures due to node motion are common. A relevant problem in this field is repairing a spanning tree after it has been broken, see [13] for a survey. An algorithm that uses partial orders on nodes to prevent cycles during simultaneous link creation is presented in [19] to compute an approximate minimum spanning tree. These algorithms handle dynamic link failures, but are not developed with the intent of dynamically producing a constraint tree to control motion.

In contrast with the papers mentioned above, the algorithm proposed here is explicitly designed to dynamically compute motion constraints for a mobile network. This is a common theme in the cooperative control and robotics literature dealing with robotic networks. Connectivity in this area appears in three contexts: (i) as an end in itself, i.e., computing network motion and configurations that optimize connectivity; (ii) connectivity-preserving solutions to specific problems in distributed control such as rendezvous, flocking, or formation control; (iii) general, broadly applicable mechanisms that can be combined with other motion planning algorithms and guarantee connectivity maintenance. With a few exceptions noted below, the references falling in (i) propose centralized solutions whereas those which fall in (ii) and (iii) are distributed.

Belonging to (i), various works have focused on designing the network motion so that some desired measure of connectivity (e.g., algebraic connectivity) is maximized under position constraints. This approach has the advantage of controlling how connected the communication graph is, rather than merely ensuring that a multi-hop path exists between all pairs of robots. [4, 10] consider convex constraints, while [20] deals with a class of nonconvex constraints. The work [35] builds a distributed estimator to control algebraic connectivity. Plenty of works belong to (ii), including research on rendezvous with limited-range communication [1, 9, 21] and with visibility constraints in nonconvex environments [15], coverage [27], motion planning [34], formation control [18], and flocking [36]. Finally, works that fall in (iii) include the distributed solution in [29] which gives connectivity maintaining constraints for second-order control systems with input magnitude bounds, the distributed connectivity service in [7] that mediates between individual agent goals and the overall requirement of maintaining connectivity, and the centralized approach in [37] allowing for a general range of agent motions. At the boundary between (i) and (iii), [38] uses potential fields to maximize algebraic connectivity while achieving some other task and our previous work [32] uses nonsmooth analysis tools to design strategies that maintain the algebraic connectivity of the network above a desired threshold. These works yield robust connectivity preservation algorithms at the cost of a substantial overhead in communication and computation.

The research cited above can also be classified according to the method adopted to study connectivity. A substantial number of papers have proceeded by identifying sets of links in the communication network whose preservation guarantees connectivity. Works along these lines can be further classified in terms of whether this set of edges remains fixed during the network operation or can dynamically evolve depending on the objectives of the robots. When connectivity edges remain fixed, [1, 9, 21, 15, 29] achieve decentralized solutions to connectivity maintenance at the cost of highly constraining the motion of the network – essentially, agents can only move in a way that does not severe any link in an appropriate proximity graph. Once a link belongs to the proximity graph, it is preserved from that point on along the evolution of the network. Approaches that dynamically compute in a distributed way a special subgraph of the communication graph whose preservation guarantees connected motions, such as [27, 34, 7], do not suffer from this shortcoming. The algorithm in this paper belongs to this category – one with the minimal possible number of edges in the dynamically computed constraint graph.

Even though the feedback nature of many of the proposed algorithmic solutions endows them with some robustness properties, few works have formally characterized them. Exceptions include the robustness against agent velocity changes studied in [7], against link failures in [9, 34], and against delays in communication in [32].

The algorithmic solution proposed in this paper is distributed (requiring individual agents to interact with neighbors in the communication graph) and can be combined with other motion planning strategies to guarantee connectivity (and therefore falls into (iii)). The algorithm identifies a dynamically evolving constraint tree that must be preserved to guarantee connectivity. The explicit characterization of the algorithm robustness against link failures and the distributed reachability property with regards to the constraint tree allows for a broad applicability of our approach.

Statement of contributions. This paper introduces the Connectivity Maintenance Algorithm for dynamically agreeing upon a spanning tree of a proximity graph, that we term constraint tree. Maintaining each edge of the tree preserves the connectivity of the robotic network. The algorithm allows for on-line topological rearrangements of the tree in a distributed manner while at the same time guaranteeing that no cycles are formed. We assume that each agent has a set of (possibly changing) preferences as to which other agents it would like to attach to. These preferences can be specified a priori or can be determined by any motion coordination algorithm the network is executing.

The Connectivity Maintenance Algorithm has several useful properties. We show that, in combination with a motion coordination algorithm, the algorithm guarantees that connectivity is maintained. We also show that the allowable rearrangements are flexible enough to make it possible for the constraint tree to transition to any desired tree which remains a subgraph of the communication graph for a sufficient time. Moreover, the algorithm is robust to link failures and is able to repair the constraint tree if the underlying interaction graph remains connected for a sufficiently large period of time. In particular, the algorithm can successfully recover from an initial disconnected constraint tree. Connectivity Maintenance Algorithm is "on-line" in the sense that it provides the agents with a "current spanning tree" at every step of the algorithm. This approach differs in both style and end goals from the standard "build a spanning tree and design an algorithm to repair it" approach in the wireless networking community – and goes beyond various algorithms in the literature which constrain robot motion to maintain a (fixed) spanning tree.

An important additional feature of our algorithm is its modularity. To formalize this property, the paper also introduces the notion of *input-output control and communication law*, building on the modeling framework introduced in [23] to analyze the properties of motion coordination algorithms. Input-output control and communication laws allow designers to accommodate reusable algorithmic components which can be combined to form complete coordination algorithms. The Connectivity Maintenance Algorithm is an example of an input-output control and communication law, and can therefore be combined with motion planning algorithms whose primary objective is to achieve some desired coordination task. Our algorithm has task completion guide how preferences for the links in the constraint tree are established. This is done in a dynamic way as the network state evolves. In this sense, our approach typically results in a set of constraints that are flexible in regards to the coordination task that the motion algorithm intends to achieve. It should be noted, however, that the optimality of the combination of Connectivity Maintenance Algorithm with another coordination algorithm is not a foregone conclusion. The performance of the resulting algorithm depends on the choice of attachment preferences and can be subtle to establish, see for instance [31] for an analysis in the point-to-point reconfiguration case. Simulations [30] show that our proposed solution works qualitatively well when combined with flocking and rendezvous algorithms. Here, we combine the Connectivity Maintenance Algorithm with a deployment algorithm originally presented in [8] to achieve optimal coverage in a convex region.

Organization. Section 2 introduces some basic graph-theoretic notions and the model for the robotic network. Section 4 presents the Connectivity Maintenance Algorithm. Section 5 analyzes the algorithm correctness. Section 6 characterizes the algorithm repair properties against link failures and disconnection. Section 7 characterizes the reachability properties. Section 8 illustrates our algorithm through simulations. Finally, Section 9 presents our conclusions and ideas for future work.

Notation. Throughout the paper, $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, and $\mathbb{R}_{>0}$ denote the sets of reals, non-negative reals, and positive reals, respectively. For a set S, $\mathbb{F}(S)$ denotes the collection of all finite subsets of S. Given sets S_1, S_2, let $\mathfrak{F}(S_1; S_2)$ denote the set of functions from S_1 to S_2. Whenever we provide algorithm pseudo-code, we use $a \leftarrow b$ to mean "a is assigned a value of b." For $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$, we use $f(n) \in O(g(n))$ to mean that there exist $N_0 \in \mathbb{N}$, $c \in \mathbb{R}$ such that $f(n) < cg(n)$ for all $n > N_0$; we use $f(n) \in \Omega(g(n))$ to mean that there exist $N_0 \in \mathbb{N}$, $c \in \mathbb{R}$ such that $f(n) > cg(n)$ for all $n > N_0$. Finally, $f(n) \in \Theta(g(n))$ means $f(n) \in O(g(n)) \cap \Omega(g(n))$. We use $\wedge$ to mean "logical and" and $\vee$ to mean "logical or." For sets $X_1, \dots, X_m$, we use $\pi_{X_j} : X_1 \times \cdots \times X_m \mapsto X_j$ to denote the canonical projection onto X_j. We use the superindex $\cdot^{[i]}$ to refer to a variable that belongs to the ith agent. For instance, $x^{[i]}$ refers to the ith agent's value of the variable x.

2. Preliminary notions. Here, we review some notions related to graph theory and introduce a formal model for robotic networks and coordination algorithms.

2.1. Graph-theoretic notions. We follow here [6, 11]. An undirected graph, or simply *graph*, $G = (V, E)$, consists of a vertex set V and a set E of unordered pairs of vertices, called *edges*. A *directed graph* or digraph, is a graph having ordered pairs of vertices as edges. A graph (V', E') is a *subgraph* of a graph (V, E) if $V' \subset V$ and $E' \subset E$; additionally, a graph (V', E') is a *spanning* subgraph if it is a subgraph and $V' = V$. From this point on, for a graph on n nodes (i.e., $|V| = n$) we assume without loss of generality that $V = \mathbb{Z}_n = \{0, \dots, n-1\}$, thus allowing us to refer to node with unique identifier (UID) 0, etc. Given a graph, $G = (V, E)$, the set of neighbors of node $i \in V$ is $\mathcal{N}(i) = \{j \in V \mid (i, j) \in E\}$.

A *tree* is a connected graph with no cycles. In this paper, we only deal with directed rooted trees. In a directed rooted tree, each edge connects a *child* node i to its *parent* node $\mathfrak{p}_{\text{curr}}^{[i]}$. The unique node with no parents is called the *root*, and the distance in a tree from a node i to the root is called the *depth* of i, denoted $\text{dp}_T^{[i]}$. The *depth of the tree*, denoted $\text{depth}(T)$, is the maximum depth among all nodes. Nodes i and j are called *siblings* in a given tree if they have the same parent, $\mathfrak{p}_{\text{curr}}^{[i]} = \mathfrak{p}_{\text{curr}}^{[j]}$. We say i is a *descendant* of j, or equivalently j is an *ancestor* of i, if there exists a sequence of nodes, $k_1, \dots, k_n$ such that $\mathfrak{p}_{\text{curr}}^{[i]} = k_1, \mathfrak{p}_{\text{curr}}^{[k_1]} = k_2, \dots, \mathfrak{p}_{\text{curr}}^{[k_n]} = j$. We find it convenient to define an ordering $<_T$ on nodes given a tree T as follows: $i <_T j$ if j is a descendant of i or if a_i, a_j are descendants or equal to i and j respectively, $\mathfrak{p}_{\text{curr}}^{[a_i]} = \mathfrak{p}_{\text{curr}}^{[a_j]}$, and $a_i < a_j$. This is equivalent to saying that $i <_T j$ if i is reached after j in a depth-first traversal of T which explores neighbors of lesser UID first. Note that $<_T$ induces a total order on the nodes of T.

We use proximity graphs as an abstraction of network interconnection among spatially distributed agents. Proximity graphs [5, 17] associate network topology with robot positions by defining mappings from finite collections of points in $\mathbb{R}^d$ to graphs. For $\mathcal{P} \in \mathbb{F}(\mathbb{R}^d)$, let $\mathbb{G}(\mathcal{P})$ denote the set of undirected graphs whose vertex set is some labeling of the elements in $\mathcal{P}$. A *proximity graph* $\mathcal{G}$ associates to $\mathcal{P} \in \mathbb{F}(\mathbb{R}^d)$, $|\mathcal{P}| = n$, an undirected graph in $\mathbb{G}(\mathcal{P})$ with vertex set isomorphic to $\mathbb{Z}_n$ and edge set $\mathcal{E}_{\mathcal{G}}(\mathcal{P})$, where $\mathcal{E}_{\mathcal{G}} : \mathbb{F}(\mathbb{R}^d) \to \mathbb{F}(\mathbb{Z}_n \times \mathbb{Z}_n)$. Examples include the complete graph, the r-disk graph, and the visibility graph, see [5].

2.2. Robotic network model. Throughout the paper, we use the modeling framework introduced in [5, 23]. This formalism allows us to provide formal definitions of the proposed coordination algorithms that are ready to be combined with other cooperative strategies. We briefly describe the main notions next.

DEFINITION 2.1 (Robotic network). *A* uniform robotic network $\mathcal{S}$ *is a tuple* $(I, \mathcal{A}, E_{\mathrm{cmm}})$ *consisting of*

(i) $I = \mathbb{Z}_n$*; the* set of unique identifiers (UIDs)*;*
(ii) $\mathcal{A} = \{A^{[i]}\}_{i \in \mathbb{Z}_n}$*, with* $A^{[i]} = (X, U, X_0, f), i \in \mathbb{Z}_n$*, the* set of physical agents*; here* X *is the state-space of each agent,* $f : X \times U \to TX$ *is the map determining its controlled dynamics,* X_0 *is the set of allowable initial states, and* U *is the control space of each control system.*
(iii) E_{cmm}*, the* communication edge map*, is a map from* $\prod_{i \in \mathbb{Z}_n} X^{[i]}$ *to the subsets of* $I \times \mathbb{Z}_n \setminus \mathrm{diag}(I \times \mathbb{Z}_n)$*.*

Next we introduce the notion of input-output control and communication law. This notion, a generalization of the concept of *control and communications law* from [5], is aimed at facilitating composition of algorithmic components.

DEFINITION 2.2. *A* (synchronous, static, uniform, feedback) input-output control and communication law $\mathcal{CC}$ *for a uniform network* $\mathcal{S}$ *consists of the sets:*

(i) $\mathbb{T} = \{t_\ell\}_{\ell \in \mathbb{N}_0} \subset \mathbb{R}_{\geq 0}$*, a communication schedule;*
(ii) L*, a communication language;*
(iii) (W, W_{in}, W_{out})*, sets of values of* logic variables, input logic variables*, and* output logic variables*,* $i \in \mathbb{Z}_n$*, respectively;*
(iv) $W_0^{[i]} \subseteq W$*,* $i \in \mathbb{Z}_n$*, subsets of* allowable initial values*;*
(v) $W_{in,0}^{[i]} \subseteq W_{in}$*, subsets of* allowable initial input values*;*

and of the maps:

(i) $\mathrm{msg} : X \times W \times W_{in} \times \mathbb{Z}_n \to L$*, the* message-generation function*;*
(ii) $\mathrm{stf} : W \times W_{in} \times L^n \to W \times W_{out}$ *the* (input-output) state-transition function*;*
(iii) $\mathrm{ctl} : X \times X \times W \times W_{in} \to U$*, the* control function.

The interpretation of the elements of the input-output control and communication law is the following. Starting from an allowable initial state as specified by the law, at each communication round specified in $\mathbb{T}$, each agent sends messages in the language L to its neighbors according to msg. With the messages received, each agent updates the value of its logic variables using stf. In between communication rounds, the motion of each agent is governed by the control function ctl.

Without loss of generality, throughout the paper we consider $\mathbb{T} = \mathbb{Z}_{\geq 0}$, and unless otherwise specified, we take $L = W$. For notational convenience, we often write an input-output state-transition function stf as the pair $(\mathrm{stf_W}, \mathrm{stf_{out}})$, where $\mathrm{stf_W}$ computes values in W and $\mathrm{stf_{out}}$ in W_{out}. We refer to $\mathrm{stf_{out}}$ as the *output state transition function*. A *control and communication law* [5] corresponds to an input-output control and communication law with $W_{\mathrm{in}} = \emptyset = W_{\mathrm{out}}$.

REMARK 2.3. We note that the algorithms presented in this paper work equally well if the slight modification is made that the control function ctl, which defines the instantaneous motion of the robot in continuous time, is replaced with a *waypoint generation function*, defining the goal position that a given robot should be at during the next communication round. Such a function can be defined like $\mathrm{waypt} : X \times W \times W_{\mathrm{in}} \times L^n \to X$. In particular, this observation implies that we can consider arbitrary agent dynamics so long as the requirements imposed by the waypoint generation function can be satisfied by the dynamics. •

A composition of two input-output laws is the natural result of substituting a subset of each law's output for a subset of the other law's input. We detail this next.

