Recent Advances in Population Protocols*

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

[1] Research Academic Computer Technology Institute (RACTI), +302610960200, Patras, Greece
[2] Computer Engineering and Informatics Department (CEID), University of Patras, 26500, Patras, Greece.
Email: {ichatz, michailo, spirakis}@cti.gr

Abstract. The *population protocol model* (PP) proposed by Angluin et al. [2] describes sensor networks consisting of passively mobile finite-state agents. The agents sense their environment and communicate in pairs to carry out some computation on the sensed values. The *mediated population protocol model* (MPP) [13] extended the PP model by communication links equipped with a constant size buffer. The MPP model was proved in [13] to be stronger than the PP model. However, its most important contribution is that it provides us with the ability to devise optimizing protocols, approximation protocols and protocols that decide properties of the communication graph on which they run. The latter case, suggests a simplified model, the GDM model, that was formally defined and studied in [11]. GDM is a special case of MPP that captures MPP's ability to decide properties of the communication graph. Here we survey recent advances in the area initiated by the proposal of the PP model and at the same time we provide new protocols, novel ideas and results.

1 Introduction

Most recent advances in microprocessor, wireless communication and sensor/actuator-technologies envision a whole new era of computing, popularly referred to as pervasive computing. Autonomous, ad-hoc networked, wirelessly communicating and *spontaneously interacting* computing devices of *small size* appearing in *great number*, and embedded into environments, appliances and objects of everyday use will deliver services adapted to the person, the time, the place, or the context of their use. The nature and appearance of devices will change to be hidden in the fabric of everyday life, invisibly networked, and will be augmenting everyday environments to form a pervasive computing landscape, in which the physical world becomes merged with a "digital world".

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

In a seminal work, Angluin et al. [2] (also [3]) considered systems consisting of very small resource limited sensor nodes that are passively mobile. Such nodes, also called agents, have no control over their own movement and interact in pairs, via a local low-power wireless communication mechanism, when they are sufficiently close to each other. The agents form a (usually huge) population that together with the agent's permissible interactions form a communication graph $G = (V, E)$, where V is a population of $|V| = n$ agents and E is the set of permissible (directed) interactions of cardinality denoted by m. In their model, finite-state, and complex behavior of the system as a whole emerges from simple rules governing pairwise interaction of the agents. The most important innovations of the model are inarguably the *constant memory* constraint imposed to the agents and the *nondeterminism* inherent to the interaction pattern. These assumptions provide us with a concrete and realistic model for future systems. Their model is called Population Protocol model and is discussed in Section 2.

The initial goal of the model was to study the *computational limitations* of cooperative systems consisting of many limited devices (agents), imposed to passive (but *fair*) communication by some *scheduler*. Much work showed that there exists an exact characterization of the computable predicates: they are precisely the *semilinear predicates* or equivalently the predicates definable by first-order logical formulas in *Presburger arithmetic* [2, 3, 5–7]. More recent work has concentrated on performance, supported by a random scheduling assumption. [12] proposed a collection of *fair* schedulers and examined the performance of various protocols. [9, 10] 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. In [9] it was also proven that there is a strong relation between classical finite population protocols and models given by ordinary differential equations.

There exist a few extensions of the basic model in the relevant literature to more accurately reflect the requirements of practical systems. In [1] they studied what properties of restricted communication graphs are stably computable, gave protocols for some of them, and proposed the model extension with *stabilizing inputs*. The results of [5] show that again the semilinear predicates are all that can be computed by this model. Finally, some works incorporated agent failures [14] and gave to some agents slightly increased computational power [8] (heterogeneous systems). For an excellent introduction to most of the preceding subjects see [7].

In [13] the Population Protocol model was extended in a natural way. Essentially the model was augmented to include a *Mediator*, i.e., a global storage capable of storing very limited information for each communication arc (the state of the arc). When pairs of agents interact, they can read and update the state of the link (arc). Interestingly, although anonymity and uniformity (for the definition of those notions the reader is referred to Section 3) are preserved in the extended model, *the presence of a mediator provides us with significantly more computational power* and gives birth to a new collection of interesting problems in the area of tiny networked and possibly moving artefacts; we can now build systems with the ability of computing subgraphs and solve optimization problems concerning the communication graph. In [13] it was shown that the new model is capable of computing non-semilinear predicates and that any stably computable predicate belongs to $NSPACE(m)$, where m denotes the number of edges of the interaction graph. The extended model is called Mediated Population Protocol model and we present it in Section 3.

One of the most interesting and applicable capabilities of the Mediated Population Protocol model is its ability to decide graph properties. To understand properties of the communication graph is an important step in almost any distributed system. In particular, in [11] the authors temporarily disregarded the input notion of the population and assumed that all agents simply start from a unique initial state (and the same holds for the edges). The obtained model is called GDM. The authors focused on protocols of the GDM model that, when executed fairly on any communication graph G, after a finite number of steps stabilize to a configuration where all agents give 1 as output if G belongs to a graph language L, and 0 otherwise. This is motivated by the idea of having protocols that eventually accept all communication graphs (on which they run) that satisfy a specific property, and eventually reject all remaining communication graphs. The motivation for the proposal of a simplified version of the Mediated Population Protocol model was that it enables us to study what graph properties are stably computable by the mediated model without the need to keep in mind its remaining parameters (which, as a matter of fact, are a lot). The GDM model is discussed in Section 4. Finally, in Section 5 we discuss some future research directions.

