The Computational Power of Simple Protocols for Self-Awareness on Graphs[☆,☆☆]

Ioannis Chatzigiannakis[a,b], Othon Michail[a,b,*], Stavros Nikolaou[c], Paul G. Spirakis[a,b]

[a] *Computer Technology Institute & Press "Diophantus" (CTI), Patras, Greece*
[b] *Computer Engineering and Informatics Department (CEID), University of Patras*
[c] *Department of Computer Science, Cornell University, Ithaca NY*

Abstract

We explore the capability of a network of extremely limited computational entities to decide properties about itself or any of its subnetworks. We consider that the underlying network of the interacting entities (devices, agents, processes etc.) is modeled by an *interaction graph* that reflects the network's connectivity. We examine the following two cases: First, we consider the case where the input graph is the whole interaction graph and second where it is some subgraph of the interaction graph given by some preprocessing on the network. In each case, we devise simple graph protocols that can decide properties of the *input graph*. The computational entities, that are called *agents*, are modeled as finite-state automata and run the same global graph protocol. Each protocol is a fixed size grammar, that is, its description is independent of the size (number of agents) of the network. This size is not known by the agents. We present two simple models (one for each case), the *Graph Decision Mediated Population Protocol* (*GDMPP*) and the *Mediated Graph Protocol* (*MGP*) models, similar to the Population Protocol model of Angluin *et al.*, where each network link (edge of the interaction graph) is characterized by a *state* taken from a finite set. This state can be used and updated during each interaction between the corresponding agents. We provide some example protocols and some interesting properties for the two models concerning the computability of graph languages in various settings (disconnected input graphs, stabilizing input graphs). We show that the computational power within the family of all (at least) weakly-connected input graphs is fairly restricted. Finally, we give an exact characterization of the class of graph languages decidable by the MGP model in the case of complete interaction graphs: it is equal to the class of graph languages decidable by a nondeterministic Turing Machine of linear space that receives its input graph by its adjacency matrix representation.

Keywords:
population protocol, diffuse computation, finite-state agent, intermittent communication, stable computation, passive mobility, graph property, self-awareness, complexity

1. Introduction

Consider an application that allows users to make voice calls over the Internet by executing a software agent. The software agents are organized in a peer-to-peer overlay network. Suppose that in order to

[☆]This work has in part been supported by the EU (European Social Fund - ESF) and Greek national funds through (i) the Operational Programme "Education and Lifelong Learning" (EdLL), under the title "Foundations of Dynamic Distributed Computing Systems" (FOCUS), and (ii) the National Strategic Reference Framework (NSRF) (Regional Operational Programme - Western Greece) under the title "Advanced Systems and Services over Wireless and Mobile Networks" (number 312179).

[☆☆]Some preliminary versions of the results in this paper have also appeared in [CMS09b, CMS10, CMNS11].

[*]Corresponding author (Telephone number: +30 2610 960200, Fax number: +30 2610 960490, Postal Address: Computer Technology Institute & Press "Diophantus" (CTI), N. Kazantzaki Str., Patras University Campus, Rio, P.O. Box 1382, 26504, Greece).

Email addresses: ichatz@cti.gr (Ioannis Chatzigiannakis), michailo@cti.gr (Othon Michail), snikolaou@cs.cornell.edu (Stavros Nikolaou), spirakis@cti.gr (Paul G. Spirakis)

achieve certain quality of service levels, statistical data have shown that each agent must have at most k concurrent incoming voice-traffic flows. We assume that the software agents are quite limited: each agent has a constant number of bits of memory and two agents can communicate only when they are required to forward voice traffic. We also assume that agents have access to a global storage in which very limited information can be stored. In this setting, software agents have no control over their interactions: users come and go, and requests for voice calls are made by the users. We assume that the underlying pattern of interactions guarantees a fairness condition on the interactions: every pair of agents in the network is repeatedly allowed to exchange control information for their users to have voice calls.

Under these assumptions, there is a simple protocol ensuring that every agent eventually learns the answer to whether the number of active incoming traffic flows surpasses k for any agent. Each agent stores a counter ($0, 1, \ldots, k+1$, where k is a constant) signifying its knowledge about the previous number. The global storage service stores 1 bit for each possible pair of interacting agents. Initially, all agents have their counter set to 0 and each bit of the global storage is set to 1. When two agents interact, e.g., to forward voice traffic, if the bit corresponding to this interaction is 1, then the receiving agent increases its counter by one and the corresponding bit is set to 0. If the bit is 0, then nothing happens (the incoming flow has been already counted). If the counter of any of the interacting agents reaches the value $k+1$, then an alert state is propagated to the population and eventually (after a finite number of interactions), all agents are informed of the existence of a potential bottleneck in the network and can take appropriate actions.

Now consider the question of whether the overlay network is fully connected or whether it is partitioned. Is there a protocol to answer these questions, without any assumptions about the size of the network? In this work, we focus on the following question: what properties of the underlying network can be computed by systems consisting of a population of computationally restricted, interacting entities? We are interested in the levels of knowledge that such systems can achieve regarding their own properties and characteristics, in other words, to what extent they can become *self-aware*. Such knowledge can be used to optimize the system's overall behavior w.r.t. resource usage, performance, etc., and to adapt to changing conditions concerning internal changes (e.g., a topology change) and context changes (e.g., a modification of user behavior).

2. Previous Work

As Angluin *et al.* observed in [AAD$^+$04], "Most work in distributed algorithms assumes that agents in a system are computationally powerful, capable of storing nontrivial amounts of data and carrying out complex calculations. But in systems consisting of massive amounts of cheap, bulk-produced hardware, or of small agents that are tightly constrained by the systems they run on, the resources available at each agent may be severely limited." In the same work, they introduced the *Population Protocol* (*PP*) model, which captures the notion of computation by a population of extremely limited communicating agents. In this model, the system consists of a collection of agents, represented as finite-state machines. The agents exchange information via pairwise interactions, which they are unable to predict or control. Via these interactions, the system organizes its computation and provides complex behavior as a whole. In [AAD$^+$06, AAER07], the computational power of the model was studied and has been proved to be exactly the class of *semilinear predicates*, consisting of all predicates definable by first-order logical formulas of Presburger arithmetic (see, e.g., [GS66]). The capability of the model to decide graph properties of restricted interaction graphs was explored in [AAC$^+$05]. For introductory texts to the area of population protocols the interested reader is referred to [AR07, Spi10, ÀCD$^+$11, MCS11b].

In an attempt to enhance the basic model, a variation was proposed in [CMS09a], called the *Mediated Population Protocol* (*MPP*) model, in which the population is also capable of storing constant size information for each pairwise interaction. This extension is fitting for modeling more complex systems where relations are formed between the interacting entities and the information generated concerning these relations is required in each interaction of the respective entities. A modern popular example are the various social networks where the interacting entities are the users (e.g., friends on Facebook) and their interactions generate information that characterize their relationship. The information concerning these relationships is stored in the system and can be used in each interaction. Biological and artificial neural networks also concern networks of interconnected simple processing elements that exhibit complex global behavior determined

by the connections between them. These connections (synapses) can store parameters called "weights" that influence the outcome of the computations. In [CMN+10], it was proven that, in complete graphs, the MPP model is computationally equivalent to a NTM [1] of $\mathcal{O}(n^2)$ space that computes symmetric predicates.

Several other extensions of the basic model have been proposed in the relevant literature to more accurately reflect the requirements of practical systems. In the *Community Protocol* model of Guerraoui and Ruppert [GR09] each agent has its own unique id and can store up to a constant number of other agents ids. Agents are only allowed to compare ids, that is, no other operation on ids is permitted. The community protocol model was proven to be extremely strong: the corresponding class consists of all symmetric predicates in $\mathbf{NSPACE}(n \log n)$, where n is the community size. The *Passively mobile Machines* (PM) model [CMN+11a, CMN+11b] made the assumption that each agent instead of being an automaton is a Turing Machine with unbounded memory. Then the authors studied computations upper-bounded by plausible space limitations. They focused on complete interaction graphs and established $\log \log n$ and $\log n$ as two important computability thresholds: at the former semilinearity ends and at the latter begins the possibility to arrange the whole population into a distributed TM.

In a slightly different direction, the *Probabilistic Population Protocol* model was proposed [AAD+06], in which the scheduler selects randomly and uniformly the next pair to interact. This random scheduling assumption allowed the study of performance (see e.g. [AAE08]). A generic definition of probabilistic schedulers along with a collection of new fair schedulers were provided in [CDF+09]. There the authors showed the need for the protocols to adapt when natural modifications of the mobility pattern occur was emphasized. [BCC+09, CS08] considered a huge population hypothesis (population going to infinity), and studied the dynamics, stability and computational power of probabilistic population protocols by exploiting the tools of continuous nonlinear dynamics.

A lot of similarities can be observed between this work and the one in [GMM04] in which the recognition problem is studied. The problem regards the computation of topological information on a network of processes. The authors explore the characterization of graph classes using the notion of local computation which resembles the notion of interactions in PPs and assuming implicit termination (not all nodes may be aware that a termination state has been reached) which also resembles of the PPs' output stabilization behavior. One of the differences of this work with ours (and PPs in general) is that each process in [GMM04] can be much stronger computationally (w.r.t. memory) than a finite-state automaton which is the case for our models' agents.

The notion of self-awareness in distributed systems has been studied in the context of self-organization and autonomic systems. Self-stabilization was one of the first self-$*$ properties discussed in the context of distributed systems and was introduced by Dijkstra [Dij74]. Since then, significant progress has been made in self-stabilization [Dol00] and in self-$*$ properties in general. In [GMK02], the authors examine the feasibility of using architectural constraints as the basis for the specification, design and implementation of self-organizing architectures for distributed systems. In an attempt to provide a roadmap for further research, in [HKLPR03], the authors try to identify the new challenges of Context-Aware-Services (CAS) management in ubiquitous environments, where the necessity of self-management is urgent. In [VvS03], following a more experimental approach, the authors proposed an epidemic protocol based on the simple Newscast protocol for managing routing tables of DHT-based peer-to-peer networks in a completely distributed and scalable way, with no need for external administration and with very high fault-tolerance.

3. Our Results - Roadmap

In Section 4, we give a formal definition of the GDMPP model in which we consider that the interaction graph itself is the input graph of the protocol. In Section 5, we focus on weakly connected interaction graphs. We prove that the class of computable graph properties is closed under complement, union, and intersection operations. Node and edge parity, bounded out-degree by a constant, existence of a node with

[1] As usual in CS literature, we abbreviate a "Turing Machine" by "TM" and by "NTM" when we want to emphasize that the TM is Nondeterministic.

more incoming than outgoing neighbors, and existence of some directed path of length $k = \mathcal{O}(1)$ are some examples of properties whose computability is proven. We also provide a protocol computing the graph language 2CYCLE, consisting of all weakly connected interaction graphs that contain some 2-cycle. Then, in Section 5.4 the existence of symmetry in two specific interaction graphs is revealed and is exploited to prove that if we restrict our protocols to stabilize their states, then 2CYCLE is not computable. In Section 10, we focus on the universe of all possible interaction graphs, containing also the disconnected ones. In this case (see Theorem 13), we prove that any nontrivial graph language (we exclude both the empty language and its complement) is not computable by the GDMPP model. As a corollary we get that GDMPPs cannot compute connectivity (Corollary 2). In Section 6, we try to approach the exact computational power of the GDMPP model by focusing on the computational capabilities of MPPs (which are practically a more general version of GDMPPs since they also accept input) on connected graphs. There we allow labels on the vertices of the input graph taken from a finite set and assigned by the network initialization function and we show that in terms of predicates on the multisets of labels only semilinear predicates can be computed. In Section 7, we additionally allow disconnected interaction graphs and arrive at a general impossibility result.

In Section 8, we formalize the case where protocols perform computations in subgraphs of their interaction graph denoted by some network initialization function and provide an example protocol illustrating the computation of the extended model which we name MGP model. We then (Section 9) present some fundamental properties of the new model. In particular, we extend (Sec. 9.1) the MGP model to allow the input graph to oscillate for a finite number of steps. This extension then allows us (Sec. 9.2) to compose protocols. In Sec. 10, we present a protocol (Sec. 10.1) that decides whether the input graph is connected and we then extend this idea (Sec. 10.2) and show that the new model is able to compute properties of disconnected input graphs, something that neither the PP nor the GDMPP model are capable of. In Sec. 11, we give an exact characterization of the computational power of the MGP model, which is the class of all graph properties decidable by a NTM of linear space that takes as input the adjacency matrix of the input graph. Finally, in Sec. 12, we conclude and discuss some future research directions.

4. The Graph Decision Mediated Population Protocol model

A *Graph Decision Mediated Population Protocol* (GDMPP) $\mathcal{A}$ consists of a *binary output alphabet* $Y = \{0, 1\}$, a finite set of *agent states* Q, an *output function* $O : Q \to Y$ mapping agent states to outputs, a finite set of *edge states* S, and a *transition function* $\delta : Q \times Q \times S \to Q \times Q \times S$. If $\delta(a, b, s) = (a', b', s')$ we call $(a, b, s) \to (a', b', s')$ a *transition*, and we define $\delta_1(a, b, s) = a'$, $\delta_2(a, b, s) = b'$ and $\delta_3(a, b, s) = s'$.

We assume that all agents are initially in an *initial agent state* $q_0 \in Q$ and all edges in an *initial edge state* $s_0 \in S$. A *graph universe* (or *graph family*) is any set of interaction graphs. We denote by $\mathcal{H}$ the graph universe consisting of all possible interaction graphs of any finite number of nodes greater or equal to 2 (we do not allow the empty graph, the graph with a unique node and infinite graphs) and by $\mathcal{G}$ the subset of $\mathcal{H}$ containing the weakly connected ones. All the following definitions hold w.r.t. some fixed graph universe $\mathcal{U}$. A *graph language* L is a subset of $\mathcal{U}$ containing interaction graphs that possibly share some common property., e.g. $L = \{G \in \mathcal{U} \mid G \text{ contains a directed hamiltonian path}\}$. A graph language L is said to be *nontrivial* if $L \neq \emptyset$ and $L \neq \mathcal{U}$.

