Castañeda *et al. Journal of the Brazilian Computer Society* (2018) 24:1
DOI 10.1186/s13173-017-0065-8

Journal of the
Brazilian Computer Society

RESEARCH Open Access

Convergence and covering on graphs for wait-free robots

Armando Castañeda[1*], Sergio Rajsbaum[1] and Matthieu Roy[2]

Abstract

The class of *robot convergence tasks* has been shown to capture fundamental aspects of fault-tolerant computability. A set of asynchronous robots that may fail by crashing, start from unknown places in some given space, and have to move towards positions close to each other. In this article, we study the case where the space is uni-dimensional, modeled as a graph G. In *graph convergence*, robots have to end up on one or two vertices of the same edge. We consider also a variant of robot convergence on graphs, *edge covering*, where additionally, it is required that not all robots end up on the same vertex. Remarkably, these two similar problems have very different computability properties, related to orthogonal fundamental issues of distributed computations: agreement and symmetry breaking. We characterize the graphs on which each of these problems is solvable, and give optimal time algorithms for the solvable cases. Although the results can be derived from known general topology theorems, the presentation serves as a self-contained introduction to the algebraic topology approach to distributed computing, and yields concrete algorithms and impossibility results.

Keywords: Robot gathering, Agreement, Symmetry breaking, Shared memory, Wait-freedom, Combinatorial topology

Introduction

The family of *robot convergence* tasks plays a fundamental role in the theory of fault-tolerant distributed computing [21]. It is used to prove the wait-free computability theorem [25] that characterizes the tasks that are wait-free solvable in a read/write shared memory environment, and is intimately related to the simplicial approximate agreement theorem of topology. Roughly speaking, asynchronous communicating robots cannot coordinate to converge to a single point because consensus is impossible in the presence of even a single crash failure [17], but robots can move towards points which are arbitrarily close to each other, using a solution to *approximate agreement* [16]. Robots can also converge in Euclidean space; see [28] for a recent treatment of a basic *multi-dimensional robot convergence* task tolerating Byzantine faults, including a discussion of applications to robots, distributed voting and optimization problems, as well as further related references. Various other applications and specific robot convergence tasks appear in, e.g., [8, 22, 23, 27, 30].

In a robot convergence problem, a collection of n robots are placed in points of a given space, $\mathcal{K}$, which can be of any shape and dimension. The robots know $\mathcal{K}$, but they do not know on which point of $\mathcal{K}$ each robot is initially placed. The goal for the robots is to move to points which are close to each other. The difficulty is that although the robots can communicate reliably with each other, and can jump from one point to any other point of $\mathcal{K}$, the robots are asynchronous and may crash. The combination of asynchrony and failures means that it is impossible to distinguish between a faulty robot that has halted and a robot subject to slow computation [5]. Thus, a robot must continue running its algorithm and decide where to move, independently of which robots it hears from at any given moment (robots do not observe each other positions directly, but only through communication). In particular, in a *solo* run, where a robot does not hear at all from other robots, the algorithm has to drive the robot to its final position based only on its initial position.

A specific robot convergence task is defined by the space $\mathcal{K}$, and rules Δ stating restrictions on the regions

*Correspondence: armando.castaneda@im.unam.mx
[1]Instituto de Matemáticas, UNAM, Ciudad Universitaria, 04510 Mexico city, Mexico
Full list of author information is available at the end of the article

Springer Open

© The Author(s). 2017 **Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

on which the robot should converge. For instance, the space $\mathcal{K}$ could be the d-dimensional Euclidean space as in [28], and robots may be required to converge on regions spanned by the convex hull of their initial positions; if all start on the same point, they should remain there, and if all start in two points, they should converge to points close to each other along the straight line connecting the two points. Another example is the *loop agreement* task [23], where there is a given loop in the space $\mathcal{K}$, and three given distinguished points v_1, v_2, v_3 on the loop. The robots are placed on any of these three points. If the robots start on the same point v_i, they should remain there, if they start on two points, they should converge on the loop segment connecting these two points. If they start on the three different points, they can converge anywhere in $\mathcal{K}$, as long as they end up being close to each other. Whether a specific robot convergence task is solvable depends on the space $\mathcal{K}$ and the convergence rules Δ. Arguably, the most basic (wait-free) unsolvable convergence task is *two-set agreement* for three processes [14], which is an instance of loop agreement where $\mathcal{K}$ is a cycle of three edges [23]. Stated using this terminology, two-set agreement for *three* processes corresponds to robots starting in any of the corresponding three vertices, and having to decide on at most two of the initial vertices.

Robot convergence problems on graphs

We are interested in studying robot convergence problems in the case the space $\mathcal{K}$ is 1-dimensional. As usual in combinatorial topology, we consider a discretization of the space, and represent it by a graph G, where two vertices are defined to be close to each other if and only if they belong to the same edge (the corresponding points can be as close as desired, by considering a subdivision of the space as fine as needed). In the *graph convergence* problem, robots may start on any of the vertices of the graph and must end up on vertices of the same edge. If they all start in close enough positions, they should stay there: if they start on vertices of an edge, they should decide vertices of this edge. Otherwise, they can decide on vertices of any edge.

We introduce a related problem, *edge covering*, where the robots have to end in positions close to each other, but not all on top of each other. Thus, while both problems require to reach a form of agreement, edge covering additionally requires symmetry breaking, as robots cannot all decide the same vertex.

Coordination problems in distributed computing can be about reaching agreement, often referred to as *colorless* problems [8] such as *consensus*, *loop agreement*, *set agreement*, *graph convergence*, or more generally *robot convergence*, or they can deal with reaching disagreement, which is usually much more difficult to analyze [19] as in *weak symmetry breaking* [11, 19, 25], *renaming* [4, 12] or *committee decision* [13].

Summary of results

The first aim of the paper is to study two basic robot convergence problems in a graph, and to expose differences between reaching agreement and symmetry breaking. We formally define and study the graph convergence and edge-covering problems. We give a full characterization of the graphs on which these problems can be solved and provide algorithms where the robots gradually move until they solve the problem. Our results are the following (summarized in Table 1):

1. Graph convergence. For the case of two robots, graph convergence can be solved iff G is connected. If the number of robots is $n \geq 3$, then graph convergence is solvable iff G is a tree.
2. Edge covering. For the case of two robots, edge covering can be solved iff G is connected and contains an odd-length cycle. If the number of robots is $n \geq 3$, then edge covering is unsolvable, whatever the graph G.

The second aim of the paper is to provide a self-contained introduction to the topological approach to distributed computing [21]. The characterization of graph convergence solvability can in principle be derived from existing theorems (e.g., Theorem 4.3.1 of the book [21]) which, roughly speaking, implies that graph convergence has a solution iff there is a continuous map from a given space to G (more details in "The topology of graph convergence" section). Our algorithms explain how such a map is constructed, and our impossibility results explain why there is no such map when G is not connected or is not acyclic. Similarly, our edge covering results can in principle be derived from the Asynchronous Computability Theorem [25] that requires additionally the continuous map to preserve identifiers of participants (because edge covering is not colorless). The topological approach to distributed computing is useful to prove time complexity results, in addition to computability results [26], and we also illustrate this aspect of the theory here. We hope our algorithms and impossibility results provide intuition

Table 1 Summary of results

	Two robots	$n \geq 3$ robots
Graph convergence	G is connected	G is a tree
	Theorem 2	Theorem 1
Edge covering	G is connected + odd-lenght cycle	Unsolvable
	Theorem 3	Lemma 6

and shed light on these topological theorems, illustrating why topological properties are so intimately related to distributed algorithms.

Related work

Distributed algorithms for robots is a very active research area (see e.g., [15] for a recent work and further references), and in particular problems about robot convergence, *gathering* at a single location, and *scattering* to different locations have been widely studied. Less work has been devoted to fault-tolerant algorithms, and mostly in the plane. Gathering algorithms for the case where at most one robot may crash, or behave in a Byzantine way, was proposed in [1], and for multiple crash failures in [10]. However, we are not aware of the use of algebraic topology techniques in the style of [21] (work about computing topological properties of a space is of a different nature, e.g., [6]). In our setting, gathering is impossible ("Impossibility results" section) because, in contrast to other settings, robots cannot observe directly the positions of other robots, they need to communicate with each other to find them. Notice that our two-robot graph convergence algorithms can be extended for any number of robots tolerating one failure using BG simulation [8].