2 The Population Protocol model

2.1 Formal Definition

Definition 1. *A* population protocol *(PP) is a 6-tuple* (X, Y, Q, I, O, δ)*, where* X*,* Y*, and* Q *are all finite sets and*

1. X *is the* input alphabet*,*
2. Y *is the* output alphabet*,*
3. Q *is the set of* states*,*
4. $I : X \to Q$ *is the* input function*,*
5. $O : Q \to Y$ *is the* output function*, and*
6. $\delta : Q \times Q \to Q \times Q$ *is the* transition function*.*

If $\delta(a, b) = (a', b')$*, we call* $(a, b) \to (a', b')$ *a* transition *and we define* $\delta_1(a, b) = a'$ *and* $\delta_2(a, b) = b'$*.*

A population protocol $\mathcal{A} = (X, Y, Q, I, O, \delta)$ runs on a communication graph $G = (V, E)$. Initially, all agents (i.e. the elements of V) receive a global start signal, sense their environment and each one receives an input symbol from X. All agents are initially in a special empty state $\sqcup \notin Q$. When an agent receives an input symbol σ, applies the input function to it and goes to its initial state $I(\sigma) \in Q$. An adversary scheduler selects in each step a directed pair of agents $(u, v) \in E$, where $u, v \in V$ and $u \neq v$, to interact. The interaction happens only if both agents are not in the empty state (they must both have been initialized). Assume that the scheduler selects the pair (u, v), that the current states of u and v are $a, b \in Q$, respectively, and that $\delta(a, b) = (a', b')$. Agent u plays the role of the *initiator* in the interaction (u, v) and v that of the *responder*. During their interaction u and v apply the transition function to their directed pair of states, and, as a result, u goes to a' and v to b' (both update their states according to δ, and specifically, the initiator applies δ_1 while the responder δ_2).

A *configuration* is a snapshot of the population states. Formally, a configuration is a mapping $C : V \to Q$ specifying the state of each agent in the population. C_0 is the initial configuration (for simplicity we assume that all agents apply the input function at the same time, which is one step before C_0, so in C_0 all empty states have been already replaced, and that' s the reason why we have chosen not to include $\sqcup$ in the model definition) and, for all $u \in V$, $C_0(u) = I(x(u))$, where $x(u)$ is the input symbol sensed by agent u. Let C and C' be 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

$$\begin{aligned} C'(u) &= \delta_1(C(u), C(v)), \\ C'(v) &= \delta_2(C(u), C(v)), \text{ and} \\ C'(w) &= C(w) \text{ for all } w \in V - \{u, v\}, \end{aligned}$$

that is, C' is the result of the interaction of the pair (u, v) under configuration C and is the same as C except for the fact that the states of u, v have been updated according to δ_1 and δ_2, respectively. 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 configurations C_0, C_1, $C_2, \ldots$, where C_0 is an initial configuration and $C_i \to C_{i+1}$, for all $i \geq 0$. We have both finite and infinite kinds of executions since the scheduler may stop in a finite number of steps or continue selecting pairs for ever. Moreover, note that, according to the preceding definitions, a scheduler may partition the agents into non-communicating clusters. If that's the case, then it is easy to see that no meaningful computation is possible. To avoid this unpleasant scenario, a strong global *fairness condition* is imposed on the scheduler to ensure the protocol makes progress. 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 C' also occurs infinitely often in the execution. A scheduler is fair if it always leads to fair executions. A *computation* is an infinite fair execution.

The following are two critical properties of population protocols:

1. *Uniformity*: Population protocols are uniform. This means that any protocol's description is independent of the population size. Since we assume that the agents have finite storage capacity, and independent of the population size, uniformity enables us to store the protocol code in each agent of the population.
2. *Anonymity*: Population protocols are anonymous. The set of states is finite and does not depend on the size of the population. This impies that there is no room in the state of an agent to store a unique identifier, and, thus, all agents are treated in the same way by the transition function.

2.2 Stable Computation

Assume a fair scheduler that keeps working forever and a protocol $\mathcal{A}$ that runs on a communication graph $G = (V, E)$. Initially, each agent receives an input symbol from X. An input assignment $x : V \to X$ is a mapping specifying the input symbol of each agent in the population. Let $\mathcal{X} = X^V$ be the set of all possible input assignments, given the population V and the input alphabet X of $\mathcal{A}$. Population protocols, when controlled by infinitely working schedulers, do not halt. Instead of halting we require any computation of a protocol to *stabilize*. An output assignment $y : V \to Y$ is a mapping specifying the output symbol of each agent in the population. Any configuration $C \in \mathcal{C} = Q^V$ is associated with an output assignment $y_C = O \circ C$. A configuration C is said to be *output-stable* if for any configuration C' such that $C \xrightarrow{*} C'$ (any configuration reachable from C) $y_{C'} = y_C$. In words, a configuration C is output-stable if all agents maintain the output symbol that have under C in all subsequent steps, no matter how the scheduler proceeds thereafter. A computation $C_0, C_1, C_2, \ldots$ is *stable* if it contains an output-stable configuration C_i, where i is finite.

