Two Convergence Problems for Robots on Graphs
Armando Castañeda, Sergio Rajsbaum, Matthieu Roy

To cite this version:

HAL Id: hal-01386628
https://hal.archives-ouvertes.fr/hal-01386628
Submitted on 24 Oct 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Two Convergence Problems for Robots on Graphs

Armando Castañeda  
Instituto de Matemáticas, UNAM, México.

Sergio Rajsbaum  
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France.

Matthieu Roy  
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France.

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 topology approach to distributed computing, and yields concrete algorithms and impossibility results.

I. INTRODUCTION

The family of robot convergence tasks plays a fundamental role in the theory of fault-tolerant distributed computing [18]. It is used to prove the wait-free computability theorem [22] 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 [14], but robots can move towards points which are arbitrarily close to each other, using a solution to approximate agreement [13]. Robots can also converge in Euclidean space; see [25] 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. [6], [19], [20], [24], [26].

In a robot convergence problem, a collection of $n$ robots are placed in points of a given space, $K$, which can be of any shape and dimension. The robots know $K$, but they do not know on which point of $K$ 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 $K$, 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 [3]. 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 $K$, and rules $\Delta$ stating restrictions on the regions on which the robot should converge. For instance, the space $K$ could be the $d$-dimensional Euclidean space as in [25], 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 2 points, they should converge to points close to each other along the line connecting the 2 points. Another example is the loop agreement task [20], where there is a given loop in the space $K$, and three given 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 $K$, as long as they end up being close to each other. Whether a specific robot convergence task is solvable depends on the space $K$ and the convergence rules $\Delta$. Arguably, the most basic (wait-free) unsolvable convergence task is 2-set agreement for three processes [11], which is an instance of loop agreement where $K$ is a cycle of three edges [20]. Stated using this terminology, 2-set agreement for 3 processes corresponds to robots starting in any of the corresponding three vertices, and having to decide on at most two of the initial vertices.

A. Robot Convergence Problems on Graphs

We are interested in studying robot convergence problems in the case the space $K$ 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 [6] 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 [16] as in *weak symmetry breaking* [9], [16], [22], renaming [2], [10] or committee decision [8]. Regarding the two problems addressed in this work, notice that graph convergence is a colorless problem, while edge covering is not.

### B. 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.

#### a) Graph Convergence:

For the case of 2 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.

#### b) Edge Covering:

For the case of 2 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 [18]. The characterization of graph convergence solvability can in principle be derived from existing theorems (e.g., Theorem 4.3.1 of the book [18]) which, roughly speaking, imply that graph convergence has a solution iff there is a continuous map from a given space to \( G \) (more details in Section V-A). Our algorithm explains 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 [22] 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 [23], and we also illustrate this aspect of the theory here. We hope our algorithms and impossibility results provide intuition and shed light on these topological theorems, illustrating why topological properties are so intimately related to distributed algorithms, and why symmetry breaking is more difficult than agreement.

### C. Related Work

Distributed algorithms for robots is a very active research area (see e.g. [12] 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, were proposed in [1], and for multiple crash failures in [7]. However, we are not aware of the use of algebraic topology techniques in the style of [18] (work about computing topological properties of a space is of a different nature e.g. [4]). In our setting gathering is impossible (Section IV) because, in contrast to the previous papers, 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 [6].

### D. Outline of the Paper

Algorithms for graph convergence are in Section II and for edge covering in Section III. The impossibility results and solvability characterizations are in Section IV. A topological perspective of our results is in Section V. Some proofs and additional material appear in the Appendix.

## II. THE GRAPH CONVERGENCE PROBLEM

We assume a standard distributed computing model where \( n \) robots, \( p_1, \ldots, p_n \), are sequential processes (state machines), that run at arbitrary speed, 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 writing and reading single-writer/multi-reader registers. We describe our algorithms using operations that can be implemented wait-free from registers, such as *snapshot* operations by which a process can read the whole shared memory in a single atomic step, or even *immediate snapshots* (see Appendix A), by which a process can write to a register and take a snapshot in a single step. We remark that this model is equivalent to a model where the processes communicate by message passing with each other, if a majority of them do not crash. See e.g. [3], [18] for additional details of these models.