A GDMPP runs on an interaction graph $G = (V, E)$, where V is a population of $|V| = n$ agents and E is an irreflexive binary relation on V. The graph on which the protocol runs is considered as the *input graph* of the protocol. The input graph of a GDMPP may be any $G \in \mathcal{U}$.

A *network configuration* (or simply *configuration*) is a mapping $C : V \cup E \to Q \cup S$ specifying the agent state of each agent in the population and the edge state of each edge in the interaction graph. Let C and C' be network configurations, and let u, v be distinct agents. We say that C goes to C' via encounter $e = (u, v)$, denoted $C \xrightarrow{e} C'$, if $C'(u) = \delta_1(C(u), C(v), C(e))$, $C'(v) = \delta_2(C(u), C(v), C(e))$, $C'(e) = \delta_3(C(u), C(v), C(e))$, and $C'(z) = C(z)$ for all $z \in (V - \{u, v\}) \cup (E - \{e\})$. Encounters are ordered, that is, the participating agents have distinct roles in each interaction. We call u the *initiator* and v the *responder* of the encounter (interaction). We say that C can go to C' in one step, denoted $C \to C'$, if $C \xrightarrow{e} C'$ for some encounter $e \in E$. We write $C \xrightarrow{*} C'$ if there is a sequence of configurations $C = C_0, C_1, \ldots, C_t = C'$, such that $C_i \to C_{i+1}$ for all i, $0 \leq i < t$, in which case we say that C' is *reachable* from C.

An *execution* is a finite or infinite sequence of network configurations $C_0, C_1, C_2, \ldots$, where C_0 is an initial configuration and $C_i \to C_{i+1}$, for all $i \geq 0$. The interactions are chosen by an adversary who is not a part of the protocol and can make any scheduling assumption on the interaction pattern as long as it keeps the execution fair. Fairness is a restriction imposed on the adversary to prevent it from avoiding a possible step forever. There are various notions of fairness for the protocols we study. In [FJ06], various fairness conditions are defined for population protocols, such as strong global and local fairness as well as their weak variations. In this work, we use the notion of *strong global fairness*, according to which an infinite execution is *fair* if for every pair of configurations C and C' such that $C \to C'$, if C occurs infinitely often in the execution, then so does C'. A *computation* is an infinite fair execution.

At any point during the execution of a GDMPP, each agent's state determines its output at that time. The output of any agent u under configuration C is $O(C(u))$. Note also that the *code* of any GDMPP is of *constant size* (since Q and S are finite) and, thus, can be stored in each agent (device) of the population. Due to the constant size descriptions of GDMPPs, the protocols we study are *uniform* (independent of the population size) and *anonymous* (agents can't store unique identifiers).

Definition 1. *Let L be a graph language consisting of all $G \in \mathcal{U}$ for which, in any computation of a GDMPP $\mathcal{A}$ on G, all agents eventually output* 1*. Then L is* the language stably recognized by $\mathcal{A}$*. A graph language is said to be* stably recognizable *by the GDMPP model (also called $GDMPP$-*recognizable*) if some GDMPP stably recognizes it.*

Thus, any protocol *stably recognizes* the graph language consisting of those graphs on which the protocol always answers "accept", i.e. eventually all agents output the value 1 (possibly the empty language).

Definition 2. *We say that a GDMPP $\mathcal{A}$* stably decides *a graph language $L \subseteq \mathcal{U}$ (or equivalently a predicate $p_L : \mathcal{U} \to \{0,1\}$ defined as $p_L(G) = 1$ iff $G \in L$) if for any $G \in \mathcal{U}$ and any computation of $\mathcal{A}$ on G, all agents eventually output* 1 *if $G \in L$ and all agents eventually output* 0 *if $G \notin L$. A graph language is said to be* stably decidable *by the GDMPP model (also called $GDMPP$-*decidable*) if some GDMPP $\mathcal{A}$ stably decides it.*

A GDMPP $\mathcal{A}$ has *stabilizing states* if in any computation of $\mathcal{A}$, after a finite number of interactions, the states of all agents stop changing. Alternatively, we call $\mathcal{A}$ a *stabilizing output graph* GDMPP.

In some cases, a protocol, instead of stably deciding a language L, may provide a different kind of guarantee. For example, whenever it runs on some $G \in L$, it may forever remain in configurations where at least one agent is in state a, and when $G' \notin L$ no agent will remain in state a. To formalize this, we say that a GDMPP $\mathcal{A}$ *guarantees* $t : Q^* \to \{0,1\}$ *w.r.t.* $L \subseteq \mathcal{U}$ if, for any $G \in \mathcal{U}$, any computation of $\mathcal{A}$ on G eventually reaches a configuration C, s.t. for all C', where $C \xrightarrow{*} C'$, it holds that $t(C') = t(C) = 1$ if $G \in L$ and $t(C') = t(C) = 0$, otherwise.[2]

5. Weakly Connected Graphs

In this section, we study the case in which the universe of input graphs does not contain disconnected graphs. Thus, here the graph universe is $\mathcal{G}$ and, thus, a graph language can only be a subset of $\mathcal{G}$. The main reason for selecting this specific universe for devising our protocols is that, if we also allow disconnected graphs, then, as we shall see, it can be proven that no graph language is stably decidable.

5.1. Example Protocol

To illustrate our model we here present an example protocol that stably decides the graph language $P_k = \{G \in \mathcal{G} \mid G \text{ has at least one directed path of at least } k \text{ edges}\}$ for any $k > 1$.

[2] We use the standard Q^* (Kleene star) notation for denoting the set of finite-length strings of elements in Q. By assuming an ordering on V we can define configurations as strings from Q^*.

Protocol 1 *DirPath*

1: $Q = \{q_0, q_1, 1, \ldots, k\}$, $S = \{0, 1\}$,
2: $O(k) = 1$, $O(q) = 0$, for all $q \in Q - \{k\}$,
3: δ:

$$\begin{aligned}
(q_0, q_0, 0) &\to (q_1, 1, 1) \\
(q_1, x, 1) &\to (x-1, q_0, 0), \text{ if } x \geq 2 \\
&\to (q_0, q_0, 0), \text{ if } x = 1 \\
(x, q_0, 0) &\to (q_1, x+1, 1), \text{ if } x+1 < k \\
&\to (k, k, 0), \text{ if } x+1 = k \\
(k, \cdot, \cdot) &\to (k, k, \cdot) \\
(\cdot, k, \cdot) &\to (k, k, \cdot)
\end{aligned}$$

Theorem 1. (Directed Path of Constant Length) *Protocol 1 stably decides the graph language* $P_k = \{G \in \mathcal{G} \mid G \text{ has at least one directed path of at least } k \text{ edges}\}$ *for any* $k > 1$.

Proof. Intuitively, the protocol expands noncommunicating active paths (they can interact but the corresponding transitions do nothing, that's why they are not appearing in δ). The head of each path counts its length. If the length of an active path ever becomes equal to k, then a state giving 1 as output is propagated. Note that, to avoid getting stuck, the protocol keeps backtracking and even completely releasing the active paths. Fairness condition ensures that if a path of length at least k exists, then $DirPath$ will eventually find it. □

5.2. Properties of GDMPPs

We provide some properties of the GDMPP model that will prove useful in showing the stably decidability of various graph languages.

Theorem 2. *The class of stably decidable graph languages is closed under complement, union and intersection operations.*

Proof. First we show that for any stably decidable graph language L its complement $\overline{L}$ is also stably decidable. From definition of stable decidability there exists GDMPP $\mathcal{A}_L$ that decides L. Thus, for any $G \in \mathcal{G}$ and any computation of $\mathcal{A}_L$ on G all agents eventually output 1 if $G \in L$ and 0 otherwise. By complementing the output map $O_{\mathcal{A}}$ of $\mathcal{A}$ we obtain a new protocol $\overline{\mathcal{A}}$, with output map defined as $O_{\overline{\mathcal{A}}}(q) = 1$ iff $O_{\mathcal{A}}(q) = 0$, for all $q \in Q_{\mathcal{A}} = Q_{\overline{\mathcal{A}}}$, whose agents eventually output 1 if $G \notin L$ and 0 otherwise, thus stably deciding $\overline{L}$.

Now we show that for any stably decidable graph languages L_1 and L_2, $L_3 = L_1 \cup L_2$ is also stably decidable. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be GDMPPs that stably decide L_1 and L_2, respectively (we know their existence). We let the two protocols operate in parallel, i.e. we devise a new protocol $\mathcal{A}_3$ whose agent and edge states consist of two components, one for protocol $\mathcal{A}_1$ and one for $\mathcal{A}_2$. Let O_1 and O_2 be the output maps of the two protocols. We define the output map O_3 of $\mathcal{A}_3$ as $O_3(q, q') = 1$ iff at least one of $O_1(q)$ and $O_2(q')$ equals to 1, for all $q \in Q_{\mathcal{A}_1}$ and $q' \in Q_{\mathcal{A}_2}$. If $G \in L_3$ then at least one of the two protocols has eventually all its agent components giving output 1, thus $\mathcal{A}_3$ correctly answers "accept", while if $G \notin L_3$ then both protocols have eventually all their agent components giving output 0, thus $\mathcal{A}_3$ correctly answers "reject". We conclude that $\mathcal{A}_3$ stably decides L_3 which proves that L_3 is stably decidable.

By defining the output map O_3 of $\mathcal{A}_3$ as $O_3(q, q') = 1$ iff $O_1(q) = O_2(q') = 1$, for all $q \in Q_{\mathcal{A}_1}$ and $q' \in Q_{\mathcal{A}_2}$, and making the same composition as before, it is easy to see that in this case $\mathcal{A}_3$ stably decides the intersection of L_1 and L_2. □

In some cases it is not easy to devise a protocol that respects the *predicate output convention* (the predicate output convention was defined in [AAD+04] and simply requires all agents to eventually agree on

the correct output value). In such cases, we can use the following variation of the Composition Theorem (Theorem 6) of [CMS09a] that facilitates the proof of existence of GDMPP protocols that stably decide a language.

Theorem 3. *If there exists a GDMPP $\mathcal{A}$ with stabilizing states that w.r.t. to a language L guarantees a semilinear predicate, then L is GDMPP-decidable.*

Proof. Immediate from the proof of the Theorem 6 of [CMS09a]. $\mathcal{A}$ can be composed with a provably existing MPP $\mathcal{B}$ whose stabilizing inputs are $\mathcal{A}$'s agent states to give a new GDMPP $\mathcal{C}$ that stably decides L w.r.t. the predicate output convention. Note that $\mathcal{B}$ is in fact a GDMPP, since its stabilizing inputs are not real inputs (GDMPPs do not have inputs). It simply updates its state components by taking also into account the eventually stabilizing state components of $\mathcal{A}$. Thus, their composition, $\mathcal{C}$, is also a GDMPP. □

5.3. Decidable Graph Languages

We here show the stable decidability of certain graph languages by providing high-level descriptions of protocols and proving their correctness.

Theorem 4. (Node Parity) *The graph languages $N_{even} = \{G \in \mathcal{G} \mid |V(G)| \text{ is even}\}$ and $N_{odd} = \overline{N}_{even}$ are stably decidable.*

Proof. Assume that the initial agent state is 1. Then there is a protocol in [AAD$^+$04] that decides N_{even} in the case where G is complete. But, according to Theorem 4 in page 295 of [AAD$^+$04], there must exist another protocol that decides N_{even} in the general case, i.e., in any graph $G \in \mathcal{G}$. Thus, N_{even} is decidable by the population protocol model that does not use inputs and whose output alphabet is $\{0, 1\}$, and since this model is a special case of the GDM model, N_{even} is decidable. Moreover, since the class of decidable graph languages is closed under complement (see Theorem 2), it follows that $N_{odd} = N_{even}$ is also decidable. □

Theorem 5. (Edge Parity) *The graph languages $E_{even} = \{G \in \mathcal{G} \mid |E(G)| \text{ is even}\}$ and $E_{odd} = \overline{E}_{even}$ are stably decidable.*

Proof. By exploiting the closure under complement, it suffices to prove that E_{even} is decidable by presenting a GDMPP that decides it. The initial agent state is $(0, 0)$ consisting of two components, where the first one is the data bit and the second the live bit following the idea of the parity protocol of [AAD$^+$06]. An agent with live bit 0 is said to be sleeping and an agent with live bit equal to 1 is said to be awake. The initial edge state is 1, which similarly means that all edges are initially awake. We divide the possible interactions in the following four groups (we also present their effect):

1. Both agents are sleeping and the edge is awake:
   - The initiator wakes up, both agents update their data bit to 1, and the edge becomes sleeping.
2. Both agents are sleeping and the edge is sleeping:
   - Nothing happens.
3. One agent is awake and the other is sleeping:
   - The sleeping agent becomes awake and the awake sleeping. Both set their data bits to the modulo 2 sum of the data bit of the agent that was awake before the interaction and the edges state, and if the edge was awake becomes sleeping.
4. Both agents are awake:
   - The responder becomes sleeping, they both set their data bits to the modulo 2 sum of their data bits and the edges state, and if the edge was awake becomes sleeping.