Outline of the paper

After introducing the model of computation in "Model of computation" section, we formally define the graph convergence problem and present two round-optimal solutions to it in "The graph convergence problem" section: an algorithm that solves graph convergence on trees, for any number of processes, and an algorithm that solves graph convergence on any connected graph, for two processes. Then, in "The edge-covering problem" section, we define the edge-gathering problem and present a round-optimal solution for two processes for on any graph. As we shall see in "Impossibility results" section, these are the only graphs where these problems are solvable. A topological perspective of our results is in "A topological perspective" section, where also the optimality of our algorithms is proven.

Model of computation

We assume a standard distributed computing model (see, e.g., [5, 21] for additional details) where *n robots*, $p_1, \ldots, p_n$, are sequential processes (state machines), that run *asynchronously*, namely, the time between any two consecutive steps of the same robot is arbitrarily long. Robots are independent from each other, and any number of them may fail by crashing at any time (and cannot recover). We use the terms interchangeably robots or *processes*. They communicate by atomically writing and reading single-writer/multi-reader registers.

Robots move on a graph G. We assume robots know the graph, and can communicate with each other their current vertex positions using the shared memory. Initially, each robot knows its initial vertex in G.

A *configuration* is a vector containing the local state of each robot. An *initial configuration* is a configuration in which all robots are in their initial states. A *step* is performed by a single robot, which executes one of its available local operations. The state machine of a robot p_i models a *local algorithm* A_i that determines p_i's next step. A *distributed algorithm* is a collection $\mathcal{A}$ of local algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_n$. When a process p_i reaches a *decided* state, it stays in its current vertex forever.

An *execution* E is an infinite alternating sequence of configurations and steps $E = C_0\ s_0\ C_1\ \ldots$, where C_0 is an initial configuration and C_{k+1} is the configuration obtained by applying step s_k to configuration C_k. The *participating* robots in an execution are those robots that take at least one step in that execution. Those robots that take a finite number of steps are *faulty* (sometimes called *crashed*), while the others are *correct* (or *non-faulty*). That is, the correct robots of an execution are those that take an infinite number of steps. Moreover, a non-participating process is a faulty process. A participating process can be correct or faulty. We are interested in *wait-free* algorithms: in every execution of the system, every correct robot decide a final vertex, regardless of delays and failures of the other robots.

We describe our algorithms using operations that can be implemented wait-free from registers such as *immediate snapshots* [9]. Each robot p_i invokes the operation with an input value v_i and obtains a set of input values S_i, its *view*, such that the following three properties are satisfied:

1. Self-inclusion: $\forall i : v_i \in S_i$.
2. Containment: $\forall i, j : S_i \subseteq S_j \ \vee \ S_j \subseteq S_i$.
3. Immediacy: $\forall i, j : i \in S_j \ \Rightarrow S_i \subseteq S_j$.

For completeness, we now describe the recursive immediate snapshot algorithm of [18] using only read and write operations. The presentation follows closely that paper and is repeated here for the convenience of the reader. In the algorithm, each process p_i with input v_i, writes the pair $\langle p_i, v_i \rangle$, to a shared memory associated to that recursive call r_i, and reads (one-by-one, in an arbitrary order) the registers of all other processes. A shorthand for the sequence of operations consisting of first writing and then collecting the inputs of all processes is **WriteCollect**. If the set s of values collected is of size r_i, it returns this set as a view and terminates the algorithm; else, the process calls the algorithm recursively with parameters $(v_i, r_i - 1)$. The first call of process p_i is with parameters (v_i, n). We stress that in each recursive call, the processes communicate with each other via a new

Algorithm 1 Code for robot/process p_i with input v_i. Initially $r_i = n$

Function RecursiveIS(v_i, r_i)

```
1: s_i ← WriteCollect(⟨p_i, v_i⟩)
2: if |s_i| = r_i then
3:     return s_i
4: else
5:     RecursiveIS(v_i, r_i − 1)
6: end if
```

array of single-writer/multi-reader registers, that is used only in that recursive call.

Consider the case of three processes, with inputs 1, 2, and 3, respectively. Interestingly, all their possible outputs (running this algorithm) can be graphically represented through a subdivision of a triangle, as shown in Fig. 1. Each triangle represents a possible execution, and the vertices are labeled with the views (outputs) of each one of the processes at the end of the execution. The fully concurrent execution, where all three processes collect the views of each other, and all terminate the algorithm without a recursive call, is represented by the triangle at the center. This triangle shares an edge with three other triangles. These represent executions where two processes see the inputs of all three processes and terminate without a recursive call, while the first process executes two recursive calls and ends up seeing only its own input. For more processes, all outputs can be represented a subdivision of a higher dimensional object, e.g., for four proceses, a subdivision of tetrahedron is needed.

The graph convergence problem

In the *graph convergence* problem on a graph G, each robot starts with an input vertex of G and, after communicating with other robots, has to eventually decide a vertex such that the following two properties are satisfied:

- Agreement: The collection of decided vertices belong to a single edge of G.
- Validity: If the input vertices are equal, then each process must decide this vertex; if the input vertices span an edge, then each process must decide a vertex of that edge.

Notice that this is exactly the definition of a *robot convergence* task [21] specialized to the case of graphs. In the *graph gathering* problem, the agreement property is replaced by requiring that the decided vertices are equal. Thus, in principle one could use general theorems in [21] to identify cases where the problem is solvable; we elaborate further on this point in "A topological perspective" section. The optimality of the two algorithms in the next two sections follows from arguments similar to those in [26]. This is explained in detail in "Round-complexity optimality" section.

Graph convergence on trees

Recall that the *eccentricity* of a vertex v is defined as the greatest distance from v to any other vertex. A *center* of a graph is a vertex with minimal eccentricity. The *radius*

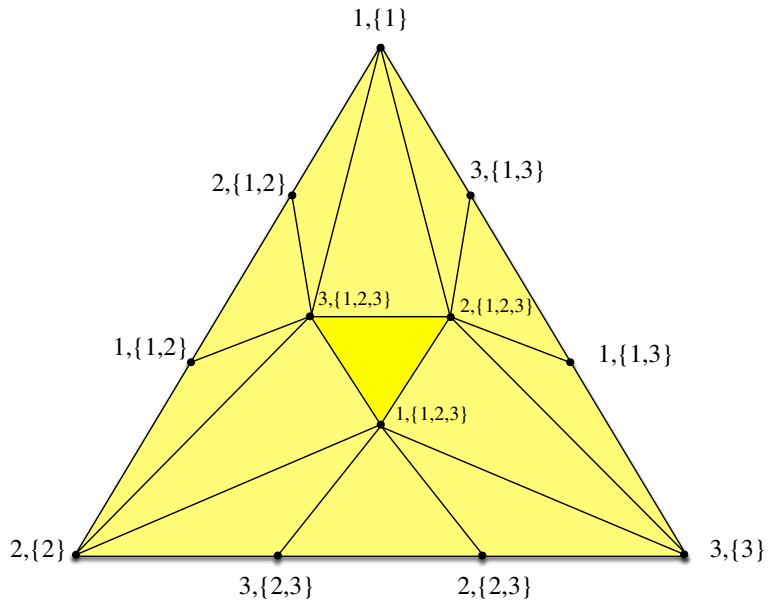

Fig. 1 Graphic description of immediate snapshots. All immediate snapshot views by three processes starting on 1, 2, and 3, respectively

Algorithm 2 $T = (V, E)$ is an arbitrary tree. Code for robot p_i.

Function GraphConvergenceTree(v_i, T)

1: **for** $r_i \leftarrow 1$ **to** $2\lceil \log diam(T) \rceil$ **do**
2: $s_i \leftarrow$ **ImmediateSnapshot**(r_i, v_i)
3: $t_i \leftarrow$ smallest tree of T containing all vertices in s_i
4: $v_i \leftarrow$ a center of t_i
5: **end for**
6: **return** v_i

of G is the minimum eccentricity among the vertices of G and the *diameter* of G is the maximum eccentricity among the vertices of G. Denoting the centers of a graph G by $center(G)$, a tree T has $|center(T)| = 1$ or $|center(T)| = 2$. If $|center(T)| = 1$, the tree is called *central*. If $|center(T)| = 2$, the tree is called *bicentral*. For any graph G, the diameter is at least the radius and at most twice the radius. Trees have the following property, which we will exploit in our graph convergence solution.

Remark 1 *For a tree T, $diam(T) = 2 \times rad(T) - 1$, if T is bicentral, and $diam(T) = 2 \times rad(T)$, if T is central.*

The algorithm **GraphConvergenceTree** (Algorithm 2) solves graph convergence on trees for any number of robots. The idea is very simple: robots proceed in a sequence of rounds, and in every round, each robot p_i communicate to the others its current vertex v_i (its input vertex in the case of the first round) and using the vertices in its snapshot s_i (which does not necessarily contain all vertices of the corresponding round, due to asynchrony), p_i computes a subtree t_i of T and "moves" to a center of t_i. Thus, processes converge by gradually moving to the center of the trees they see during the computation. In the algorithm, each round has an associated immediate snapshot operation which is indexed with the round number passed to the operation as an input parameter.

