The Computational Power of Simple Protocols for Self-Awareness on Graphs*

Ioannis Chatzigiannakis[1,2], Othon Michail[1], Stavros Nikolaou[1,2], and Paul G. Spirakis[1,2]

[1] Research Academic Computer Technology Institute (CTI), Patras, Greece
[2] Computer Engineering and Informatics Department (CEID), University of Patras
Email: {ichatz, michailo, nikolaou, spirakis}@cti.gr

Abstract. We explore the capability of a network of extremely limited computational entities to decide properties about any of its subnetworks. We consider that the underlying network of the interacting entities (devices, agents, processes etc.) is modeled by a complete *interaction graph* and we devise simple graph protocols that can decide properties of some *input subgraph* provided by some preprocessing on the network. The 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 propose a simple model, the *Mediated Graph Protocol* (*MGP*) model, similar to the Population Protocol model of Angluin *et al.*, in which each network link 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 interesting properties of the MGP model among which is the ability to decide properties on stabilizing (initially changing for a finite number of steps) input graphs and we show that the MGP model has the ability to decide properties of disconnected input graphs. We show that the computational power within the connected components is fairly restricted. Finally, we give an exact characterization of the class **GMGP**, of graph languages decidable by the MGP model: 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.

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 achieve certain quality of service levels, statistical data have shown that each agent must

* This work has been partially supported by the ICT Programme of the European Union under contract number ICT-2008-215270 (FRONTS).

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 contains the correct answer. Each agent stores a counter $(0, 1, \ldots, k + 1$, where k is a constant) signifying the number of active incoming traffic flows. 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 some counter reaches the value $k+1$, then an alert state is propagated to the population and eventually, 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 not. Is there a protocol to answer such 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 populations 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

In [3], Angluin *et al.* 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 [3, 4], 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., [10]). The capability of the model to decide graph properties of restricted interaction graphs was explored in [2].

In an attempt to enhance the basic model, an interesting variation was proposed in [7], 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. For example, biological and artificial neural networks 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 [6], it was proven that, in complete graphs, the MPP model is computationally equivalent to a Nondeterministic Turing Machine (NTM) of $\mathcal{O}(n^2)$ space that computes symmetric predicates.

In [8], the *Graph Decision Mediated Population Protocols* (*GDMPPs*) were introduced, which are essentially MPPs without input, that may run on any graph from a specified family, trying to decide some property of that graph. GDMPPs were proven unable to compute any nontrivial property of disconnected input graphs. For introductory texts to the area of population protocols the interested reader is referred to [5, 12, 1].

3 Our Results - Roadmap

In Section 4, we give a formal definition of the proposed model, (MGP), provide an example protocol illustrating the computation of the model and present some important definitions that are used throughout this work. We then (Section 5) present some fundamental properties of the new model. In particular, we extend (Sec. 5.1) the MGP model to allow the input graph to oscillate for a finite number of steps. This extension then allows us (Sec. 5.2) to compose protocols. In Sec. 6, we present a protocol (Sec. 6.1) that decides whether the input graph is connected and we then extend this idea (Sec. 6.2) and show that the new model is able to compute properties of disconnected input graphs, something that neither the PP nor the GDMPP model were capable of. By studying the computations in each connected component, we provide a first indication that in unrestricted (not necessarily complete) connected graphs the computational power dramatically drops. In Sec. 7, 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. 8, we conclude and discuss some future research directions.

4 A Formal Model: Mediated Graph Protocols

A *Mediated Graph Protocol* (*MGP*) consists of a *finite set* of *agent states* Q, where $q_0, q_1 \in Q$ are the *initial agent states*, an *output function* $O : Q \to \{0, 1\}$ mapping agent states to binary outputs, a finite set of *edge states* S, where $s_0, s_1 \in S$ are the *initial edge states*, 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 $\delta_1(a, b, s) = a'$, $\delta_2(a, b, s) = b'$, $\delta_3(a, b, s) = s'$.

An MGP runs on an *interaction graph* $G = (V, E)$, where V is a population and E is an irreflexive binary relation on V. Throughout this work, we assume that this graph is *complete*, so that an MGP may run on any $K_n = (V, E)$, where $|V| = n$ and $E = V^2 \backslash \{(u, u) \mid u \in V\}$.

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 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 the *input graph* to the protocol.