It is easy to see that the initial modulo 2 sum of the edge bits (initially, they are all equal to 1) is preserved and is always equal to the modulo 2 sum of the bits of the awake agents and the awake edges. The first interaction creates the first awake agent and from that time there is always at least one awake agent and eventually remains only one. Moreover, all edges eventually become sleeping which simply means that eventually the one remaining awake agent contains the modulo 2 sum of the initial edge bits which is 0 iff the number of edges is even. All the other agents are sleeping, which means that they copy the data bit of the awake agent, thus eventually they all contain the correct data bit. The output map is defined as $O(0,\cdot)=1$ (meaning even edge parity) and $O(1,\cdot)=0$ (meaning odd edge parity). □

Theorem 6. (Constant Neighbors - Some Node) *The graph language $N_k^{out}=\{G\in\mathcal{G}\mid G$ has some node with at least k outgoing neighbors$\}$ is stably decidable for any $k=\mathcal{O}(1)$ (the same holds for $\overline{N}_k^{out}$).*

Proof. Initially all agents are in q_0 and all edges in 0. The set of agent states is $Q=\{q_0,\dots,q_k\}$ the set of edge states is binary and the output function is defined as $O(q_k)=1$ and $O(q_i)=0$ for all $i\in\{0,\dots,k-1\}$. We now describe the transition function. In any interaction through an edge in state 0, the initiator visits an unvisited outgoing edge, so it marks it by updating the edge's state to 1 and increases its own state index by one, e.g. initially $(q_0,q_0,0)$ yields $(q_1,q_0,1)$, and, generally, $(q_i,q_j,0)\to(q_{i+1},q_j,1)$, if $i+1<k$ and $j<k$, and $(q_i,q_j,0)\to(q_k,q_k,1)$, otherwise. Whenever two agents meet through a marked edge they do nothing, except for the case where only one of them is in the special alert state q_k. If the latter holds, then both go to the alert state, since in this case the protocol has located an agent with at least k outgoing neighbors. To conclude, all agents count their outgoing edges and initially output 0. Iff one of them marks its k-th outgoing edge, both end points of that edge go to an alert state q_k that propagates to the whole population and whose output is 1, indicating that G belongs to N_k^{out}. □

Note that $\overline{N}_k^{out}$ contains all graphs that have no node with at least $k=\mathcal{O}(1)$ outgoing neighbors, in other words, all nodes have fewer than k outgoing edges, which is simply the well-known bounded by k out-degree predicate. The same statement for population protocols appears as Lemma 3 in [AAC+05].

Theorem 7. (Constant Neighbors - All Nodes) *The graph language $K_k^{out}=\{G\in\mathcal{G}\mid$ Any node in G has at least k outgoing neighbors$\}$ is stably decidable for any $k=\mathcal{O}(1)$ (the same holds for $\overline{K}_k^{out}$).*

Proof. Note, first of all, that another way to think of K_k^{out} is $K_k^{out}=\{G\in\mathcal{G}\mid$ No node in G has less than k outgoing neighbors$\}$, for some $k=\mathcal{O}(1)$. The protocol we describe is similar to the one described in the proof of Theorem 6. The only difference is that when an agent counts its k-th outgoing neighbor as the initiator of an interaction, it goes to the special alert state q_k, but the alert state is not propagated (e.g. the responder of this interaction keeps its state). It follows that eventually any node that has marked at least k outgoing edges will be in the alert state, while any other node that has less than k outgoing edges will be in some state q_i, where $i<k$. Clearly the protocol has stabilizing states and provides the following semilinear guarantee:

- If $G\notin K_k^{out}$ then at least one agent remains in some state q_i, where $i<k$.
- If $G\in K_k^{out}$ no such state remains.

Thus, Theorem 3 applies, implying that there exists some GDMPP stably deciding K_k^{out} w.r.t. the predicate output convention. Thus, both K_k^{out} and $\overline{K}_k^{out}$ are stably decidable and the proof is complete. □

Theorem 8. (Compare Incoming and Outgoing Neighbors) *The graph language $M_{out}=\{G\in\mathcal{G}\mid G$ has some node with more outgoing than incoming neighbors$\}$ is stably decidable (the same holds for $\overline{M}_{out}$).*

Proof. Consider the following protocol: Initially all agents are in state 0 which is the *equality* state. An agent can also be in state 1 which is the *more-outgoing* state. Note that the latter state essentially reflects the agent's knowledge about the number of outgoing edges being larger than the number of incoming edges by a constant (here 1). That is because agents cannot keep track of arbitrarily large differences. Initially all

edges are in state s_0 and S contains also o, i and b, where state o means that the edge has been used by the protocol only as outgoing so far, i means only as incoming and b is for "both". Any agent always remembers if it has seen so far more outgoing edges or the same number of incoming and outgoing edges. So, if it is in equality state and is the initiator in an interaction where the edge has not been used at all (state s_0) or has been used only as an incoming edge (state i), which simply means that only the responder has counted it, then the agent goes to the more-outgoing state and updates the edge accordingly to remember that it has counted it. Similarly, if an agent in the more-outgoing state is the responder of an interaction and the edge is in one of the states s_0 or o, then it goes to the equality state and updates the edge accordingly. If we view the interaction from the edge's perspective, then we distinguish the following cases:

1. The edge is in state s_0. Both the initiator and the responder can use it. If only the initiator uses it (both initiator and responder in equality state), then the edge goes to state o. If only the responder uses it (both in more-outgoing state) then the edge goes to state i. If both use it (initiator in equality and responder in more-outgoing) then it goes to state b. If no one uses it it remains in s_0.
2. The edge is in state o. The initiator cannot use it, since it has already counted it. If the responder is in more-outgoing state, then it counts it and goes to equality state, thus the edge goes to state b. If, instead, it is in the equality state, the edge remains in state o.
3. The edge is in state i. The responder cannot use it. If the initiator is in equality state, then it counts it, thus the edge goes to state b. If, instead, it is in the more-outgoing state, the edge remains in state i.
4. The edge is in state b. Both the initiator and the responder have used it, thus nothing happens.

The equality state outputs 0 and the more-outgoing state outputs 1. If there exists a node with more outgoing edges, then it will eventually remain in the more-outgoing state giving 1 as output, otherwise all nodes will eventually remain in equality state, thus giving 0 as output. Computing that at least one more-outgoing state eventually remains is semilinear and the protocol, obviously, has stabilizing states, thus Theorem 3 applies and we conclude that M_{out} is stably decidable. Closure under complement implies that $\overline{M}_{out}$ is also stably decidable. □

Remark 1. *By symmetry, the corresponding languages N_k^{in}, $\overline{N}_k^{in}$, K_k^{in} and $\overline{K}_k^{in}$ concerning incoming neighbors, $M_{in} = \{G \in \mathcal{G} \mid G$ has some node with more incoming than outgoing neighbors$\}$ and $\overline{M}_{in}$ are also stably decidable.*

We now present a GDMPP that stably decides the graph language 2CYCLE $= \{G \in \mathcal{G} \mid G$ has at least two nodes u, v s.t. both $(u, v), (v, u) \in E(G)$ (in other words, G has at least one 2-cycle)$\}$ for any at least weakly connected G that has at least two nodes.

The state of an agent consists of two components, a *live bit* and a *counter*. The live bit is either l or t meaning *leader* (awake) and *nonleader* (asleep), respectively. The counter takes integer values from 0 to 2 inclusive. The state of an edge is 0 or 1 which are interpreted as *inactive* and *active*, respectively. Intuitively, the counter of an agent counts the number of active edges incident to it. Initially, all agents are leaders and their counters are set to 0 and all edges are inactive. The output of an agent is the output of its counter, where $O(0) = O(1) = 0$ and $O(2) = 1$.

We describe now the transition function of the protocol. When a leader with zero counter interacts as the initiator with an agent with zero counter via an inactive edge, the initiator becomes a nonleader, the responder becomes a leader, both set their counters to 1, and the edge becomes active. When a nonleader with zero counter initiates an interaction with a leader with zero counter via an inactive edge, the agents swap their live bits. When a leader with counter 1 initiates an interaction with a nonleader with counter 1 via an inactive edge, the agents increase their counters by one and swap their live bits and the edge becomes active. When a leader and a nonleader with counters 2 interact via an active edge, they swap their live bits. All other interactions via active edges decrease the counters of both agents by one and deactivate the edge and if both agents were leaders, the initiator becomes a nonleader.

An *active subgraph* is a weakly connected subgraph of the interaction graph induced by active edges.

Lemma 1. *The counter component of an agent is always equal to the number of active edges incident to it.*

Proof. Initially all counters are zero. Whenever an inactive edge is activated, both endpoints increase their counters by one and whenever an active edge is deactivated, both endpoints decrease their counters by one. □

Lemma 2. *Counters do not overflow.*

Proof. If the counter of an agent is zero, there is no interaction that decreases it (the only transition that decreases it applies if the edge is active, which is never the case with 0 counter) and if it is equal to 2, there is no interaction that increases it (simply inspect the transition function). □

Lemma 3. *By ignoring the directions of the edges, active subgraphs are either lines or cycles.*

Proof. By Lemma 1 the counter of an agent is equal to the number of active edges incident to it and by Lemma 2 its value is at most 2. Consequently, any agent of an active subgraph has at most two neighbors and also it must have at least one for the subgraph to be connected. □

Lemma 4. *If $(l, 2)$ and $(t, 2)$ appear at the endpoints of an active (subgraph which is a) 2-cycle, the configuration of the 2-cycle remains forever the same up to swaps of the live bits.*

Proof. Immediate from the transition function. □

Lemma 4 establishes that active 2-cycles are stable. Moreover, it is clear that the endpoints of such a 2-cycle forever output 1. Note, however, that $(l, 2)$ and $(t, 2)$ may also appear in active subgraphs that are not 2-cycles. The next lemma proves that active subgraphs that are not 2-cycles are unstable in the sense that they will eventually be eliminated.

Lemma 5. *In every active subgraph that is not a 2-cycle it is always possible that an active edge is eventually deactivated.*

Proof. By Lemma 3, any active subgraph is either a line or a cycle. If the active subgraph is a line, then its endpoints have counters 1 and all other nodes have counters 2. So there is always a transition between an endpoint and its neighbor that deactivates the corresponding edge. If the active subgraph is a cycle, then all nodes have counters equal to 2 and the only case that this might be stable would be to have a cycle of even length with alternating $(l, 2)$s and $(t, 2)$s. However, even in this case, an interaction between them swaps the live bits and eventually two $(l, 2)$s or two $(t, 2)$s interact and an edge is deactivated. □

Lemma 6. *If (the interaction graph) $G \notin$ 2CYCLE, then eventually a unique leader remains.*

Proof. No active 2-cycle is ever formed (which, by Lemmas 4 and 5 is the only possible stable active subgraph), the leaders keep wondering around (even by forming unstable active graphs that will eventually be eliminated) and whenever two of them meet (and only then) one of them becomes a nonleader. □

Lemma 7. *If $G \in$ 2CYCLE, eventually the output stabilizes with at least two agents giving 1 as output and if $G \notin$ 2CYCLE, all agents eventually output 0.*

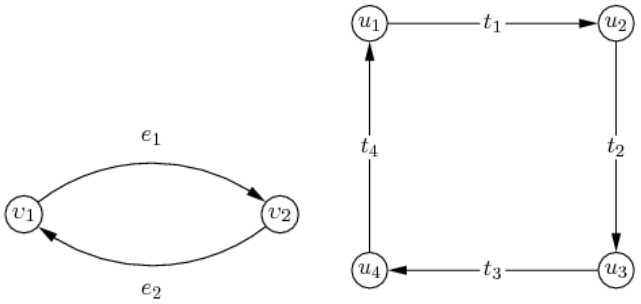
Proof. If there is a 2-cycle, then at least one leader will remain wondering around since only one leader is eliminated after each interaction between (wondering) leaders. Given such a leader an active 2-cycle will eventually be formed and, by Lemma 4, both its endpoints will forever output 1. Eventually any leader, in general, either remains the unique leader with counter less than 2 or forms an active 2-cycle, which implies that eventually all nonleaders that are not part of an active 2-cycle will have maximum counter value 1 (consequently their output will stabilize to 0 and the same will hold for the unique wondering leader). Finally, if there is no 2-cycle, then by Lemma 6 a unique leader eventually remains, all 2 counters are eventually eliminated (see Lemma 5), and, from that point on, no counter can become 2 again. □

Theorem 9. *The above GDMPP stably decides 2CYCLE.*

Proof. Lemma 7 and the fact that the predicate $(N_1 \geq 1)$ is semilinear, where N_1 denotes the number of 1s in the GDMPP's output assignment, satisfy the preconditions of Theorem 3 thus implying that 2CYCLE is GDMPP-decidable. □

5.4. An Impossibility Result

Now we are about to prove that a specific graph language cannot be stably decided by GDMPPs with stabilizing states. First we state and prove a useful lemma.



(a) Graph G (b) Graph G'

Figure 1: $G \in$ 2CYCLE and $G' \notin$ 2CYCLE.

Lemma 8. *For any GDMPP $\mathcal{A}$ and any computation (infinite fair execution) $C_0, C_1, C_2, \ldots$ of $\mathcal{A}$ on G (Figure 1(a)) there exists a computation $C'_0, C'_1, C'_2, \ldots, C'_i, \ldots$ of $\mathcal{A}$ on G' (Figure 1(b)) s.t.*

$$C_i(v_1) = C'_{2i}(u_1) = C'_{2i}(u_3)$$
$$C_i(v_2) = C'_{2i}(u_2) = C'_{2i}(u_4)$$
$$C_i(e_1) = C'_{2i}(t_1) = C'_{2i}(t_3)$$
$$C_i(e_2) = C'_{2i}(t_2) = C'_{2i}(t_4)$$

for any finite $i \geq 0$.

Proof. The proof is by induction on i. We assume that initially all nodes are in q_0 and all edges in s_0 (initial states). So the base case (for $i = 0$) holds trivially. Now we make the following assumption: Whenever the scheduler of $\mathcal{A}$ on G (call it S_1) selects the edge e_1 we assume that the scheduler, S_2, of $\mathcal{A}$ on G' takes two steps; it first selects t_1 and then selects t_3. Whenever S_1 selects the edge e_2, S_2 first selects t_2 and then t_4. Formally, if $C_{i-1} \xrightarrow{e_1} C_i$ then $C'_{2(i-1)} \xrightarrow{t_1} C'_{2i-1} \xrightarrow{t_3} C'_{2i}$ and if $C_{i-1} \xrightarrow{e_2} C_i$ then $C'_{2(i-1)} \xrightarrow{t_2} C'_{2i-1} \xrightarrow{t_4} C'_{2i}$ for every finite $i \geq 1$. Obviously, S_2 is not a fair scheduler so to be able to talk about computation we only require this predetermined behavior to be followed by S_2 for a finite number of steps. After this finite number of steps, S_2 goes on arbitrarily but in a fair manner.