Figure 2 depicts the tress that three robots, p_1, p_2, and p_3, can obtain in a single round of an execution of algorithm **GraphConvergenceTree** (the execution involves more than those processes). In the example, p_1 executes the round before p_2 and p_2 executes the round before p_3. The properties of the ImmediateSnapshot primitive imply that the snapshot of p_1 contains less information than the snapshot of p_2 (particularly, the snapshot of p_1 does not contain the position of p_2 at the beginning of the round) and, similarly, the snapshot of p_2 contains less information than the snapshot of p_3. Namely, $s_1 \subset s_2 \subset s_3$. Each snapshot s_i induces a tree T_i, and thus $T_1 \subset T_2 \subset T_3$. Finally, the position of each p_i for the next round is the center c_i of T_i. Note that the centers induce a spanning tree that is strictly contained in T_3, the largest tree of the round.

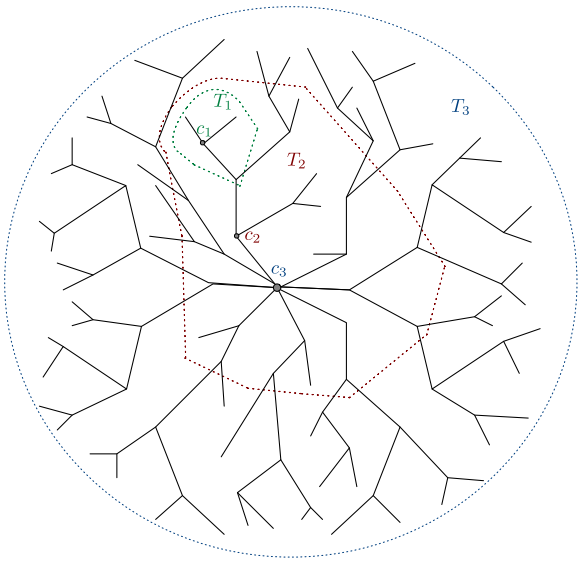

Fig. 2 Optimal graph convergence on a tree. In every round, robots compute a sequence of trees ordered by containment

Clearly, every non-crashed robot terminates in **GraphConvergenceTree** as the number of rounds depends only on the shared tree T. It is also easy to see that the algorithm satisfies the validity property of the graph convergence problem: If all the initial vertices already span a vertex, then each non-crashed robot p_i returns its initial vertex as its snapshot in every round t_i, its contains the single initial vertex. Similarly, if all initial vertices span an edge e, then p_i returns a vertex of e because its snapshot in every round t_i is either e or a vertex of e.

The correctness of algorithm **GraphConvergenceTree** follows from the following lemma showing that every round roughly halves the distance between any two vertices, hence implying that the algorithm satisfies the agreement property. For simplicity, in what follows, we only consider executions in which all robots decide a final vertex. Observe that there is no loss of generality by assuming this, as any execution in which every non-crashed robot returns, a vertex can be extended to an execution in which every robot returns a vertex. Intuitively, in the extended execution, we think of "faulty" robots as very slow that take steps only after all "correct" robots are decided.

Lemma 1 *For $j = 1, \ldots, 2\lceil \log diam(T) \rceil$, let v_i^j be the value of v_i at the beginning of the j-th iteration, and let T_j be the smallest subtree of T containing $v_1^j, \ldots, v_n^j$. Then, $diam(T_{j+1}) \leq (diam(T_j) + 1)/2$.*

Proof The inclusion property of the ImmediateSnapshot operation implies that the snapshots of the robots in the j-th round are ordered by containment, and hence there

are sets $S_1 \subset S_2 \subset \ldots \subset S_k = \left\{v_1^j, \ldots, v_n^j\right\}$, for some $k \geq 1$, such that, for each robot p_i, its snapshot in the j-th round is equal to some S_l. Moreover, by the immediacy property, every robot whose vertex is in S_l has its snapshot equal to S_l. For each S_l, let R_l be the smallest subtree of T_j containing every vertex in S_l. Thus, we have that $R_1 \subset R_2 \subset \ldots \subset R_k = T_j$, and consequently, $v_1^{j+1}, \ldots, v_n^{j+1}$ are nothing else than the centers of these trees, $center(R_1), \ldots, center(R_k)$ (see Fig. 2). Hence, T_{j+1} is the smallest subtree of T_j containing $center(R_1), \ldots, center(R_k)$. Let R'_m be the smallest subtree of T_j containing the first m centers, $center(R_1), \ldots, center(R_m)$ (thus $R'_k = T_{j+1}$). By induction of l, one can show that $diam(R'_l) \leq rad(R_l)$. The base case $l = 1$ is obvious as R'_1 is a single vertex. Once we have assumed the claim holds for l, to show it holds for $l+1$, it is enough to observe that the distance between any pair $center(R_s)$ and $center(R_t)$, distinct both from $center(R_k)$, is at most $rad(R_l)$, by induction hypothesis, which is at most $rad(R_{j+1})$, by definition; and the distance from $center(R_{l+1})$ to any other $center(R_s)$ is at most $rad(R_{l+1})$ by definition. Thus, when $l = k$, we have that $diam(T_{j+1}) \leq rad(T_j)$. By Remark 1, it follows that $rad(T_j) \leq (diam(T_j) + 1)/2$, and hence $diam(T_{j+1}) \leq (diam(T_j) + 1)/2$. □

Lemma 1 directly implies that after $O(\log(diam(T)))$ rounds, all robots end up spanning a vertex or an edge of T.

Theorem 1 *For any tree T, algorithm **GraphConvergenceTree** solves graph convergence on T for $n \geq 2$ robots.*

Graph convergence for two robots on connected graphs

Let us now focus on two robots solving graph convergence on a connected graph. In the following, we describe a modification of **GraphConvergenceTree**, tailored for this case.

In **GraphConvergenceTwoRobots** (Algorithm 3), we assume the robots know (or deterministically compute) a pre-defined shortest path in G between any pair of vertices; thus if the two vertices are the same, the path is the vertex itself, and if the vertices are adjacent, the path is the edge between them. The two robots first take an immediate snapshot to communicate their input vertices. If a robot p_i sees the input of the other, it sets P_i to the corresponding precomputed path. Due to the view containment property of immediate snapshots, it cannot be that both P_i and P_j are equal to $\perp$, and if both are distinct to $\perp$, then $P_i = P_j$. Then, the robots proceed similarly as in **GraphConvergenceTree**: they move to the center of the subpath between their current vertices. Observe that the first round guarantees that the robots move along the precomputed between the initial vertices in all subsequent rounds. Moreover, the path process p_i sets in P_i in the first round is written in every round because in the case the other process p_j goes faster than p_i in the first round, it will be impossible to p_j to know the initial vertex of p_i, and hence set P_j to the path P_i (since a process does not read a shared memory of a previous round).

Algorithm 3 $G = (V, E)$ is a connected graph. Code for p_i.

Preprocessing: $\forall (u, v) \in V^2$ pre-define a shortest path from u to v

Function GraphConvergenceTwoRobots(v_i, G)

```
1:  s_i ← ImmediateSnapshot(1, v_i)
2:  P_i ← ⊥
3:  if |s_i| = 2 then
4:      P_i ← precomputed path between vertices in s_i
5:  end if
6:  for r_i ← 2 to 2⌈log diam(G)⌉ do
7:      s_i ← ImmediateSnapshot(r_i, ⟨v_i, P_i⟩)
8:      if |s_i| = 2 then
9:          if P_i = ⊥ then
10:             P_i ← other robot path in s_i
11:         end if
12:         t_i ← smallest subpath of P_i containing s_i
13:         v_i ← a center of t_i
14:     end if
15: end for
16: return v_i
```

Theorem 2 *For any connected graph G, algorithm **GraphConvergenceTwoRobots** solves graph convergence on G for two robots.*

The edge-covering problem

This section introduces the edge-covering problem and then presents an algorithm that solves it for two processes on any connected graph G that has a cycle of odd length. Our edge-covering solution, **EdgeCoveringTwoRobots** (Algorithm 4), is an adaptation of the **GraphConvergenceTwoRobots** algorithm in the previous section. Surprisingly, the algorithm cannot be generalized for more than two robots; as we shall see, edge covering for three or more processes is impossible on any graph.

In the *edge covering* problem on a given graph G, each robot starts with an input vertex of G and has to eventually decide a vertex such that:

- Agreement: The collection of decided vertices belong to a single edge, and not all decided vertices are equal.