After introducing the graph convergence problem, we present two optimal-time solutions to it: 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. As we shall see in Section IV, these are the only situations where graph convergence is solvable, and it is impossible to solve gathering to a single vertex in this context.

We assume robots know the graph, and can communicate with each other their current vertex positions using the shared memory.

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:
\textbf{Agreement:} The collection of decided vertices belong to a single edge of $G$.

\textbf{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 \cite{18} specialized to the case of graphs\footnote{In the case where the space is an arbitrary complex $\mathcal{K}$, the validity condition states that if the robots initial positions span a simplex, they have to decide vertices on that simplex.}, while in the graph gathering problem, the \textit{agreement} property is replaced by requiring that the decided vertices are equal. The optimality of the two algorithms in the next two sections follows from arguments similar to those in \cite{23}.

\section*{A. Graph Convergence on Trees}

Recall that the \textit{eccentricity} of a vertex $v$ is defined as the greatest distance from $v$ to any other vertex. A \textit{center} of a graph is a vertex with minimal eccentricity. The \textit{radius} of $G$ is the minimum eccentricity among the vertices of $G$ and the \textit{diameter} of $G$ is the maximum eccentricity among the vertices of $G$. Denoting the centers of a graph $G$ by $\text{center}(G)$, a tree $T$ has $|\text{center}(T)| = 1$ or $|\text{center}(T)| = 2$. If $|\text{center}(T)| = 1$, the tree is called \textit{central}. If $|\text{center}(T)| = 2$, the tree is called \textit{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.

\textbf{Remark 1:} For a tree $T$, $\text{diam}(T) = 2 \times \text{rad}(T) - 1$, if $T$ is bicentral, and $\text{diam}(T) = 2 \times \text{rad}(T)$, if $T$ is central.

\begin{algorithm}[h]
\caption{$T = (V, E)$ is an arbitrary tree. Code for robot $p_i$.}
\textbf{Function} GraphConvergenceTree($v_i, T$)
1: for $i \leftarrow 1$ to $|\text{diam}(T)| + 1$ do
2: $s_i \leftarrow \text{ImmediateSnapshot}(v_i, v_i)$
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 $v_i$
\end{algorithm}

The algorithm $\text{GraphConvergenceTree}$ (Algorithm 1) solves graph convergence on trees for any number of processes. 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 its graph convergence 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, shared memory is organized in layers, and every round has a fixed associated a fresh shared memory layer. Figure 1 depicts three possible trees robots might see in a single round (notice that trees are ordered by containment, as induced by the ImmediateSnapshot primitive).

Clearly, every non-crashed robot terminates in $\text{GraphConvergenceTree}$ as the number of rounds is fixed. If all the initial vertices already span a vertex, then each non-crashed robot $p_i$ returns its initial vertex as in every round $t_i$ contains a single vertex. Similarly, if all initial vertices span an edge $e$, then $p_i$ returns a vertex of $e$ because in every round $t_i$ is either $e$ or a vertex of $e$.

The correctness of the algorithms follows from the following lemma showing that every round roughly halves the distance between any two vertices. For simplicity, in what follows we only consider executions in which all robots return 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.

\begin{lemma}
For $j = 1, \ldots, \lceil\text{diam}(T)\rceil + 1$, 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, $\text{diam}(T_{j+1}) \leq \text{diam}(T_j) + 1/2$.
\end{lemma}