DEFINITION 2.4 (Composition of input-output laws). *The composition of two* *input-output control and communication laws, $\mathcal{CC}_1$ and $\mathcal{CC}_2$, that satisfy*

$$\mathcal{CC}_1[W_{in}] = X \times Y, \quad \mathcal{CC}_1[W_{in0}] = X_0 \times Y_0, \quad \mathcal{CC}_1[W_{out}] = B \times C,$$
$$\mathcal{CC}_2[W_{in}] = A \times B, \quad \mathcal{CC}_2[W_{in0}] = A_0 \times B_0, \quad \mathcal{CC}_2[W_{out}] = Y \times Z,$$

for some sets X, Y, Z, A, B, C, with $A_0 \subset A$, $B_0 \subset B$, $X_0 \subset X$, and $Y_0 \subset Y$, is the input-output control and communications law, $\mathcal{CC}_1 \otimes \mathcal{CC}_2$, with sets

$$\begin{aligned}
(\mathcal{CC}_1 \otimes \mathcal{CC}_2)[L] &= \mathcal{CC}_1[L] \times \mathcal{CC}_2[L], & (\mathcal{CC}_1 \otimes \mathcal{CC}_2)[W_{out}] &= Z \times C,\\
(\mathcal{CC}_1 \otimes \mathcal{CC}_2)[W] &= \mathcal{CC}_1[W] \times \mathcal{CC}_2[W] \times Y \times B, & (\mathcal{CC}_1 \otimes \mathcal{CC}_2)[W_{in}] &= X \times A,\\
(\mathcal{CC}_1 \otimes \mathcal{CC}_2)[W_0] &= \mathcal{CC}_1[W_0] \times \mathcal{CC}_2[W_0] \times Y_0 \times B_0, & (\mathcal{CC}_1 \otimes \mathcal{CC}_2)[W_{in0}] &= X_0 \times A_0,
\end{aligned}$$

and functions

$$\begin{aligned}
\mathrm{msg}(x, w, w_{in}) &= (\mathcal{CC}_1[\mathrm{msg}](x, \mathcal{CC}_1[w], \mathcal{CC}_1[w_{in}]), \mathcal{CC}_2[\mathrm{msg}](x, \mathcal{CC}_2[w], \mathcal{CC}_2[w_{in}])),\\
\mathrm{stf}_{\mathrm{W}}(w, w_{in}, l) &= (\mathcal{CC}_1[\mathrm{stf}_{\mathrm{W}}](\mathcal{CC}_1[w], \mathcal{CC}_1[w_{in}], \mathcal{CC}_1[l]),\\
&\qquad \mathcal{CC}_2[\mathrm{stf}_{\mathrm{W}}](\mathcal{CC}_2[w], \mathcal{CC}_2[w_{in}], \mathcal{CC}_1[l]), \pi_B(\mathcal{CC}_1[\mathrm{stf}_{\mathrm{out}}](\mathcal{CC}_1[w], \mathcal{CC}_1[w_{in}])),\\
&\qquad \pi_Y(\mathcal{CC}_2[\mathrm{stf}_{\mathrm{out}}](\mathcal{CC}_2[w], \mathcal{CC}_2[w_{in}]))),\\
\mathrm{stf}_{\mathrm{out}}(w, w_{in}, l) &= (\pi_Z(\mathcal{CC}_2[\mathrm{stf}_{\mathrm{out}}](\mathcal{CC}_2[w], \mathcal{CC}_2[w_{in}]), \mathcal{CC}_2[l]),\\
&\qquad \pi_C(\mathcal{CC}_1[\mathrm{stf}_{\mathrm{out}}](\mathcal{CC}_1[w], \mathcal{CC}_1[w_{in}]), \mathcal{CC}_1[l])),
\end{aligned}$$
$$\mathrm{ctl}(x_{t_\ell}, x, w^{[i]}, w_{in}^{[i]}) = \mathcal{CC}_1[\mathrm{ctl}](x_{t_\ell}, x, \mathcal{CC}_1[w], \mathcal{CC}_1[w_{in}]) + \mathcal{CC}_2[\mathrm{ctl}](x_{t_\ell}, x, \mathcal{CC}_2[w], \mathcal{CC}_2[w_{in}]).$$

3. Main ideas behind the algorithm design. In this section, we provide an informal description of the algorithm design that is developed in the forthcoming sections. Our objective here is to provide intuition to help the understanding of the technical developments that come later.

3.1. Maintaining connectivity by preserving a spanning tree. We consider a robotic network executing a motion coordination algorithm to perform some spatial task in an environment of interest. The robotic network is modeled as a graph with robots as nodes and pairwise communication links between robots as edges. Our objective is to guarantee that connectivity is preserved while the network moves. The idea to do this is based on the simple observation that if a spanning subgraph of the robotic network remains connected at all times, then the network as a whole remains connected. Since any connected graph contains at least one spanning tree as a subgraph, we use it as the spanning subgraph. The reason for using a tree is that it has the minimal number of edges (hence posing the fewest number of constraints on the robotic network motion. It is worth noting that a network with the fewest number of constraints is not necessarily the same as the least constrained network). We address this by asking our underlying motion coordination algorithm to provide us with a ranking of preferences of which constraints it would prefer to have to maintain. Our algorithm picks the constraint tree guided by these preferences. We formalize how the constraints are represented in a distributed network in Definition 4.3. In our discussion, we make the assumption that the motion coordination algorithm has the property that, if a robot is given a list of neighbors in the spanning tree, the coordination algorithm will prescribe a robot motion that ensures that the links between itself and its neighbors are not broken. We refer to such algorithms as motion-compatible.

In order to preserve the edges of the tree, it is sufficient that each node is aware of the links incident to itself, and moves in such a way that these links are preserved. However, preserving the same set of links at all times might impose constraints on the network that are too tight and preclude it from achieving the spatial task the motion coordination algorithm is designed for. To cope with this problem, we allow the spanning tree to evolve with time according to a set of preferences that are determined by the motion coordination algorithm (how these preferences are established is discussed in detail in the later sections). Our main objective can then be stated as follows: design a distributed algorithm which can make such changes to the spanning tree while guaranteeing global graph connectivity. The following basic graph-theoretic facts are useful in prescribing how the links of the spanning tree can change:

(i) a tree on n nodes has exactly $n-1$ edges;

(ii) any connected graph on n nodes with $n-1$ edges has no cycles, or alternatively, any graph on n nodes with $n-1$ edges containing a cycle is disconnected;

In our algorithm, every robot other than the root is responsible for storing the link to its parent and deciding when this link changes. As a corollary of fact (i) above, this guarantees that each link in the tree is the responsibility of exactly one and only one robot. Because of fact (ii), we structure our algorithms for choosing the next parent of each robot with the express intent of preventing a new robot-parent connection from forming a cycle in the spanning tree.

3.2. Preventing cycles from forming via depth estimates. In our strategy, each robot has an estimate of its depth, i.e., the number of nodes in the tree along its path to the root. The depth estimate of the root is 0, and every other robot periodically runs an update rule to adjust its depth estimate by adding 1 to the depth estimate of its parent. With this update law, if the tree remains fixed for a sufficient number of steps, each robot's depth estimate eventually converges to its actual depth in the tree.

A simple way to enforce connectivity while allowing for the possibility of changing links is to set up the following rule: a node i can change its parent to a new node j if the depth estimate of j is strictly smaller than the depth estimate of i. While this rule would guarantee that no cycles are formed, it results in a too strict requirement. If each robot is only allowed to connect to parents with strictly smaller depth estimates, only a finite number of re-arrangements are possible before the tree becomes a star topology with each robot attached to the root. It turns out that allowing new connections from i to j even when both nodes have the same depth estimates is sufficient for flexible re-arrangements. However, this must be done with care to prevent cycles from forming among sequences of nodes having the same depth. Figure 3.1 provides a visual explanation of when connecting at the same depth does and does not disconnect the graph.

We have two mechanisms that work together to prevent the appearance of cycles. These mechanisms are based on the observation that, for any prospective cycle among agents of the same depth estimate, there must be one agent with a maximum unique identifier along the cycle. The mechanisms are as follows:

(i) a tie-breaking scheme based on agent unique identifiers. We prescribe that each robot announces some portion of its state and its intent to connect before actually changing the identity of its parent. We prevent a robot i from attaching to another robot j of the same depth estimate if $j < i$ and there is another robot k, with $k < i$ and the same depth estimate as i, which is trying

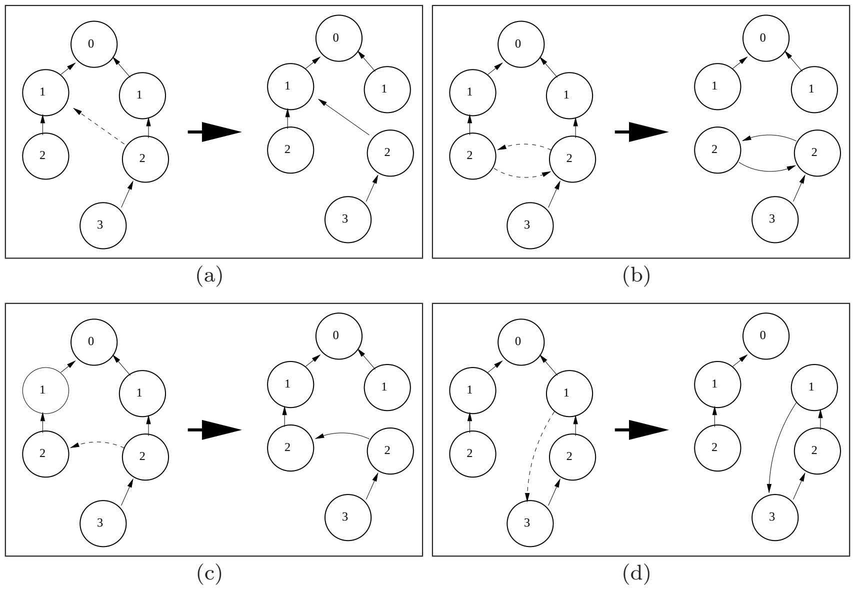


FIG. 3.1. *Illustrations of nodes attempting to attach to parents with a) lesser depth estimates b) equal depth estimates c) equal depth estimates d) greater depth estimates.*

to attach to i.

(ii) the above tie-breaking scheme only works if we can guarantee that the agent of maximum UID along the prospective cycle is capable of breaking the cycle by refusing to attach to its proposed new parent. We ensure this by using a boolean flag for each robot that indicates whether $\mathrm{dp}_{\mathrm{est}}^{[\mathtt{p}_{\mathrm{curr}}^{[i]}]} < \mathrm{dp}_{\mathrm{est}}^{[i]}$ on the previous round. We use this to ensure that any cycle created as a result of the addition of proposed new edges is entirely composed of newly created edges, and thus any agent (in our case, the one with the max UID) along the cycle can break it by failing to connect to its proposed new parent.

The strategy that results from the combination of the above ideas is guaranteed to preserve the connectivity of the robotic network, while allowing for the possibility of rearrangements in the set of links to be preserved. The CONNECTIVITY MAINTENANCE ALGORITHM presented in Section 4 is based on this strategy, but it incorporates two important modifications to deal with the issues that we discuss next.

3.3. Repair. The algorithm described in Section 3.2 requires that the robotic network starts from a valid spanning tree. If, due to some unanticipated failure, two robots lose one of the links they are supposed to preserve, the algorithm makes no guarantee that the network will become connected again. An extreme case of this situation would be a spanning tree initialization where all $n-1$ edges have been lost.

Our modifications to the above algorithm to handle unanticipated failures and tree initializations are based on the idea of allowing the tree algorithm to represent multiple trees with multiple roots. Each robot stores the id of the robot it thinks is the root of its tree. The robot updates the latter variable by setting it equal to the root of its parent. If the robot has no parent, it is either because it is the root itself, or because it has lost touch with its parent. In both cases, it sets the root identity equal to itself. We modify the rules on which pairs of robots are allowed to break their communication links to forbid breaking any edge between robots with different roots. We modify the set of preferences that determine how robots attach so that robots strongly prefer attaching to robots with smaller root ids, thus causing the number of robots with distinct roots to decrease over time. In our Connectivity Maintenance Algorithm proposed in Section 4, the modifications to handle unanticipated failures and handle tree initialization are denoted with the symbol ♣ in Table 4.1. Proofs that these modifications repair a large set of possible failures are presented in Section 6.

3.4. Reachability. The algorithm described in Section 3.2 has the property that, given any target spanning tree T, the robotic network can reach a state in which its current stored spanning tree is T. However, to establish this fact, one needs to require that the underlying graph allows any robot i to choose any other arbitrary robot j as its parent. Clearly, in a realistic scenario, there are some pairs of robots which cannot communicate, or cannot communicate without moving. Instead, we would like an algorithm which allows the network to reach a target spanning tree without relying on edges which may or may not be in the network.

The Connectivity Maintenance Algorithm proposed in Section 4 allows the network to transition from a state in which the spanning tree is T_1 to one in which the spanning tree is T_2, for any T_1 and T_2, so long as T_2 remains in the graph. We accomplish this by recognizing that there is nothing particularly special about the use of the depth as a tool to prevent cycles: we could use in its place any other notion to associate a number to each robot as long as it satisfies a simple set of properties across the tree (these properties are formally described in Proposition 5.1).

Therefore, we specify a new update rule for the depth estimate which, based on input from another algorithm, either updates the estimate by (a) adding 1 to the depth estimate of the robot's parent or (b) increasing its current depth estimate by 1. We refer to an algorithm that determines which update rule to use as a depth-compatible algorithm. In Table 7.1 we provide an instance of a depth-compatible algorithm which specifies that (b) is used in situations when a robot i wants to attach to another robot j and the depth estimate of j is larger than the depth estimate of i. Such situations can give rise to the formation of cycles, as, for instance, some other robot k, on a path from j to the root, might be simultaneously increasing its own depth estimate to be able to attach to i. The algorithm in Table 7.1 incorporates a tie-breaking rule which prevents such pathologies.

In the algorithm proposed in Section 4, these modifications to allow for arbitrary re-arrangements are denoted with the symbol ♠ in Table 4.1. Proofs of the reachability properties of the algorithm that results from the combination with the strategy in Table 7.1 are presented in Section 7.

4. The Connectivity Maintenance Algorithm. This section introduces the Connectivity Maintenance (abbreviated CM) Algorithm. By itself, the algorithm does not invoke either physical agents or their mobility, and fits within common frameworks of distributed algorithms.

4.1. Algorithm description. Given a coordination task and a motion coordination algorithm to achieve it, maintaining a fixed set spanning tree throughout the evolution will guarantee connectivity preservation but, in general, will interfere in the optimal achievement of the task. The Connectivity Maintenance Algorithm is a procedure that, coupled with individual motion control strategies of the network agents, maintains an evolving spanning tree of the communication graph. The underlying idea is that if the motion of the robots is constrained to not break any links of the spanning tree, the communication graph will remain connected as well. Let us begin by describing the algorithm informally.

> *[Informal description:]* Each robot maintains a reference to its parent in the spanning tree and an estimate of its depth, i.e., the distance to the root. At pre-arranged times, each robot is allowed to change its parent, in accordance with a set of preferences specified in suitable way (normally, by a motion coordination algorithm). In order to preserve connectivity, robots are not allowed to pick a robot of greater depth than its parent's (which ensures that no robot will pick one of its current descendants as a parent node). To allow robots to attach to potential parents of the same depth estimate, a tie-breaking algorithm based on UIDs is used to prevent the formation of cycles.

Next, we provide a formal definition using the formalism described in Section 2.2. Given a network $\mathcal{S}$ with communication links determined by proximity graph $\mathcal{G}$, the Connectivity Maintenance Algorithm is an input-output control and communication law $\mathcal{CC}$ for $\mathcal{S}$ consisting of the sets:

(i) $W = \mathbb{N} \times \mathbb{Z}_4 \times \mathbb{Z}_n^3 \times \mathbb{Z}_n \cup \{\texttt{null}\} \times \mathbb{Z}_2$, are sets of values of the variables $w^{[i]} = (\mathrm{dp}_{\mathrm{est}}^{[i]}, \mathrm{phase}^{[i]}, \mathfrak{p}_{\mathrm{curr}}^{[i]}, \mathfrak{p}_{\mathrm{next}}^{[i]}, n_{\mathrm{root}}^{[i]}, n_{\mathrm{dep\text{-}targ}}^{[i]}, \mathbb{I}_{\mathrm{par\text{-}less}}^{[i]})$, for $i \in \mathbb{Z}_n$.