- Validity: If the initial input vertices span an edge, then each robot must decide a vertex of that edge. If a robot runs alone, it should decide its input vertex.

Edge-covering for two robots on connected graphs with odd length cycles

Algorithm **EdgeCoveringTwoRobots** needs a deterministic preprocessing phase that, for any pair of vertices v_i and v_j, computes a simple odd length path from v_i to v_j; if $v_i = v_j$ then it is needed a simple odd length cycle (without such a cycle, finding such a path is impossible). First, if (v_i, v_j) is an edge, then the path from v_i to v_j is this edge. Otherwise, consider a simple cycle $C = w_1, w_2, \ldots, w_x, w_1$ of G of odd length (whose existence is guaranteed by out initial hypotheses). Since G is connected, there are paths P_i and P_j from v_i to w_1 and from w_1 to v_j, respectively. If the length of the composed path $P_i{-}P_j$ is odd, we are done, otherwise, the length of the path $P_i - C - P_j$ must be odd. In any case, for every v_i and v_j, there is a odd length simple path between them. The cycle C and paths P_1 and P_2 can be efficiently computed with a classical Breadth First Search. Note that every precomputed path is bicentral.

Now, as in **GraphConvergenceTwoRobots**, the two robots first take a snapshot to communicate its input vertex to the other and if a robot p_i sees the input of the other, it sets P_i to the corresponding precomputed path (at least one of P_1 and P_2 is distinct from $\perp$). Then, the robots move to a center of the subpath between its current vertices. The difference with algorithms in the previous section is that each robot now picks a center that guarantees that the new positions at the beginning of the next round are at odd distance from each other. Figure 3 describes the associated process: if the distance between v_1 and v_2 is $2d + 1$ then, depending on the parity of d, a robot will move to a position such that the distance to the other is lowered and still odd, independently of whether the other moves or not (in Fig. 3, the top case corresponds to an odd value of d, while the bottom case is when d is even). This invariant, which holds at the beginning of any execution because of the precomputed paths (and ultimately because of the existence of an odd length cycle in G), implies that robots end up at distance exactly one.

Algorithm 4 $G = (V, E)$ contains an odd length simple cycle.

Preprocessing: $\forall (u, v) \in V^2$, compute a simple and shortest odd length path of G from u to v

Function EdgeCoveringTwoRobots(v_i, G)

```
1:  s_i ← ImmediateSnapshot(1, v_i)
2:  P_i ← ⊥
3:  if |s_i| = 2 then
4:      P_i ← precomputed path between the vertices in s_i
5:  end if
6:  for r_i ← 2 to 2⌈log diam(G)⌉ do
7:      s_i ← ImmediateSnapshot(r_i, ⟨v_i, P_i⟩)
8:      if |s_i| = 2 then
9:          if P_i = ⊥ then
10:             P_i ← other robot path in s_i
11:         end if
12:         v_j ← vertex of the other robot in s_i
13:         t_i ← smallest subpath of P_i containing v_i and v_j
14:         v_i ← center of t_i such that the length of the
    subpath of t_i from that center to v_j is odd
15:     end if
16: end for
17: return v_i
```

Lemma 2 *Let P be the precomputed path between the initial vertices v_1 and v_2. For $j = 1, \ldots, 2\lceil diam(G)\rceil$, let v_1^j and v_2^j be the values of v_1 and v_2 at the beginning of the j-th iteration, and let P_j be the smallest subpath of P containing them. Then, $|P_{j+1}|$ is odd and less or equal to $(|P_j| + 1)/2$.*

Proof First, observe that $P_1 = P$, hence, by construction, $|P_1|$ is odd. Thus, we assume $|P_j|$ is odd. Note that P_j is a bicentral tree as its length is odd. Let (u, v) be the edge containing its two centers. Let s_1 and s_2 be the immediate snapshots of p_1 and p_2 in the j-th iteration. We have three cases:

- $|s_1| = |s_2| = 2$. In this case, both s_1 and s_2 contain v_1^j and v_2^j. Then, both robots p_1 and p_2 have t_1 and t_2 equal to T_j. Observe that if the length of subpath of T_j from u to v_1^j is odd (resp. even), then the length of the subpath from u to v_2^j is even (resp. odd). It similarly happens with v (see Fig. 3). Then, it must be that either $v_1^{j+1} = u$ and $v_2^{j+1} = v$, or $v_1^{j+1} = v$ and $v_2^{j+1} = u$. In either case $T_{j+1} = (u, v)$.
- $|s_1| = 1$ and $|s_2| = 2$. In this case, s_1 only contains v_1^j while s_2 contains v_1^j and v_2^j. Then, $v_1^{j+1} = v_1^j$ and v_2^{j+1} is the center of P_j such that the length of the subpath Q from that center to v_1^j is odd (as explained in the previous case, only one center has that property).

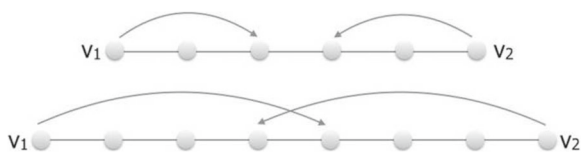

Fig. 3 Edge covering for two robots. Each robot moves to a central vertex that preserves the odd-path path invariant

Thus, $T_{j+1} = Q$. Remark 1 implies that $Q = (|T_j| + 1)/2$.

- $|s_1| = 2$ and $|s_2| = 1$. This case is symmetric to the previous case.

□

Lemma 2 shows that at the end of **EdgeCoveringTwoRobots**, the two robots end up on vertices that span an edge of G, hence we have the following.

Theorem 3 *For any connected graph G containing a simple cycle of odd length, algorithm **EdgeCoveringTwoRobots** solves the edge covering problem for two robots.*

Impossibility results

In this section, we present a series of impossibility results that fully characterize the solvability of graph convergence and edge covering.

We first show that if G is disconnected, then graph convergence and edge covering are impossible. The reason is that a solution to any of these problems on G can be used to solve wait-free *binary consensus*, which is known to be impossible [5, 20]. The following style of proof is known since the first (1-resilient) task characterization results [7].

Lemma 3 *If G is disconnected, then graph convergence and edge covering on G are impossible for any number of robots $n \geq 2$.*

Proof Assume, for the sake of contradiction, that there is an algorithm A that solves graph convergence on G for $n \geq 2$ robots (the proof is the same when A solves edge covering). Using A, we solve binary consensus among n robots, which is known to be impossible [5, 20]. For sake of simplicity, we focus on the *binary consensus* problem, where each robot proposes either 0 or 1, and robots are required to decide proposed values so that all decisions are equal.

Let v_0 and v_1 be vertices of G belonging to distinct connected components. Let C_0 be the connected component v_0 belongs to. We solve binary consensus as follows. Each robot p_i with proposal $j \in \{0, 1\}$, invokes A with input v_j. Let w_i be the value A outputs to p_i. Then, p_i decides 0 if w_i belongs to C_0 and 1 otherwise.

If all proposal are equal to $j \in \{0, 1\}$, then every robot receives v_j from A, since A is a graph convergence solution. Then, every robot decides j, which solves consensus. If robots propose distinct values, then they invoke A with distinct inputs, v_0 and v_1. Since A solves graph convergence, it outputs vertices that span a vertex or an edge of G. Note that it cannot be that some of these vertices belong to C_0 and the rest to $G \setminus C_0$. Therefore, all robots decides either 0 or 1 and all decisions are the same, which solves consensus. □

The following lemma shows that cycles are an obstacle for solving graph convergence and edge covering when the number of robots is greater or equal than three. The structure of the proof is similar to proof of the previous lemma: if there is a solution to a graph with cycles, then one can solve the well-known *set agreement* problem [14], which has been proved to be unsolvable (see [21]).

Lemma 4 *If G has a cycle, then graph convergence and edge covering on G are unsolvable for $n \geq 3$ robots.*

Proof By contradiction, suppose that there is an algorithm A that solves graph convergence on G for $n \geq 3$ robots (the proof for edge covering is the same). We use A to solve two-set agreement for three processes hence reaching a contradiction. For the sake of simplicity, we focus on the inputless version of the $(n-1)$*-set agreement* problem each process p_i has as input its index i, and every correct process is required to decide an index of a process that participates in the execution such that at most $n-1$ distinct indexes are decided by the processes. It is well-known that $(n-1)$-set agreement is unsolvable. Thus, A cannot exist because it implies a solution to $(n-1)$-set agreement.

Below we use the following remark that directly follows from the specification of graph convergence and edge covering, which are adaptive by nature.