\textbf{Proof.} The snapshots of the robots in the $j$-th round are ordered by containment, and hence there are sets $S_1 \subset S_2 \subset \ldots \subset S_k = \{v_1^j, \ldots, v_n^j\}$, 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, every robot whose vertex is in $S_l$ has its snapshot equal to $S_l$ (see Appendix A). 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, $\text{center}(R_1), \ldots, \text{center}(R_k)$ (see Figure 1). Hence, $T_{j+1}$ is the smallest subtree of $T_j$ containing $\text{center}(R_1), \ldots, \text{center}(R_k)$. Let $R_{\ell}$ be the smallest subtree of $T_j$ containing the first $m$ centers, $\text{center}(R_1), \ldots, \text{center}(R_m)$ (thus $R_0 = T_{j+1}$). By induction of $l$, one can show that $\text{diam}(R_l) \leq \text{rad}(R_0)$. The base case $l = 1$ is obvious as $R_1$ is a single vertex. We have already assumed the claim holds for $l$, to show it holds for $l + 1$, it is enough to observe that the distance between any pair $\text{center}(R_s)$ and $\text{center}(R_t)$, distinct both from $\text{center}(R_k)$, is at most $\text{rad}(R_0)$, by induction hypothesis, which is at most $\text{rad}(R_{l+1})$, by definition; and the distance from $\text{center}(R_{l+1})$ to any other $\text{center}(R_s)$ is at most $\text{rad}(R_{l+1})$ by definition.

![Fig. 1. Optimal graph convergence on a tree](image-url)
Thus, when \( l = k \), we have that \( \text{diam}(T_j + 1) \leq \text{rad}(T_j) \). By Remark 1, it follows that \( \text{rad}(T_j) \leq (\text{diam}(T_j) + 1)/2 \), and hence \( \text{diam}(T_j + 1) \leq (\text{diam}(T_j) + 1)/2 \).

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

**Theorem 1:** For any tree \( T \), algorithm \textbf{GraphConvergenceTree} solves graph convergence on \( T \) for \( n \geq 2 \) robots.

### B. 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 \textbf{GraphConvergenceTree}, tailored for this case.

**Algorithm 2** \( G = (V, E) \) is a connected graph. Code for \( p_i \).

**Preprocessing:** \( V(u,v) \in V^2 \) pre-compute a shortest path from \( u \) to \( v \)

**Function GraphConvergenceTwoRobots** (Algorithm 2).

\[
\begin{align*}
1: & \quad s_i \leftarrow \text{ImmediateSnapshot}(1, v_i) \\
2: & \quad P_i \leftarrow ⊥ \\
3: & \quad \text{if } |s_i| \neq 2 \text{ then} \\
4: & \quad P_i \leftarrow \text{precomputed path between vertices in } s_i \\
5: & \quad \text{end if} \\
6: & \quad \text{for } v_i \leftarrow 2 \text{ to } [\text{diam}(G)] + 1 \text{ do} \\
7: & \quad s_i \leftarrow \text{ImmediateSnapshot}(r_i, (v_i, P_i)) \\
8: & \quad \text{if } |s_i| \neq 2 \text{ then} \\
9: & \quad \text{if } P_i = ⊥ \text{ then} \\
10: & \quad P_i \leftarrow \text{other robot path in } s_i \\
11: & \quad \text{end if} \\
12: & \quad t_i \leftarrow \text{smallest subpath of } P_i \text{ containing } s_i \\
13: & \quad v_i \leftarrow \text{center of } t_i \\
14: & \quad \text{end if} \\
15: & \quad \text{end for} \\
16: & \quad \text{return } v_i
\end{align*}
\]

In \textbf{GraphConvergenceTwoRobots} (Algorithm 2), we assume the robots know (or 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 a snapshot to communicate their input vertex. 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 snapshots, it cannot be that both \( P_i \) and \( P_j \) are equal to \( ⊥ \). Then, the robots proceed similarly as in \textbf{GraphConvergenceTree}: they move to the center of the subpath between their current vertices.

**Theorem 2:** For any connected graph \( G \), algorithm \textbf{GraphConvergenceTwoRobots} solves graph convergence on \( G \) for two robots.

### III. 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 simple cycle of odd length. Our edge covering solution, \textbf{EdgeCoveringTwoRobots} (Algorithm 3), is an adaptation of the \textbf{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.

#### A. Edge Covering for Two Robots on Connected Graphs with Odd Length Cycles

Algorithm \textbf{EdgeCoveringTwoRobots} needs a deterministic preprocessing phase that, for any pair of vertices \( v_i \) and \( v_j \) (possibly equal), computes a simple odd length path from \( v_i \) to \( v_j \). 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 \( C = w_1, w_2, \ldots, w_n, w_1 \) of \( G \) of odd length. 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.