Now assume that all conditions are satisfied for some finite step i (inductive hypothesis). We will prove that the same holds for step $i + 1$ to complete the proof (inductive step). There are two cases:

1. $C_i \xrightarrow{e_1} C_{i+1}$ *(i.e. in step $i + 1$ S_1 selects the edge e_1)*: Then we know that S_2 first selects t_1 and then t_3 (in its corresponding steps $2i + 1$ and $2i + 2$). That is, its first transition is $C'_{2i} \xrightarrow{t_1} C'_{2i+1}$ and its second is $C'_{2i+1} \xrightarrow{t_3} C'_{2(i+1)}$. But from the inductive hypothesis we know that $C'_{2i}(u_1) = C_i(v_1)$, $C'_{2i}(u_2) = C_i(v_2)$ and $C'_{2i}(t_1) = C_i(e_1)$ which simply means that interaction e_1 on G has the same effect as interaction t_1 on G' (u_1 has the same state as v_1, u_2 as v_2 and t_1 as e_1). Thus, $C'_{2i+1}(u_1) = C_{i+1}(v_1)$, $C'_{2i+1}(u_2) = C_{i+1}(v_2)$ and $C'_{2i+1}(t_1) = C_{i+1}(e_1)$. Moreover, in this step t_3 and both its endpoints do not change state (since the interaction concerned t_1), thus $C'_{2i+1}(u_3) = C'_{2i}(u_3) = C_i(v_1)$ (the last equation comes from the inductive hypothesis), $C'_{2i+1}(u_4) = C'_{2i}(u_4) = C_i(v_2)$ and $C'_{2i+1}(t_3) = C'_{2i}(t_3) = C_i(e_1)$. When in the next step S_2 selects t_3, t_1 and both its endpoints do not change state, thus $C'_{2(i+1)}(u_1) = C'_{2i+1}(u_1) = C_{i+1}(v_1)$, $C'_{2(i+1)}(u_2) = C'_{2i+1}(u_2) = C_{i+1}(v_2)$ and $C'_{2(i+1)}(t_1) = C'_{2i+1}(t_1) = C_{i+1}(e_1)$. Now let's see what happens to t_3 and its endpoints. Before the interaction the state of u_3 is $C_i(v_1)$, the

state of u_4 is $C_i(v_2)$ and the state of t_3 is $C_i(e_1)$, which means that, in $C'_{2(i+1)}$, u_3 has gone to $C_{i+1}(v_1)$, u_4 to $C_{i+1}(v_2)$ and t_3 to $C_{i+1}(e_1)$. Finally, t_2 and t_4 have not participated in any of the two interactions of S_2 and thus they have maintained their states, that is $C'_{2(i+1)}(t_2) = C'_{2i}(t_2) = C_i(e_2) = C_{i+1}(e_2)$ (the last two equations follow from the inductive hypothesis and the fact that, in step $i+1$, S_1 selects e_1 which means that e_2 maintains its state, respectively), and similarly $C'_{2(i+1)}(t_4) = C_{i+1}(e_2)$.

2. $C_i \xrightarrow{e_2} C_{i+1}$ *(i.e. in step $i+1$ S_1 selects the edge e_2)*: This case is symmetric to the previous one.

□

Let now $\mathcal{A}$ be a GDMPP that stably decides the graph language 2CYCLE. So for any computation of $\mathcal{A}$ on G, after finitely many steps, both v_1 and v_2 go to some state that outputs 1, since $G \in$ 2CYCLE, and do not change their output value in any subsequent step (call the corresponding output stable configuration C_i, where i is finite). But according to Lemma 8 there exists a computation of $\mathcal{A}$ on G' that under configuration C'_{2i} has u_1, u_2, u_3 and u_4 giving output 1. We use this fact to prove the following impossibility result.

Theorem 10. *There exists no GDMPP with stabilizing states to stably decide the graph language 2CYCLE= $\{G \in \mathcal{G} \mid G$ has at least two nodes u, v s.t. both $(u,v),(v,u) \in E(G)\}$.*

Proof. Let $\mathcal{A}$ be a GDMPP with stabilizing states that stably decides 2CYCLE. It follows that when $\mathcal{A}$ runs on G (Figure 1(a)) after a finite number of steps v_1 and v_2 obtain two states w.l.o.g. q_1 and q_2, respectively, that output 1 (since $\mathcal{A}$ stably decides 2CYCLE) and do not change in any subsequent step (since $\mathcal{A}$ has stabilizing states). Assume that at that point e_1 is in state s_1 and e_2 in state s'_1. Assume also that there exists a subset $S_1 = \{s_1, s_2, \ldots, s_k\}$ of S, of edge states that can be reached by subsequent interactions of the pair (v_1, v_2) and a subset $S_2 = \{s'_1, s'_2, \ldots, s'_l\}$ of S, of edge states that can be reached by subsequent interactions of the pair (v_2, v_1), where k and l are both constants independent of n (note that S_1 and S_2 are not necessarily disjoint). It follows that for all $s_i \in S_1$, $(q_1, q_2, s_i) \to (q_1, q_2, s_j)$, where $s_j \in S_1$, and for all $s'_i \in S_2$, $(q_2, q_1, s'_i) \to (q_2, q_1, s'_j)$, where $s'_j \in S_2$. In words, none of these reachable edge states can be responsible for a change in some agent's state. According to Lemma 8 there exists a computation of $\mathcal{A}$ on G' (Figure 1(b)) such that after a finite number of steps u_1, u_3 are in q_1, u_2, u_4 are in q_2, t_1, t_3 are in s_1 and t_2, t_4 are in s'_1. Since $\mathcal{A}$ stably decides 2CYCLE, at some subsequent finite step (after we let the protocol run in a fair manner in G'), some agent obtains a new state q_3, since if it didn't then all agents would always remain in states q_1 and q_2 that output 1 (but in G' there is no 2-cycle and such a decision is wrong). This must happen through some interaction of the following two forms: (i) (q_1, q_2, s_i), where $s_i \in S_1$ and (ii) (q_2, q_1, s'_i), where $s'_i \in S_2$. But this is a contradiction, since we showed earlier that no such interaction can modify the state of any of its end points. Intuitively, if there exists some way for $\mathcal{A}$ to modify one of q_1 and q_2 in G' then there would also exist some way for $\mathcal{A}$ to modify one of q_1 and q_2 in G, after the system has obtained stabilizing states there, which is an obvious contradiction. □

6. The computational power of GDMPPs in $\mathcal{G}$

In this section, we discuss the exact computational power of the GDMPP model. Here we will approach this problem by extending the notion of input graphs to vertex-labeled graphs [3] whose labels are assigned by the network initialization function. We consider that these labels are chosen from a finite set X. Such input graphs are in fact more general since they bear additional information that allows more complex computations taking place on the multiset of labels (since multiple agents may have identical labels and neighborhoods with identical labels sets). Now instead of just deciding whether e.g. an agent has k incoming neighbors we can decide whether there is an 'a'-labeled agent that has k 'b'-labeled incoming neighbors. Protocols running on such input graphs have been studied in the MPP model [CMS09a, MCS11a] which assumes that the previously defined input graph is the interaction graph, and the labels are the input of the protocol. Note that GDMPPs are practically MPPs whose labels are the nodes' states and where the initial

[3] Graphs whose vertexes are assigned labels by some function $f : V \to S$ where S is a set of labels.

labels are the same for all nodes/edges. Thus for any GDMPP $\mathcal{A}$ running on an interaction graph $G \in \mathcal{G}$ there is an identical MPP which is similarly initialized and that provides the same output when running on G. Therefore, in order to understand the capabilities of GDMPP model we can study the computational capabilities of MPPs on $\mathcal{G}$. We will conclude that MPPs on unrestricted graphs are equivalent to PPs, and since GDMPPs must also operate on any possible input graph, they are strongly expected to be weak, like PPs. The proof idea is as follows. Let p be computable by an MPP $\mathcal{A}$. $\mathcal{A}$ still computes p if we restrict our attention on star graphs, which consist of 1 internal node of degree n and n external nodes of degree 1. The latter situation can be simulated by a PP, running on complete graphs, with a unique leader in its initial configuration since the leader can play the role of the internal node and also can safely store the states of the edges on the external nodes (since external nodes have no effective interactions with each other). The latter, in turn, can be simulated by a PP which is the composition of a leader election protocol, that elects a leader while leaving the inputs unaffected, and the stabilizing inputs implementation of a PP, whose transition function is the same as the simulated protocol, extended appropriately by ineffective transitions. A more detailed proof is described below.

We first provide some preliminaries: For any finite alphabet X, we call *predicate* any function $p : X^* \to \{0, 1\}$. A predicate p on X^* is called *symmetric* if for every $x \in X^*$ and any x' which is a permutation of x's symbols it holds that $p(x) = p(x')$. In other words, permuting the input symbols does not affect the symmetric predicate's outcome. We say that *a predicate over X^* is stably computable by an MPP $\mathcal{A}$* on a graph family $\mathcal{F}$, if for each $G = (V, E) \in \mathcal{F}$ and $x : V \to X$ and for any computation of $\mathcal{A}$ on G beginning from the initial configuration where each agent is in a state denoted by its label, all agents output $p(x)$. Note that p depends only on the labeling x and not G.

An *out-star* is a digraph where one node has out-degree $n - 1$ (internal node) and in-degree zero and $n - 1$ nodes have in-degree one and out-degree zero (leaves). Denote by $\mathcal{G}_{star}$ the graph family of all out-stars, by $\mathcal{G}_{com}$ that of all complete digraphs, and by $\mathcal{G}$ that of all weakly-connected digraphs (as before). Denote by LPP *the PP model that additionally has a unique leader in any initial configuration* and by **LPC** ("LPP stably computable predicates in the family of Complete graphs") the class of all predicates that are stably computable by the LPP model in $\mathcal{G}_{com}$. We assume that the initial leader of any LPP in any initial configuration is provided by a *leader initialization function* I_{le} which is not a part of the protocol (can be e.g. a preprocessing on the network). We also consider that the state of any LPP consists of two state components, one assigned by I_{le} as a preprocessing of the network and the other initialized according to the agent's input symbol and updated during the execution of the protocol. Finally, denote by **MPU** ("Mediated stably computable Predicates in the Unrestricted family of graphs") the class of all predicates that are stably computable by the MPP model in $\mathcal{G}$.

Lemma 9. *Any predicate that is stably computable by the MPP model in $\mathcal{G}_{star}$ is also stably computable by the LPP model in $\mathcal{G}_{com}$.*

Proof. Take any such predicate p. Let $\mathcal{A}$ be the MPP that stably computes it. We present a LPP $\mathcal{B}$ that stably computes p in $\mathcal{G}_{com}$ by simulating $\mathcal{A}$.

- $X_{\mathcal{B}} = \{l, n\} \times X_{\mathcal{A}}$,
- $Y_{\mathcal{B}} = Y_{\mathcal{A}} = \{0, 1\}$,
- $Q_{\mathcal{B}} = (\{l\} \times Q_{\mathcal{A}}) \cup (\{n\} \times Q_{\mathcal{A}} \times S_{\mathcal{A}})$,
- $I_{\mathcal{B}}(n, \sigma) = (n, I_{\mathcal{A}}(\sigma), s_0)$, where s_0 is the initial edge state of $\mathcal{A}$, and $I_{\mathcal{B}}(l, \sigma) = (l, I_{\mathcal{A}}(\sigma))$,
- $O_{\mathcal{B}}((\cdot, q)) = O_{\mathcal{B}}((\cdot, q, s)) = O_{\mathcal{A}}(q)$, for all, $q \in Q_{\mathcal{A}}$ and $s \in S_{\mathcal{A}}$, and
- $\delta_{\mathcal{B}} : (l, q_1), (n, q_2, s) \to (l, \delta_{\mathcal{A}_1}(t)), (n, \delta_{\mathcal{A}_2}(t), \delta_{\mathcal{A}_3}(t))$, where t denotes (q_1, q_2, s) for all $q_1, q_2 \in Q_{\mathcal{A}}$ and $s \in S_{\mathcal{A}}$.

It is not hard to see that the LPP $\mathcal{B}$ also forms an out-star by using the leader agent as the internal node and the nonleaders as the leaves and by allowing interactions only via the edges leaving the leader. The

information that $\mathcal{A}$ stores on the edges is stored by $\mathcal{B}$ on the nonleaders. Note that we provide the leader assignment, given by some function I_{le} that is not part of the protocol, in the input of $\mathcal{B}$, and define $I_{\mathcal{B}}$ appropriately. In the previously given LPP model's definition, we state that the leader component of an LPP's state is set directly from I_{le}. To simplify the description of $\mathcal{B}$, we extend the input alphabet to include the leader assignment of I_{le} and avoid complex notations in the input function's definition. Note that $I_{\mathcal{B}}$ remains constant in size, since $|\{l, n\} \times X_{\mathcal{A}}| = \mathcal{O}(1)$ and thus this modification is w.l.o.g. □

Lemma 10. $\mathbf{MPU} \subseteq \mathbf{LPC}$.

Proof. Take any $p \in \mathbf{MPU}$ and let $\mathcal{A}$ be the MPP that stably computes it in $\mathcal{U}$. Obviously, $\mathcal{A}$ also stably computes p in any $\mathcal{U} \subseteq \mathcal{G}$. Thus, p is also stably computable by the MPP model in $\mathcal{G}_{star}$ and, by applying Lemma 9, p is also stably computable by the LPP model in $\mathcal{G}_{com}$. □

Note that since $\mathcal{A}$ stably computes p in $\mathcal{G}$ it also stably computes it in $\mathcal{G}_{com}$ and thus p must be symmetric due to the completeness of the interaction graph. Consequently, **MPU** is, in fact, a subset of the symmetric subclass of **LPC**. We denote this subclass as **SLPC** (*Symmetric LPC*). It therefore follows that:

Corollary 1. $\mathbf{MPU} \subseteq \mathbf{SLPC}$.

In the sequel, we denote by **SEM** the class of semilinear predicates.

Theorem 11. $\mathbf{SLPC} = \mathbf{SEM}$.

Proof. $\mathbf{SEM} \subseteq \mathbf{SLPC}$ is trivial, since LPPs are essentially PPs with the additional knowledge of a leader and thus can compute any predicate in $\mathcal{G}_{com}$ that PPs can in the same class. For the other direction, we show that any LPP running in $\mathcal{G}_{com}$ can be simulated by the PP model in the same family of graphs; the latter can only compute semilinear predicates. Take any predicate $p : \mathbb{N}^X \to \{0, 1\}$ that is stably computable by an LPP $\mathcal{S} = (X_{\mathcal{S}}, Y_{\mathcal{S}}, Q_{\mathcal{S}}, I_{\mathcal{S}}, O_{\mathcal{S}}, \delta_{\mathcal{S}})$. Note that the state set of $\mathcal{S}$, $Q_{\mathcal{S}}$ is of the form $\{l, n\} \times Q$ where $|Q| = \mathcal{O}(1)$.