A (*network*) *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. Note first that a network initialization function ι specifies the initial configuration. Let C and C' be configurations, and let u, υ be distinct agents. We say that C goes to C' via encounter $e = (u, \upsilon)$, denoted $C \overset{e}{\to} C'$, if $C'(u) = \delta_1(C(u), C(\upsilon), C(e))$,

$C'(\upsilon) = \delta_2(C(u), C(\upsilon), C(e))$, $C'(e) = \delta_3(C(u), C(\upsilon), C(e))$, and $C'(z) = C(z), \forall z \in (V - \{u, \upsilon\}) \cup (E - \{e\})$, that is, C' is the result of the interaction of the pair (u, υ) under configuration C and is the same as C except for the fact that the states of u, υ, and (u, υ) have been updated according to δ_1, δ_2, and δ_3, respectively. Note that each interaction (u, υ) is an *ordered pair* that is each agent has a distinct role in the interaction, u that of the *initiator* and υ that of the *responder*. We say that C can go to C' in one step, denoted $C \rightarrow C'$, 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 $C_i \rightarrow C_{i+1}$ for all i, $0 \leq i < t$, in which case we say that C' is *reachable* from C. The *transition graph* $T(\mathcal{A}, G)$ of an MGP $\mathcal{A}$ running on G is a directed graph whose nodes are all possible configurations and whose edges are all possible transitions on those nodes.

An *execution* is a finite or infinite sequence of configurations $C_0, C_1, C_2, \ldots$, where C_0 is an initial configuration and $C_i \rightarrow C_{i+1}$, 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 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 $C \rightarrow C'$, if C occurs infinitely often in the execution, then so does C' (cf. [9]). A *computation* is an infinite fair execution. The output of any agent u under configuration C is $O(C(u))$.

Due to the constant size descriptions of MGPs (Q and S are finite), the protocols we study are *uniform* (independent of the population size) and *anonymous* (agents can't store unique identifiers).

In this work, we are interested in determining properties of the subgraph that is specified by the network initialization function. To formalize this, let $\mathcal{H}$ be the *family* of all simple directed graphs with no isolated nodes. Note that $\mathcal{H}$ also includes disconnected graphs whose connected components have at least two nodes. A *graph language* is any $L \subseteq \mathcal{H}$.

Definition 1. *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. 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. We denote by **SEM** the class of *semilinear predicates*.

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 $S = \{s_0, s_1\}$. 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 $\delta(q_i, q_j, s_1) = (q_i, q_{j+1}, s_0)$; 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.

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.

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. 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.

5 Properties of MGPs

We present some useful properties of the model that will help us unfold its computational potential. Let $L^{-1} = \{H \mid \exists G \in L$ such that H is the inverse of $G\}$ [3] be the inverse of a language L.

[3] Let $G = (V, E)$ be a simple digraph and K the irreflexive subset of V^2. Then the *inverse* or *complement* of G is defined as $H = (V, K \backslash E)$.

Theorem 1 (Closure). **GMGP** *is closed under union, intersection, complement, and inversion.*

5.1 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 [2]), 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 [2], 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 (all the possible values, however, constantly belong to $\{s_0, s_1\}$ for the edges and to $\{q_0, q_1\}$ for the agents) between any two subsequent steps. The transition function is now of the form $\delta : ((Q\times\{q_0, q_1\})\times(Q\times\{q_0, q_1\})\times(S\times\{s_0, s_1\})) \rightarrow (Q\times Q\times S)$ and a configuration is a mapping $C : V \cup E \rightarrow (Q\times\{q_0, q_1\})\cup(S\times\{s_0, s_1\})$ taking into account the current initial states of the agents and edges.

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 [6, 11]. That protocol is also executed on complete interaction graphs. It constructs a correctly labeled spanning pseudo-path subgraph of the interaction graph and then exploits this construction to simulate a NTM on its input assignment. Informally, a pseudo-path graph is a straight line with arbitrary link directions. 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:

Theorem 2. *L is stably SIMGP-decidable iff it is MGP-decidable.*

Proof. The straight direction holds trivially due to our focus on languages of stabilized input graphs. For the inverse, let $\mathcal{A}$ be an MGP that stably decides L. We can construct an SIMGP $\mathcal{B}$ which consists of protocol $\mathcal{A}$ and the protocol of [6] (see above) running in parallel so that the population can be organized into a pseudo-path graph. This construction ends in a finite number of interactions with the reinitialization of $\mathcal{A}$'s execution and can be used to perform further reinitializations whenever the input graph changes. Since the input graph stabilizes, $\mathcal{A}$ will eventually be executed correctly given the stabilized graph as input. $\square$

5.2 Composition of MGPs

We will now present another interesting 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 3. *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 $\mathcal{A}$.*

Proof. From Theorem 2, 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. $\square$

6 Disconnected graphs

We now discuss the capability of our model to decide languages on disconnected graphs. Neither the PP model [2] nor the GDMPP model [8] 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. 6.1, we present a simple protocol that can decide whether the input graph is a connected graph and, in Sec. 6.2, 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.