The meaning of the components of $w^{[i]}$ is as follows. $\mathrm{dp}_{\mathrm{est}}^{[i]}$ is a depth estimate. $\mathrm{phase}^{[i]}$ is a round counter indicating the current mode of the algorithm. $\mathfrak{p}_{\mathrm{curr}}^{[i]}$ is a parent identifier. We refer to the graph induced by $(i, \mathfrak{p}_{\mathrm{curr}}^{[i]})$, $i \in \mathbb{Z}_n$ as the *constraint tree*. This is with a slight abuse of terminology, since we will also consider the possibility of this graph not being initially connected. $\mathfrak{p}_{\mathrm{next}}^{[i]}$ is a proposed next parent. $n_{\mathrm{root}}^{[i]}$ indicates the UID of the root of the tree containing agent i. $n_{\mathrm{dep\text{-}targ}}^{[i]}$ is the UID of an agent which i would like to attach to as a parent. It is the most preferred parent among those which the algorithms coupled with CM Algorithm allow (more details are provided below when we discuss $W_{\mathrm{in}}^{[i]}$). Note that i may not be allowed by the algorithm to propose that $n_{\mathrm{dep\text{-}targ}}^{[i]}$ be its next parent, for which reason $\mathfrak{p}_{\mathrm{next}}^{[i]}$ might not equal $n_{\mathrm{dep\text{-}targ}}^{[i]}$. Finally, $\mathbb{I}_{\mathrm{par\text{-}less}}^{[i]}$ is a Boolean indicator denoting whether i's parent $\mathfrak{p}_{\mathrm{curr}}^{[i]}$ had a strictly smaller depth estimate than i as of the most recent communication round. If $n_{\mathrm{root}}^{[i]}$ is different from 0 for some agent i, then the tree needs to be repaired. $n_{\mathrm{dep\text{-}targ}}^{[i]}$ is also used as an output logic variable, signaling that robot i wishes to increase $\mathrm{dp}_{\mathrm{est}}^{[i]}$ with the goal of attaching to $n_{\mathrm{dep\text{-}targ}}^{[i]}$;

(ii) $W_0^{[i]} = \{(0, 0, i, i, i, \texttt{null}, \mathrm{false})\} \subseteq W$, $i \in \mathbb{Z}_n$. Note that $\mathfrak{p}_{\mathrm{curr}}^{[i]} = i$ and $\mathrm{dp}_{\mathrm{est}}^{[i]} = 0$ for $i \in \mathbb{Z}_n$. Also note that the tree is initially in need of repair;

(iii) W_{in}, are sets of values of input variables, $w_{\mathrm{in}}^{[i]}$, $i \in \mathbb{Z}_n$, given by

$$\begin{aligned} w_{\mathrm{in}}^{[i]} =& (\mathbb{I}_{\mathrm{incr\text{-}dep}}^{[i]}, n_{\mathrm{incr\text{-}sgnl}}^{[i]}, R_{\mathrm{pref}}^{[i]}, f_{\mathrm{allow}}^{[i]}) \in \\ & \mathbb{Z}_2 \times \mathbb{Z}_n \cup \{\texttt{null}\} \times R(\mathbb{Z}_n, \mathbb{Z}_n) \times \mathfrak{F}(\mathbb{Z}_n; \{\mathrm{true}, \mathrm{false}\}). \end{aligned}$$

The meaning of the components of w_{in} is as follows. $\mathbb{I}_{\mathrm{incr\text{-}dep}}^{[i]}$ is a Boolean variable that determines whether the agent's depth estimate update is in a special mode to allow for more flexible re-arrangements. $n_{\mathrm{incr\text{-}sgnl}}^{[i]}$ is the identity of an agent which is in the special mode indicated by $\mathbb{I}_{\mathrm{incr\text{-}dep}}^{[\cdot]}$. $R_{\mathrm{pref}}^{[i]}$ is a strict order relation ranking the order in which node i would prefer to attach to each other node as its parent: $(j,k) \in R^{[i]}_{\text{pref}}$ means node i would prefer to attach to j over k. We stipulate that each $R^{[i]}_{\text{pref}}$ must satisfy $(j,i) \in R^{[i]}_{\text{pref}}$ for all $j \in \{0,\dots,n-1\} \setminus \{i\}$ and that $n^{[j]}_{\text{root}} < n^{[k]}_{\text{root}}$ implies $(j,k) \in R^{[i]}_{\text{pref}}$. $R^{[i]}_{\text{pref}}$ can be thought of as a symbol string representing computation to be performed to evaluate whether $(j,k) \in R^{[i]}_{\text{pref}}$. Finally, $f^{[i]}_{\text{allow}} : \mathbb{Z}_n \to \{\text{true}, \text{false}\}$ is a function which maps a node j to true if and only if the algorithm supplying $f^{[i]}_{\text{allow}}$ will allow $\text{dp}^{[i]}_{\text{est}}$ to increase in order for i to attach to j under the CM Algorithm;

(iv) $W^{[i]}_{\text{in},0} = \{(\text{false}, \texttt{null}, R_{i-\text{init}}, f^{[i]}_{\text{allow}})\}$, where $(j,k) \in R_{i-\text{init}}$ if $n^{[j]}_{\text{root}} < n^{[k]}_{\text{root}}$ or $n^{[j]}_{\text{root}} = n^{[k]}_{\text{root}}$ and $((k = i \text{ and } j \neq i) \text{ or } j < k)$, and $f^{[i]}_{\text{allow}} \equiv \text{false}$;

(v) $W_{\text{out}} = \mathbb{Z}_n \times \mathbb{Z}_n \times \mathbb{F}(\mathbb{Z}_n) \times \mathbb{F}(\mathbb{Z}_n) \times \mathbb{Z}_n \cup \{\texttt{null}\}$ are sets of values of output variables $w^{[i]}_{\text{out}} = (\mathfrak{p}^{[i]}_{\text{curr}}, \mathfrak{p}^{[i]}_{\text{next}}, S^{[i]}_{\text{constraints}}, S^{[i]}_{\text{children}}, n^{[i]}_{\text{dep-targ}})$, for $i \in \mathbb{Z}_n$.

The meaning of the components of W_{out} is as follows. $\mathfrak{p}^{[i]}_{\text{curr}}$ (taken from $W^{[i]}$) specifies the parent of agent i to the motion control algorithm. $S^{[i]}_{\text{constraints}} = \{j \in \mathcal{N}(i) \mid n^{[j]}_{\text{root}} \neq n^{[i]}_{\text{root}}\}$ is the set of agents whose estimate of their root nodes are different from those of i and $S^{[i]}_{\text{children}} = \{j \in \mathbb{Z}_n \mid \mathfrak{p}^{[j]}_{\text{curr}} = i\}$ are the children of i in the constraint tree. If every i maintains connectivity with each agent in $\{\mathfrak{p}^{[i]}_{\text{curr}}\} \cup S^{[i]}_{\text{constraints}} \cup S^{[i]}_{\text{children}}$, then the communication graph remains connected. $n^{[i]}_{\text{dep-targ}} \in \mathbb{Z}_n \cup \{\texttt{null}\}$ is copied from W and used to signal which agents i wants to increase depth to connect to; and of the maps:

(i) **function** $\text{msg}(x, w, w_{\text{in}}, j) = w$;

(ii) **function** $\text{stf}_{\text{W}}(w, w_{\text{in}}, l)$ as defined in Table 4.1. In the update step of $\text{phase}^{[\text{id}]} = 2$, there are two possible depth update steps, one on line `12` and the other on line `14`. Line `12` is an update rule which, if universally followed, causes each agent's depth estimate, $\text{dp}^{[\cdot]}_{\text{est}}$, to converge on its actual depth in the tree. Line `14` is used when one agent wants to deliberately increase its depth estimate above its actual depth in order to attach to another desired agent. Usage of line`14` will be discussed in more detail in Section 7.2. Note that `9` is labeled as the "re-attach" step. We will use this nomenclature throughout the paper;

(iii) **function** $\text{stf}_{\text{out}}(w, w_{\text{in}}, l) = (\mathfrak{p}^{[\text{id}]}_{\text{curr}}, \mathfrak{p}^{[\text{id}]}_{\text{next}}, \{j \in \mathcal{N}(\text{id}) \mid n^{[j]}_{\text{root}} \neq n^{[\text{id}]}_{\text{root}}\}, \{j \in \mathcal{N}(\text{id}) \mid \mathfrak{p}^{[j]}_{\text{curr}} = \text{id} \vee \mathfrak{p}^{[j]}_{\text{next}} = \text{id}\}, n^{[i]}_{\text{dep-targ}})$.

(iv) $\text{ctl}(x_{t_\ell}, x, w, w_{\text{in}}) = 0$.

4.2. Depth-compatible and motion-compatible algorithms. We intend for the Connectivity Maintenance Algorithm to be coupled with two classes of input-output laws. The first class of algorithms, termed depth-compatible, specify the update depth rule to be used by the Connectivity Maintenance Algorithm by setting the value of the variable $\mathbb{I}^{[i]}_{\text{incr-dep}}$ for each agent after receiving the value of $n^{[i]}_{\text{dep-targ}}$. The second class of algorithms, termed motion-compatible, specify the dynamics of the network agents and the set of preferences $R^{[i]}_{\text{pref}}$ for each agent.

We begin by formalizing the notion of depth-compatible algorithm.

Definition 4.1. *An input-output control and communication law* CC_d *is* depth CM-compatible *if the following hold:*

- $W_{in} = \mathbb{Z}_n \cup \{\texttt{null}\}$ *is a set of values for* $w^{[i]}_{in} = n^{[i]}_{dep\text{-}targ}$ *for* $i \in \mathbb{Z}_n$*;*
- W_{out}*, are sets of values of* output logic variables*,* $(\mathbb{I}^{[i]}_{incr\text{-}dep}, n^{[i]}_{incr\text{-}sgnl}, f^{[i]}_{allow}) \in \mathbb{Z}_2 \times \mathbb{Z}_n \cup \{\texttt{null}\} \times \mathfrak{F}(\mathbb{Z}_N; \{true, false\})$*;*

function $\mathrm{stf}_{\mathrm{W}}(w_{\mathrm{in}}, (\mathfrak{p}^{[\mathrm{id}]}_{\mathrm{curr}}, \mathrm{dp}^{[\mathrm{id}]}_{\mathrm{est}}, \mathrm{phase}^{[\mathrm{id}]}, \mathfrak{p}^{[\mathrm{id}]}_{\mathrm{next}}), l)$

---

if $n^{[\mathrm{id}]}_{\mathrm{root}} = \mathrm{id}$ $\clubsuit$ **then**
    Set $n_{\mathrm{max}} \leftarrow \mathrm{id}(n+1)$ $\clubsuit$
else
    Set $n_{\mathrm{max}} \leftarrow n^{[\mathrm{id}]}_{\mathrm{root}}(n+1) + n$ $\clubsuit$

/**If proposed parent has same depth as id, is trying to change its parent, and id is greater than my proposed parent's UID and the UIDs of any nodes that have proposed attaching to me as a parent, then keep current parent. Otherwise choose proposed parent**/

if $\mathrm{phase}^{[\mathrm{id}]} = 0$ **then**
    if $\mathfrak{p}^{[\mathfrak{p}^{[\mathrm{id}]}_{\mathrm{next}}]}_{\mathrm{next}} \neq \mathfrak{p}^{[\mathfrak{p}^{[\mathrm{id}]}_{\mathrm{next}}]}_{\mathrm{curr}}$, $\mathrm{dp}^{[\mathfrak{p}^{[\mathrm{id}]}_{\mathrm{next}}]}_{\mathrm{est}} = \mathrm{dp}^{[\mathrm{id}]}_{\mathrm{est}}$, $\mathrm{id} > \mathfrak{p}^{[\mathrm{id}]}_{\mathrm{next}}$ and $\mathrm{id} > \min\{j \mid \mathfrak{p}^{[j]}_{\mathrm{next}} = \mathrm{id}\}$ **then**
        Set $\mathfrak{p}^{[\mathrm{id}]}_{\mathrm{next}} \leftarrow \mathfrak{p}^{[\mathrm{id}]}_{\mathrm{curr}}$
    else
        Set $\mathfrak{p}^{[\mathrm{id}]}_{\mathrm{curr}} \leftarrow \mathfrak{p}^{[\mathrm{id}]}_{\mathrm{next}}$ /**We call this the "re-attach" step**/
if $\mathrm{phase}^{[\mathrm{id}]} = 1$ /**Update depth estimate**/ **then**
    if $\mathbb{I}^{[i]}_{\text{incr-dep}} = \mathrm{false}$ $\spadesuit$ **then**
        Set $\mathrm{dp}^{[\mathrm{id}]}_{\mathrm{est}} \leftarrow \min(\mathrm{dp}^{[\mathfrak{p}^{[\mathrm{id}]}_{\mathrm{curr}}]}_{\mathrm{est}} + 1, n_{\mathrm{max}})$ $\clubsuit$
    else
        Set $\mathrm{dp}^{[\mathrm{id}]}_{\mathrm{est}} \leftarrow \min(\mathrm{dp}^{[\mathrm{id}]}_{\mathrm{est}} + 1, n_{\mathrm{max}})$ $\spadesuit$
if $\mathrm{phase}^{[\mathrm{id}]} = 2$ **then**
    Set $\mathbb{I}^{[\mathrm{id}]}_{\text{par-less}} \leftarrow \mathrm{false}$
    if $\mathrm{dp}^{[\mathfrak{p}^{[\mathrm{id}]}_{\mathrm{curr}}]}_{\mathrm{est}} < \mathrm{dp}^{[\mathrm{id}]}_{\mathrm{est}}$ **then**
        Set $\mathbb{I}^{[\mathrm{id}]}_{\text{par-less}} \leftarrow \mathrm{true}$

/**As a worst-case solution, consider attaching to yourself. By definition of $R^{[i]}_{pref}$, this does not happen unless there are no other options**/

if $\mathrm{phase}^{[\mathrm{id}]} = 3$ /**Here we pick $\mathfrak{p}^{[i]}_{next}$ and $n^{[i]}_{dep\text{-}targ}$ according to $R^{[i]}_{pref}$ among the appropriate sets of allowable values**/ **then**
    Let $S_{<} = \{j \in \mathcal{N}(\mathrm{id}) \mid \mathrm{dp}^{[j]}_{\mathrm{est}} < \mathrm{dp}^{[\mathrm{id}]}_{\mathrm{est}}$ or $\mathfrak{p}^{[j]}_{\mathrm{curr}} = \mathfrak{p}^{[\mathrm{id}]}_{\mathrm{curr}}$ or $\mathrm{dp}^{[j]}_{\mathrm{est}} = \mathrm{dp}^{[\mathrm{id}]}_{\mathrm{est}}$ and $\mathbb{I}^{[j]}_{\text{par-less}} = \mathrm{true}$ or $j = i\}$
    Let $\mathfrak{p}^{[\mathrm{id}]}_{\mathrm{next}} \leftarrow$ some element of $\{j \in S_{<} \mid \nexists k, (k,j) \in R^{[\mathrm{id}]}_{\mathrm{pref}}, k \in S_{<}\}$
    if $\{j \in S_{<} \mid \nexists k, (k,j) \in R^{[\mathrm{id}]}_{\mathrm{pref}}, k \in S_{<}\} = \{\mathfrak{p}^{[\mathrm{id}]}_{\mathrm{curr}}\}$ $\spadesuit$ **then**
        Let $S_{\mathrm{sgnl}} \leftarrow \{j \in \mathcal{N}(\mathrm{id}) \mid f^{[i]}_{\mathrm{allow}}(j) = \mathrm{true}\}$ $\spadesuit$
        Let $n^{[\mathrm{id}]}_{\text{dep-targ}} \leftarrow \{j \in S_{\mathrm{sgnl}} \mid \nexists k \in S_{\mathrm{sgnl}}, (k,j) \in R^{[\mathrm{id}]}_{\mathrm{pref}}\}$ $\spadesuit$
        if $n^{[\mathrm{id}]}_{\text{dep-targ}} = \mathrm{id}$ or $n^{[\mathrm{id}]}_{\text{dep-targ}} = \mathfrak{p}^{[\mathrm{id}]}_{\mathrm{curr}}$ $\spadesuit$ **then**
            $n^{[\mathrm{id}]}_{\text{dep-targ}} \leftarrow$ some element of $\{j \in \mathcal{N}(\mathrm{id}) \mid \nexists k \in \mathcal{N}(\mathrm{id}), (k,j) \in R^{[\mathrm{id}]}_{\mathrm{pref}}\}$ $\spadesuit$
    else
        $n^{[\mathrm{id}]}_{\text{dep-targ}} \leftarrow$ `null` $\spadesuit$
if $\mathfrak{p}^{[\mathrm{id}]}_{\mathrm{curr}} = \mathrm{id}$ $\clubsuit$ **then**
    Set $n^{[\mathrm{id}]}_{\mathrm{root}} \leftarrow \mathrm{id}$ $\clubsuit$
else
    Set $n^{[\mathrm{id}]}_{\mathrm{root}} \leftarrow n^{[\mathfrak{p}^{[\mathrm{id}]}_{\mathrm{curr}}]}_{\mathrm{root}}$ $\clubsuit$
Set $\mathrm{phase}^{[\mathrm{id}]} \leftarrow (\mathrm{phase}^{[\mathrm{id}]} + 1) \bmod 4$
return $(\mathrm{dp}^{[\mathrm{id}]}_{\mathrm{est}}, \mathrm{phase}^{[\mathrm{id}]}, \mathfrak{p}^{[\mathrm{id}]}_{\mathrm{curr}}, \mathfrak{p}^{[\mathrm{id}]}_{\mathrm{next}}, n^{[\mathrm{id}]}_{\mathrm{root}}, n^{[\mathrm{id}]}_{\text{dep-targ}}, \mathbb{I}^{[\mathrm{id}]}_{\text{par-less}})$ $\clubsuit\spadesuit$

TABLE 4.1
$\mathrm{stf}_{\mathrm{W}}$ *for* CONNECTIVITY MAINTENANCE ALGORITHM.

- *The composition with* CM Algorithm *guarantees that the maximum depth estimate $dp_{est}^{[\cdot]}$ that any agent i having $n_{root}^{[i]} = 0$ holds for more than 4 rounds is bounded from above by a function of the number of agents n.*

Combining an algorithm which is *depth* CM-*compatible* yields a new input-output control and communication law. The simplest algorithm which is *depth* CM-*compatible*, and the one we assume unless otherwise stated, is Null Depth Increment Algorithm. Null Depth Increment Algorithm ignores the preference input from CM Algorithm and always forces CM Algorithm to follow the $\mathrm{dp}_{\mathrm{est}}^{[\mathrm{id}]} \leftarrow \min(\mathrm{dp}_{\mathrm{est}}^{[\mathfrak{p}_{\mathrm{curr}}^{[\mathrm{id}]}]} + 1, n_{\max})$ depth update rule. Essentially, the only variables that Null Depth Increment Algorithm specifies are $\mathbb{I}_{\text{incr-dep}}^{[i]} = \text{false}$ and $f_{\text{allow}}^{[i]} \equiv \text{false}$ for all $i \in \mathbb{Z}_n$. A formal definition of Null Depth Increment Algorithm can be given in the form of an input-output control and communication law as in Definition 2.2, but we omit it for brevity.

Note that we cap the depth estimate of any given node at a number (denoted by the temporary variable $n_{\max}$ within the description of CM) determined by the number of agents. We show that this does not affect the operation under Null Depth Increment Algorithm in the next result.