Consider a PP $\mathcal{B} = (X_{\mathcal{B}}, Y_{\mathcal{B}}, Q_{\mathcal{B}}, I_{\mathcal{B}}, O_{\mathcal{B}}, \delta_{\mathcal{B}})$ defined as $X_{\mathcal{B}} = \{0, 1\} \times X_{\mathcal{S}}$, $Y_{\mathcal{B}} = \{0, 1\} = Y_{\mathcal{S}}$, $Q_{\mathcal{B}} = Q_{\mathcal{S}}$, $I_{\mathcal{B}} : X_{\mathcal{B}} \to Q_{\mathcal{B}}$ so that $I_{\mathcal{B}}(0, \sigma) = (n, I_{\mathcal{S}}(\sigma))$, $I_{\mathcal{B}}(1, \sigma) = (l, I_{\mathcal{S}}(\sigma))$ and $\sigma \in X_{\mathcal{S}}$, $O_{\mathcal{B}} = O_{\mathcal{S}}$ and $\delta_{\mathcal{B}}$ is the same as $\delta_{\mathcal{S}}$ extended with all transitions of the form: $((l, q_1), (l, q_2)) \to ((l, q_1), (l, q_2))$ where $q_1, q_2 \in Q$. The extended $\delta_{\mathcal{S}}$ practically states that any interaction between two leaders is ineffective. Observe that $\mathcal{B}$ given input assignment $x_{\mathcal{B}} \in X_{\mathcal{B}}^*$ where there is exactly one symbol of the form $1 \times \sigma$ and the rest are of the form $0 \times \sigma$, $\sigma \in X_{\mathcal{S}}$, has exactly the same output as $\mathcal{S}$ starting from the initial configuration $I_{\mathcal{B}}(x_{\mathcal{B}})$ (since $Q_{\mathcal{S}} = Q_{\mathcal{B}}$ all initial configurations defined by such $x_{\mathcal{B}}$s will also be possible initial configurations of $\mathcal{S}$). This is because the previously defined PP works in the same way $\mathcal{S}$ does, given a single leader (symbol of the form $1 \times \sigma$) in its input.

Now take a population protocol PP $\mathcal{A} = (X_{\mathcal{A}}, Y_{\mathcal{A}}, Q_{\mathcal{A}}, I_{\mathcal{A}}, O_{\mathcal{A}}, \delta_{\mathcal{A}})$ defined as $X_{\mathcal{A}} = X_{\mathcal{S}}$, $Y_{\mathcal{A}} = \{0, 1\} \times X_{\mathcal{S}}$, $Q_{\mathcal{A}} = \{0, 1\} \times X_{\mathcal{S}}$, $I_{\mathcal{A}}(\sigma) = (1, \sigma)$, $O_{\mathcal{A}}(y, \sigma) = (y, \sigma)$ where $y \in \{0, 1\}$ and $\sigma \in X_{\mathcal{S}}$ and $\delta_{\mathcal{A}}$ includes only the following rule: $((1, \sigma_1), (1, \sigma_2)) \to ((1, \sigma_1), (0, \sigma_2))$ where $\sigma_1, \sigma_2 \in X_{\mathcal{S}}$. $\mathcal{A}$ stabilizes to a configuration where a single agent is in $1 \times X_{\mathcal{S}}$ and all others in $0 \times X_{\mathcal{S}}$ (which practically means that eventually there is a single leader) and leaves the input unchanged in each agent.

From [AAC+05] and [AAER07], we have that for any PP $\mathcal{B}$ there is a stabilizing inputs PP $\mathcal{B}'$ that stably computes the same predicate in the same family of graphs, which is in fact a semilinear predicate. The composition of $\mathcal{A}$ and $\mathcal{B}'$, let us call it $\mathcal{D}$, is also a population protocol that runs on the complete graph, takes the same input as $\mathcal{A}$ and has the output of $\mathcal{B}'$ given as input the stabilizing output of $\mathcal{A}$. $\mathcal{A}$ guarantees that a unique leader remains in its stable output containing all the initial input values, and $\mathcal{B}'$ runs on this output performing as $\mathcal{B}$ given a single leader on the input. But $\mathcal{B}$ given a single leader on its input does what $\mathcal{S}$ does starting from the corresponding initial configuration. □

Given Corollary 1 and Theorem 11, we can prove the following:

Theorem 12. $\mathbf{MPU} = \mathbf{SEM}$.

Proof. It follows from Theorem 16 of [AAD+06] that any semilinear predicate can be computed by the PP model in the unrestricted family of graphs. Since the MPP model is a generalization of the PP model **SEM** $\subseteq$ **MPU** follows. The other direction follows from the inclusions **MPU** $\subseteq$ **SLPC** = **SEM** of Lemma 10 and Theorem 11. □

The consequences of the above theorem are twofold. First of all, the GDMPP model on the family of weakly-connected graphs seems to be computationally weak; this is a first step towards providing an exact characterization of GDMPPs computational power. Secondly, as a side effect, we conclude that the existence of a leader does not help population protocols running on complete graphs.

7. Graphs not even weakly connected

So far we have only studied the case of at least weakly-connected graphs. A reasonable question is what happens when the connectivity of the input graph is broken. In this section, we show that nontrivial graph languages are not stably computable in this case. We consider that the universe is $\mathcal{H}$ and, thus, a graph language can only be a subset of $\mathcal{H}$. Any disconnected graph G in $\mathcal{H}$ consists of (weakly or strongly connected) components $G_1, G_2, \ldots, G_t$, where $t \geq 2$ (note also that any component must have at least two nodes, to allow computation).

Lemma 11. *For any nontrivial graph language L, there exists some disconnected graph G in L where at least one component of G does not belong to L or there exists some disconnected graph G' in $\overline{L}$ where at least one component of G' does not belong to $\overline{L}$, or both.*

Proof. Let L be such a nontrivial graph language and assume that the statement does not hold. Then for any disconnected graph in L, all of its components also belong to L and for any disconnected graph in $\overline{L}$, all of its components also belong to $\overline{L}$. There are two main cases:

1. L contains all connected graphs. But $\overline{L}$ is nontrivial which means that it must contain at least one disconnected graph. We know that for any disconnected graph in $\overline{L}$ all of its connected components belong to $\overline{L}$, but this is a contradiction, since all connected graphs belong to L.
2. L does not contain all connected graphs. There are now two possible subcases:
   (a) L contains at least one connected graph (but not all). This means that $\overline{L}$ contains also at least one connected graph. Let $G' = (V', E')$ be a connected graph from L and $G'' = (V'', E'')$ be a connected graph from $\overline{L}$. The disjoint union of G' and G'', $U = (V' \cup V'', E' \cup E'')$ is a disconnected graph consisting of two connected components, one belonging to L and one to $\overline{L}$. U itself must belong in one of L and $\overline{L}$ implying that all of its components must belong to L or all to $\overline{L}$, which is a contradiction.
   (b) L contains no connected graph. Thus, since L is nontrivial, it contains at least one disconnected graph whose connected components belong to L. But all connected graphs belong to $\overline{L}$ which is a contradiction.

□

Theorem 13. *Any nontrivial graph language $L \subset \mathcal{H}$ is not stably decidable by the GDMPP model.*

Proof. Let L be such a language and assume that there exists a GDMPP $\mathcal{A}_L$ that stably decides it. Thus, $\mathcal{A}_L$ has eventually all the agents of G giving output 1 if $G \in L$ and all giving output 0 if $G \notin L$. Moreover, the protocol $\mathcal{A}_{\overline{L}}$ that has the output map of $\mathcal{A}_L$ complemented stably decides $\overline{L}$. Those GDMPPs (and in fact any GDMPP) have no way to transmit data between agents of different components when run on disconnected graphs. In fact it is trivial to see that, when run on disconnected graphs, those protocols essentially run individually on the different components of those graphs. This means that when, for example, $\mathcal{A}_L$ runs on a disconnected graph G, where G has at least two components $G_1, G_2, \ldots, G_t$, then $\mathcal{A}_L$ runs in t different copies, one for each component, and each such copy stably decides the membership of the corresponding component (on which it runs on) in L. The same holds for $\mathcal{A}_{\overline{L}}$. By Lemma 11 there exists at least one

disconnected graph in L with at least one component in $\overline{L}$ or at least one disconnected graph in $\overline{L}$ with at least one component in L. If L contains such a disconnected graph then, obviously, $\mathcal{A}_L$ when run on this graph, call it G, has eventually all the nodes of the component(s) in $\overline{L}$ giving 0 as output. This is a contradiction, because $G \in L$ and $\mathcal{A}_L$ stably decides L, which means that all agents should eventually output 1. If $\overline{L}$ contains such a disconnected graph then the contradiction is symmetric. □

As an immediate consequence we get the following corollary:

Corollary 2. *The graph language $C = \{G \in \mathcal{H} \mid G$ is (weakly) connected$\}$ is not GDMPP-decidable.*

Proof. C is a nontrivial graph language and Theorem 13 applies. □

8. An alternative notion of input

We now make a different assumption about the input graph. We consider the case where the input graph is any subgraph of the interaction graph on which the corresponding protocol runs. This *input subgraph* is denoted by some special agents' and edges' initial states which are a result of some preprocessing on the network. We call the resulting protocols *Mediated Graph Protocol*s (*MGP*s). From this point on, we only consider MGPs and we explore how the previous assumption affects the computability of graph languages. In the following subsection, we provide a brief definition and notations of this new extension.

8.1. Mediated Graph Protocols

A *Mediated Graph Protocol* (*MGP*) is defined as a GDMPP where the finite sets of agent and edge states Q, S are augmented by the initial states $q_0, q_1 \in Q$ and $s_0, s_1 \in S$ respectively.

As in the case of GDMPPs, an MGP runs on an interaction graph $G = (V, E)$, which we assume to be complete for the rest of this work, so that an MGP may run on any $K_n = (V, E)$, where $|V| = n$ and $E = V^2 - \{(u, u) \mid u \in V\}$.

We assume that the initial states of the agents and the edges of the network are specified by some function $\iota : V \cup E \to \{q_0, q_1, s_0, s_1\}$, which, similarly to the GDMPP model, is not part of the protocol but models some preprocessing on the network. ι is called a *network initialization function* if $\iota(e) \in \{s_0, s_1\}$ for all $e \in E$, $\iota(u) = q_1$ if u is incident to at least one edge in s_1 according to E and $\iota(u) = q_0$ otherwise, for all $u \in V$. Given an interaction graph $K_n = (V, E)$ and a network initialization function ι, we may define the *subgraph of K_n specified by ι* as $G_\iota[K_n] = (V', E')$, where $V' = \{u \in V \mid \iota(u) = q_1\}$ and $E' = \{e \in E \mid \iota(e) = s_1\}$. $G_\iota[K_n]$ is considered as the *input graph* to the protocol.

The notions of configuration, reachability, execution and fairness are the same as in GDMPPs. Moreover, MGPs are uniform and anonymous. For the rest of this work, our focus is to determine properties of the subgraph that is specified by the network initialization function.

Definition 3. *We say that an MGP $\mathcal{A}$* stably decides *a graph language L if, for any complete interaction graph $K_n = (V, E)$, any network initialization function ι, and any computation of $\mathcal{A}$ on K_n beginning from the initial configuration specified by ι, all agents eventually output 1 (accept) if $G_\iota[K_n] \in L$ and 0 (reject) otherwise. A graph language is said to be* stably decidable *by the MGP model (or MGP-*decidable*) if there is an MGP $\mathcal{A}$ that stably decides it.*

We call a protocol $\mathcal{A}$ a *stabilizing output graph* MGP if, in any computation of $\mathcal{A}$, all *agents' outputs* and *edges' states* eventually stop changing. Note that this stabilization assumption is weaker than the stabilizing states assumption made for GDMPPs where both edges' and agents' states are required to stabilize. We define **GMGP** to be the class of all stably decidable graph languages by the MGP model. We denote by **LGNSPACE** the class of all decidable graph languages by a NTM of linear space which receives the input graph by its adjacency matrix representation.

As a simple illustration, we formalize a version of the count-(k+1)-in-neighbors protocol that was outlined in the introduction (for $k = 2$). The set of agent states is $Q = \{q_0, q_1, q_2, q_3, q_4\}$ and the set of edge states is $S = \{s_0, s_1\}$. Note that agent states, q_0, q_1, denote the inclusion of the agents in the input subgraph

while the rest of the agent states denote, as we will see next, the numbers counted by the agents of that subgraph. The output function O maps all states except q_4 to 0 and the state q_4 to 1. The transition function δ is defined as follows: if $i < 4$ and $j < 3$ then $\delta(q_i, q_j, s_1) = (q_i, q_{j+1}, s_0)$; [4] if $i = 4$ or $j \geq 3$ then $\delta(q_i, q_j, s_1) = (q_4, q_4, s_0)$; if $i = 4$ or $j = 4$ then $\delta(q_i, q_j, s_0) = (q_4, q_4, s_0)$. All remaining transitions leave all three components unaffected.

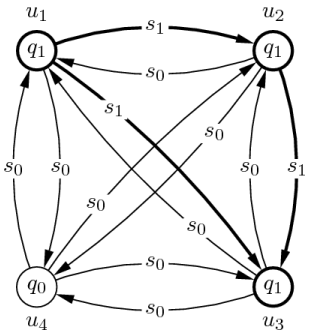


Figure 2: An example. The input graph is identified by the bold nodes and edges.

Assume now that the agents are u_1, u_2, u_3, u_4. Since the interaction graph is complete, the edges are $(1,2), (1,3), (1,4), (2,1), (2,2), \ldots, (4,3)$. Let the initial configuration, as described by some network initialization function, be $((q_1, q_1, q_1, q_0), \{(1,2), (1,3), (2,3)\})$, where the tuple describes the state of each agent and the set contains the s_1 edges and is used for simplicity. These are also illustrated in Fig. 2. Note that the tuples of the form (i, j) correspond to the edges (u_i, u_j) in the graph.