6.1 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 1 *Connectivity Protocol (CP)*

```
1: $Q = \{q_0, q_1, t, t', l, l'\}$, $S = \{s_0, s_1\}$,
2: $O(q_0) = 0, O(q_1) = 0, O(t) = 1, O(t') = 0, O(l) = 1, O(l') = 0$,
3: $\delta$:
      a single leader is generated
      $(q_1, q_1, s_1) \rightarrow (l, t, s_1)$

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

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

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

      two non-adjacent single leaders meet in the same connected component of the input graph or in different connected components (in the case of disconnected input graph); they become non-unique leaders
      $(l, l, s_0) \rightarrow (l', l', s_0)$

      the non-unique leaders turn non-leaders into their followers
      $(l', t, s_1) \rightarrow (t', l', s_1)$, $(t, l', s_1) \rightarrow (l', t', s_1)$, $(l', q_1, s_1) \rightarrow (t', l', s_1)$, $(q_1, l', s_1) \rightarrow (l', t', s_1)$, $(l', q_0, s_0) \rightarrow (l', t', s_0)$, $(l', t, s_0) \rightarrow (l', t', s_0)$

      any two leaders meet in the same component; one single leader remains
      $(l', l, s_1) \rightarrow (l, t, s_1), (l, l', s_1) \rightarrow (l, t, s_1), (l', l', s_1) \rightarrow (l, t, s_1)$

      any two leaders meet in different components; both become non-unique
      $(l', l, s_0) \rightarrow (l', l', s_0), (l, l', s_0) \rightarrow (l', l', s_0)$

      a single leader restores all followers of multiple leaders to followers of single leaders
      $(l, t', s_0) \rightarrow (l, t, s_0), (l, t', s_1) \rightarrow (t, l, s_1), (t', l, s_1) \rightarrow (l, t, s_1)$
```

Theorem 4. *Protocol CP stably computes L_C for any input graph $G_\iota[K_n]$ on the complete interaction graph K_n.*

Proof. As can be observed by the description of Protocol 1 each connected component elects a leader which eventually becomes unique for the component. Therefore, if there are more than one connected components, their leaders will interact via an s_0 edge and all agents will be informed that there are at least 2 components in the input graph and will output 0. If no such interaction takes place, all agents output 1. □

6.2 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. We are not interested in components consisting of a single agent (input graphs with isolated nodes). 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 graph protocol* described in Section 5.1 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: $(q_1, q_1, s_1) \rightarrow (l, f, s_1)$ in which a leader is generated; $(l, q_1, s_1) \rightarrow (f, l, s_1)$ and $(q_1, l, s_1) \rightarrow (l, f, s_1)$ via which the leader turns non-leaders to followers; $(l, f, s_1) \rightarrow (f, l, s_1)$ and $(f, l, s_1) \rightarrow (l, f, s_1)$ which allow the leader state to move among the agents of a connected component; $(l, l, s_1) \rightarrow (l, f, s_1)$ which removes multiple leaders. Interactions between agents not defined by the previous transitions are ineffective (leave the states of both agents unchanged).

Lemma 1. *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 1 of Sec. 6.1 without the interactions between leaders of different connected components. The remaining leader moves constantly due to the transitions $(l, f, s_1) \rightarrow (f, l, s_1)$ and $(f, l, s_1) \rightarrow (l, f, s_1)$. □

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 [8] since it runs on any connected graph (instead of a complete one). *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 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 2. *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 [2]) 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 protocols: The composition is similar to the one described in Section 5.2. Firstly, RP constructs the spanning pseudo-path graph of the interaction graph reinitializing all other protocols during the process, then LE and BGP run in parallel to generate the representatives reinitializing REP, and finally, REP runs on the population of the representatives. We call the protocol resulting from the previous composition $GLADIS$ (*Graph LAnguages on DIsconnected graphS*). Since BGP and REP can be any GDMPP and PP, respectively, we denote the composition as $GLADIS(BGP, REP)$. The conclusion of this section is captured by the Theorem 5.

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 those that do not.