Theorem 4.2. *If $\mathbb{I}_{incr\text{-}dep}^{[i]} = false$ and $n_{root}^{[i]} = 0$ for all $i \in \mathbb{Z}_n$, the following invariant is maintained: at most $n - k$ nodes have depth estimates greater than or equal to k at any time.*

Proof. Assume the result holds for round t. To show the invariant holds at round $t+1$, we induct on the depth estimate, using as the base case the fact that at most n nodes have depth estimate 0. Let any k nodes, $i_i, \dots, i_k$ each increase depth (i.e., $\mathrm{dp}_{\mathrm{est}}^{[i_j]}(t{+}1) > \mathrm{dp}_{\mathrm{est}}^{[i_j]}(t)$ for all $j \in \{1, \dots, k\}$) according to $\mathrm{dp}_{\mathrm{est}}^{[i_j]}(t{+}1) \leftarrow \mathrm{dp}_{\mathrm{est}}^{[\mathfrak{p}_{\mathrm{curr}}^{[i_j]}]}(t){+}1$, $j \in \{1, \dots, k\}$ ($n_{\max}$ does not appear, as it is greater than the maximum value of $\mathrm{dp}_{\mathrm{est}}^{[\cdot]}$ of any agent we are considering). To have greater depth at $t+1$ than on round t, each i_j must have had a parent, $\mathfrak{p}_{\mathrm{curr}}^{[i_j]}$, having $\mathrm{dp}_{\mathrm{est}}^{[\mathfrak{p}_{\mathrm{curr}}^{[i_j]}]}(t) = \mathrm{dp}_{\mathrm{est}}^{[i_j]}(t)$. Without loss of generality, let i_1 be the agent with the least depth at round t. By our induction hypothesis, there were at most $n - \mathrm{dp}_{\mathrm{est}}^{[i_1]}(t)$ nodes at depth $\mathrm{dp}_{\mathrm{est}}^{[i_1]}(t)$ or greater on round t. Thus, there were at most $n - \mathrm{dp}_{\mathrm{est}}^{[i_1]}(t) - k - 1$ already at depth estimates $\geq \mathrm{dp}_{\mathrm{est}}^{[i_1]}(t)$, since k must have been at depth estimate $\mathrm{dp}_{\mathrm{est}}^{[i_1]}$ to increase at round $t+1$, and at least one node must have been at depth $\mathrm{dp}_{\mathrm{est}}^{[\mathfrak{p}_{\mathrm{curr}}^{[i_1]}]}$. This gives us at most $n - \mathrm{dp}_{\mathrm{est}}^{[i_1]}(t) - k - 1 + k$ nodes at depth $\mathrm{dp}_{\mathrm{est}}^{[i_1]}(t) + 1$ or greater at time $t+1$. ∎

The operation of CM Algorithm composed with Null Depth Increment Algorithm is illustrated in Figure 4.1.

We end this section by formalizing the notion of motion-compatible algorithms. As we will see later, certain properties of the dynamics of the communication graph need to hold for the correctness properties of CM Algorithm to hold, and these are captured in the following notion.

Definition 4.3. *An input-output control and communication law* CC_d *is* motion CM-compatible *if the following hold:*

- $W_{in} = \mathbb{Z}_n \times \mathbb{Z}_n \times \mathbb{F}(\mathbb{Z}_n) \times \mathbb{F}(\mathbb{Z}_n) \times \mathbb{Z}_n \cup \{\texttt{null}\}$ *is a set of values for* $w_{in}^{[i]} = (\mathfrak{p}_{curr}^{[i]}, \mathfrak{p}_{next}^{[i]}, S_{constraints}^{[i]}, S_{children}^{[i]}, n_{dep\text{-}targ}^{[i]})$*, with* $i \in \mathbb{Z}_n$*;*
- $W_{out} = R(\mathbb{Z}_n, \mathbb{Z}_n)$*,* $w_{out}^{[i]} = R_{pref}^{[i]}$*, with* $i \in \mathbb{Z}_n$*;*
- *the input-output law that results from the combination with* CM Algorithm *is guaranteed never to induce a motion which causes $(i, \mathfrak{p}_{next}^{[i]})$ or $(i, \mathfrak{p}_{curr}^{[i]})$ to cease to be an edge of the underlying proximity graph. We require that these*

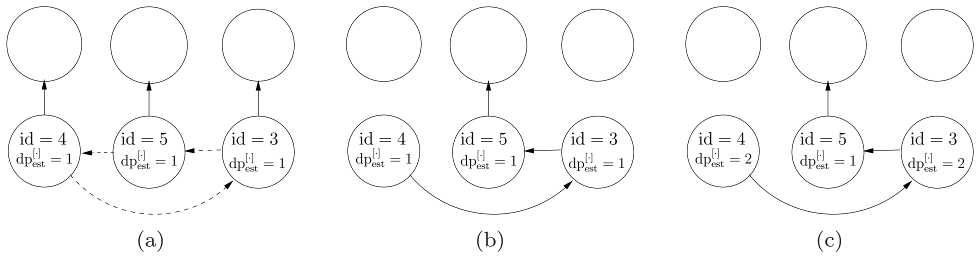


FIG. 4.1. *Illustration of* CM ALGORITHM *combined with* NULL DEPTH INCREMENT ALGORITHM. *Frame (a) indicates connections of the form* $(i, \mathfrak{p}_{curr}^{[i]})$ *as solid arrows and* $(i, \mathfrak{p}_{next}^{[i]})$ *as dashed arrows. Frame (b) illustrates the constraint tree after steps 5 through 9 of* CM ALGORITHM. *Frame (c) shows the result of the depth update in lines 10 through 14.*

also guarantee that no edge, (i, j), *of the underlying graph, having* $n_{root}^{[i]} \neq n_{root}^{[j]}$, *ever be broken. Note that* $S_{constraints}^{[i]} = \{j \in \mathcal{N}(i) \mid n_{root}^{[j]} \neq n_{root}^{[i]}\}$.

Figure 4.2 shows how the input-output laws introduced above interact.

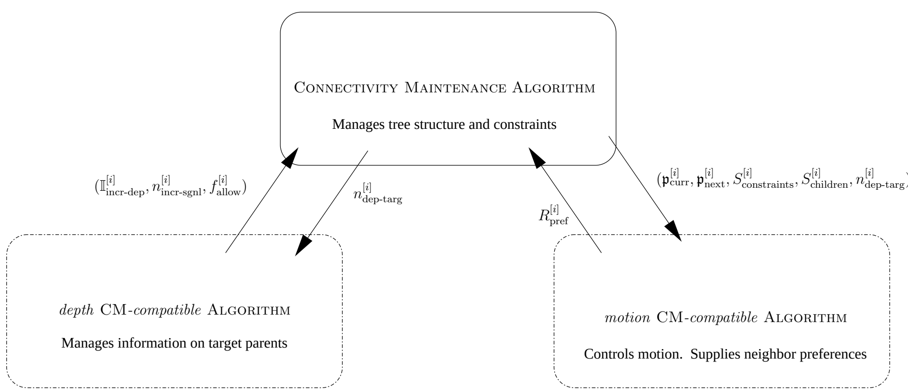


FIG. 4.2. *Illustration of the interaction between the* CONNECTIVITY MAINTENANCE ALGORITHM, *a* depth CM-compatible ALGORITHM *and a* motion CM-compatible ALGORITHM. *The* depth CM-compatible ALGORITHM *and the* CONNECTIVITY MAINTENANCE ALGORITHM *provide to each other the update depth rule to be used and the information on parent targets, respectively. The* motion CM-compatible ALGORITHM *and the* CONNECTIVITY MAINTENANCE ALGORITHM *provide to each other the preference set and the connectivity constraints that must be enforced, respectively.*

5. Correctness analysis. In this section, we analyze the correctness of CONNECTIVITY MAINTENANCE ALGORITHM when combined with any algorithm which is *motion* CM-*compatible* and any algorithm which is *depth* CM-*compatible*. We show that connectivity is preserved throughout the execution of the algorithm and also analyze its tree repair properties. We start by characterizing the topological properties of the constraint tree under CONNECTIVITY MAINTENANCE ALGORITHM.

For convenience, in the forthcoming analysis, we let $\text{rnd}(t) \in \mathbb{N}$ be the number of times the assignment $\text{phase}^{[i]} \leftarrow 2$ has been made at time t. We also, informally, refer to $\text{rnd}(t)$ as the number of iterations at time t and a cycling through all four values of $\text{phase}^{[i]}$ as an "iteration" of the algorithm. We denote the value of, say $\text{dp}_{\text{est}}^{[i]}$ at iteration $\text{rnd}(t)$, by $\text{dp}_{\text{est}}^{[i]}(\text{rnd}(t))$.

PROPOSITION 5.1. *The execution of the* CM ALGORITHM *verifies that* *(i)* $dp^{[i]}_{est}(rnd(t)) \leq dp^{[i]}_{est}(rnd(t)-1)+1$,

(ii) $dp^{[i]}_{est}(rnd(t)) \geq dp^{[\mathfrak{p}^{[i]}_{curr}]}_{est}(rnd(t))$,

for $i \in \mathbb{Z}_n$, *where for convenience,* $dp^{[i]}_{est}(r) = dp^{[i]}_{T}(t_0)$ *for all iterations* $r \leq 0$. *Thus, at any time* $t \geq 0$, *if* k *is an ancestor of* i, *then* $dp^{[i]}_{est}(rnd(t)) \geq dp^{[k]}_{est}(rnd(t))$.

Proof. Note that either lines `12` or `14` of $\mathrm{stf_W}$ for CM ALGORITHM, cf. Table 4.1, are the only steps where the value of $\mathrm{dp}^{[i]}_{\mathrm{est}}$ is modified. We refer to either of these steps as the *update rule*. Step `14` trivially maintains *(i)*. To show `12` also maintains *(i)*, we induct on the current iteration, $\mathrm{rnd}(t)$. Let j be $\mathfrak{p}^{[i]}_{\mathrm{curr}}$ at iteration $\mathrm{rnd}(t)$. This can only happen because either (a) j became i's parent due to a re-attach or (b) j was i's parent at $\mathrm{rnd}(t)-1$. In case (a), the re-attach requires $\mathrm{dp}^{[j]}_{\mathrm{est}}(\mathrm{rnd}(t)-1) \leq \mathrm{dp}^{[i]}_{\mathrm{est}}(\mathrm{rnd}(t)-1)$, and this implies that $\mathrm{dp}^{[i]}_{\mathrm{est}}(\mathrm{rnd}(t)) \leq \mathrm{dp}^{[i]}_{\mathrm{est}}(\mathrm{rnd}(t)-1)+1$. In case (b), $\mathrm{dp}^{[i]}_{\mathrm{est}}(\mathrm{rnd}(t)-1) = \mathrm{dp}^{[j]}_{\mathrm{est}}(\mathrm{rnd}(t)-2)+1$. The induction hypothesis implies that $\mathrm{dp}^{[j]}_{\mathrm{est}}(\mathrm{rnd}(t)-1) \leq \mathrm{dp}^{[j]}_{\mathrm{est}}(\mathrm{rnd}(t)-2)+1$, and therefore $\mathrm{dp}^{[i]}_{\mathrm{est}}(\mathrm{rnd}(t)) \leq \mathrm{dp}^{[i]}_{\mathrm{est}}(\mathrm{rnd}(t)-1)+1$. To prove *(ii)*, we note that either i has attached to $\mathfrak{p}^{[i]}_{\mathrm{curr}}$ more recently than $\mathrm{dp}^{[\mathfrak{p}^{[i]}_{\mathrm{curr}}]}_{\mathrm{est}}$ has changed, or, by the update rule, $\mathrm{dp}^{[i]}_{\mathrm{est}}(\mathrm{rnd}(t)) \geq \mathrm{dp}^{[\mathfrak{p}^{[i]}_{\mathrm{curr}}]}_{\mathrm{est}}(\mathrm{rnd}(t)-1)+1$, from which we get $\mathrm{dp}^{[i]}_{\mathrm{est}}(\mathrm{rnd}(t)) \geq \mathrm{dp}^{[\mathfrak{p}^{[i]}_{\mathrm{curr}}]}_{\mathrm{est}}(\mathrm{rnd}(t)$, which concludes the result. □

The above result is key in showing that if the constraint tree begins with k connected components, then, under certain technical conditions, the CM ALGORITHM is guaranteed to preserve their connectivity. In particular, if the constraint tree begins connected, $k=1$, then it remains connected along the execution, as we show next.

THEOREM 5.2. *Assume the constraint tree starts with* k *disjoint connected components. Then, at all times during the execution of* CM ALGORITHM, *the constraint tree contains no cycles other than those it started with and retains at most* k *disjoint connected components, so long as there are no further communications failures (i.e., no edge of the form* $(i, \mathfrak{p}^{[i]}_{curr})$ *or* $(i, \mathfrak{p}^{[i]}_{next})$ *disappears from the underlying proximity graph).*

Proof. The removal of an edge of the form $(i, \mathfrak{p}^{[i]}_{\mathrm{curr}})$ from the underlying graph causes the algorithm to set $\mathfrak{p}^{[i]}_{\mathrm{curr}} \leftarrow i$, potentially introducing a new cycle. Barring this possibility, let us proceed with the rest of the proof. We assume the introduction of a new cycle and proceed by contradiction. Any cycle in the graph must consist entirely of agents having the same depth estimate (this follows from $\mathrm{dp}^{[i]}_{\mathrm{est}} \geq \mathrm{dp}^{[\mathfrak{p}^{[i]}_{\mathrm{curr}}]}_{\mathrm{est}}$ from Proposition 5.1 and the fact that for a cycle C, we have $\sum_{i\in C} \mathrm{dp}^{[i]}_{\mathrm{est}} - \mathrm{dp}^{[\mathfrak{p}^{[i]}_{\mathrm{curr}}]}_{\mathrm{est}} = 0$). During the four communication rounds, cf. Table 4.1, leading up to the creation of any cycle, all agents involved in the cycle must have the same depth estimate as well, otherwise there exists either an agent $i \in C$ having $\mathrm{dp}^{[\mathfrak{p}^{[i]}_{\mathrm{curr}}]}_{\mathrm{est}} > \mathrm{dp}^{[i]}_{\mathrm{est}}$, which violates Proposition 5.1, or an agent $i \in C$ having $\mathrm{dp}^{[\mathfrak{p}^{[i]}_{\mathrm{next}}]}_{\mathrm{est}} > \mathrm{dp}^{[i]}_{\mathrm{est}}$, a choice which is prohibited by step `20` of the CM ALGORITHM. When a cycle first appears, it must appear due to some agent i connecting to a new parent j with the same depth estimate. However, this new parent must, in turn, attach to an agent of the same depth estimate. It cannot have had a parent of the same depth estimate before the cycle was created, as that would have set $\mathbb{I}^{[j]}_{\text{par-less}} = \text{false}$ and prevented i from connecting to j. This argument can be carried all the way around the cycle, C, and we can conclude that for each agent $k \in C$, k attached to a new parent at the time the cycle was formed. Here we invoke the UID-based tie-breaking scheme of step `6`, and note that some agent i in this cycle must have been greater than either of its neighbors in C, which explicitly triggers the execution of step 7, preventing i from attaching to $\mathfrak{p}^{[i]}_{\mathrm{next}}$ and thus preventing the formation of a cycle. □

The following result is a direct consequence.

COROLLARY 5.3. *If no edges of the form $(i, \mathfrak{p}^{[i]}_{curr})$ or $(i, \mathfrak{p}^{[i]}_{next})$ are removed from the underlying graph, the connected component of the constraint tree corresponding to $n^{[i]}_{root} = 0$ remains connected and never decreases in size.*

6. Tree repair properties. In this section we show that the CM ALGORITHM can repair links of the constraint tree under some conditions on the evolution of the underlying graph. The results in this section hold for CM ALGORITHM when combined with any algorithm which is *depth* CM-*compatible* ALGORITHM. The importance of our results is emphasized by the observation that it is not possible to design a repair algorithm which allows links to break while at the same time handling all possible agent failures, as we show next.

THEOREM 6.1. *There is no distributed repair algorithm which can allow links to break and, at the same time, recover from all possible underlying hardware failures which leave the communication graph connected.*

Proof. Consider an edge between agents i and j which can safely be broken at time t in the absence of underlying hardware failures. Let there be a cut of the graph which includes (i, j) and no other links sharing nodes i and j. The remaining edges in the cut could be severed by hardware failure immediately after the algorithm informs i and j that their link is safe to break, and the information would require at least one round to reach i and j. □

This result justifies our ensuing study of the link repair properties of CM ALGORITHM. We divide our analysis in two parts. In Section 6.1, we show that repair works under the CM ALGORITHM if the underlying network remains connected. The agents' mobility might break the connectivity of the underlying network. Section 6.2 establishes the repair properties of CM ALGORITHM even if the underlying graph is dynamically changing and might get disconnected at some times.

6.1. Repair properties when the underlying proximity graph remains connected. We study first the repair properties when the underlying graph is connected.