Now, as in \textbf{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 \( ⊥ \)). Then, the robots move to a center of the subpath between its current vertices but each of them picks a center that guarantees that the new current vertices are at odd distance (see Figure 2).

**Algorithm 3** \( G = (V, E) \) contains an odd length simple cycle.

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

**Function EdgeCoveringTwoRobots** (Algorithm 3).

\[
\begin{align*}
1: & \quad s_i \leftarrow \text{ImmediateSnapshot}(1, v_i) \\
2: & \quad P_i \leftarrow ⊥ \\
3: & \quad \text{if } |s_i| \neq 2 \text{ then} \\
4: & \quad P_i \leftarrow \text{precomputed path between vertices in } s_i \\
5: & \quad \text{end if} \\
6: & \quad \text{for } v_i \leftarrow 2 \text{ to } [\text{diam}(G)] + 1 \text{ do} \\
7: & \quad s_i \leftarrow \text{ImmediateSnapshot}(r_i, (v_i, P_i)) \\
8: & \quad \text{if } |s_i| \neq 2 \text{ then} \\
9: & \quad \text{if } P_i = ⊥ \text{ then} \\
10: & \quad P_i \leftarrow \text{other robot path in } s_i \\
11: & \quad \text{end if} \\
12: & \quad v_i \leftarrow \text{vertex of the other robot in } s_i \\
13: & \quad t_i \leftarrow \text{smallest subpath of } P_i \text{ containing } v_i \text{ and } v_j \\
14: & \quad v_i \leftarrow \text{center of } t_i \text{ such that the length of the subpath of } t_i \text{ from that center to } v_j \text{ is odd} \\
15: & \quad \text{end if} \\
16: & \quad \text{end for} \\
17: & \quad \text{return } v_i
\end{align*}
\]

**Lemma 2:** Let \( P \) be the precomputed path between the initial vertices \( v_1 \) and \( v_2 \). For \( j = 1, \ldots, [\text{diam}(G)] + 1 \), 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, \(|P_{j+1}|\) 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 \(|P_2|\) is odd. Note that \( P_j \) is a bicentral tree. Let \((u, v)\) be the edge containing its two centers. Let \( s_1 \) and \( s_2 \) be the 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 \( v_1^1 \) and \( v_2^1 \). 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 \( v_1^1 \) is odd (resp. even), then the length of the subpath from \( u \) to \( v_2^1 \) is even (resp. odd). It similarly happens with \( v \) (see Figure 2). 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 \( v_j^1 \) while \( s_2 \) contains \( v_1^1 \) and \( v_2^1 \). Then \( v_1^{j+1} = v_j^1 \) and \( v_2^{j+1} = v_1^1 \) is the center of \( P_j \) such that the length of the subpath \( Q \) from that center to \( v_1^1 \) is odd (as explained in the previous case, only one center has that property). 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.

\( \square \)

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.

### IV. 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 graph convergence or edge covering solution for \( G \) can be used to solve wait-free binary consensus, which is known to be impossible [3], [17]. The following style of proof is known since the first (1-resilient) task characterization results [5].

**Lemma 3:** If \( G \) is disconnected, then graph convergence and edge covering on \( G \) is 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 [3], [17]. In the binary consensus problem, 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.

A similar proof shows the following

**Lemma 4:** If \( G \) has at least two vertices, then graph gathering on \( G \) is impossible for any number of robots \( n \geq 2 \).

The following lemma shows that cycles are an obstacle for solving graph convergence and covering when the number of robots is greater or equal than three. The structure of the proof is similar to the previous one: if there is a solution to a graph with cycles, then one can solve the well-known set agreement problem [11], which has been proved to be unsolvable (see [18]).

**Lemma 5:** If \( G \) has a cycle, then graph convergence and edge covering on \( G \) is 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 when \( A \) solves edge covering is the same). We use \( A \) to solve 2-set agreement for three processes hence reaching a contradiction. In the \((n - 1)\)-set agreement problem [11], which has been proved to be unsolvable.