Remark 2 *Let G be a graph and suppose there is an algorithm A that solves graph convergence (edge covering) on G for $n \geq 3$ robots. Then, A solves graph convergence (edge covering) on G for $n-1$ robots.*

Therefore, this last remark implies that we can assume A solves graph convergence on G for three robots.

Algorithm 5 Code for process p_i.

```
Function SetAgreement(i)
1:  if i = 3 then
2:      x_i ← v_3
3:  else
4:      x_i ← B.GraphConvergence(v_i, P)
5:  end if
6:  y_i ← A.GraphConvergence(x_i, G)
7:  if for some j = 1, 2, 3, y_i = v_j then
8:      return j
9:  else
10:     return 1
11: end if
```

Let C be a simple cycle of G and let v_1, v_3, v_2 be three distinct and consecutive vertices of C. Let P be the simple odd path obtained by removing v_3 from C (see Fig. 4). By Theorem 2, let B be an algorithm that solves graph convergence on P for processes p_1 and p_2. We use A and B to solve two-set agreement for three robots.

Algorithm 5 solves two-set agreement for three robots, p_1, p_2, and p_3, using algorithms A and B. The idea of the solution is that robots use A to "agree" on a vertex or an edge of G, namely, on at most two distinct vertices, and then use these information to return at most two distinct indexes of participating processes. Thus, vertices of G are mapped to indexes of processes: v_3 is mapped to 3, v_2 is mapped to 2 and the remaining vertices are mapped to 1, as illustrated in Fig. 4. The properties of A make easy to achieve agreement: at most to distinct indexes are decided since A solves graph convergence. What is more complicated to achieve is validity: only indexes from participating processes can be decided. That is the aim of algorithm B and actually the most complicated case is when p_1 and p_2 participate: they use B to cover a vertex or an edge of P and these vertices are the inputs the use for A; this step guarantees that none of them gets v_3 from A, so each of them returns either 1 or 2.

We now show that Algorithm 5 is correct.

- Termination. By assumption, A and B terminate in all invocations, thus in every execution, a nonfaulty robot returns a value.
- Validity. We identify three cases, according to the number of robots that participate and decide in a given execution.
  - One robot p_i participates in an execution. If $p_i = p_3$, then it invokes A with input v_3, and consequently obtains v_3 from it, by validity property of edge convergence. Thus, p_3 returns 3.
    If p_i is either p_1 or p_2, it invokes B with input v_i, and consequently obtains v_i from B, by validity of graph convergence; and hence p_i invokes A with v_i and obtains v_i as well, for the same reason, therefore it returns i.
  - Two processes p_i and p_j participate. If $p_i = p_3$ and p_j is either p_1 or p_2, then p_j invokes solo B with input v_j, hence it obtains v_j. Thus, p_i and p_j invoke A with inputs v_i and v_j, respectively, and thus they obtains these vertices from A since, by assumption, A solves graph convergence and, by definition, these vertices are an edge of G. We conclude that p_i returns i and p_j return j.
    If $p_i = p_1$ and $p_j = p_2$, then they obtain from B two vertices x_i and x_j, respectively, that cover a vertex or an edge of P, since B solves graph convergence on P. Then, p_i and p_j get the very same vertices from A since it solves graph convergence on G, by assumption. Thus, each of p_i and p_j returns either 1 or 2 because the only way a process returns 3 is if it gets v_3 from G, but $v_3 \notin V(P)$.
  - The three robots participate. From the pseudocode, it is easy to see that a robot can only decide 1, 2, or 3. If all processes participate, any of these decisions satisfy validity.
- Agreement. The only interesting case is when the three processes return a value. Since A solves graph convergence on G, the values it returns y_1, y_2 and y_3 to p_1, p_2, and p_3, respectively, cover a vertex or an edge of G, hence at most two distinct indexes are decided.

We conclude that Algorithm 5 solves two-set agreement for three processes, which, as already explained, is a contradiction, from which follows that such an algorithm A cannot exist. □

Lemmas 3 and 4 together with Theorems 1 and 2 in "The graph convergence problem" section completely characterize the solvability of graph convergence.

Theorem 4 (Solvability of Graph Convergence) *For two robots, graph convergence on a graph G is solvable if and only if G is connected. For three or more robots, graph convergence on G is solvable if and only if G is acyclic.*

We now fully characterize the solvability of the edge-covering problem. As we will see, the extra requirement of edge covering that processes always have to cover an edge, precludes solutions for three or more robots, for any graph. The next lemma shows a necessary condition for the solvability of edge covering for two robots.

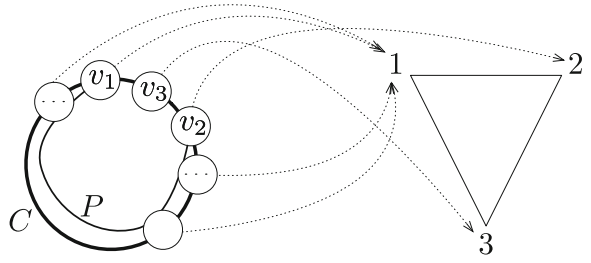

Fig. 4 Impossibility of graph convergence on cycles. Mapping vertices of the cycle C to values for set agreement

Lemma 5 *Let G be a graph. If there is an algorithm that solves edge covering on G for two processes, then G is connected and has a simple cycle of odd length.*

Proof Let G be any graph and suppose there is an algorithm A that solves edge covering on G for two processes. Lemma 3 implies that G is connected. To show that G has a simple cycle of odd length, suppose the contrary, namely, suppose that G has no simplex cycle of odd length. We will use A to solve *weak symmetry breaking* (WSB) for two robots [19]. The WSB for two robots is an inputless problem in which each robot has to decide 0 or 1 such that in solo executions, the decision is the same and if the two robots participate, they decide distinct values. It is known that WSB for two robots is unsolvable (see [21]).

Before solving WSB, we observe that G is bipartite since it has no odd length cycles, hence it has a proper vertex binary coloring c. To solve WSB, each robot p_i invokes A with a fixed vertex v (the same for both robots), and decides $c(w)$, where w is the vertex A outputs to p_i. Clearly, robots decide 0 or 1, since c is a binary coloring. In a solo execution of any robot p_i, A outputs v to p_i, by validity of edge covering, and hence it decides $c(v)$. Finally, if the two robots participate, they decide distinct values because c is a binary coloring and A outputs to the robots vertices of G that span an edge. Thus, using A, we can solve WSB, which is a contradiction. □

Using the previous lemma, we can show that edge covering is unsolvable for three or more processes. The proof is that if there is an algorithm that solves edge covering on a graph G for three or more processes, then, by the adaptive nature of edge covering, this algorithm solves edge covering on G for two processes, and hence G has a cycle, by Lemma 5. But this contradicts Lemma 4.

Lemma 6 *For any graph G, there is no algorithm that solves edge covering on G for three or more robots.*

Proof Suppose por contradiction that there is an algorithm A that solves edge covering on G. As observed by Remark 2 in the proof of Lemma 4, A solves edge covering on G for two robots. Thus, G has a cycle, by Lemma 5. Lemma 4 implies that A cannot exist. □

Finally, from Lemmas 5 and 6 and Theorem 3 in "The edge-covering problem" section, we derive a full characterization for the solvability of edge covering.

Theorem 5 (Solvability of Edge Covering) *For two robots, edge covering on G is solvable if and only if G is connected and has a simple cycle of odd length. For three or more robots, edge covering is unsolvable on any graph G.*

A topological perspective

The topological approach to distributed computing [21] has been useful to understand the nature of fault-tolerant distributed computing, to prove impossibility results and to understand why in some case there exists a distributed algorithm to solve a problem. In this section, we briefly discuss a topological perspective of graph convergence and edge covering.

The topology of graph convergence

Our graph convergence problem is a special case of the *robot convergence task* defined in [21], which is specified as follows. A collection of n robots are placed on the vertices of a graph G. The robots are asynchronous, communicate through read/write shared registers and eventually (in a wait-free manner) each one chooses a final vertex and halts. The final vertices must belong to the same edge (in the book, to the same simplex of an arbitrary complex). If they are all placed initially on the same vertex or edge, then they stay there (although they may move from one vertex to the other, even they may all move to the same vertex). If the robots are placed on vertices that do not belong to the same edge, they can move to any vertices of G, as long as the vertices belong to an edge.