LEMMA 6.2. *For any node $i \in \mathbb{Z}_n$, if $n^{[i]}_{root}$ stays constant for $k \leq n^{[i]}_{root}(n+1)$ iterations, and no link failures happen in the intervening time, then, at the end of these iterations, $dp^{[i]}_{est} \geq k$.*

Proof. We induct on the number of iterations. At iteration 0 each node, i, satisfies $\mathrm{dp}^{[i]}_{\mathrm{est}} \geq 0$. By the update rule, $\mathrm{dp}^{[i]}_{\mathrm{est}}$ at iteration k gets 1 more than the depth estimate of $\mathfrak{p}^{[i]}_{\mathrm{curr}}$ at iteration $k-1$. Since no node ever attaches to a parent with greater $n^{[\cdot]}_{\mathrm{root}}$, $n^{[\mathfrak{p}^{[i]}_{\mathrm{curr}}]}_{\mathrm{root}}$ must have been greater than or equal to $n^{[i]}_{\mathrm{root}}$ for all $k-1$ previous iterations, and, by induction, $\mathrm{dp}^{[\mathfrak{p}^{[i]}_{\mathrm{curr}}]}_{\mathrm{est}}$ must have been at least $k-1$. This makes $\mathrm{dp}^{[i]}_{\mathrm{est}}$ at least $1 + (k-1) = k$ at the end of the kth iteration. □

The next result shows that this algorithm can repair breaks in the spanning tree whenever the underlying graph remains connected.

THEOREM 6.3. *Let K be a connected component of the underlying proximity graph with n_K agents. Let id_K be the smallest UID of the nodes in K. Assume K remains connected during the evolution and isolated from any other connected component, and that the (not necessarily connected) constraint tree is such that the only cycles are self-loops. Then, within $id_K(n+1) + n + n_K$ iterations of* CM ALGORITHM, *every* *node i in K has $n^{[i]}_{root} = id_K$.*

Proof. By Lemma 6.2 within $(\mathrm{id}_K + 1)(n+1)$ iterations each node i with $n^{[i]}_{\mathrm{root}} > \mathrm{id}_K$ will have $\mathrm{dp}^{[i]}_{\mathrm{est}} \geq (\mathrm{id}_K + 1)(n+1) > \mathrm{id}_K(n+1)$. Since the underlying graph remains connected at all times, there is some node i having $n^{[i]}_{\mathrm{root}} \neq \mathrm{id}_K$ with an edge (in the underlying graph) between itself and some j having $n^{[j]}_{\mathrm{root}} = \mathrm{id}_K$. Agent i will prefer to attach to nodes $\{j \in \mathbb{Z}_n \mid n^{[j]}_{\mathrm{root}} = \mathrm{id}_K\}$ over all other nodes (the preference function is constrained so that i prefers to attach to nodes with smaller $n^{[\cdot]}_{\mathrm{root}}$, and the smallest $n^{[\cdot]}_{\mathrm{root}}$ available is id_K). Each of these nodes has $\mathrm{dp}^{[j]}_{\mathrm{est}} \leq \mathrm{id}_K(n+1) + n$, so the attach is allowed. This can happen for at most n iterations before every node i satisfies $n^{[i]}_{\mathrm{root}} = \mathrm{id}_K$. □

This result implies that the constraint tree connects all the nodes of K (otherwise, some two nodes would have different values of $n^{[\cdot]}_{\mathrm{root}}$). Figure 6.1 shows a sample instance where tree repair occurs during the execution of the CM ALGORITHM. The

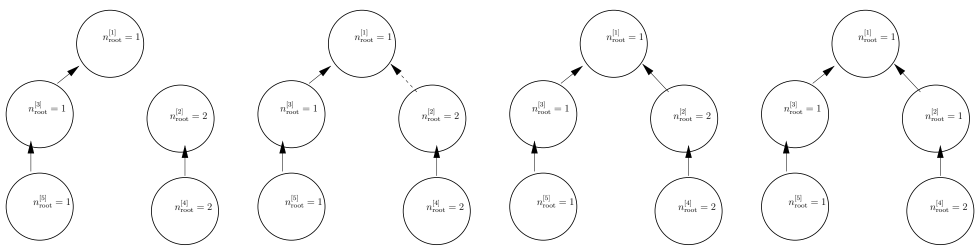


FIG. 6.1. *Illustration of tree repair during the execution of the* CM ALGORITHM. *Two components with different root ids join and eventually decide to merge into a single component.*

above analysis is not valid if the underlying graph does not remain connected. Disconnection in the underlying graph can occur as a result of changes in graph topology induced by the motion of the robots. We deal with this more complex situation next.

6.2. Repair properties under dynamic graph conditions. Here, we study the repair properties of the CM ALGORITHM when the connectivity of the underlying graph is evolving. We start by introducing the notion of restricted graph.

DEFINITION 6.4. *The* restricted graph on k, *denoted $G_{restr(k)}$, is the graph consisting of all nodes i having $n^{[i]}_{root} \leq k$ and edges (i,j), where either $j = \mathfrak{p}^{[i]}_{curr}$, $i = \mathfrak{p}^{[j]}_{curr}$, or $n^{[i]}_{root} \neq n^{[j]}_{root}$.*

The next result is the crux of our connectivity argument for partially disconnected constraint trees, as it is the basis for linking the repair properties of CM ALGORITHM to the guarantees provided by any *motion* CM-*compatible* algorithm. The proof, along with some auxiliary results, is given in Appendix A.

PROPOSITION 6.5. *Given a network which initially satisfies the property that $n^{[\cdot]}_{root}$ is monotonically non-increasing along edges from child to parent. Then, the following property is an invariant of the* CM ALGORITHM *when combined with any* motion CM-compatible *algorithm: for any two robots i and j having $n^{[i]}_{root} = n^{[j]}_{root} = k$, there is a path between i and j in $G_{restr(k)}$.*

Next, we use Proposition 6.5 to show that the algorithm maintains the connectivity of any network which starts with $G_{\mathrm{restr}(k)}$ connected for $k \in \{0,\ldots,n\}$, provided $n^{[\cdot]}_{\mathrm{root}}$ is initially monotonically non-increasing along links from child to parent.

THEOREM 6.6. *Consider the composition of* CM ALGORITHM *with a* motion CM-compatible *algorithm. Assume the network starts in a configuration where*

(i) the restricted graph on n *is connected,*

(ii) any two agents, $i, j \in \mathbb{Z}_n$, *having* $n_{root}^{[i]} = n_{root}^{[j]} = k$ *are connected in* $G_{restr(k)}$,

(iii) along each path from child node to parent node, the value of $n_{root}^{[\cdot]}$ *is monotonically non-increasing.*

Then, the network remains connected for all time.

Proof. Consider the links of the graph between iterations $t-1$ and t. Each link in the graph is either between a pair of nodes with the same value of $n_{\text{root}}^{[\cdot]}$ or between a pair of nodes with different values of $n_{\text{root}}^{[\cdot]}$. By Proposition 6.5, any pair of nodes having $n_{\text{root}}^{[\cdot]} = k$ are connected in $G_{\text{restr}(k)}$ (and thus are connected in $G_{\text{restr}(n)}$) at all times for any such starting configuration. Any pair of nodes, i, j, having $n_{\text{root}}^{[i]} \neq n_{\text{root}}^{[j]}$ will be preserved during the motion phase (by properties of *motion* CM-*compatible*). During the computation phase, either $n_{\text{root}}^{[i]}$ and $n_{\text{root}}^{[j]}$ will become the same, which guarantees a path exists between the two in $G_{\text{restr}(n_{\text{root}}^{[i]})}$ and thus in $G_{\text{restr}(n)}$, or $n_{\text{root}}^{[i]}$ and $n_{\text{root}}^{[j]}$ will remain different, in which case the edge will remain in $G_{\text{restr}(n)}$. Since any link (i, j) either remains in $G_{\text{restr}(n)}$ or disappears, but provably maintains a path between i and j in $G_{\text{restr}(n)}$, any nodes starting connected in $G_{\text{restr}(n)}$ remains so. □

Theorem 6.6 shows that a wide class of initial configurations result in the network remaining connected for all time. Showing that the constraints guaranteed by the notion of *motion* CM-*compatible* algorithm achieve $G_{\text{restr}(\cdot)}$-connectivity is not enough to show that failed links are repaired along the execution. We now show that connectivity of the underlying graph holds under a more flexible set of conditions than those identified in Theorem 6.6.

First, let us show that the constraints imposed by the notion of *motion* CM-*compatible* algorithm suffice to guarantee connectivity in the case where we use the repair properties of CM ALGORITHM to build our initial spanning tree from the default start state of the algorithm.

COROLLARY 6.7. *When* CM ALGORITHM *is coupled with a* motion CM-compatible *algorithm the following holds: if the network starts in a state where every node* i *has* $\mathfrak{p}_{curr}^{[i]} = n_{root}^{[i]} = i$ *(a configuration which satisfies the conditions of Theorem 6.6) and the communication graph is connected, the swarm will stay connected at all time.*

As a prelude to the main result, we show that n iterationsare sufficient time to satisfy the conditions of Theorem 6.6.

LEMMA 6.8. *Use* n_k *to denote the number of iterationsit takes for each node,* i, *having* $n_{root}^{[i]} = k$ *to be connected to* k *in* $G_{restr(k)}$ *given that the network starts with a connected underlying graph having* $n_{root}^{[i]} \geq n_{root}^{[\mathfrak{p}_{curr}^{[i]}]}$ *holds for each agent* i.. *This number obeys the following relationship :* $n_k \leq n - n_{k-1}$ *(i.e., within at most* $n - n_{k-1}$ *iterations, each node,* i, *having* $n_{root}^{[i]} = k$ *will be connected to* k *in* $G_{restr(k)}$, *leaving at most* $n - n_k$ *nodes with* $n_{root}^{[\cdot]} > k$*).*

Proof. From Lemma A.1 note that $n_{\text{root}}^{[i]} \geq n_{\text{root}}^{[\mathfrak{p}_{\text{curr}}^{[i]}]}$ holds for each agent i. We will proceed by induction on k. As in Proposition 6.5, our base case, with $k = 0$ follows from Corollary 5.3 of Theorem 5.2. Assume our hypothesis holds for all $\kappa < k$. So long as there are nodes with $n_{\text{root}}^{[\cdot]} = k$ not connected to k in $G_{\text{restr}(k)}$, at least one such node must have a parent with $n_{\text{root}}^{[\cdot]} < k$ (any such node must have a parent other than itself, otherwise it would have UID equal to k). So for each round during which there are such nodes, one such node must permanently get $n_{\mathrm{root}}^{[\cdot]} < k$. No node having $n_{\mathrm{root}}^{[\cdot]} = k$ which is connected to k in $G_{\mathrm{restr}(k)}$ after round $n(k-1)$ ever becomes disconnected because the only way for edges in $G_{\mathrm{restr}(k)}$ to become severed are

- Through re-attach: however, this guarantees the child gets a new root. Unless the parent gets a new root as well, the child remains attached to the parent.
- An edge between i and j in $G_{\mathrm{restr}(k)}$ disappears when i and j get the same value of $n_{\mathrm{root}}^{[\cdot]}$, or i detaches from its old parent, j, and both i and j get the same value of $n_{\mathrm{root}}^{[\cdot]}$. Since i and j were already in $G_{\mathrm{restr}(k)}$, they must get new $n_{\mathrm{root}}^{[\cdot]}$ values less than k, but by induction, must therefore be connected in $G_{\mathrm{restr}(n_{\mathrm{root}}^{[i]})}$.

Since there are at most $n - n_{k-1}$ nodes which could possibly have $n_{\mathrm{root}}^{[\cdot]} > k-1$, after at most $n - n_{k-1}$ more iterations, no nodes having $n_{\mathrm{root}}^{[\cdot]} = k$ remain unless they are connected to k in $G_{\mathrm{restr}(k)}$. For each iterationbetween round n_{k-1} and iterationn_k, where n_k is the minimum time at which each node having $n_{\mathrm{root}}^{[\cdot]} = k$ is connected to k in $G_{\mathrm{restr}(k)}$, at least one node per iterationgoes from $n_{\mathrm{root}}^{[\cdot]} = k$ to $n_{\mathrm{root}}^{[\cdot]} < k$, leaving at most $n - n_k$ nodes with $n_{\mathrm{root}}^{[\cdot]} > k$. ∎

Note that the above result implies $0 \leq n_k \leq n$ for each n_k. Finally, the next result shows that after $2n$ iterationswith no underlying hardware failures, once the communication graph becomes connected, it will remain connected for all time.

THEOREM 6.9. *If the underlying graph starts in any initial state (even if the underlying graph is disconnected), and the evolution of the network follows the constraints imposed by the notion of* motion CM-compatible *algorithm for* $2n$ *iterations, then, if the underlying graph becomes connected again, it will stay connected for all time, thus building a connected constraint tree.*

Proof. After n iterations, the value of $n_{\mathrm{root}}^{[\cdot]}$ along links from children to parents is monotonically non-increasing by Lemma A.2. By Lemma 6.8, another n iterationssuffices to satisfy the conditions of Theorem 6.6. Hence, the result follows. ∎

7. Reachability properties.

Here we examine the reachability properties of the CM ALGORITHM. Roughly speaking, by reachability we mean that the algorithm is capable of switching between any two given trees. Without this property, the preservation of the connectivity of the constraint tree might lead the network to get stuck at some undesirable configuration.

7.1. Reachability notions.

We begin by defining what it means for one tree to be reachable from another.

DEFINITION 7.1 (Reachable trees). *A constraint tree T_2 is reachable from a constraint tree T_1 with a sequence of underlying graphs $\{\mathcal{G}(t)\}_{t\in\mathbb{N}}$ under* CM ALGORITHM *coupled with a* depth CM-compatible *algorithm if the following conditions hold for any initial allocation of $dp_{est}^{[i]}, i \in \mathbb{Z}_n$ satisfying $dp_{est}^{[0]} = 0$, $dp_{est}^{[i]} \geq dp_{est}^{[\mathfrak{p}_{curr}^{[i]}]}$ for $i \in \mathbb{Z}_n$,*

- *i is allowed to set $\mathfrak{p}_{curr}^{[i]} = j$ or $\mathfrak{p}_{next}^{[i]} = j$ at round t only if $(i,j) \in \mathcal{E}(\mathcal{G}(t))$,*
- *T_1 and T_2 are subgraphs of each $\mathcal{G}(t)$,*
- *there exists a set of link preferences (values of $R_{pref}^{[i]}, i \in \mathbb{Z}_n$) such that the constraint tree will eventually settle on T_2 if it starts in T_1,*

Next, we define what it means for an algorithm to satisfy the reachability property.

DEFINITION 7.2. *The* CM ALGORITHM *combined with a* depth CM-compatible *algorithm satisfies the* reachability property *if, for every pair of trees T_1 and T_2,* *and any sequence of graphs $\mathcal{G}(t)$, having T_1 and T_2 subgraphs of $\mathcal{G}(t)$ for all t, T_2 is reachable from T_1 with underlying graphs $\{\mathcal{G}(t)\}_{t\in\mathbb{N}}$.*

Therefore, an algorithm which satisfies the reachability property can drive the constraint tree to any desired tree given a suitable preference function. As we show next, Null Depth Increment Algorithm does not satisfy this property.

Theorem 7.3. *The* CM Algorithm *coupled with* Null Depth Increment Algorithm *does not have the reachability property.*

Proof. Let G be the cycle graph on n vertices, with $(i,(i+1) \bmod n) \in \mathcal{E}(G)$ for every $i \in \mathbb{Z}_n$. Let T_1 have $\mathfrak{p}_{\text{curr}}^{[n-1]} = 0$ and $\mathfrak{p}_{\text{curr}}^{[i]} = i-1$ for all i other than 0 and $n-1$. Let T_2 have $\mathfrak{p}_{\text{curr}}^{[1]} = 0$ and $\mathfrak{p}_{\text{curr}}^{[i]} = (i+1) \bmod n$ for all i other than 0 and 1. Let the initial depth estimates, $\text{dp}_{\text{est}}^{[i]}$, be equal to the exact depth of i in the tree T_1. Node $n-1$ cannot attach to anything other than 0 until node $n-2$ has a depth $\text{dp}_{\text{est}}^{[n-2]} \leq 1$. But the only nodes $n-2$ can attach to are $n-3$ and $n-1$. If it attaches to $n-1$ it will get a depth estimate $\text{dp}_{\text{est}}^{[n-2]} = 2$. Since $n-3$ cannot attach to the root, it will always have a depth estimate $\text{dp}_{\text{est}}^{[n-3]} \geq 2$, so if $n-2$ attaches to $n-3$ it will have $\text{dp}_{\text{est}}^{[n-2]} \geq 3 > 1$ thus preventing node $n-1$ from ever attaching to it. ∎

The negative result of Theorem 7.3 motivates the design of a new depth-compatible algorithm whose combination with CM Algorithm satisfies the reachability property. This is what we do next.

7.2. Cycle-Detecting Depth Increment Algorithm. Let us we introduce the Cycle-Detecting Depth Increment Algorithm. Unlike Null Depth Increment Algorithm, this algorithm allows CM Algorithm to follow the depth update rule $\text{dp}_{\text{est}}^{[\text{id}]} \leftarrow \min(\text{dp}_{\text{est}}^{[\text{id}]} + 1, n_{\max})$, potentially allowing some robots to connect to other robots with initially larger depth estimates. Informally, this algorithm performs two operations, described as follows.

> *[Informal description:]* Each robot stores a "start number", a "number of descendants" and a "mapping from child UID to child start number." At each round, in addition to the tree constraint information and its number of descendants, each node sends the following information to each neighbor. If the neighbor is a child, it sends the corresponding entry in its mapping, or, if the child is not in the mapping, it sends its own start number. If the neighbor is not a child, it sends its own start number. With the messages received, each node updates its numbers as follows. Its number of descendants is the sum of the number of descendants information received from each child, plus one (for itself). Its start number is the one received from its parent. For each child it receives a message from, it adds an entry to its map that is indexed by that child's UID and has a value of the sum, over all children with lesser UID, of the number of descendants of those children, plus one plus its own start number.