Consider now the following possible computation: $((q_1, q_1, q_1, q_0), \{(1,2), (1,3), (2,3)\}) \xrightarrow{(2,3)} ((q_1, q_1, q_2, q_0), \{(1,2), (1,3)\}) \xrightarrow{(4,3)} ((q_1, q_1, q_2, q_0), \{(1,2), (1,3)\}) \xrightarrow{(1,2)} ((q_1, q_2, q_2, q_0), \{(1,3)\}) \xrightarrow{(2,3)} ((q_1, q_2, q_2, q_0), \{(1,3)\}) \xrightarrow{(4,1)} ((q_1, q_2, q_2, q_0), \{(1,3)\}) \xrightarrow{(1,3)} ((q_1, q_2, q_3, q_0), \{\})$. In the last configuration, all agents output 0, and this configuration is *output stable* in the sense that, from that point on, no agent can change its output. Note that here, it also happens that the states will not be modified again in the future, but generally this is a stronger requirement. In our model, we only require for the outputs to stop changing (the former implies the latter but the opposite is not true). So, in this case, the protocol rejects the input graph that was specified by the network initialization function and this is a correct decision because none of its nodes has more than 2 in-neighbors.

9. Properties of MGPs

The properties discussed in Theorems 2 and 3 for GDMPPs extend trivially to MGPs. We here present some useful unique properties of the MGP model that will help us unfold its computational potential. Let $G = (V, E)$ be a simple digraph and K the maximal irreflexive subset of V^2. Then the *inverse* or *complement* of G is defined as $H = (V, K \backslash E)$. Let $L^{-1} = \{H \mid \exists G \in L$ such that H is the inverse of $G\}$ be the inverse of a language L.

Theorem 14 (Closure under Inversion)**. GMGP** *is closed under inversion.*

Proof. Let $\mathcal{A}$ be the protocol that stably decides L. Interchange in $\mathcal{A}$, the roles of the initial edges states s_1 and those s_0 that are incident only to q_1 nodes. If $G_\iota[K_n] \in L^{-1}$ then its inverse is in L and is the graph consisting of all q_1 nodes and all s_0 edges incident to them. Clearly, the described protocol accepts. The "reject" case is symmetric. □

[4] The careful reader will already have noticed that both i and j are not required to be greater than 0 because no legal network initialization function allows a q_0 agent to have an s_1 incident edge.

Consider now the (stably decidable) language consisting of all graphs that have at least one edge. Its inverse consists of all graphs that are missing at least one edge, or, in other words, all graphs that are not complete. Closure under inversion makes it possible to decide whether a graph is not a clique and then closure under complement enables us to determine whether a graph is a clique. Such a decision seems impossible to be achieved by PPs and GDMPPs. The reason is that, in those models, a missing edge is not directly detectable and the only way for an agent to detect its absence is by observing and possibly counting the existing edges. As an agent cannot distinguish $n-2$ from $n-1$ incident edges (due to limited local storage), it seems that it cannot distinguish a clique from a non-clique in which every agent has $n-2$ incoming and $n-2$ outgoing incident edges (proving this is an interesting open problem). This indicates that extra power can possibly be obtained by the ability of the MGP model to read/write edges "missing" from the input graph. In the sequel, we shall see that indeed there is plenty of extra power obtained (Theorem 19).

9.1. MGPs with Stabilizing Input Graphs

In this section, we define the *stabilizing input graphs MGP* (*SIMGP*) model (similar to the PP model with stabilizing inputs [AAC$^+$05]), in which the initial state of each agent and edge of the interaction graph (and thus the input graph) may change finitely many times before it stabilizes to a final value. Here, we consider the computational capabilities of the MGP model when the network initialization function is working in parallel (and not as a preprocessing on the network) with the execution of an MGP, as if another protocol eventually designates the input graph for an MGP. We are interested in stably deciding membership of the stabilized input graph in a graph language. Intuitively, one can think of the case where we are concerned about properties of a dynamic overlay network (which could be a result of a protocol running on a complete network infrastructure, e.g., a peer-to-peer network over the Internet) where the overlay can initially change but eventually stabilizes.

In a similar way to that of [AAC$^+$05], let each agent/edge store its current initial state (which corresponds to the initial value given by ι) to a special component of its state. This state is available to the agent/edge at every computation step and may change arbitrarily between any two subsequent steps. All the possible values, however, belong constantly to $\{s_0, s_1\}$ for the edges and to $\{q_0, q_1\}$ for the agents. The transition function is now of the form $\delta : ((Q \times \{q_0, q_1\}) \times (Q \times \{q_0, q_1\}) \times (S \times \{s_0, s_1\})) \to (Q \times Q \times S)$ and a configuration is a mapping $C : V \cup E \to (Q \times \{q_0, q_1\}) \cup (S \times \{s_0, s_1\})$ specifying the state and current initial state of each agent and edge in the interaction graph. If $\delta(a, b, s) = (a', b', s')$, where $a, b \in (Q \times \{q_0, q_1\})$, $a', b' \in Q$, $s \in (S \times \{s_0, s_1\})$ and $s' \in S$, we call $(a, b, s) \to (a', b', s')$ a *transition* and we define $\delta_1(a, b, s) = a'$, $\delta_2(a, b, s) = b'$, $\delta_3(a, b, s) = s'$. We denote by $C^q(t)$ and $C^\iota(t)$ the state and initial state respectively of any $t \in V \cup E$ in C. The initial configuration has any agent u and edge s in a (*state, initial state*) tuple, (q_i, q_i) and (s_i, s_i) respectively where $q_i \in \{q_0, q_1\}$ and $s_i \in \{s_0, s_1\}$ are the initial agent and edge states given by the network initialization function. The output of a configuration C is obtained by applying the output function O to the agent state components C^q of C.

The definition of one-step transition via a certain encounter is the same as in the GDMPP model but instead we use the $\delta_1, \delta_2, \delta_3$ definitions as described above. That is, if C and C' are configurations, and u, v be distinct agents we say that C goes to C' via encounter $e = (u, v)$, denoted $C \xrightarrow{e} C'$, if $C'^q(u) = \delta_1(C(u), C(v), C(e))$, $C'^q(v) = \delta_2(C(u), C(v), C(e))$, $C'^q(e) = \delta_3(C(u), C(v), C(e))$, $C'^q(w) = C^q(w), \forall w \in (V - \{u, v\})$, and $C'^q(z) = C^q(z), \forall z \in (E - \{e\})$. Note that there is no restriction on the initial states C^ι of C.

Executions, fairness and computations are defined as in the GDMPP model. We distinguish the computations in which the input graph stabilize as in [AAC$^+$05]. We say a computation $C_0, C_1, \ldots$ is a stabilizing input graph computation if there is some finite step m after which no initial state is modified on any agent or edge. More formally, there is a final assignment of agents' and edges' states given by the network initialization function ι, $\iota_f : V \cup E \to \{q_0, q_1, s_0, s_1\}$ such that $C^\iota_n(t) = \iota_f(t)$ for every $t \in V \cup E$ and every $n \geq m$. A stabilizing input graph computation $C_0, C_1, \ldots$ is fair if for every configuration C that occurs infinitely often in the computation, and every configuration C' such that the assignment of ι in C' is equal to the assignment of ι in C and $C \to C'$, C' also occurs infinitely often in the computation.

In the next theorem, we show that every MGP-decidable language is stably decidable even if the input graph is initially changing for a finite number of steps. To do so, we construct a protocol similar to the

one presented in [CMN+10, MCS11a]. That protocol is also executed on complete interaction graphs. It constructs a correctly labeled spanning pseudo-subpath [5] of the interaction graph and then exploits this construction to simulate a NTM on its input assignment. Informally, a pseudo-path is a straight line with arbitrary link directions (see Figure 3 for some examples of correctly labeled pseudo-subpaths of a complete interaction graph). The agents become ordered according to this line and all the remaining edges of the complete interaction graph (those that are not part of the line) form the tape cells of the TM. These can be visited in an ordered fashion due to the ordering of the agents. Let L be any graph language:

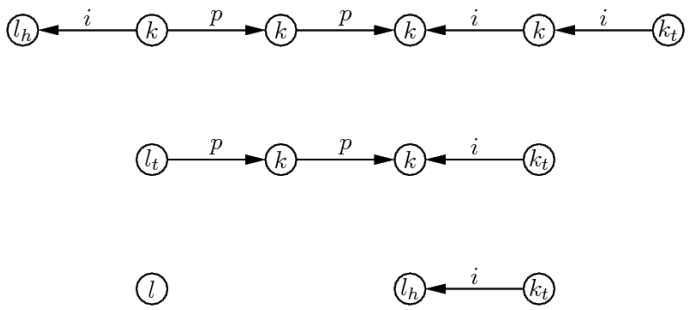


Figure 3: We assume that the above depicted graph, call it K_n, is complete. We have chosen not to draw the inactive edges for the sake of visibility. Therefore, all edges not appearing have label 0, that is, they are inactive. The top and middle six-node and four-node graphs respectively form two correctly labeled pseudo-paths. Each such graph features a leader endpoint (denoted by either l_h or l_t labels), a non-leader endpoint, (denoted by label k_t), non-leader intermediate nodes (with label k) and a set of edges with labels p or i according to their orientation w.r.t. the two endpoints. All other edges (which do not appear in the figure) incident to the previous nodes are inactive (labeled by 0).Similarly, all other graphs that appear are pseudo-subpaths of the complete graph K_n. Note that the left node at the bottom that seems to be isolated, is in fact a node of K_n whose incident edges are all inactive. Moreover, it has label l, so it constitutes a trivial pseudo-subpath of K_n.

Theorem 15. *L is stably decidable by the SIMGP model iff it is MGP-decidable.*

Proof. The "if" direction is trivial since we focus on languages of stabilized input graphs (thus once the input graph has stabilized there is an MPG that works as the corresponding SIMGP). For the "only if" direction, let $\mathcal{A}(Q_{\mathcal{A}}, S_{\mathcal{A}}, O_{\mathcal{A}}, \delta_{\mathcal{A}})$ be an MGP that stably decides L. To prove the above statement we are going to use the reinitialization technique used in [CMN+10]. We will construct a stabilizing input graphs protocol $\mathcal{B}$ that runs in parallel $\mathcal{A}$ and a protocol $\mathcal{I}$ (similar to that of [CMN+10]) which organizes the population into a correctly labeled, spanning pseudo-path and reinitializes $\mathcal{A}$'s execution every time the input graph changes. The state of each agent of protocol $\mathcal{B}$ has 3 components. The first stores a state in $\{q_0, q_1\}$ which is the current initial state according to the network initialization function ι, the second stores a state $q \in Q_{\mathcal{A}}$ and is used for the execution of $\mathcal{B}$ (this is practically the component where $\mathcal{A}$'s execution takes place) and the third is used for the construction of the spanning pseudo-path and the application of reinitialization process. The case of the edges' states is similar. Note that both the construction of the pseudo-path and the reinitialization process require a complete interaction graph which we have in the case of the MGP model. The output alphabet of $\mathcal{B}$ is the same as $\mathcal{A}$'s. $\mathcal{B}$'s output is the output taken by applying $\mathcal{A}$'s output function to the second state component of the agents. We give a high level description of $\mathcal{B}$'s transition function.

The main idea is the following: $\mathcal{B}$ starts with constructing a spanning pseudo-path of its agents using the third state-component. This takes place by labeling the edges of the interaction graph as active or inactive according to whether they belong to the pseudo-path or not, and by marking (setting in special states) the two endpoints of the pseudo-path as well as the intermediate agents. All intermediate agents are set in the same special state (*simple nonleader*) whereas each endpoint has its own special state, *leader* and *nonleader*. The process of organizing all agents of the population into a pseudo-path with distinct endpoints

[5] It was called a *line graph* in [CMN+10], but since this has another standard meaning in graph theory, that named changed to *pseudo-path* in [MCS11a].

is called *spanning process*. During this process, multiple pseudo-subpaths can co-exist before all agents of the population are part of a single pseudo-path. Such subgraphs are merged through the interaction of their leader endpoints (only); this way, cycles formed when a head interacts with its own tail are avoided. Once the leaders of two subgraphs are engaged in the merge of their respective pseudo-paths they go to a special blocking state to prevent any further merge interactions and thus the process avoids the formation of cycles. In addition, one of the two subgraphs gradually reverses its orientation in order to form and extended pseudo-path consisting of the agents and edges of both previous graphs plus an additional edge via which the leaders interaction took place. This process is described in great detail in [CMN$^+$10].

As in [CMN$^+$10] whenever the construction is updated (by merging pseudo-subpaths of the interaction graph) a reinitialization takes place on the population restoring the initial states of $\mathcal{B}$. We say that $\mathcal{B}$'s initial states are restored by copying in each agent and edge the contents of the first state-component to the second. The *reinitialization process* is performed by having two other special states traversing (via the active edges) the pseudo-path in an orderly fashion from one endpoint to the other, marking in this way the ends of the next edge to be reinitialized (the agents get reinitialized just by getting to these special states and the corresponding edge on their next interaction). The ordering of the reinitialization process encodes a form of timestamp for its state (w.r.t. to which agents have been reinitialized so far). When previously mentioned states traverse the complete pseudo-path (for more details on the exact execution see [CMN$^+$10]) then all agents and edges have been reinitialized. The order prevents reinitialized nodes to interact effectively (i.e. resulting in modification of an agent's state) with non-reinitialized nodes as such interactions would result in wrong executions (outdated info interacting with up to date info). According to process described in [CMN$^+$10] the contents of the 3^{rd} component will eventually stabilize (when the spanning process is complete).

In addition, whenever an agent's or edge's first-component is modified, that is the input graph provided by ι changes, a new reinitialization mark (a special state as it is called in [CMN$^+$10]) is generated and travels towards the leader endpoint (via the active edges). This mark generation happens in the next interaction in which the corresponding agent/edge participates.[6] Once the mark reaches the endpoint it initiates a reinitialization process (as the one described above). Note that many reinitialization marks may travel towards the leader endpoint. The general rule is that priority is given to the one that is directed towards the leader endpoint last (or equivalently the one closest to the nonleader endpoint). If a mark meets another one during its motion towards the leader endpoint it erases it. Moreover, if the mark moving towards the leader meets the marks (special states) used for the reinitialization process it erases them.