Formally, a robot convergence task for a graph G is given by a triple $(\mathcal{I}, G, \Delta)$, where $\mathcal{I}$ consists of all the subsets of V of at most n vertices of G. Such a set $\mathcal{I}$, consisting of a family of sets closed under containment is called in topology a *simplicial complex*. An element σ of $\mathcal{I}$ is called a *simplex*, and its *dimension* is $|\sigma| - 1$. Thus, $\sigma \subseteq V$, $|\sigma| \leq n$. For each simplex σ in $\mathcal{I}$, representing possible simultaneously starting vertices of G, Δ encodes the convergence rules. Namely, $\Delta(\sigma)$ is a subgraph of G where the robots may end up, if their initial positions are in σ. Thus, $\Delta(\sigma) = \sigma$ if σ is either a vertex or an edge of G, and otherwise, $\Delta(\sigma) = G$. The following is from the book [21] (page 88) (see also [24]).

Theorem 6 (4.3.1 [21]) *The graph convergence task $(\mathcal{I}, G, \Delta)$ has a wait-free n-process read/write protocol if and only if there is a continuous map $f : |\mathcal{I}| \to |G|$ carried by Δ.*

This theorem considers G as a continuous space, denoted by $|G|$, as if G was embedded in some sufficiently large Euclidean space: vertices of G are points of the space, and edges are lines connecting their corresponding vertex-points. Similarly, the space $|\mathcal{I}|$ consists of the points where the vertices of $\mathcal{I}$ are placed in Euclidean space, and linear subspaces spanned by vertices belonging to the same simplex σ of $\mathcal{I}$. The continuous map f respects the input/output specification of the task, Δ (i.e., it is carried by Δ), in the sense that for each simplex $\sigma \in \mathcal{I}, f(|\sigma|)$ is in $|\Delta(\sigma)|$. In particular, $f(v) = v$, and $f(e) = e$ for any

edge e. But f may send a simplex σ which is not an edge anywhere in G.

Theorem 6 can be used to derive simple impossibility proofs, as shown below.

Corollary 1 *If G is disconnected, graph convergence on G is unsolvable for $n \geq 2$ robots.*

Proof Let u, v be vertices of different connected components, G_u, G_v. Consider the simplex $\sigma = \{u, v\}$ of $\mathcal{I}$. Suppose there is a solution to the task, let f be its associated map by Theorem 6. Then, $f(u)$ is in connected component G_u and $f(v)$ is in connected component G_v. It is impossible to extend f to all of σ, because σ is connected, while $G_u \cup G_v$ is disconnected, a contradiction. □

Corollary 2 *If G has a cycle, graph convergence on G is unsolvable for $n \geq 3$ robots.*

Proof Let G be a graph with a cycle $C = v_1, v_2, \ldots, v_p, v_1$ (see Fig. 5 for an illustration with $p = 4$). Then, C is a cycle of $\mathcal{I}$. Once again, suppose there is a solution to the task, let f be its associated map by Theorem 6. We have that $f(C) = C$, where for each $v_i, f(v_i) = v_i$, and $f(v_i, v_{i+1}) = (v_i, v_{i+1})$, by the validity requirement of graph convergence. Notice that $\mathcal{I}$ is the $(n-1)$-skeleton of a simplex with vertices V, i.e., the set of all simplexes of dimension at most $n-1$. We can view V as a complex (consisting of the set V and all its faces) which represents a solid ball of dimension $|V|-1$. Hence, any cycle on $\mathcal{I}$ (its $n-1$, dimensional skeleton, with $n-1 \geq 2$) is contractible. Thus, C is contractible in $\mathcal{I}$, but its image, $f(C)$ is not contractible, because $f(C) = C$. □

The intuition for Theorem 6 becomes clear considering the effect of processes taking immediate snapshots. As shown in Fig. 1, all possible views obtained by three processes have the effect of subdividing a triangle, which in turn represent the inputs of the processes. As the immediate snapshot algorithm is repeated more and more times, finer and finer subdivisions are obtained, and hence a better approximation to a continuous map (see from [21]).

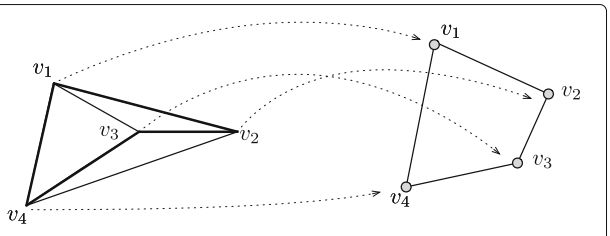

Fig. 5 Topological perspective of graph convergence. Impossibility of graph convergence on cycle $C = v_1, v_2, v_3, v_4, v_1$ by three processes

For more processes, subdivisions of higher dimensional simplexes are obtained.

Figure 6 illustrates how a solution is obtained for a tree with three processes, first with a continuous map, which has to then be approximated by using subdivisions (right-hand side). Notice that a simplicial map from a subdivision to the output complex represents the decisions taken in each vertex.

The topology of edge covering

The underlying topological behavior of the edge covering problem is more complex, because the problem cannot be described solely by the vertices of G where robots may start and may end. In addition, it is necessary to specify which robot starts or ends in which vertex. The *colored* version of a graph G for two robots A, B, required to model the edge covering problem, denoted $\tilde{G} = (\tilde{V}, \tilde{E})$, consists of all pairs of vertices of the form $\langle id, v\rangle$, where $v \in V$ and $id \in \{A, B\}$.

An edge (x, x') belongs to $\tilde{E}$ iff $x = \langle id, v\rangle$ $x' = \langle id', v'\rangle$ such that $(v, v') \in E$ $(v \neq v')$, and $id \neq id'$. A vertex $\langle id, v\rangle$ represents the situation where robot id ends in vertex v.

Recall that robots have to end in adjacent vertices of $\tilde{G}$. Figure 7 illustrates that robots A and B have to end in vertices belonging to the same edge of G, but cannot both end in the same vertex.

The two-robot edge covering problem for a graph $G = (V, E)$ is formally modelled as a colored task $\langle \tilde{I}, \tilde{G}, \tilde{\Delta}\rangle$, where the input graph $\tilde{I}$ is the colored version of the complete graph (including self-loops) on $|V|$ vertices. Thus, for each pair of vertices of V, (v, v'), not necessarily distinct, there is an edge $(x, x') \in \tilde{E}$ with $x = \langle A, v\rangle$, $x' = \langle B, v'\rangle$, meaning that it is possible that A starts in v and B starts in v'. Then, the relation $\tilde{\Delta}$ is defined as follows. First, if a robot runs solo, it stays in its initial vertex, $\tilde{\Delta}(\langle id, v\rangle) = \{\langle id, v\rangle\}$, for every vertex $\langle id, v\rangle$. Second, if the robots start in an edge, they stay there, i.e., $\forall (v, v') \in E$:

$$\tilde{\Delta}(\langle A, v\rangle, \langle B, v'\rangle) = \left\{\langle A, v\rangle, \langle B, v'\rangle, \langle B, v\rangle, \langle A, v'\rangle\right\}$$

Finally, if they do not start in vertices of the same edge, they can decide on any vertices belonging to the same edge, i.e., $\forall \left(v, v'\right) \notin E, \tilde{\Delta}\left(\langle A, v\rangle, \langle B, v'\rangle\right) = \tilde{G}$.

The wait-free solvability theorem [25] implies that the edge covering problem for two robots has a solution if and only if there is a subdivision X of $\tilde{I}$ and a simplicial map δ from X to $\tilde{G}$, such that δ preserves ids and edge adjacencies, and δ respects $\tilde{\Delta}$. It is called a *decision* map because it represents the outputs of the distributed algorithm that the robots execute. Namely, when the robots start in an edge $(x, x') \in \tilde{I}$, $x = \langle A, v\rangle$ and $x' = \langle B, v'\rangle$, it is known that the distributed algorithm induces a subdivision of (x, x'), essentially creating a path, where each edge

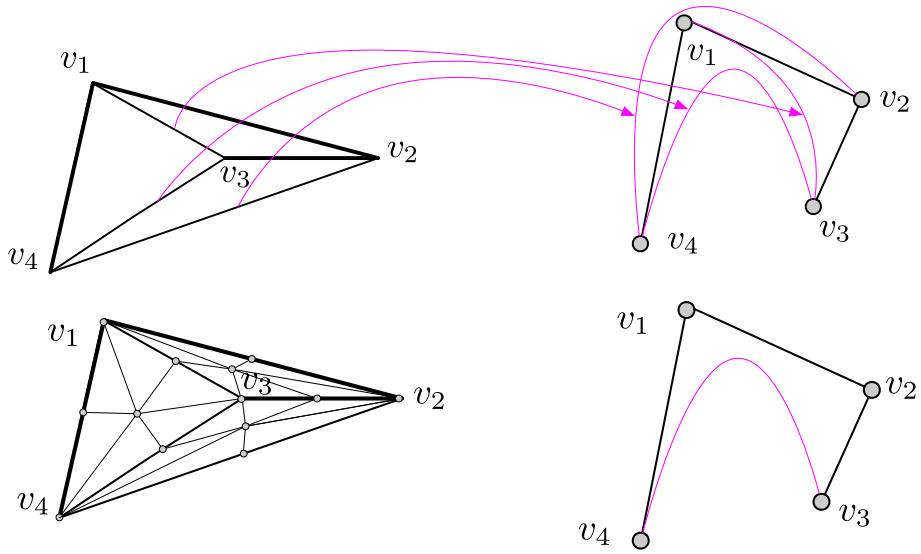

Fig. 6 Topological perspective of graph convergence. Top Solving a tree by three processes. Bottom Trying to solve a tree in one round

of the path represents the final states of the robots in one of the possible executions starting in (x, x').