The formal description of Cycle-Detecting Depth Increment Algorithm as an input-output control and communication law consists of the sets

(i) $L = (\mathbb{Z}_n \cup \{\texttt{null}\}) \times \mathbb{Z}$;
(ii) $W = (\mathbb{Z}_n \cup \{\texttt{null}\}) \times \mathbb{Z}_2 \times \mathbb{Z} \times \mathbb{Z} \times \mathfrak{F}(\mathbb{Z}_n;\mathbb{Z})$, are sets of values of logic variables $w^{[i]} = (n_{\text{incr-sgnl}}^{[i]}, \mathbb{I}_{\text{incr-dep}}^{[i]}, n_{\text{start}}^{[i]}, n_{\text{num-desc}}^{[i]}, f_{\text{start}}^{[i]})$, $i \in \mathbb{Z}_n$, where $n_{\text{incr-sgnl}}^{[i]}$ is used to break cycles, while $n_{\text{start}}^{[i]}$, $n_{\text{num-desc}}^{[i]}$ and $f_{\text{start}}^{[i]}$ are used to detect descendants;

function $\mathrm{stf}_{\mathrm{W}}(n^{[\mathrm{id}]}_{\text{dep-targ}}, (n^{[\mathrm{id}]}_{\text{incr-sgnl}}, \mathbb{I}^{[\mathrm{id}]}_{\text{incr-dep}}, n^{[\mathrm{id}]}_{\text{start}}, n^{[\mathrm{id}]}_{\text{num-desc}}, f^{[\mathrm{id}]}_{\text{start}}), l)$

$n^{[\mathrm{id}]}_{\text{num-desc}} \leftarrow \sum_{j \in \{j \in \mathcal{N}(\mathrm{id}) \mid \mathfrak{p}^{[j]}_{\text{curr}} = i\}} (1 + n^{[j]}_{\text{num-desc}})$

$n^{[\mathrm{id}]}_{\text{start}} \leftarrow f^{[\mathfrak{p}^{[\mathrm{id}]}_{\text{curr}}]}_{\text{start}}(\mathrm{id})$ /**From message transmitted from* $\mathfrak{p}^{[id]}_{curr}$*/

$f^{[\mathrm{id}]}_{\text{start}}(j) \leftarrow n^{[\mathrm{id}]}_{\text{start}} + \sum_{\{k \in S^{[\mathrm{id}]}_{\text{desc}} \mid k<j\}} n^{[k]}_{\text{num-desc}}$ for each $j \in \mathcal{N}(\mathrm{id})$ /*$(n^{[id]}_{start}, n^{[id]}_{num\text{-}desc})$ *used in* $\mathrm{stf}_{\text{out}}$*/

Set $\mathbb{I}^{[\mathrm{id}]}_{\text{incr-dep}} \leftarrow$ true

if $n^{[\mathrm{id}]}_{\text{dep-targ}} \neq$ `null` and $n^{[\mathrm{id}]}_{\text{incr-sgnl}} =$ `null` **then**

  Set $n^{[\mathrm{id}]}_{\text{incr-sgnl}} \leftarrow \mathrm{id}$ /**Signal originates**/

if $n^{[\mathfrak{p}^{[\mathrm{id}]}_{\text{curr}}]}_{\text{incr-sgnl}} \neq$ `null` **then**

  Set $n^{[\mathrm{id}]}_{\text{incr-sgnl}} \leftarrow n^{[\mathfrak{p}^{[\mathrm{id}]}_{\text{curr}}]}_{\text{incr-sgnl}}$; Set $\mathbb{I}^{[\mathrm{id}]}_{\text{incr-dep}} \leftarrow$ false /**Signal propagates**/

else if $n^{[\mathrm{id}]}_{\text{dep-targ}} \neq$ `null` and $n^{[n^{[\mathrm{id}]}_{\text{dep-targ}}]}_{\text{incr-sgnl}} \neq$ `null` **then**

  if $n^{[n^{[\mathrm{id}]}_{\text{dep-targ}}]}_{\text{incr-sgnl}} < n^{[\mathrm{id}]}_{\text{incr-sgnl}}$ or $n^{[\mathrm{id}]}_{\text{incr-sgnl}} =$ `null` **then**

    Set $n^{[\mathrm{id}]}_{\text{incr-sgnl}} \leftarrow n^{[n^{[\mathrm{id}]}_{\text{dep-targ}}]}_{\text{incr-sgnl}}$ /**Signal propagates**/

  else if $n^{[n^{[\mathrm{id}]}_{\text{dep-targ}}]}_{\text{incr-sgnl}} = \mathrm{id}$ **then**

    Set $\mathbb{I}^{[\mathrm{id}]}_{\text{incr-dep}} \leftarrow$ false /**Cycle detected**/

return $(n^{[\mathrm{id}]}_{\text{incr-sgnl}}, \mathbb{I}^{[\mathrm{id}]}_{\text{incr-dep}}, n^{[\mathrm{id}]}_{\text{start}}, n^{[\mathrm{id}]}_{\text{num-desc}}, f^{[\mathrm{id}]}_{\text{start}})$

TABLE 7.1
$\mathrm{stf}_{\mathrm{W}}$ *for* CYCLE-DETECTING DEPTH INCREMENT ALGORITHM.

(iii) $W^{[i]}_0 = \{(\texttt{null}, \text{false}, 0, 0, f^{[i]}_{\text{start}})\} \subset W$, with $f^{[i]}_{\text{start}} \equiv 0$;
(iv) $W_{\text{in}} = \mathbb{Z}_n \cup \{\texttt{null}\}$, $w^{[i]}_{\text{in}} = n^{[i]}_{\text{dep-targ}}$, $i \in \mathbb{Z}_n$;
(v) $W^{[i]}_{\text{in},0} = \{\texttt{null}\}$;
(vi) $W_{\text{out}} = \mathbb{Z}_2 \times (\mathbb{Z}_n \cup \{\texttt{null}\}) \times \mathfrak{F}(\mathbb{Z}_n; \{\text{true}, \text{false}\})$, are sets of values of output logic variables $w^{[i]}_{\text{out}} = (\mathbb{I}^{[i]}_{\text{incr-dep}}, n^{[i]}_{\text{incr-sgnl}}, f^{[i]}_{\text{allow}})$, $i \in \mathbb{Z}_n$;

and of the maps:

(i) $\mathrm{msg}(x, w, w_{\text{in}}, j) = \begin{cases} (n^{[\mathrm{id}]}_{\text{incr-sgnl}}, f^{[\mathrm{id}]}_{\text{start}}(j)) & \mathfrak{p}^{[j]}_{\text{curr}} = \mathrm{id} \\ (n^{[\mathrm{id}]}_{\text{incr-sgnl}}, n^{[\mathrm{id}]}_{\text{num-desc}}) & \mathfrak{p}^{[j]}_{\text{curr}} \neq \mathrm{id} \end{cases}$; Each robot, id sends $n^{[\mathrm{id}]}_{\text{incr-sgnl}}$ to each of its neighbors, and, to each neighbor j, sends either $f^{[\mathrm{id}]}_{\text{start}}(j)$ to j if $\mathfrak{p}^{[j]}_{\text{curr}} = \mathrm{id}$ or sends $n^{[\mathrm{id}]}_{\text{num-desc}}$ to j otherwise, with the intent that $n^{[j]}_{\text{start}}$ will be set to $f^{[\mathrm{id}]}_{\text{start}}(j)$;
(ii) $\mathrm{stf}_{\mathrm{W}}$ as described in Table 7.1;
(iii) $\mathrm{stf}_{\text{out}}(w_{\text{in}}, w, l) = (\mathbb{I}^{[\mathrm{id}]}_{\text{incr-dep}}, n^{[i]}_{\text{incr-sgnl}}, f^{[\mathrm{id}]}_{\text{allow}})$, where $f^{[\mathrm{id}]}_{\text{allow}}(j) = ([n^{[\mathrm{id}]}_{\text{start}}, n^{[\mathrm{id}]}_{\text{start}} + n^{[\mathrm{id}]}_{\text{num-desc}}] \cap [n^{[j]}_{\text{start}}, n^{[j]}_{\text{start}} + n^{[j]}_{\text{num-desc}}] = \texttt{null})$ if $j \in \mathcal{N}(\mathrm{id})$ and $f^{[\mathrm{id}]}_{\text{allow}}(j) = \text{false}$ otherwise;
(iv) $\mathrm{ctl}(x_{t_\ell}, x, w, w_{\text{in}}) = 0$.

We note that the variables $n^{[i]}_{\text{num-desc}}$ and $n^{[i]}_{\text{start}}$ of the processor state can be used to correctly answer queries of the form "is i a descendant of j?", see Lemma B.1.

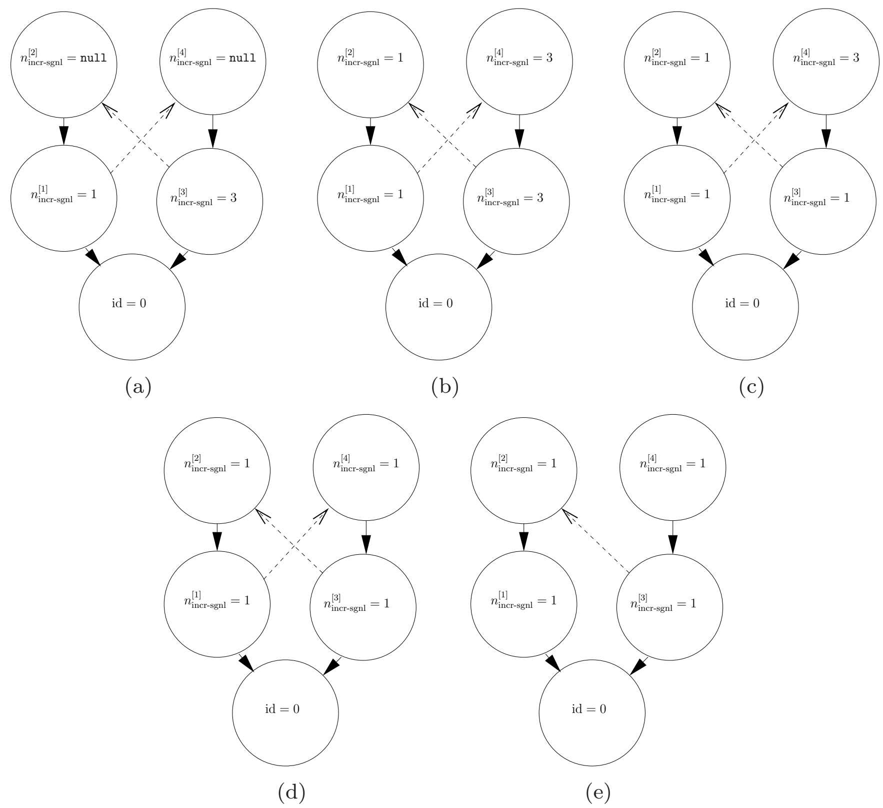


FIG. 7.1. *From (a) to (e), propagation of* $n^{[id]}_{incr\text{-}sgnl}$ *during* CYCLE-DETECTING DEPTH INCREMENT ALGORITHM. *A solid arrow indicates a link from id to* $\mathfrak{p}^{[id]}_{curr}$ *while a dashed arrow indicates a link from id to* $n^{[id]}_{dep\text{-}targ}$*. The agents with* $id = 1$ *and* $id = 3$ *want to attach to the agents with* $id = 4$ *and* $id = 2$*, respectively. This induces a cycle in the graph of desired re-arrangements, which is detected, and averted.*

A sample execution of CYCLE-DETECTING DEPTH INCREMENT ALGORITHM is illustrated in Figure 7.1.

7.3. Reachability analysis. Here, we establish the reachability properties of the CM ALGORITHM coupled with CYCLE-DETECTING DEPTH INCREMENT ALGORITHM. Given a target tree T_2, we give the network the simplest set of preferences that yield T_2 as the most desirable tree. To show that this set of preferences yields T_2 given sufficient time, we classify the edges of T_2 into (i) those which are in the current constraint tree, (ii) those which will be in the current constraint tree within a finite amount of time, and (iii) those which cannot currently be added to the current constraint tree. The key observation is that the existence of edges in case (iii) requires the existence of edges in case (ii). This observation is a consequence of the following two results, whose proofs, together with some auxiliary results, are given in Appendix B.

The first result discusses a set of operations that are known to break the tree, see Figure 3.1 for an illustration.

LEMMA 7.4. *Let T a tree. Any sequence of operations that replaces edges of T*

with edges from nodes back to their descendants in T yields a graph which is not connected.

The next result states that, if any re-arrangement other than those discussed in Lemma 7.4 is available, a re-arrangement will eventually happen.

PROPOSITION 7.5. *If the underlying graph and the constraint tree do not change for* $2\,\mathrm{depth}(T)$ *iterations, and some node desires to attach to a node other than its descendants, one such node will eventually do so (causing the constraint tree to change).*

We are now ready to state the reachability properties of our algorithm.

THEOREM 7.6. *The* CM ALGORITHM *coupled with* CYCLE-DETECTING DEPTH INCREMENT ALGORITHM *satisfies the reachability property.*

Proof. Let T_1 and T_2 be two trees of the underlying graph G. For each robot i, let $R^{[i]}_{\mathrm{pref}}$ be defined as follows: $(p^{[i]}_2, p^{[i]}_1) \in R^{[i]}_{\mathrm{pref}}$ and $(p^{[i]}_1, j) \in R^{[i]}_{\mathrm{pref}}$ for all $j \notin \{p^{[i]}_2, p^{[i]}_1\}$, where $p^{[i]}_1$ and $p^{[i]}_2$ denote the parents of i under T_1 and T_2, respectively. No node will ever attach to a parent other than its parent in one of $\{T_1, T_2\}$ (since it starts attached to its T_1 parent, only prefers its T_2 parent over that, and the graph does not change). At all times, the edges (i, j) of T_2 ($j = \mathfrak{p}^{[i]}_{\mathrm{curr}}$ in T_2) belong to one of the following categories

(i) $j = \mathfrak{p}^{[i]}_{\mathrm{curr}}$ in the current tree,
(ii) j is not a descendant of i in the current tree,
(iii) j is a descendant of i in the current tree (i cannot attach to j).

If T_2 consists of edges only in case (i), then the result follows. If this is not the case, then the edges not in case (i) cannot all be in case (iii). This follows from Lemma 7.4, since in such a case, the tree T_2 would be achievable from T_1 by a sequence of moves which re-attach nodes to their descendants, and thus T_2 would be a disconnected graph with at least one cycle. So if the edges of T_2 do not all fall into case (i), then there must be at least one edge (i, j), where j is not a descendant of i. By Proposition 7.5 one such edge will be added to the current tree in finite time. This will stop happening when each edge in the current constraint tree is also in T_2. ☐

8. Simulations. Here we illustrate the performance of the CONNECTIVITY MAINTENANCE ALGORITHM in several simulations. We combine the algorithm with CYCLE-DETECTING DEPTH INCREMENT ALGORITHM and the deployment algorithm presented in [8]. We consider a network of n agents moving in $\mathbb{R}^2$ according to

$$\dot{x}^{[i]} = u^{[i]},$$

where $\|u^{[i]}\| \leq v_{\mathrm{speed}}$ is the maximum velocity of the robots. The proximity graph of the robotic network is the r-disk proximity graph. The deployment algorithm assumes that each robot has a sensor coverage disk – i.e., the sensor we are interested in on each robot covers a disk of radius $r_{\mathrm{sns}} \leq r/2$ centered about the robot position. The algorithm moves the robots to maximize sensor coverage of a "region of interest" represented by a density function $\rho : \mathbb{R}^2 \to \mathbb{R}$. In particular, it maximizes the integral, over the union of all sensor coverage disks, of the density function.