Theorem 5. *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. $GLADIS(BGP, REP)$ 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). $\square$

The applications of Theorem 5 are various, depending on the BGP and REP we use. Given a GDMPP-decidable graph language, e.g., $L = \{G \mid G \text{ contains at least one 2-cycle }\}$ (the decidability of L was an important question left open by [8]; in fact, it turns out that L is decidable even by PPs), 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 some 2-cycle. Whether the whole graph contains some 2-cycle can be simply decided by an OR population protocol on the representatives' population.

A fair question that arises is: what graph properties are stably decidable by the GDMPPs running on the components of the input graph? The exact computational power of the GDMPP model has not been characterized yet [8]. We approach the answer indirectly, by extending the notion of input graphs to vertex-labeled input graphs (whose labels are taken from a finite set X and are assigned by ι) and by considering computations on the multisets of the labels. These are, in fact, computations performed by any MPP (see [7]) on any connected graph given as input the multiset of labels. We call the corresponding computational class **MPU**. We provide the following exact characterization of **MPU**.

Theorem 6. **MPU** = **SEM**.

Proof. Let $p \in \mathbf{MPU}$ 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 (since external nodes have no effective interactions with each other) store the states of the edges on the external nodes. The latter, in turn, can be simulated by a PP (as we have proved recently) 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. $\square$

The consequences of the above theorem are twofold. First of all, the GDMPP model on unrestricted connected graphs seems to be computationally weak; this is a first step towards answering a major question left open by [8]. Secondly, this characterization, if compared to the one for the MGP model that is provided in the following section, shows that the MGPs seem to be significantly more powerful than the GDMPPs.

7 An Exact Characterization: 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 $L \in \mathbf{LGNSPACE}$ is stably decidable by the MGP model, or equivalently that $\mathbf{LGNSPACE} \subseteq \mathbf{GMGP}$. By showing that the inverse is also possible, we conclude that the inclusion holds with equality.

Now, consider the following 3-component initial configuration: According to the first component, called the *label*, there is a unique spanning correctly labeled pseudo-path subgraph $L = (V, A)$ of a complete interaction graph K_n. [4] 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 has initially the value $\sqcup^k$ for some predetermined constant k.

Lemma 2. *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. We use the protocol of [6, 11] that constructs a spanning pseudo-path graph of K_n, L which allows the orderly visit of K_n's edges and the full use of the distributed memory of the population. To store the adjacency matrix, we visit the edges in an orderly fashion and store each time the distances of the two ends of the visited edge. The distance of an agent is the length of the unique pseudo-path from the fixed leader endpoint of L to that agent. Thus, the distance of both ends of a visited edge can be stored in two $\mathcal{O}(n)$ counters which are the indexes of the entry corresponding to the edge on the adjacency matrix. If the visited edge belongs to the input graph, then a 1 is stored in the $\mathcal{O}(n^2)$ distributed memory and 0 otherwise. □

Now the exact characterization follows easily:

Theorem 7. GMGP = LGNSPACE.

8 Conclusions - Future Research Directions

Many interesting issues arise by the findings of our work. The ability of the new 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

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

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?

Acknowledgements. We would like to specifically thank Theofanis Raptis for his useful comments throughout the writing of this work.

References

1. Carme Àlvarez, Ioannis Chatzigiannakis, Amalia Duch, Joaquim Gabarró, Othon Michail, Serna Maria, and Paul G. Spirakis. Computational models for networks of tiny artifacts: A survey. *Computer Science Review*, 5(1), January 2011.
2. Dana Angluin, James Aspnes, Melody Chan, Michael J. Fischer, Hong Jiang, and René Peralta. Stably computable properties of network graphs. In *1st IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2005)*, volume 3560 of *LNCS*, pages 63–74. Springer-Verlag, June 2005.
3. Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, pages 235–253, mar 2006.
4. Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, November 2007.
5. James Aspnes and Eric Ruppert. An introduction to population protocols. *Bulletin of the EATCS*, 93:98–117, October 2007.
6. Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul 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 *LNCS*, pages 270–281. Springer-Verlag, August 23–27 2010.
7. Ioannis Chatzigiannakis, Othon Michail, and Paul 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.
8. Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. Stably decidable graph languages by mediated population protocols. In *Stabilization, Safety, and Security of Distributed Systems*, volume 6366 of *LNCS*, pages 252–266. September 2010.
9. Michael Fischer and Hong Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In *Principles of Distributed Systems*, volume 4305 of *LNCS*, pages 395–409. Springer-Verlang, 2006.
10. S. Ginsburg and E. H. Spanier. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.
11. Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Mediated population protocols. *Theor. Comput. Sci.*, 412:2434–2450, May 2011.
12. Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. *New Models for Population Protocols.* N. A. Lynch (Ed), Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2011.