In the proof 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.

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 Figure 3). By...
Algorithm 4 Code for process $p_i$.

Function SetAgreement($i$)
1: if $i = 3$ then
2: $x_i \leftarrow v_3$
3: else
4: $x_i \leftarrow B$.GraphConvergence($v_1, P$)
5: end if
6: $y_i \leftarrow A$.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

Fig. 3. Mapping vertices of the cycle $C$ to values for Set Agreement

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 2-set agreement for three robots.

Algorithm 4 solves 2-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 Figure 3. 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 4 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 $p_1$ participates in an execution. If $p_1 = p_3$, then it invokes $A$ with input $v_3$, and consequently obtains $v_3$ from it, by validity property of edge covering. Thus, $p_3$ returns 3. If $p_1$ is either $p_1$ or $p_2$, it invokes $B$ with input $v_3$, and consequently obtains $v_3$ from $B$, by validity of graph convergence; and hence $p_1$ 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 $v_3$, hence it obtains $v_3$. Thus, $p_i$ and $p_j$ invoke $A$ with inputs $v_i$ and $v_j$, respectively, and thus they obtain 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 $j$.
  - 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.

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 4 solves 2-set agreement for 3 processes, which, as already explained, is a contradiction, from which follows that such an algorithm $A$ cannot exist.

Lemmas 3 and 5 together with Theorems 1 and 2 in Section II completely characterize the solvability of graph convergence.

Theorem 4: For two robots, graph convergence on $G$ 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 edge covering 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.

Lemma 6: 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 [16]. 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 [18]).
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 $e(w)$, 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 $e(v)$. 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 6. But this contradicts Lemma 5.

Lemma 7: For any graph $G$, there is no algorithm that solves edge covering on $G$ for three or more robots.

Proof. Suppose by contradiction that there is an algorithm $A$ that solves edge covering on $G$. As observed by Remark 2 in the proof of Lemma 5, $A$ solves edge covering on $G$ for two robots. Thus, $G$ has a cycle, by Lemma 6. Lemma 5 implies that $A$ cannot exist. □

Finally, from Lemmas 6 and 7 and Theorem 3 in Section III, we derive a full characterization for the solvability of edge covering.

Theorem 5: 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 $G$.

V. A TOPOLOGICAL PERSPECTIVE

The topological approach to distributed computing [18] has been useful to understand the nature of fault-tolerant distributed computing, to prove impossibility results and to understand why in some cases there exists a distributed algorithm to solve a problem. In this section we briefly discuss a topological perspective of graph convergence and edge covering.

A. The Topology of Graph Convergence

Our graph convergence problem is a special case of the robot convergence task defined in [18], 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 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 $\sigma$ of $\mathcal{I}$ is called a simplex, and its dimension is $|\sigma|-1$. Thus, $\sigma \subseteq V, |\sigma| \leq n$. For each simplex $\sigma$ in $\mathcal{I}$, representing possible simultaneously starting vertices of $G$, $\Delta$ encodes the convergence rules. Namely, $\Delta(\sigma)$ is a subgraph of $G$ where the robots may end up, if their initial positions are in $\sigma$. Thus, $\Delta(\sigma) = \sigma$ if $\sigma$ is either a vertex or an edge of $G$, and otherwise, $\Delta(\sigma) = G$. The following is from the book [18](page 88), see also [21].

Theorem 6 (4.3.1 [18]): 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 $f : |\mathcal{I}| \rightarrow |G|$ carried by $\Delta$.

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 $\sigma$ of $\mathcal{I}$. The continuous map $f$ respects the input/output specification of the task, $\Delta$, in the sense that for each simplex $\sigma \in \mathcal{I}$, $f(|\sigma|) \in |\Delta(\sigma)|$. In particular, $f(v) = v$, and $f(e) = e$ for any edge $e$. But $f$ may send a simplex $\sigma$ 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.

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 $\sigma$, because $\sigma$ 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.