Formally, the deployment algorithm is a motion-compatible algorithm that can be described by the sets

(i) $L = \mathbb{Z}_n \times \mathbb{R}^2$;
(ii) $W = \mathbb{R}^2$ set of values for the variable $w^{[i]}$ that contains the "target" of agent i's motion, $P^{[i]}_{\mathrm{targ}}$, $i \in \mathbb{Z}_n$;
(iii) $W^{[i]}_0 = W$, $i \in \mathbb{Z}_n$;

(iv) $W_{\text{in}} = \mathbb{Z}_n \times \mathbb{Z}_n \times \mathbb{F}(\mathbb{Z}_n) \times \mathbb{F}(\mathbb{Z}_n)$, $w_{\text{in}}^{[i]} = (\mathfrak{p}_{\text{curr}}^{[i]}, \mathfrak{p}_{\text{next}}^{[i]}, S_{\text{constraints}}^{[i]}, S_{\text{children}}^{[i]})$, $i \in \mathbb{Z}_n$;

(v) $W_{\text{out}} = R(\mathbb{Z}_n, \mathbb{Z}_n)$, $w_{\text{out}}^{[i]} = R_{\text{pref}}^{[i]}$ defined as follows. For agent $i \in \mathbb{Z}_n$,

$$(j,k) \in R_{\text{pref}}^{[i]} \text{ for } j,k \neq i \text{ if } n_{\text{root}}^{[i]} = n_{\text{root}}^{[j]} = n_{\text{root}}^{[k]} \text{ and } \|x^{[j]} - x^{[i]}\| < \|x^{[k]} - x^{[i]}\|;$$

(vi) $W_{\text{in},0}^{[i]} = \{(i, \mathcal{N}(i), \texttt{null})\}$;

and the maps:

(i) $\text{msg}(x^{[i]}, w^{[i]}, w_{\text{in}}^{[i]}, i) = (i, x^{[i]})$;

(ii) $\text{stf}(w^{[i]}, w_{\text{in}}^{[i]}, l) = P_{\text{targ}}^{[i]} \leftarrow \text{argmin}_{p \in D_C^{[i]}}(\|p - \texttt{cm}^{[i]}\|)$, where $\texttt{cm}^{[i]} = \frac{\int_D q\rho}{\int_D \rho}$ and

$$D^{[i]} = \{p \in \mathbb{R}^2 \mid \|p - x^{[i]}\| \leq r_{\text{sns}} \wedge \|p - x^{[i]}\| \leq \|p - x^{[j]}\| \; \forall j \in \mathcal{N}(i)\},$$

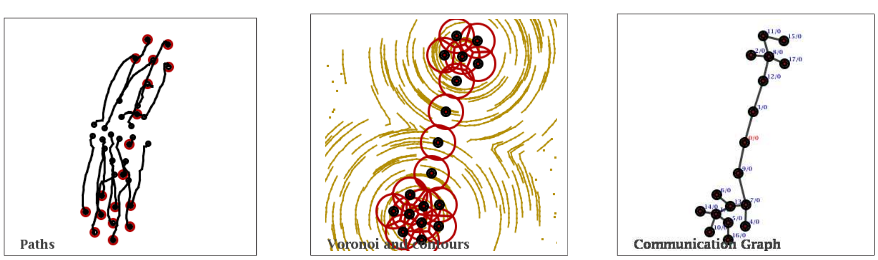
$$D_C^{[i]} = \cap_{j \in \{\mathfrak{p}_{\text{curr}}^{[i]}\} \cap S_{\text{constraints}}^{[i]} \cap S_{\text{children}}^{[i]} \{\mathfrak{p}_{\text{next}}^{[i]}\}} B\Big(\frac{x^{[j]} + x^{[i]}}{2}, \frac{r}{2}\Big);$$

(iii) $\text{ctl}(x^{[i]}, x^{[i]}(t_{\text{last}}), w^{[i]}, w_{\text{in}}^{[i]}) = v_{\text{speed}}\Big(\frac{P_{\text{targ}}^{[i]} - x^{[i]}}{\|P_{\text{targ}}^{[i]} - x^{[i]}\|}\Big)$.

Links of the constraint tree are preserved adapting the procedure described in [1]. To preserve a link between two robots, we constrain the motion of the two robots to a circle of radius $\frac{r}{2}$ centered at the midpoint of the line between their positions. Because each robot has a "target" it moves towards, we can find the closest point to the target in the intersection of the circles generated by the constraint edges (an extra constraint circle is added, centered about the position of the robot making the motion decision, with a radius equal to its maximum travel range between communication rounds). We do this by considering as candidate points each intersection between the boundaries of each pair of constraint circles, and each closest point between a given constraint circle and the target. By filtering out those candidate points which are outside one or more constraint circles, and picking the remaining point with the minimum distance to the target, we find the closest feasible point to the target. Each robot is then instructed to move towards its closest feasible point instead of its original target.

The algorithm resulting from the combination of the deployment algorithm with the Connectivity Maintenance Algorithm is executed in our Java simulation platform [30]. This platform has been developed to provide a software implementation of the modeling framework introduced in [23].

Figure 8.1 shows a sample execution of the algorithm.



(a) (b) (c)

Fig. 8.1. *Execution of* Connectivity Maintenance Algorithm, *showing (a) the paths taken by the robots, (b) a contour plot of the density field and the sensor coverage regions of the robots, (c) the final network constraint tree.*

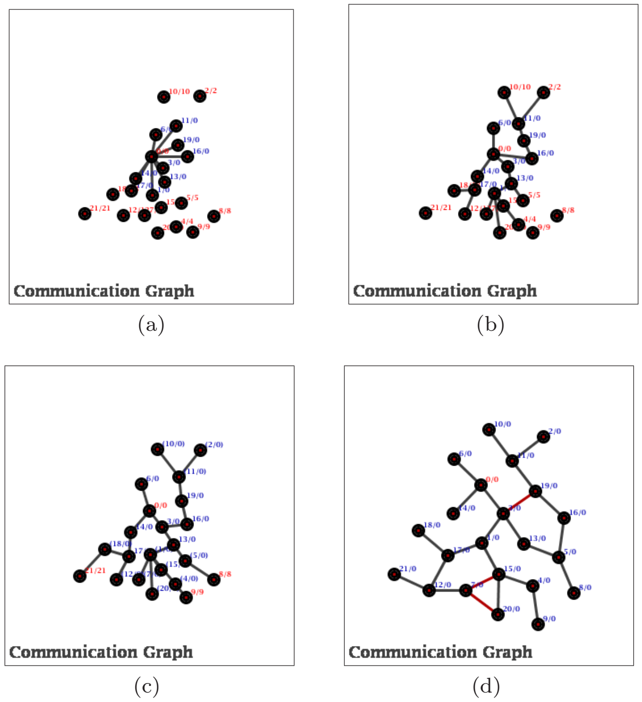
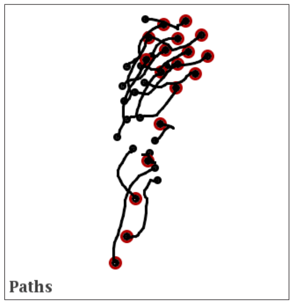


Fig. 8.2. *From left to right, top to bottom, progress of repair starting with an initially disconnected constraint tree. Agents are labeled by "agent id/root id": those in red have not yet completed repair. Edges in red are of the form* $(id, \mathfrak{p}_{next}^{[id]})$ *and denote a proposed new edge. Edges in gray are of the form* $(id, \mathfrak{p}_{curr}^{[id]})$ *and denote current edges of the constraint tree.*

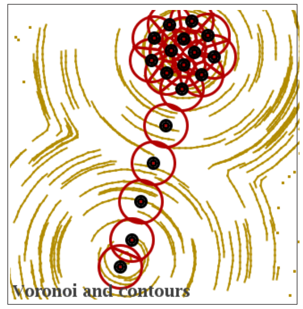
Figure 8.2 shows the evolution of the algorithm repairing an initially disconnected constraint tree.

For comparison, Figure 8.3 shows the evolution of the deployment algorithm, with and without coupling with the Connectivity Maintenance Algorithm, starting from the same initial condition. As it may be expected, the deployment algorithm by itself leads the network to a final configuration where the group is disconnected. However, when coupled with the Connectivity Maintenance Algorithm, a minimal evolving set of edges are maintained in order to guarantee group connectivity.

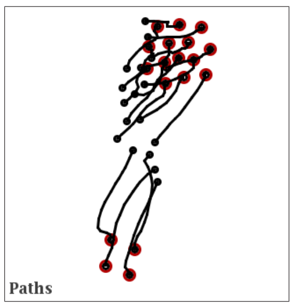
9. Conclusions and future work. We have studied connectivity maintenance of robotic networks performing spatially distributed tasks. We have proposed a distributed strategy based on maintaining the links of an evolving tree that is specified by the motion control algorithm. We have analyzed the correctness of the proposed algorithm and characterizes its repair and reachability and properties. The algorithm has been formalized in a general modeling framework for robotic networks with the purpose of facilitating easy composition with other coordination algorithms. Future work will be devoted to understanding how the resulting trees of our algorithm compare to minimum spanning trees, developing systematic ways to encode preference re-arrangements in connection with other coordination algorithms and exploring the efficacy of the proposed ideas in conjunction with other proximity graphs relevant in motion coordination, such as the visibility graph.



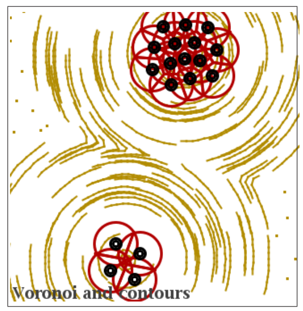
(a)



(b)



(c)



(d)

FIG. 8.3. *Comparison between executions of the deployment algorithm with –(a) agent trajectories, (b) final agent configuration with density contours and sensing disks– and without –(c) agent trajectories, (d) final agent configuration with density contours and sensing disks– coupling with the* CM ALGORITHM. *All simulations start from the same initial condition.*

REFERENCES

[1] H. ANDO, Y. OASA, I. SUZUKI, AND M. YAMASHITA, *Distributed memoryless point convergence algorithm for mobile robots with limited visibility*, IEEE Transactions on Robotics and Automation, 15 (1999), pp. 818–828.

[2] B. AWERBUCH, *Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems*, in Proceedings of the ACM Symposium on Theory of Computing, New York City, NY, May 1987, pp. 230–240.

[3] B. AWERBUCH, I. CIDON, AND S. KUTTEN, *Optimal maintenance of a spanning tree*, Journal of the Association for Computing Machinery, 55 (2008), pp. 1–45.

[4] S. BOYD, *Convex optimization of graph Laplacian eigenvalues*, in Int. Congress of Mathematicians, Madrid, Spain, Aug. 2006, pp. 1311–1319.

[5] F. BULLO, J. CORTÉS, AND S. MARTÍNEZ, *Distributed Control of Robotic Networks*, Applied Mathematics Series, Princeton University Press, 2009. Electronically available at http://coordinationbook.info.

[6] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN, *Introduction to Algorithms*, MIT Press, 2 ed., 2001.

[7] A. CORNEJO AND N. LYNCH, *Connectivity service for mobile ad-hoc networks*, in IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, Venice, Italy, 2008, pp. 292–297.

[8] J. CORTÉS, S. MARTÍNEZ, AND F. BULLO, *Spatially-distributed coverage optimization and control with limited-range interactions*, ESAIM. Control, Optimisation & Calculus of Variations, 11 (2005), pp. 691–719.

[9] ———, *Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions*, IEEE Transactions on Automatic Control, 51 (2006), pp. 1289–1298.

[10] M. C. DE GENNARO AND A. JADBABAIE, *Decentralized control of connectivity for multi-agent systems*, in IEEE Conf. on Decision and Control, San Diego, CA, Dec. 2006, pp. 3628–3633.

[11] R. DIESTEL, *Graph Theory*, vol. 173 of Graduate Texts in Mathematics, Springer, 2 ed., 2000.

[12] G. N. Frederickson, *Data structures for on-line updating of minimum spanning trees*, in Proceedings of the fifteenth annual ACM symposium on Theory of computing, New York, NY, USA, 1983, ACM, pp. 252–257.
[13] F. C. Gaertner, *A survey of self-stabilizing spanning-tree construction algorithms*, tech. rep., Ecole Polytechnique Fdrale de Lausanne, 2003. Available electronically at `http://infoscience.epfl.ch/search.py?recid=52545`.
[14] R. G. Gallager, P. A. Humblet, and P. M. Spira, *A distributed algorithm for minimum-weight spanning trees*, ACM TOPLAS, 5 (1983), pp. 66–77.
[15] A. Ganguli, J. Cortés, and F. Bullo, *Multirobot rendezvous with visibility sensors in nonconvex environments*, IEEE Transactions on Robotics, 25 (2009), pp. 340–352.
[16] J. Garay, S. Kutten, and D. Peleg, *A sub-linear time distributed algorithm for minimum-weight spanning trees*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, 1993, pp. 659–668.
[17] J. W. Jaromczyk and G. T. Toussaint, *Relative neighborhood graphs and their relatives*, Proceedings of the IEEE, 80 (1992), pp. 1502–1517.
[18] M. Ji and M. Egerstedt, *Distributed control of multiagent systems while preserving connectedness*, IEEE Transactions on Robotics, 23 (2007), pp. 693–703.
[19] M. Khan and G. Pandurangan, *A fast distributed approximation algorithm for minimum spanning trees*, Distributed Computing, 20 (2008), pp. 391–402.
[20] Y. Kim and M. Mesbahi, *On maximizing the second smallest eigenvalue of a state-dependent graph Laplacian*, IEEE Transactions on Automatic Control, 51 (2006), pp. 116–120.
[21] J. Lin, A. S. Morse, and B. D. O. Anderson, *The multi-agent rendezvous problem. Part 1: The synchronous case*, SIAM Journal on Control and Optimization, 46 (2007), pp. 2096–2119.
[22] N. A. Lynch, *Distributed Algorithms*, Morgan Kaufmann, 1997.
[23] S. Martínez, F. Bullo, J. Cortés, and E. Frazzoli, *On synchronous robotic networks - Part I: models, tasks, and complexity*, IEEE Transactions on Automatic Control, 52 (2007), pp. 2199–2213.
[24] ———, *On synchronous robotic networks - Part II: time complexity of rendezvous and deployment algorithms*, IEEE Transactions on Automatic Control, 52 (2007), pp. 2214–2226.
[25] D. Peleg, *Distributed Computing. A Locality-Sensitive Approach*, Monographs on Discrete Mathematics and Applications, SIAM, 2000.
[26] D. Peleg and V. Rubinovich, *A near tight lower bound on the time complexity of distributed minimum spanning tree construction*, SIAM Journal of Computing, 30 (2000), pp. 1427–1442.
[27] S. Poduri and G. S. Sukhatme, *Constrained coverage for mobile sensor networks*, in IEEE Int. Conf. on Robotics and Automation, New Orleans, LA, USA, May 2004, pp. 165–172.
[28] C. W. Reynolds, *Flocks, herds, and schools: A distributed behavioral model*, Computer Graphics, 21 (1987), pp. 25–34.
[29] K. Savla, G. Notarstefano, and F. Bullo, *Maintaining limited-range connectivity among second-order agents*, SIAM Journal on Control and Optimization, 48 (2009), pp. 187–205.
[30] M. D. Schuresko, *CCLsim. A simulation environment for robotic networks*, 2008. Electronically available at http://carmenere.ucsd.edu/mds/cclsim.
[31] M. D. Schuresko and J. Cortés, *Safe graph rearrangements for distributed connectivity of robotic networks*, in IEEE Conf. on Decision and Control, New Orleans, LA, 2007, pp. 4602–4607.
[32] ———, *Distributed motion constraints for algebraic connectivity of robotic networks*, Journal of Intelligent and Robotic Systems, 56 (2009), pp. 99–126.
[33] ———, *Distributed tree rearrangements for reachability and robust connectivity*, in International Conference on Hybrid Systems: Computation and Control, R. Majumdar and P. Tabuada, eds., vol. 5469 of Lecture Notes in Computer Science, New York, 2009, Springer, pp. 470–474.
[34] D. P. Spanos and R. M. Murray, *Motion planning with wireless network constraints*, in American Control Conference, Portland, OR, June 2005, pp. 87–92.
[35] P. Yang, R. A. Freeman, G. Gordon, K. M. Lynch, S. Srinivasa, and R. Sukthankar, *Decentralized estimation and control of graph connectivity for mobile sensor networks*, in American Control Conference, Seattle, WA, 2008.
[36] M. M. Zavlanos, A. Jadbabaie, and G. J. Pappas, *Flocking while preserving network connectivity*, in IEEE Conf. on Decision and Control, New Orleans, LA, 2007, pp. 2919–2924.
[37] M. M. Zavlanos and G. J. Pappas, *Controlling connectivity of dynamic graphs*, in IEEE Conf. on Decision and Control and European Control Conference, Seville, Spain, Dec. 2005, pp. 6388–6393.

[38] ———, *Potential fields for maintaining connectivity of mobile networks*, IEEE Transactions on Robotics, 23 (2007), pp. 812–816.

Appendix A. *Restricted graph on* k and its properties.

Here, we provide the proof of Proposition 6.5. Before doing so, we need to state a couple of important intermediate results. We start with an invariance property of CM Algorithm.

Lemma A.1. *If no edge of the form $(i, \mathfrak{p}^{[i]}_{curr})$ breaks, then* CM Algorithm *preserves the property that $n^{[\cdot]}_{root}$ is monotonically non-increasing along edges from child to parent. As a corollary, the value of $n^{[i]}_{root}$ for any given i is monotonically non-increasing over time.*

Proof. $n^{[i]}_{\mathrm{root}}$ can only change by taking on $n^{[\mathfrak{p}^{[i]}_{\mathrm{curr}}]}_{\mathrm{root}}$, which does not affect the invariant. The value of $n^{[\cdot]}_{\mathrm{root}}$ along the path from child to parent can change when a node re-attaches, but each node can only attach to nodes with $n^{[\cdot]}_{\mathrm{root}}$ less than or equal to its own value of $n^{[\cdot]}_{\mathrm{root}}$. $\square$

We next show that any network converges, in a finite number of rounds, to a configuration where the property required in Lemma A.1 is satisfied.