After all initial states are stabilized, no more reinitialization marks will be generated. Once the initial states stabilize and the spanning process is complete, $\mathcal{B}$'s execution will not get reinitialized again and $\mathcal{A}$ will be executed undisturbed (in the second state component of $\mathcal{B}$) on the stabilized (final) input graph. The resulting execution will provide the output of $\mathcal{A}$ as if it was (separately) running on the same stable input graph. □

9.2. Composition of MGPs

We will now present another property of MGPs. According to this property, which we call *MGP-composition*, given two MGPs $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{A}$ is a stabilizing output graph MGP, we can compose the two protocols to a new protocol $\mathcal{D}$. $\mathcal{D}$ will have the same output as $\mathcal{B}$ as if the latter was running on the stabilizing input graph defined by $\mathcal{A}$'s execution. $\mathcal{B}$'s input graph, provided by $\mathcal{A}$'s execution, is stabilizing since the edges' states eventually stabilize (by $\mathcal{A}$'s definition) and $\mathcal{A}$'s outputs 0 and 1 can be trivially mapped to initial agent states q_0 and q_1 respectively. The property is formalized below:

Theorem 16. *For any two MGPs $\mathcal{A}, \mathcal{B}$ where $\mathcal{A}$ is a stabilizing output graph MGP, there is an MGP $\mathcal{D}$ which is a composition of $\mathcal{A}$ and $\mathcal{B}$, has as input the input graph of $\mathcal{A}$ and as output the output of $\mathcal{B}$ running on the stabilized input graph provided by $\mathcal{A}$.*

[6] If an agent's first state component is modified, the mark is generated at that agent. If an edge's first state component is modified, then the mark is generated on the initiator of the interaction.

Proof. From Theorem 15, we have that $\mathcal{B}$ can be replaced by a stabilizing input graphs protocol $\mathcal{B}'$ that runs on the stabilizing graph defined by $\mathcal{A}$ and works exactly like $\mathcal{B}$ running on the same graph. □

10. MGPs on Disconnected Input Graphs

We now discuss the capability of the MGP model to decide languages on disconnected graphs. Neither the PP model [AAC+05] nor the GDMPP model are capable of supporting this feature. We here exploit the complete infrastructure to communicate information between the connected components of the disconnected input graph and make a decision according to the exchanged information. In Sec. 10.1, we present a simple protocol that can decide whether the input graph is a connected graph and, in Sec. 10.2, we generalize the idea to prove that any semilinear predicate on the multiset of decisions of any GDMPP running on the connected components (where the decision of each component is counted only once) that constitute the input graph is stably decidable.

10.1. Deciding Connectivity

In this section, we present an MGP CP that decides the language $L_C = \{G \mid G \textit{ is a connected graph}\}$.

Protocol 2 *Connectivity Protocol (CP)*

1: $Q = \{q_0, q_1, f, f', l, l'\}$, $S = \{s_0, s_1\}$,
2: $O(q_0) = 0, O(q_1) = 0, O(f) = 1, O(f') = 0, O(l) = 1, O(l') = 0$,
3: δ:

a single leader is generated
$(q_1, q_1, s_1) \to (l, f, s_1)$

the single leader turns all nodes of the input graph it can reach to followers
$(l, f, s_1) \to (f, l, s_1), (f, l, s_1) \to (l, f, s_1), (l, q_1, s_1) \to (f, l, s_1), (q_1, l, s_1) \to (l, f, s_1)$

the single leader turns all nodes that do not belong to the input graph to followers of a single leader
$(l, q_0, s_0) \to (l, f, s_0)$

two single leaders meet in the same connected component of the input graph; one is turned to follower
$(l, l, s_1) \to (l, f, s_1)$

two nonadjacent single leaders meet (via an edge that is not in the input graph) in the same connected component of the input graph or in different connected components (in the case of disconnected input graph); they become nonunique leaders
$(l, l, s_0) \to (l', l', s_0)$

the nonunique leaders turn all the nonleaders they meet into their followers
$(l', f, s_1) \to (f', l', s_1), (f, l', s_1) \to (l', f', s_1), (l', q_1, s_1) \to (f', l', s_1), (q_1, l', s_1) \to (l', f', s_1), (l', q_0, s_0) \to (l', f', s_0), (l', f, s_0) \to (l', f', s_0)$

two leaders of any type meet in the same component; only one single leader remains
$(l', l, s_1) \to (l, f, s_1), (l, l', s_1) \to (l, f, s_1), (l', l', s_1) \to (l, f, s_1)$

two leaders of any type meet in different components; they both become nonunique leaders
$(l', l, s_0) \to (l', l', s_0), (l, l', s_0) \to (l', l', s_0)$

a single leader restores all followers of multiple leaders to followers of single leaders
$(l, f', s_0) \to (l, f, s_0), (l, f', s_1) \to (f, l, s_1), (f', l, s_1) \to (l, f, s_1)$

Theorem 17. *Protocol CP stably computes L_C for any input graph $G_\iota[K_n]$ on the complete interaction graph K_n.*

Proof. Note that CP has eventually one leader in each connected component. If the input graph is connected there is a single connected component and therefore only a single leader remains. This leader turns all other agents of K_n to followers of the single leader and the protocol stabilizes to 1. If the input graph is not connected it consists of multiple connected components and therefore multiple leaders will exist in K_n. These leaders will eventually interact with each other via an edge that does not belong in the input graph (since the graph that CP runs on is complete) and they will eventually become nonunique leaders. Since, however, they belong in different connected components they cannot interact via an edge of the input graph (in state s_1) and thus they will remain nonunique forever. The nonunique leaders will eventually turn the rest of the population to followers of multiple leaders (states t') and all agents will stabilize to 0. □

10.2. Computing Graph Languages on Disconnected Graphs

In this section, we are interested in languages that describe properties of disconnected input graphs, that is, graphs with > 1 connected components. We use the term "connected components" for weakly-connected components as well. To achieve this, we propose a construction which combines the functionality of four MGPs that allow information exchange between the connected components by exploiting the complete underlying infrastructure. In what follows, we give a description of these protocols.

First, we have the *spanning pseudo-path protocol* described in Section 9.1 that is required for the composition of protocols. We will call it RP (*Reinitialization Protocol* since it is mainly used to reinitialize the execution of the composed protocols).

Then, there is a *Leader Election MGP* (LE) that practically runs on the connected components of the input graph (leaving all q_0 agents of the population intact). $LE = \{Q_{LE}, S_{LE}, \delta\}$, where $Q_{LE} = \{q_0, q_1, l, f\}$, $S_{LE} = \{s_1\}$ and δ has the following transitions: $(q_1, q_1, s_1) \to (l, f, s_1)$ in which a leader is generated; $(l, q_1, s_1) \to (f, l, s_1)$ and $(q_1, l, s_1) \to (l, f, s_1)$ via which the leader turns nonleaders to followers; $(l, f, s_1) \to (f, l, s_1)$ and $(f, l, s_1) \to (l, f, s_1)$ which allow the leader state to move among the agents of a connected component; $(l, l, s_1) \to (l, f, s_1)$ which removes multiple leaders; $(q_1, q_1, s_0) \to (l, l, s_0)$ and $(l, q_1, s_0) \to (l, l, s_0)$ in which leaders are generated from initial states through interaction of agents of different components (these interactions are useful for generating leaders in components consisting of a single isolated node). Interactions between agents not defined by the previous transitions are ineffective (leave the states of both agents unchanged).

Lemma 12. *LE eventually elects a unique, constantly moving leader in each connected component of the input graph.*

Proof. The functionality is similar to the one of Protocol 2 of Sec. 10.1 without the interactions between leaders of different connected components and with the additional interactions between agents in different connected components in which at least one of the agents is in the initial state. These additional interactions enable the eventual creation of a leader in components consisting of a single agent. The remaining non-isolated leader moves constantly due to the transitions $(l, f, s_1) \to (f, l, s_1)$ and $(f, l, s_1) \to (l, f, s_1)$. □

The third protocol is a parameter to the composition. It can be any MGP that works only within the connected components (effective interactions take place only between agents linked with s_1 edges). This protocol, that we call BGP (*Basic Graph Protocol*) hereafter, is practically a GDMPP extended to also run on graphs with isolated nodes (remember that in section 4, GDMPPs were defined for the graph universe $\mathcal{H}$ which only includes graphs with node set size greater than 2). Such an extension can be trivially performed since any protocol running on an isolated node can only decide the trivial language of graphs consisting of a single isolated node. Such a case can be trivially detected by a simple population protocol (where every agent initially outputs 1 and changes its state and output to 0 once an interaction takes place). BGP runs in parallel with LE within the connected components of the input graph and decides the same graph property within each component. This means that, *once all components of the input graph stabilize w.r.t. BGP's execution, all agents within each component will output* 1 *if the component satisfies the property and* 0 *otherwise*. Obviously, agents of different components may have different outputs.

The parallel execution of LE and BGP ends up with a unique moving or static leader in each connected component, all other agents are followers and every agent knows the decision of BGP for the component it

belongs to. An agent that is not part of the input graph (q_0) is not affected and outputs by default 0. For each connected component of the input graph:

Definition 4. *We call an agent representative of the component if it is the unique leader and its output w.r.t. BGP has stabilized.*

In other words, once the number of leaders stops changing and BGP stabilizes, the unique leader of each connected component becomes a representative (the elected agent that bears the decision of the component w.r.t. the graph property that BGP decides). Note that regardless of the movement of the leader state within each component, once BGP stabilizes, the leader's output w.r.t. BGP remains the same no matter which agent is the leader.

The final protocol is also a parameter to the composition and is practically a population protocol (see [AAC+05]) running on the population of representatives. We call this protocol REP (*REpresentative Protocol*) and it runs in parallel with RP, LE and BGP. Since the interaction graph of MGPs is complete the representatives' population will be fully connected via the s_0 edges. The inputs of REP will be the outputs (decisions on the satisfiability of the graph property) of the agents w.r.t. BGP. We consider that effective interactions w.r.t. to REP can take place only between the representatives (via s_0 edges) of the population. In addition, we demand that whenever a representative moves to a neighboring agent within its component (since the leaders constantly move), it also copies its REP-state component to that agent. We consider that the output of REP is the output of the whole composition and we extend REP so that the representatives propagate their state when interacting with q_0 agents. In this way, all agents (the followers in each component due to representatives' movement and the q_0 agents due to the previous extension) will eventually have in their REP-state components the contents of the representatives' REP-state components.

The composition of the 4 protocols: In order for REP to be executed correctly the population of representatives must have been formed. Thus, LE must generate unique moving (or static) leaders and BGP must stabilize before REP can start its execution. To achieve this we do the following: RP, LE, BGP and REP run in parallel as stated above. RP constructs in a finite number of steps the spanning pseudo-path that allows the reinitialization of components. Any reinitialization taking place during RP's execution restore all LE, BGP and REP state components to their initial values. Once RP stabilizes, LE and BGP are executed in parallel without being reinitialized. Every time the output of an agent w.r.t. BGP changes or a leader is turned to a follower by the transition $(l, l, s_1) \to (l, f, s_1)$ of LE a new special reinitialization takes place that only restores the REP state components of the agents. Once the number of leaders stops changing and BGP stabilizes the population of representatives has been formed and no more reinitialization can take place. Then REP is executed on the correctly reinitialized representatives.

For all G, denote by $N_{G,L}$ the number of components of G that belong to a language L and by $N_{G,\overline{L}}$ the number of components of G that do not.

Theorem 18. *Let L be a GDMPP-decidable language. Let p be a semilinear predicate on $\mathbb{N}^2$. Then $L' = \{G \mid p(N_{G,L}, N_{G,\overline{L}}) = 1\}$ is MGP-decidable.*

Proof. The previously described composition of LE, RP, BGP, REP takes an input graph given by some network initialization function and computes any semilinear predicate (due to REP) on the decisions (outputs of BGP) of the connected components (each component's decision is counted only once) of this input graph concerning any GDMPP-decidable graph property (since BGP is a essentially a GDMPP). □

The applications of Theorem 18 are various, depending on the BGP and REP we use. Given a GDMPP-decidable graph language, e.g., $L = \{G \mid G$ contains at least one 2-cycle$\}$, we can now answer questions about predicates on the number of components that satisfy L; questions like whether at least 25% of the components contain at least one 2-cycle. Whether the whole graph contains at least one 2-cycle can be simply decided by an OR population protocol [7] on the representatives' population. Moreover, the previous discussion shows that the MGP model is computationally stronger than the GDMPP and thus PP models

[7] A population protocol implementing the logical OR function on its inputs.

for graph language decidability since it allows decidability of nontrivial graph languages on disconnected input graphs. This may seem somewhat expected given the completely connected infrastructure available in MGPs but the previous construction/composition shows how we can systematically organize our protocols for the case of disconnected graphs and how we can study their computability in this case.

11. An Exact Characterization of MGPs: GMGP = LGNSPACE

In this section, we will develop an MGP that is capable of simulating a linear-space NTM on input $G_\iota[K_n] = (V', E')$. In this manner, we establish that any graph language $L \in$ **LGNSPACE** is stably decidable by the MGP model, or equivalently that **LGNSPACE** $\subseteq$ **GMGP**. By showing that the converse is also true, we conclude that the inclusion holds with equality. The main idea for establishing the lower bound is the following. The protocol is similar to the one presented in Sec. 9.1 (which is similar to the one described in [CMN$^+$10, MCS11a]). Here, we preserve the process that constructs the correctly labeled spanning pseudo-subpath, called the *spanning process*, we also preserve the *reinitialization process*, and we slightly modify the *simulation process*. The reinitialization process was responsible for reinitializing the simulation process whenever the spanning process takes some step; it restores the tape cells and the state of the TM. The simulation process is the actual simulation of the TM. In [CMN$^+$10, MCS11a], the simulation process is responsible of writing the input values of all agents to the leftmost cells of the distributed simulation tape, just before starting the actual simulation. Here, the main difference is that we do not have some input assignment to store, rather we will store to the TM's tape the input graph $G_\iota[K_n]$ by its adjacency matrix representation. When this is done, as in [CMN$^+$10, MCS11a], the simulation process starts simulating the TM that decides the corresponding graph language.