Let $G$ be a graph with a cycle $C = v_1, v_2, \ldots, v_p, v_1$ (see Figure 4 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 not 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 colorless immediate snapshot algorithm and subdivision, as explained in Appendix B. A round of colorless immediate-snapshots subdivides produces a barycentric subdivision of
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 \( G = (V, E) \), consists of all pairs of vertices of the form \( (id, v) \), where \( v \in V \) and \( id \in \{A, B\} \). An edge \( (x, x') \) belongs to \( E \) iff \( x = (id, v) \) and \( x' = (id', v') \) such that \( (v, v') \in E \) (\( v \neq v' \)), and \( id \neq id' \). A vertex \( (id, v) \) represents the situation where robot \( id \) ends in vertex \( v \).

Recall that robots have to end in adjacent vertices of \( G \). Figure 6 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 \( (\tilde{I}, \tilde{G}, \tilde{\Delta}) \), 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 \( x = (A, v) \) and \( x' = (B, v') \), 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, \( \tilde{\Delta}( (id, v)) = \{(id, v)\} \), for every vertex \( (id, v) \). Second, if the robots start in an edge, they stay there, i.e., \( \forall (v, v') \in E \):

\[
\tilde{\Delta}( (A, v), (B, v')) = \{(A, v), (B, v'), (A, v')\}
\]

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., \( \forall (v, v') \notin E, \tilde{\Delta}( (A, v), (B, v')) = G \).

The wait-free solvability theorem [22] 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 \( \delta \) from \( X \) to \( \tilde{G} \), such that \( \delta \) preserves ids and edge adjacencies, and \( \delta \) 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 \( (x, x') \in \tilde{I} \), \( x = (A, v) \) and \( x' = (B, v') \), it is known that the distributed algorithm induces a subdivision of \( (x, x') \), essentially creating a path, where each edge of the path represents the final states of the robots in one of the possible executions starting in \( (x, x') \). More details in Appendix A.

**Theorem 7 (3.1 [22]):** 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 \( \delta : X(\mathcal{I}) \rightarrow \mathcal{O} \) s.t. for each \( \sigma \in X(\sigma) \), \( \delta(\sigma) \) is carried by \( \Delta(\sigma) \).

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 \( G \) alternating vertices with id \( A \) and \( B \).

Notice that the formal specification of the edge covering problem as a triple \( (\tilde{I}, \tilde{G}, \tilde{\Delta}) \), 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 [22] is about general dimension combinatorial topology simplicial complexes. Roughly speaking, edge covering (and in general non-colorless tasks) is more difficult than graph convergence, because of a seemingly innocuous, but surprisingly “difficult” requirement: the simplicial decision map \( \delta \) is color-preserving. Namely, while in a colorless task, robots can always adopt each other outputs (\( \delta \) can send simplices to lower dimensional output simplices), this is not possible in general tasks (\( \delta \) sends a final state’s algorithm simplex to an output simplex of the same dimension).

![Fig. 4. Impossibility of graph convergence on cycle \( C = v_1, v_2, v_3, v_4, v_1 \) by 3 processes](image)

![Fig. 5. Top: Solving a tree by 3 processes. Bottom: Trying to solve a tree in one round](image)

![Fig. 6. \( G \) and colored graph \( \tilde{G} \), with input \( \tilde{I} \) and views after one round \( X(\tilde{I}) \), for robots \( A, B \)](image)
VI. CONCLUSION

In this paper we study two robot convergence tasks in an asynchronous read/write shared memory crash-prone system, where the base space is a finite graph. The tasks 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.

In practice, robots have no shared memory to communicate with each other; typically, they communicate through a weaker communication medium. However, we believe that our results will help in understanding the limits of what robots can compute in an asynchronous crash-prone environment.

ACKNOWLEDGEMENTS

The 1st and 2nd authors are supported by UNAM-PAPIIT IA101015 and IN107714. The 1st author is also supported by the project CONACYT C394/2016/271602. The 2nd author also received support from ECOS-CONACYT and LAISLA. The 3rd author is supported by CPSlab project H2020-ICT-644400.