Theorem 7 (3.1 [25]) *A task* $(\mathcal{I}, \mathcal{O}, \Delta)$ *has a wait-free read/write protocol iff there is a chromatic subdivision* X *of* $\mathcal{I}$ *and a color-preserving simplicial map* $\delta : X(\mathcal{I}) \to \mathcal{O}$ *s.t. for each* $\sigma \in X(\sigma)$, $\delta(\sigma)$ *is carried by* $\Delta(\sigma)$ *(see Fig. 8).*

A consequence is that the task has a solution for A, B iff for any two vertices of G there is an odd length path, corresponding to a path in $\tilde{G}$ alternating vertices with id A and B.

Notice that the formal specification of the edge covering problem as a triple $\langle \tilde{I}, \tilde{G}, \tilde{\Delta} \rangle$, depends on the number of robots, and indeed we defined it above for A, B. To go beyond two robots, to three robots, it is necessary to add to triangles to the graphs, representing positions of three robots A, B, C, and more generally, simplices of n vertices, labeled with distinct robot ids. Then, the wait-free solvability theorem [25] is about general dimension combinatorial topology simplicial complexes. Roughly speaking, edge covering (and in general non-colorless tasks) is more difficult that graph convergence, because of a seemingly innocuous, but surprisingly "difficult" requirement: the simplicial decision map δ is color-preserving. Namely, while in a colorless task, robots can always adopt each other outputs (δ can send simplexes to lower dimensional output simplexes), this is not possible in general tasks (δ sends a final state's algorithm simplex to an output simplex of the same dimension).

Round-complexity optimality

Once the topology of the graph convergence and edge covering is understood, it is easy to argue that our algorithms in previous sections are asymptotically round-complexity optimal.

For graph convergence, consider the simple case in which two robots, say p_u and p_v, start on vertices u and v at distance $diam(G)$. As explained above, this initial configuration is represented with an edge, S^0, with its vertices representing the processes initially standing on those vertices. After a first round, this line is subdivided into three edges, S^1, each of them modeling the three possible immediate snapshots: p_u goes first or viceversa, or p_u and p_v go together. For example, the bottom edges in the subdivision in Fig. 1 correspond to the immediate snapshot for robots 2 and 3. In a second round, each of these three edges is subdivided again into three edges to obtain a subdivision, S^2, with nine edges, and so on. In general, the subdivision S^r of the edge S^0 obtained after r rounds has

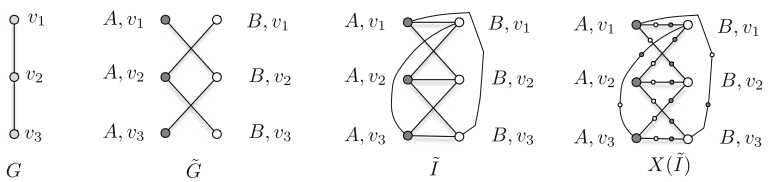

Fig. 7 Topological perspective of edge-covering. G and colored graph $\tilde{G}$, with input $\tilde{I}$ and views after one round $X(\tilde{I})$, for robots A, B

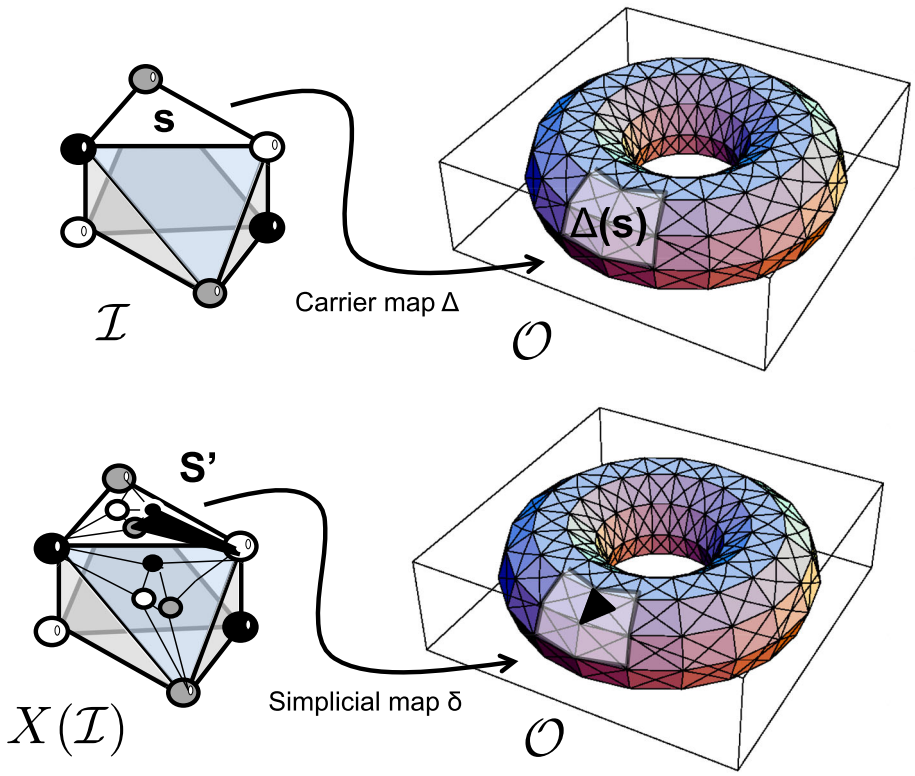

Fig. 8 The wait-free computability theorem. A graphical description of the wait-free computability theorem

3^r edges. Thus, the length of the subdivision exponentially grows on the number of rounds, which implies that after $O(\log diam(G))$ rounds the length of $S^{O(\log diam(G))}$ is $\Omega(diam(G))$.

Now, due to the agreement property of graph convergence, any algorithm that solves this problem must map the vertices in $S^{O(\log diam(G))}$ to vertices of G satisfying that the vertices of any edge are mapped to vertices of G at distance at most 1. Moreover, the validity property implies that the extreme vertices of the subivision must be mapped to u and v: these vertices in $S^{O(\log diam(G))}$ represent the solo executions of the processes, one of p_u and one of p_v, and the validity property states that in solo executions a process must decides its initial vertex. Thus, the extreme vertices of $S^{O(\log diam(G))}$ are mapped to u and v and all other vertices are mapped to vertices of G, satisfying the agreement property. In other words, $S^{O(\log diam(G))}$ is mapped to a path of G from u to v where each edge of the subdivision is either mapped to an edge of the path or "collapsed" to a vertex of the path. This mapping is possible because the length of $S^{O(\log diam(G))}$ is $\Omega(diam(G))$, as explained above.

It is now easy to see that if robots perform only $o(\log diam(G))$ rounds, they cannot solve graph-convergence: in such a case the diameter of $S^{o(\log diam(G))}$ is $o(diam(G))$, and hence it is impossible to map $S^{o(\log diam(G))}$ to a path of G from u to v as described above. Hence, the round-complexity optimality of our graph-convergence algorithms.

The analysis for edge covering is essentially the same, the only difference is that, due to the agreement property of the problem, each edge of the subdivision must be mapped to an edge of G. From this perspective, we can say more. It is needed the existence of an odd-length path in G from u to v: $S^{O(\log diam(G))}$ is mapped to a path in G from u to v, and each edge of the subdivision is mapped to an edge of the path; however, as explained above, the length of $S^{O(\log diam(G))}$ is odd (more precisely, $3^{O(\log diam(G))}$), from which follows that the length of the path $S^{O(\log diam(G))}$ is mapped to must be odd.

Conclusion

In this paper, we study two robot convergence problems in an asynchronous read/write shared memory crash-prone system, where the base space is a finite graph. The problems are the graph convergence (robots decide vertices that belong to the same edge) and edge covering (robots decide vertices that cover an edge). For both tasks we show possibility and impossibility results that fully characterize the graphs on which these problems can be solved. Additionally, we give a topological perspective of the solvability of both problems.