Now, consider the following 3-component initial configuration: According to the first component, called the *label*, there is a unique spanning correctly labeled pseudo-subpath $L = (V, A)$ of a complete interaction graph K_n. [8] The second component stores the values of some network initialization function ι and is called the *membership indicator*. The third component is the (simulation) *tape* and is initially in a blank state.

Lemma 13. *There is an MPP that in any computation on K_n, beginning from such a configuration, stores the input graph $G_\iota[K_n]$ in the leftmost cells of the tape and halts in a finite number of steps having preserved the initial states of both the label and the membership indicator components.*

Proof. In [CMN$^+$10, MCS11a], it was proved that, given such an initial configuration, there is an MPP running on the complete graph K_n, capable of visiting one by one all of its edges and nodes. The main idea is the following. There is always a unique agent with a star mark in each state that orders its incident edges by the position of their opposite endpoints on the pseudo-path. To visit its outgoing edges one after the other according to this ordering, the star agent places a dot mark that traverses the pseudo-path from left to right; it waits to interact with the dot agent and then moves the dot one step to the right. Formally, define the distance of each node $u \in V$ as the length of the unique pseudo-path that begins from the left endpoint of L (which is the unique node in V with label l) and leads to u and denote it by $d(u)$. Nodes become stars one after the other until all edges of K_n are visited. It is not hard to extend the above protocol to store the adjacency matrix of $G_\iota[K_n]$ in the leftmost cells of the distributed tape. Initially, the protocol creates in the leftmost cells of its tape a $n \times n$ adjacency matrix initialized to 0. To do that, it simply has to count in its tape the number of nodes in V. Then the protocol creates in its tape two binary counters *star* and *dot* and initializes both to 0. The counters *star* and *dot* are responsible for storing the distances of the current star agent and the current dot agent, respectively. Whenever the star or dot mark is moved one position to the right the corresponding counter is incremented by one. Whenever an interaction between the star agent and the dot agent takes place via an edge whose membership indicator is s_1, the cell $(star, dot)$ of the adjacency matrix is set to 1. It is clear that at the end of this process, a cell (i, j) of the adjacency matrix is 1 iff $(i, j) \in E'$. Moreover, the outlined process never alters the membership indicator components and

[8] Note that, by definition of a correctly labeled pseudo-subpath, it holds that for all $e \in E - A$, e is inactive.

any modification of the label components is done via the placement of a star or a dot mark, which can be thought as occurring in a separate component than never alters the actual labels. Fairness ensures that the process halts in a finite number of steps. Note that the input graph can be stored in a more succinct manner by creating a $t \times t$ adjacency matrix, where t is the order of $G_\iota[K_n]$, and by counting in the distance of a node only the nodes whose membership indicator is q_1. However, in the worst case, the two representations are equivalent. □

We are now ready to establish the following exact characterization of **GMGP**:

Theorem 19. GMGP = LGNSPACE.

Proof. For the **LGNSPACE** $\subseteq$ **GMGP** direction, take any graph language $L \in$ **LGNSPACE**, where the graphs are provided in terms of their adjacency matrix representations, and let $\mathcal{M}$ be the NTM that decides it in $\mathcal{O}(n)$ space. We describe an MGP that stably decides L by simulating $\mathcal{M}$. Take any complete interaction graph K_n and any input graph $G_\iota[K_n] \subseteq G$. The MGP consists of the 3 processes previously described in this Section, and the simulation process begins by executing the graph storage procedure as described in Lemma 13. It was proved in [CMN$^+$10, MCS11a] that the composition of these processes correctly constructs a spanning pseudo-subpath of G. Since the simulation is again here executed in its own separate component that never alters the labels and the membership indicators, the aforementioned result is carried over to our construction. When this happens, the simulation process stores the adjacency matrix of $G_\iota[K_n]$ to the leftmost cells of the distributed tape. In the worst case, $G_\iota[K_n] = G$ and the size of the adjacency matrix is $\mathcal{O}(n^2)$ (one bit for each edge), thus there is always enough distributed space to store the input graph. Moreover, each of the *star* and *dot* counters is upper bounded by n ($\log n$ bits in binary), thus again there is enough room to store them. $\mathcal{M}$ runs in linear space, consequently, capability of storing the graph implies capability of executing the simulation in the available space. Finally, nondeterminism can be easily drawn from the one inherent in the interaction pattern. However, since only one nondeterministic path can be followed at a time, whenever a rejecting state is reached the simulation starts over from the beginning. Fairness ensures that a correct search on the tree of $\mathcal{M}$'s nondeterministic computation is eventually performed by the protocol.

The inclusion **GMGP** $\subseteq$ **LGNSPACE** follows easily. Take any protocol $\mathcal{A}$ that stably decides a language L and any graph G. $\mathcal{A}$ stably decides membership of G in L also in the worst case in which the node set of the interaction graph is the node set of the input graph G (that is, the only possible additional space comes from the edges missing from G). The NTM $\mathcal{N}$ takes G in its input by its adjacency matrix representation which means that if G has k nodes then $\mathcal{N}$ has $\mathcal{O}(k^2)$ available space which is equal to the space available to $\mathcal{A}$ in its worst case. Then $\mathcal{N}$ simply performs a nondeterministic search on the worst-case transition graph of $\mathcal{A}$ by always storing at most one configuration (see, e.g., Theorem 8 in [CMS09a]). □

12. Conclusions - Future Research Directions

Many interesting issues arise by the findings of our work. The ability of the MGP model to use its complete network infrastructure enables us to compose protocols and decide graph properties of disconnected graphs. The additional memory provided by the extra edges of the complete interaction graph gives an important advantage to MGPs in comparison to GDMPPs. However, extra nodes do not seem to help: after all, in our model, the worst-case interaction graph of any input graph is itself made complete. Various questions arise from the above conclusions. How would the computability be affected if we had allowed more memory in each agent or each edge? Which interaction graph topologies allow the full use of the distributed memory? Do we truly require a complete interaction graph to decide graph languages in disconnected graphs or a connected infrastructure would suffice? How can we exploit the presence of extra nodes for increasing the computational power?In addition, our results use a fairness model that provides no insight to the time-complexity of our protocols. It would be interesting to assume restricted, probabilistic, or worst-case adversaries so that we can also analyze our protocols w.r.t. time-performance. Such an approach was followed in [AAD$^+$06], in which probabilistic schedulers were assumed that enabled a time-complexity study. Worst-case adversaries

for dynamic networks have recently been considered in [OW05, KLO10, MCS12]. In each of these works, some temporal restriction on end-to-end communication is imposed on the adversary (essentially an upper bound on the time needed for information to influence another node) that allows for a worst-case analysis of time-performance.

It turns out that, in the family of all complete digraphs, the existence of a leader in the population does not increase the computational power of the PP model. It would be interesting to show that it suffices for the leader to access every other agent's memory exactly once. This behavior is similar to that of some automata known as *multiset automata*, like, e.g., MFAs [CVMVM01], where the input multiset is consumed by a coordinator before an acceptance decision is made. In addition, the ability of MPPs to organize the population into a NTM (as shown in [CMN$^+$10]) shows that a leader does not help MPPs either. Are the above true also for other families of interaction graphs? Finally, what is the computational power of the extensions of the above models in which the protocols have access to a mechanism notifying them of the time needed to causally influence and get influenced by the states of the whole population? It seems that such protocols can perform iterative computations and possibly halt in some cases.

References

[AAC$^+$05] D. Angluin, J. Aspnes, M. Chan, M. J. Fischer, H. Jiang, and R. Peralta. Stably computable properties of network graphs. In V. K. Prasanna, S. Iyengar, P. G. Spirakis, and M. Welsh, editors, *Distributed Computing in Sensor Systems: First IEEE International Conference, DCOSS 2005, Marina del Rey, CA, USE, June/July, 2005, Proceedings*, volume 3560 of *Lecture Notes in Computer Science*, pages 63–74. Springer-Verlag, June 2005.

[AAD$^+$04] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 290–299, New York, NY, USA, 2004. ACM.

[AAD$^+$06] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, pages 235–253, mar 2006.

[AAE08] D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21[3]:183–199, September 2008.

[AAER07] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20[4]:279–304, November 2007.

[ÀCD$^+$11] C. Àlvarez, I. Chatzigiannakis, A. Duch, J. Gabarró, O. Michail, S. Maria, and P. G. Spirakis. Computational models for networks of tiny artifacts: A survey. *Computer Science Review*, 5[1], January 2011.

[AR07] J. Aspnes and E. Ruppert. An introduction to population protocols. *Bulletin of the European Association for Theoretical Computer Science*, 93:98–117, October 2007.

[BCC$^+$09] O. Bournez, P. Chassaing, J. Cohen, L. Gerin, and X. Koegler. On the convergence of population protocols when population goes to infinity. *Applied Mathematics and Computation*, 2009. To appear.

[CDF$^+$09] I. Chatzigiannakis, S. Dolev, S. P. Fekete, O. Michail, and P. G. Spirakis. Not all fair probabilistic schedulers are equivalent. In *OPODIS '09: Proceedings of the 13th International Conference on Principles of Distributed Systems*, pages 33–47, Berlin, Heidelberg, 2009. Springer-Verlag.

[CMN+10] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P. G. Spirakis. All symmetric predicates in $NSPACE(n^2)$ are stably computable by the mediated population protocol model. In *35th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 6281 of *Lecture Notes in Computer Science*, pages 270–281. Springer-Verlag, August 23–27 2010.

[CMN+11a] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P. G. Spirakis. Passively mobile communicating machines that use restricted space. In *Proceedings of the 7th ACM ACM SIGACT/SIGMOBILE International Workshop on Foundations of Mobile Computing*, FOMC '11, pages 6–15, New York, NY, USA, 2011. ACM.

[CMN+11b] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P. G. Spirakis. Passively mobile communicating machines that use restricted space. *Theor. Comput. Sci.*, 412:6469–6483, October 2011.

[CMNS11] I. Chatzigiannakis, O. Michail, S. Nikolaou, and P. G. Spirakis. The computational power of simple protocols for self-awareness on graphs. In *Proceedings of the 13th international conference on Stabilization, safety, and security of distributed systems*, SSS'11, pages 135–147, Berlin, Heidelberg, 2011. Springer-Verlag.

[CMS09a] I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Mediated population protocols. In *36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*, volume 5556 of *LNCS*, pages 363–374, July 2009.

[CMS09b] I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Recent advances in population protocols. In *MFCS '09: Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science 2009*, pages 56–76, Berlin, Heidelberg, 2009. Springer-Verlag.

[CMS10] I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Stably decidable graph languages by mediated population protocols. In S. Dolev, J. Cobb, M. Fischer, and M. Yung, editors, *Stabilization, Safety, and Security of Distributed Systems*, volume 6366 of *Lecture Notes in Computer Science*, pages 252–266. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-16023-3_21.

[CS08] I. Chatzigiannakis and P. G. Spirakis. The dynamics of probabilistic population protocols. In *DISC '08: Proceedings of the 22nd international symposium on Distributed Computing*, pages 498–499, Berlin, Heidelberg, 2008. Springer-Verlag.

[CVMVM01] E. Csuhaj-Varjú, C. Martin-Vide, and V. Mitrana. Multiset automata. In *Proceedings of the Workshop on Multiset Processing: Multiset Processing, Mathematical, Computer Science, and Molecular Computing Points of View*, WMP '00, pages 69–84, London, UK, 2001. Springer-Verlag.

[Dij74] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17:643–644, November 1974.

[Dol00] S. Dolev. *Self-stabilization*. MIT Press, Cambridge, MA, USA, 2000.

[FJ06] M. Fischer and H. Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In A. Shvartsman, editor, *Principles of Distributed Systems*, volume 4305 of *Lecture Notes in Computer Science*, pages 395–409. Springer Berlin / Heidelberg, 2006. 10.1007/11945529_28.

[GMK02] I. Georgiadis, J. Magee, and J. Kramer. Self-organising software architectures for distributed systems. In *Proceedings of the first workshop on Self-healing systems*, WOSS '02, pages 33–38, New York, NY, USA, 2002. ACM.

[GMM04] E. Godard, Y. Mtivier, and A. Muscholl. Characterizations of classes of graphs recognizable by local computations. *Theory of Computing Systems*, 37:249–293, 2004. 10.1007/s00224-003-1062-1.

[GR09] R. Guerraoui and E. Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In *ICALP (2)*, pages 484–495, 2009.

[GS66] S. Ginsburg and E. H. Spanier. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.

[HKLPR03] H.-G. Hegering, A. Küpper, C. Linnhoff-Popien, and H. Reiser. Management challenges of context-aware services in ubiquitous environments. In M. Brunner and A. Keller, editors, *Self-Managing Distributed Systems*, volume 2867 of *Lecture Notes in Computer Science*, pages 321–339. Springer Berlin / Heidelberg, 2003. 10.1007/978-3-540-39671-0_26.

[KLO10] F. Kuhn, N. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 513–522, New York, NY, USA, 2010. ACM.

[MCS11a] O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Mediated population protocols. *Theor. Comput. Sci.*, 412:2434–2450, May 2011.

[MCS11b] O. Michail, I. Chatzigiannakis, and P. G. Spirakis. *New Models for Population Protocols*. N. A. Lynch (Ed), Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2011.

[MCS12] O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Causality, influence, and computation in possibly disconnected dynamic networks. *Arxiv preprint arXiv:1206.1290*, 2012.

[OW05] R. O'Dell and R. Wattenhofer. Information dissemination in highly dynamic graphs. In *Proceedings of the 2005 joint workshop on Foundations of mobile computing*, DIALM-POMC '05, pages 104–110, New York, NY, USA, 2005. ACM.

[Spi10] P. G. Spirakis. *Theoretical Aspects of Distributed Computing in Sensor Networks*, chapter Population Protocols and Related Models. Springer-Verlag, 2010.

[VvS03] S. Voulgaris and M. van Steen. An epidemic protocol for managing routing tables in very large peer-to-peer networks. In M. Brunner and A. Keller, editors, *Self-Managing Distributed Systems*, volume 2867 of *Lecture Notes in Computer Science*, pages 299–308. Springer Berlin / Heidelberg, 2003. 10.1007/978-3-540-39671-0_5.