REFERENCES


APPENDIX

A. Topology of Colored Computation and Immediate Snapshots

Here we provide more details about the topology perspective in Section V-B, related to general tasks such as edge covering. In Section B we discuss how these ideas are simplified, when the problem is colorless, such as graph convergence.

We have discussed in Section V-B the wait-free computability theorem [22], that describes when a task \((I, O, \Delta)\) has a wait-free read/write protocol. We reproduce here Figure 11.7 from [18] for the reader’s convenience, where the theorem is illustrated.

Fig. 7. The wait-free computability theorem, Fig 11.7 from [18].
The theorem states that $(I, O, \Delta)$ has a wait-free read/write protocol if and only if there is a chromatic subdivision $X$ of $I$ and a color-preserving simplicial map $\delta : X(I) \to O$ such that for each simplex $s' \in X(s)$, $\delta(s')$ is carried by $\Delta(s)$. In Figure 7 we see an input simplex $s$ in $I$, at the top, and how a distributed algorithm subdivides it into $X(s)$, part of the subdivision $X(I)$. Each vertex of $X(s)$ represents the final state of a process in one of the executions starting in $s$. In the figure, $s'$ is an example of a set of process' final states in one of the possible executions starting in $s$. The decision map $\delta$ defines the decisions taken by the set of processes in $s$'. These decisions are $\delta(s')$ thus, for each $s' \in X(s)$, the decisions $\delta(s')$ should belong to $\Delta(s)$, as the rule $\Delta$ states the legal outputs when starting in each initial simplex $s$.

We now describe the recursive immediate snapshot algorithm of [15], both to illustrate how a distributed algorithm is able to produce a chromatic subdivision of a complex, and to show how immediate snapshots are simulated using only read and write operations. The presentation follows closely that paper, and is repeated here for the convenience of the reader. Then we present a colorless version of this algorithm, suited for robot convergence tasks.

**Algorithm 5** Code for robot/process $p_i$ with input $v_i$. Initially $r_i = n$

Function RecursiveIS($v_i, r_i$)
1: $s_i \leftarrow$ 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

The algorithm is in Figure 5. Each process $p_i$ with input $v_i$, writes the pair $(p_i, v_i)$, to the shared memory, 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 $n$, it returns this set as a view and terminates the algorithm; else, the process calls the algorithm recursively. We stress that in each recursive call the processes communicate with each other via a new array of single-writer/multi-reader registers, that is used only in that recursive call.

When three processes invoke the algorithm with inputs $1, 2, 3$ respectively, the simplicial complex obtained is in Figure 8. Each triangle (2-simplex) represents a possible execution, and the vertices are labeled with the views 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 3 processes and terminate without a recursive call, while the first process executes two recursive calls and ends up seeing only its own input.

The algorithm guarantees that the views of the processes, at the end of an execution, satisfy the following properties. Denoting by $sm_i$, the view of process $p_i$ at the end of the algorithm, we have: Self-inclusion: $\forall i : i \in sm_i$. Containment: $\forall i,j : sm_i \subseteq sm_j \lor sm_j \subseteq sm_i$. Immediacy: $\forall i,j : i \in sm_j \Rightarrow sm_i \subseteq sm_j$.

The immediacy property can be rewritten as $\forall i,j : (i \in sm_j \land j \in sm_i) \Rightarrow sm_i = sm_j$. Thus, concurrent invocations of the task obtain the same view. A snapshot task is required to satisfy only the first two properties.

**B. Colorless Snapshots**

The colorless version of algorithm RecursiveIS simply discards the ids of the processes. It assumes any of the $n \geq 3$ robots can start on any of possible inputs, $v_i$ for $p_i$. It executes the same code, except that when a robot produces an output, it returns only a set values seen. In Figure 9 the possible triangles are represented, when the algorithm is executed with input vertices $1, 2, 3$, and any number of processes greater than two. Each vertex, or edge, or triangle, represents the possible outputs of the processes. For example, the edge with vertices $\{1\}$ and $\{1, 2, 3\}$ is possible in an execution where the processes with input 1 see only themselves, and the other processes see all three inputs. Notice that this figure is obtained from Figure 8 by discarding ids from vertices, and merging vertices which have the same associated set.