The study of the two robot convergence problems presented in this paper is of a theoretical nature. The purpose is to understand which problems are in principle solvable, and show inherent limitations of what is possible. We show that the two robot convergence problems are unsolvable if G has a cycle, when the number of robots is at least 3 (Lemma 4). This implies an inherent limitation about robot coordination, even if the robots know G and can communicate with each other reliably. The limitation comes from the fact that the robots are asynchronous and they may crash. This means that if one abstracts away a two-dimensional space that has a hole that must be avoided by the robots by a graph, the tasks are unsolvable when $n \geq 3$. Another contribution, is to expose the inherent difficulty of requiring the robots to coordinate to cover an edge: edge covering is unsolvable when the number of robots is at least 3 (Lemma 6). Indeed, edge covering is a more subtle problem. We have shown that for two robots, solving it requires the existence of an odd length cycle (Lemma 5).

Although the study presented is theoretical, we hope it motivates to solve solutions to robot convergence problems in realistic settings. For the cases we provide impossibility results, it would be interesting to search for probabilistic solutions. A limitation of our model is that robots communicate reliably using a shared memory, see the exact positions of other robots, and may jump directly to any vertex. In reality robots may have limited visibility of other robots and see only their approximate positions. We investigate a more detailed model where robots can move only to adjacent vertices in G, in [2]. Extending our algorithms to such situations requires interesting future research. Also, we consider only crash failures. Interesting future work may consider Byzantine failure techniques such as [28]. In contrast, we expect that the assumption that robots have identifiers is common in practice. In any case, our algorithms do not make an essential use of identifiers.

We assume that robots are located on a space that can be modelled as a graph. Some spaces encountered in applications can be naturally modeled as graph, or a graph can be extracted, see [29] for a survey of techniques for extracting graph representations of the environment. Further research is needed to extend our results to two or even three dimensional spaces. We expect that more sophisticated topological techniques may be needed.

Our two-robot convergence tasks are abstractions that capture the fundamental difficulty of coordination. Understanding the inherent difficulty in coordination and providing sound and proven solutions is a prerequisite for designing algorithms for real applications; many real-life situations in distributed robotics, such as cooperative vehicular communication, or rescue missions, require that robots coordinate to get to positions close to each other.

In practice, robots do not have a physical shared memory to communicate with each other; typically, they communicate through a weaker communication medium, sending messages to each other, observing each other, etc. The model we consider can be simulated on top of a system where processes communicate by sending messages to each other, where a majority of them do not crash [3]. Thus, all our impossibility results hold also in that setting and our algorithms can be translated to such network system using the simulation in [3].

Funding
The first author is supported by UNAM-PAPIIT IA102417. The second author is supported by UNAM-PAPIIT IN109917 and also received support from ECOS-CONACYT and LAISLA. The third author is supported by CPSLab project H2020-ICT-644400. The first and second authors want to thank INRIA for its support in the context of the INRIA-UNAM "Équipe Associée" *LiDiCo* (At the Limits of Distributed Computing).

Availability of data and materials
Not applicable.

Authors' contributions
All authors read and approved the final manuscript.

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details
[1]Instituto de Matemáticas, UNAM, Ciudad Universitaria, 04510 Mexico city, Mexico. [2]LAAS-CNRS, 7, av du Colonel Roche, 31077 Toulouse, France.

Received: 19 May 2017 Accepted: 6 December 2017
Published online: 08 January 2018

References

1. Agmon N, Peleg D (2006) Fault-tolerant gathering algorithms for autonomous mobile robots. SIAM J Comput 36(1):56–82
2. Alcántara M, Castañeda A, Peñaloza DF, Rajsbaum S (2017) Fault-tolerant robot gathering problems on graphs with arbitrary appearing times. In: 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE. pp 493–502
3. Attiya H, Bar-Noy A, Dolev D (1995) Sharing memory robustly in message-passing systems. J ACM 42(1):124–142
4. Attiya H, Bar-Noy A, Dolev D, Peleg D, Reischuk R (1990) Renaming in an asynchronous environment. J ACM 37(3):524–548
5. Attiya H, Welch J (2004) Distributed computing: fundamentals, simulations, and advanced topics. Wiley
6. Bhattacharya S, Lipsky D, Ghrist R, Kumar V (2013) Invariants for homology classes with application to optimal search and planning problem in robotics. Ann Math Artif Intell 67(3):251–281
7. Biran O, Moran S, Zaks S (1990) A combinatorial characterization of the distributed 1-solvable tasks. J Algoritm 11(3):420–440
8. Borowsky E, Gafni E, Lynch N, Rajsbaum S (2001) The BG distributed simulation algorithm. Distrib Comput 14(3):127–146
9. Borowsky E, Gafni E (1993) Generalized FLP impossibility result for t-resilient asynchronous computations. In: Proceedings of the

Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA. pp 91–100
10. Bouzid Z, Das S, Tixeuil S (2013) Gathering of mobile robots tolerating multiple crash faults. In: Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems, ICDCS '13. IEEE Computer Society, Washington. pp 337–346
11. Castañeda A, Rajsbaum S (2012) New combinatorial topology bounds for renaming: the upper bound. J ACM 59(1):3:1–3:49
12. Castañeda A, Rajsbaum S, Raynal M (2011) The renaming problem in shared memory systems: an introduction. Comput Sci Rev 5(3):229–251
13. Castañeda A, Imbs D, Rajsbaum S, Raynal M (2016) Generalized symmetry breaking tasks and nondeterminism in concurrent objects. SIAM J Comput 45(2):379–414
14. Chaudhuri S (1993) More choices allow more faults: set consensus problems in totally asynchronous systems. Inf Comput 105(1):132–158
15. Das S, Flocchini P, Prencipe G, Santoro N, Yamashita M (2016) Autonomous mobile robots with lights. Theor Comput Sci 609(P1):171–184
16. Dolev D, Lynch NA, Pinter SS, Stark EW, Weihl WE (1986) Reaching approximate agreement in the presence of faults. J ACM 33(3):499–516
17. Fischer MJ, Lynch NA, Paterson M (1985) Impossibility of distributed consensus with one faulty process. J ACM 32(2):374–382
18. Gafni E, Rajsbaum S (2010) Recursion in distributed computing. In: Proceedings of the 12th International Conference on Stabilization, Safety, and Security of Distributed Systems, SSS'10. Springer-Verlag, Heidelberg. pp 362–376
19. Gafni E, Rajsbaum S, Herlihy M (2006) Subconsensus tasks: renaming is weaker than set agreement. In: Proceedings of the 20th International Conference on Distributed Computing, DISC'06. Springer-Verlag, Heidelberg. pp 329–338
20. Herlihy M (1991) Wait-free synchronization. ACM Trans Program Lang Syst 13(1):124–149
21. Herlihy M, Kozlov D, Rajsbaum S (2013) Distributed computing through combinatorial topology, 1st edition. Morgan Kaufmann Publishers Inc., San Francisco
22. Herlihy M, Rajsbaum S (1997) The decidability of distributed decision tasks (extended abstract). In: Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing, STOC '97. ACM, New York. pp 589–598
23. Herlihy M, Rajsbaum S (2003) A classification of wait-free loop agreement tasks. Theor Comput Sci 291(1):55–77
24. Herlihy M, Rajsbaum S, Raynal M, Stainer J (2014) LATIN 2014: Theoretical Informatics: 11th Latin American Symposium, Montevideo, Uruguay, March 31–April 4, 2014. In: Proceedings, chapter Computing in the Presence of Concurrent Solo Executions. Springer Berlin Heidelberg, Heidelberg. pp 214–225
25. Herlihy M, Shavit N (1999) The topological structure of asynchronous computability. J ACM 46(6):858–923
26. Hoest G, Shavit N (2006) Toward a topological characterization of asynchronous complexity. SIAM J Comput 36(2):457–497
27. Liu X, Xu Z, Pan J (2009) Classifying rendezvous tasks of arbitrary dimension. Theor Comput Sci 410(21-23):2162–2173
28. Mendes H, Herlihy M, Vaidya N, Garg VK (2015) Multidimensional agreement in byzantine systems. Distrib Comput 28(6):423–441
29. Portugal D, Rocha RP (2013) Retrieving topological information for mobile robots provided with grid maps, Number 358 in Communications in Computer and Information Science. Springer Berlin Heidelberg, Heidelberg
30. Saraph V, Herlihy M The relative power of composite loop agreement tasks. In: Proceedings of the International Conference on Principles of Distributed systems (OPODIS), LIPIcs: Leibniz Int. Proc. Informatics, Germany, 2015. Dagstuhl

Submit your manuscript to a SpringerOpen journal and benefit from:

► Convenient online submission
► Rigorous peer review
► Open access: articles freely available online
► High visibility within the field
► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com