Lemma A.2. *Given any starting condition, any* motion CM-compatible *algorithm guarantees that, under the* CM *algorithm, each agent i will have the property that $n^{[i]}_{root} \geq n^{[\mathfrak{p}^{[i]}_{curr}]}_{root}$ within n iterations.*

Proof. Let $S = \{i \in \mathbb{Z}_n \mid n^{[\mathfrak{p}^{[i]}_{\mathrm{curr}}]}_{\mathrm{root}} > n^{[i]}_{\mathrm{root}}\}$ and $S_> = \{j \in \mathbb{Z}_n \mid \exists i \in S, n^{[j]}_{\mathrm{root}} \geq n^{[i]}_{\mathrm{root}}\}$. Note that, since each $i \in S$ satisfies that $n^{[i]}_{\mathrm{root}} \geq n^{[j]}_{\mathrm{root}}$ for some $j \in S$ (namely $j = i$), we deduce that $S \subset S_>$. We argue that $|S_>|$ decreases by at least one every iteration. No node, k, will increase $n^{[k]}_{\mathrm{root}}$ unless its parent connection is severed (prohibited by *motion* CM-*compatible*) or its parent had previously satisfied $n^{[\mathfrak{p}^{[k]}_{\mathrm{curr}}]}_{\mathrm{root}} > n^{[k]}_{\mathrm{root}}$, thus putting $k \in S_>$. Thus no node gets added to $S_>$. Let $l = \mathrm{argmax}_{i \in S}\{n^{[\mathfrak{p}^{[i]}_{\mathrm{curr}}]}_{\mathrm{root}}\}$ and note that $n^{[\mathfrak{p}^{[l]}_{\mathrm{curr}}]}_{\mathrm{root}} > n^{[i]}_{\mathrm{root}}$ for all $i \in S$. Since the update rule, $n^{[l]}_{\mathrm{root}} \leftarrow n^{[\mathfrak{p}^{[l]}_{\mathrm{curr}}]}_{\mathrm{root}}$ applies to l, at least one node must be removed from $S_>$ every iteration. When $|S_>| = 0$, we conclude that $S = \emptyset$. $\square$

We are now ready to give the proof of Proposition 6.5.

Proof of Proposition 6.5. We will proceed by induction on k. For $k = 0$, the result follows from Corollary 5.3. Assume the result holds for $\kappa < k$. Let the property hold for k on round $t - 1$. To show it holds for k on round t, consider the ways in which an edge can be removed from $G_{\mathrm{restr}(k)}$ on round t:

- An edge, (i, j), in $G_{\mathrm{restr}(k)}$ having $j \neq \mathfrak{p}^{[i]}_{\mathrm{curr}}$ and $i \neq \mathfrak{p}^{[j]}_{\mathrm{curr}}$ can disappear when $n^{[i]}_{\mathrm{root}}$ becomes equal to $n^{[j]}_{\mathrm{root}}$.

In this case, either $n^{[i]}_{\mathrm{root}}$ or $n^{[j]}_{\mathrm{root}}$ must have been less than k at round $t - 1$. By Lemma A.1, the value of $n^{[i]}_{\mathrm{root}}$ and $n^{[j]}_{\mathrm{root}}$ can only decrease, making each of these less than k at round t. By the induction hypothesis, there must be a path between the two at round t in $G_{\mathrm{restr}(n^{[i]}_{\mathrm{root}})}$ which is also a path in $G_{\mathrm{restr}(k)}$.

- Node i, having $\mathfrak{p}^{[i]}_{\mathrm{curr}} = h$ at round $t - 1$, can change $\mathfrak{p}^{[i]}_{\mathrm{curr}}$.
  (i) to a new node, j, having $n^{[j]}_{\mathrm{root}} < k$.

In this case, either $n^{[h]}_{\mathrm{root}} \neq n^{[i]}_{\mathrm{root}}$, leaving the two connected, or $n^{[h]}_{\mathrm{root}} = n^{[i]}_{\mathrm{root}}$, but since $n^{[i]}_{\mathrm{root}}$ is the value $n^{[j]}_{\mathrm{root}}$ had at round $t - 1$, $n^{[i]}_{\mathrm{root}}$ and $n^{[j]}_{\mathrm{root}}$ must both be less than k, leaving the two connected, by the induction hypothesis; or

(ii) to a new node, j, having $n^{[j]}_{\text{root}} = k$.

If $n^{[h]}_{\text{root}} \neq k$ at round t, i and h are still connected in $G_{\text{restr}(k)}$. Otherwise, since $n^{[h]}_{\text{root}}$ never decreases, we have $n^{[h]}_{\text{root}} = n^{[j]}_{\text{root}} = n^{[i]}_{\text{root}} = k$ at round $t-1$ meaning all three of i, j, h had paths between one another in $G_{\text{restr}(k)}$ at round $t-1$. We show that this sort of attach operation does not break the property we are trying to establish by ordering the attach and update operations in the following way. First we perform, in (an arbitrary) sequence, all parent re-attach operations from an old parent to a new parent with $n^{[\cdot]}_{\text{root}} = k$. Since this operation does not change the value of $n^{[\cdot]}_{\text{root}}$ for the node switching parents, we do not worry about the $n^{[\cdot]}_{\text{root}}$ update for this kind of event. After showing that none of these operations individually break the connectivity among agents with $n^{[\cdot]}_{\text{root}} = k$ in $G_{\text{restr}(k)}$, we then proceed with the remainder of attach and update operations in the synchronous order in which they occur in the algorithm, relying on the correctness properties already established to show these operations correct. First, to show none of the $n^{[i]}_{\text{root}} = n^{[j]}_{\text{root}} = n^{[h]}_{\text{root}} = k$ operations break connectivity we note, as shown above, that at round $t-1$ there is a path between any pair of i, j and h before the attach. After changing $\mathfrak{p}^{[i]}_{\text{curr}}$ from h to j, there is still a path from h to j in $G_{\text{restr}(k)}$, meaning there are paths between i and j (trivially) and between i and h (through path composition) after the event. Since all three affected nodes are still interconnected after the event, the connectivity of any pair of nodes in $G_{\text{restr}(k)}$ is not affected by the re-attach, and we can use the same line of reasoning for the next re-attach in our sequence. Finally we note that when a new node, i, sets $n^{[i]}_{\text{root}} = k$ for the first time, it is entering $G_{\text{restr}(k)}$ for the first time. It is doing so by attaching (or staying attached) to a node, j, which had $n^{[j]}_{\text{root}} = k$ at round $t-1$, and thus j was connected to all nodes having $n^{[\cdot]}_{\text{root}} = k$ at round $t-1$. If we order the sequence of updates again, putting these attaches and $n^{[\cdot]}_{\text{root}}$ updates before the ones we have already considered, we show that they do not break the connectivity among nodes with $n^{[\cdot]}_{\text{root}} = k$ in $G_{\text{restr}(k)}$. □

Appendix B. Properties of the evolutions under Cycle-Detecting Depth Increment Algorithm.

Here we gather some properties of the executions of the CM Algorithm combined with Cycle-Detecting Depth Increment Algorithm. These results are instrumental to establish the reachability properties satisfied by this combination. We begin by introducing several auxiliary graphs.

Lemma B.1. *If the constraint tree T remains fixed for $2\,\text{depth}(T)$ rounds, then each i satisfies*

(i) $n^{[i]}_{\textit{num-desc}}$ *is equal to the total number of descendants of i,*

(ii) $n^{[i]}_{\textit{start}} = \sum_{j <_T i} 1$,

(iii) if i is an ancestor of j, eventually $n^{[i]}_{\textit{start}} \leq n^{[j]}_{\textit{start}}$ and $n^{[i]}_{\textit{start}} + n^{[i]}_{\textit{num-desc}} \geq n^{[j]}_{\textit{start}} + n^{[j]}_{\textit{num-desc}}$ will hold, and

(iv) if i is not an ancestor of j, either $n^{[i]}_{\textit{start}} > n^{[j]}_{\textit{start}} + n^{[j]}_{\textit{num-desc}}$ or $n^{[j]}_{\textit{start}} > n^{[i]}_{\textit{start}} + n^{[i]}_{\textit{num-desc}}$ will hold.

Proof. Each node, i, has the proper value of $n^{[i]}_{\text{num-desc}}$ one round after the last of its children have the proper $n^{[\cdot]}_{\text{num-desc}}$. This takes at most $\text{depth}(T)$ rounds, thus showing *(i)*. To show *(ii)*, each node has the proper value of $n^{[i]}_{\text{start}}$ one round after its siblings each have the proper value of $n^{[\cdot]}_{\text{num-desc}}$ and its parent, $\mathfrak{p}^{[i]}_{\text{curr}}$, has the proper value of $n_{\text{start}}^{[\mathfrak{p}_{\text{curr}}^{[i]}]}$. This takes at most depth(D) additional rounds after all nodes satisfy *(i)*. Regarding fact *(iii)*, after $2\,\text{depth}(T)$ rounds, for each node $m \in \mathbb{Z}_n$, $n_{\text{num-desc}}^{[m]}$ will be the total number of descendants of m and $n_{\text{start}}^{[\cdot]}$ will be $\sum_{k<_T m} 1$. If i is not an ancestor of j, without loss of generality let $j > i$. Let k be the nearest common ancestor of i and j. Let k_i and k_j be the children of k having i and j as descendants respectively. At some point k_i will get some $n_{\text{start}}^{[k_i]}$ and k_j will get $n_{\text{start}}^{[k_j]} \geq n_{\text{start}}^{[k_i]} + n_{\text{num-desc}}^{[k_i]}$ thus showing *(iii)*. Regarding fact *(iv)*, if i is an ancestor of j, for every link, (k_1, k_2), between i and j, k_2 will get some number between $n_{\text{start}}^{[k_1]}$ and $n_{\text{start}}^{[k_1]} + n_{\text{num-desc}}^{[k_1]} - n_{\text{num-desc}}^{[k_2]}$ for $n_{\text{start}}^{[k_2]}$. Thus, at every step along the way, $n_{\text{start}}^{[k_1]} \leq n_{\text{start}}^{[k_2]} \leq n_{\text{start}}^{[k_2]} + n_{\text{num-desc}}^{[k_2]} \leq n_{\text{start}}^{[k_1]} + n_{\text{num-desc}}^{[k_1]}$. By inducting on the number of links from i, this gives *(iv)*. □

Definition B.2. *Consider the subgraphs of the underlying proximity graph:*

- *The* increase depth graph *is the graph containing all edges of the form* $(i, n_{\textit{dep-targ}}^{[i]})$ *for* $n_{\textit{dep-targ}}^{[i]} \neq \texttt{null}$ *and all edges of the form* $(i, \mathfrak{p}_{\textit{curr}}^{[i]})$ *where* $n_{\textit{incr-sgnl}}^{[i]} \neq \texttt{null}$ *and* $n_{\textit{incr-sgnl}}^{[\mathfrak{p}_{\textit{curr}}^{[i]}]} \neq \texttt{null}$;
- *The* lowest subgraph (in the increase depth graph) *consists of all edges* $(i, n_{\textit{dep-targ}}^{[i]})$ *where no ancestor,* j*, of* i *has* $n_{\textit{dep-targ}}^{[j]} \neq \texttt{null}$ *and all edges of the form* $(i, \mathfrak{p}_{\textit{curr}}^{[i]})$ *where some ancestor of* i *is also in the lowest subgraph;*
- *The* signal graph on i *is the graph containing all edges of the form* (j, k) *where* $n_{\textit{incr-sgnl}}^{[j]} = i$, $n_{\textit{incr-sgnl}}^{[k]} = i$ *and either* $k = \mathfrak{p}_{\textit{curr}}^{[j]}$ *or* $k = n_{\textit{dep-targ}}^{[j]}$.

Each of these graphs contains only those nodes adjacent to an edge in the respective graph. The increase depth graph and its lowest subgraph only change when the underlying graph, the constraint tree, or $n_{\text{dep-targ}}^{[i]}$ for some i change. The signal graph on i can change even when the aforementioned structures remain constant.

We proceed to show that whenever the set of agent preferences are such that each node in the network would prefer its parent in T_2 to its parent in the current tree, some node will change parents. We break this down by cases.

(i) The "lowest subgraph of the increase depth graph" contains no cycles.
(ii) The "lowest subgraph of the increase depth graph" contains at least one cycle.

Case (i) is treated in Lemma B.3, which describes what happens when the set of preferences do not induce a cycle in the "lowest subgraph of the increase depth graph." Case (ii) is handled in Lemma B.5.

Lemma B.3. *If there is no cycle in the "lowest subgraph in the increase depth graph" then some node* j *in this graph will attach to* $n_{\textit{dep-targ}}^{[j]}$ *within a finite number of time steps.*

Proof. If there is no such cycle, than some edge of the form $(i, n_{\text{dep-targ}}^{[i]})$ exists such that neither $n_{\text{dep-targ}}^{[i]}$ nor any of its ancestors has $n_{\text{dep-targ}}^{[\cdot]} \neq \texttt{null}$. Since i is in the lowest subgraph of the increase depth graph, each ancestor, j, of i has $n_{\text{dep-targ}}^{[j]} = \texttt{null}$. If this is the case, than $n_{\text{dep-targ}}^{[i]}$ will settle to a depth estimate, $\text{dp}_{\text{est}}^{[n_{\text{dep-targ}}^{[i]}]}$, of its actual depth in the tree (less than $n+1$). i will never receive $n_{\text{incr-sgnl}}^{[\cdot]} = i$ and will keep increasing depth until it can attach to $n_{\text{dep-targ}}^{[i]}$ (at a value of, at most, $n+1$). □

If there is a cycle in the "lowest subgraph of the increase depth graph" the signaling mechanism of Cycle-Detecting Depth Increment Algorithm will detect it, as shown in Lemma B.4.

LEMMA B.4. *If there is a cycle in the "lowest subgraph of the increase depth graph" then within a finite number of timesteps, there exists some k such that the "signal graph on k" contains that cycle.*

Proof. There are two cases in which a node can switch from $n^{[i]}_{\text{incr-sgnl}} = j$ to $n^{[i]}_{\text{incr-sgnl}} = k$. One is when its parent sends $n^{[\mathfrak{p}^{[i]}_{\text{curr}}]}_{\text{incr-sgnl}} = k$. This does not happen for any i having $(i, n^{[i]}_{\text{dep-targ}})$ in the lowest subgraph of the increase depth graph. The other is when $n^{[n^{[i]}_{\text{dep-targ}}]}_{\text{incr-sgnl}} = k$ and $k < j$. The nodes i in any cycle of lowest subgraph having $(i, n^{[i]}_{\text{dep-targ}})$ in the lowest subgraph have a unique minimum value of $n^{[i]}_{\text{incr-sgnl}}$, k. When some edge, $(i, \mathfrak{p}^{[i]}_{\text{curr}})$ in the subgraph has $n^{[\mathfrak{p}^{[i]}_{\text{curr}}]}_{\text{incr-sgnl}} = k$, in the next round, $n^{[i]}_{\text{incr-sgnl}}$ gets k. Likewise when some $(i, n^{[i]}_{\text{dep-targ}})$ in the subgraph gets $n^{[n^{[i]}_{\text{dep-targ}}]}_{\text{incr-sgnl}} = k$, in the next round, $n^{[i]}_{\text{incr-sgnl}} = k$. Since these edges describe every edge in the lowest subgraph of the increase depth graph, eventually all nodes in this cycle will have $n^{[\cdot]}_{\text{incr-sgnl}} = k$. □

This mechanism makes the proof of case (ii) above possible, as we show next.

LEMMA B.5. *If there is an edge, (i, j) in one of the cycles of the "lowest subgraph of the increase depth graph" where j is not an ancestor of i and vice versa, then within a finite number of time steps, some node, k, in this graph will attach to $n^{[k]}_{\text{dep-targ}}$.*

Proof. By Lemma B.4, every edge of this cycle will get $n^{[\cdot]}_{\text{incr-sgnl}} = k$ for some k. Node k will reset, as it receives $n^{[n^{[k]}_{\text{dep-targ}}]}_{\text{incr-sgnl}} = k$, as will every descendant trying to attach to some node dependent on this cycle. Some node which is not a descendant of k must be trying to attach to a descendant of k, and will keep increasing depth while k's ancestors stay reset. □

Finally, Lemma B.6 and Proposition 7.5 tie the two cases together.

LEMMA B.6. *If the graph and the constraint tree do not change for* $2\,\text{depth}(T)$ *rounds, either the conditions for Lemma B.5 or Lemma B.3 will hold, or no node, i, will have $n^{[i]}_{\text{dep-targ}} \neq$* `null`

Proof. By Lemma B.1, after $2\,\text{depth}(T)$ time steps, no i and j where j is a descendant of i will have $f^{[i]}_{\text{allow}}(j) = \text{true}$. CM ALGORITHM will not allow $n^{[i]}_{\text{dep-targ}}$ to be j and thus any cycle in the lowest subgraph of the increase depth graph must contain at least one of the type of edge described in Lemma B.5. □

We are now ready to provide the proof of Proposition 7.5.

Proof of Proposition 7.5. Either there is some node, i which wants to attach to a node j having $\text{dp}^{[j]}_{\text{est}} < \text{dp}^{[i]}_{\text{est}}$, or, by Lemma B.6, the lowest subgraph of the increase depth graph contains no cycles, or the lowest subgraph of the increase depth graph contains at least one cycle. The first case is trivial, the second is handled by Lemma B.3 and the third is handled by Lemma B.5. □

Proof of Lemma 7.4. The first such edge we replace will disconnect all the descendants of some node, i, from the root. Any replacement involving one of the now disconnected nodes changing its parent will not connect these nodes back to the root, since the disconnected node is attaching back to one of its ancestors. Any replacement involving a currently connected node will also not connect these nodes to the root, as the currently connected node loses its path to the root when the old edge from itself to its parent is removed. □