Definition 2. *A population protocol $\mathcal{A}$ running on a communication graph $G = (V, E)$* stably computes a predicate $p : \mathcal{X} \to \{0, 1\}$*, if, for any $x \in \mathcal{X}$, every computation of $\mathcal{A}$ on G beginning in $C_0 = I \circ x$ reaches in a finite number of steps an output-stable configuration C_{stable} such that $y_{C_{stable}}(u) = p(x)$ for all $u \in V$. A predicate is* stably computable *if some population protocol stably computes it.*

Assume that a computation of $\mathcal{A}$ on G begins in the configuration corresponding to an input assignment x. Assume, also, that $p(x) = 1$. If $\mathcal{A}$ stably computes p, then we know that after a finite number of steps (if, of course, the scheduler is fair) all agents will give 1 as output, and will continue doing so for ever. This means, that if we wait for a sufficient, but finite, number of steps we can obtain the correct answer of p with input x by querying any agent in the population.

Definition 3. *The* basic population protocol model *(or* standard*) assumes that the communication graph G is always directed and complete.*

Semilinear predicates are predicates whose support is a semilinear set. A *semilinear set* is the finite union of linear sets. A set of vectors in $\mathbb{N}^k$ is *linear* if it is of the form

$$\{\boldsymbol{b} + l_1\boldsymbol{a_1} + l_2\boldsymbol{a_2} + \cdots + l_m\boldsymbol{a_m} \mid l_i \in \mathbb{N}\},$$

where $\boldsymbol{b}$ is a base vector, $\boldsymbol{a_i}$ are basis vectors, and l_i are non-negative integer coefficients. Moreover, semilinear predicates are precisely those predicates that can be defined by first-order logical formulas in *Presburger arithmetic*, as was proven by Ginsburg and Spanier in [15].

In [2] and [3] it was proven that any semilinear predicate is stably computable by the basic population protocol model and in [5] that any stably computable predicate, by the same model, is semilinear, thus together providing an exact characterization of the class of stably computable predicates:

Theorem 1. *A predicate is stably computable by the basic population protocol model iff it is semilinear.*

An immediate observation is that predicates like "the number of c's is the product of the number of a's and the number of b's (in the input assignment)" and "the number of 1's is a power of 2" are not stably computable by the basic model.

A *graph family*, or *graph universe*, is any set of communication graphs. Let $\mathcal{G}$ be a graph family. For any $G \in \mathcal{G}$ and given that X is the input alphabet of some protocol $\mathcal{A}$, there exists a set of all input assignments *appropriate* for G, denoted $\mathcal{X}_G = X^{V(G)}$. Let now $\mathcal{X}_\mathcal{G} = \bigcup_{G \in \mathcal{G}} (\mathcal{X}_G \times \{G\})$ or, equivalently, $\mathcal{X}_\mathcal{G} = \{(x, G) \mid G \in \mathcal{G} \text{ and } x \text{ is an input assignment appropriate for } G\}$. Then we have the following definition:

Definition 4. *A population protocol* $\mathcal{A}$ stably computes a predicate $p : \mathcal{X}_\mathcal{G} \to \{0, 1\}$ *in a family of communication graphs* $\mathcal{G}$*, if, for any* $G \in \mathcal{G}$ *and any* $x \in \mathcal{X}_G$*, every computation of* $\mathcal{A}$ *on* G *beginning in* $C_0 = I \circ x$ *reaches in a finite number of steps an output-stable configuration* C_{stable} *such that* $y_{C_{stable}}(u) = p(x, G)$ *for all* $u \in V(G)$*.*

Moreover, if p is a mapping from $\mathcal{G}$ to $\{0, 1\}$, that is, a *graph property*, then we say that $\mathcal{A}$ stably computes property p.

Note that we can also consider undirected communication graphs. In the case of an undirected graph we only require that E is symmetric, but we keep the initiator-responder assumption. The latter is important to ensure deterministic transitions, since otherwise we would not be able to now which agent applies/gets the result of δ_1 and which that of δ_2.

2.3 An Example

Let us illustrate what we have seen so far by an example.

Problem 1. (*Undirected Star*) Given a communication graph $G = (V, E)$ from the unrestricted family of undirected graphs (any possible connected and undirected simple graph), find whether the topology is an undirected star.

We devise a protocol, named $UndirectedStar$, that stably computes property Undirected Star, that is, it eventually decides whether the underlying communication graph $G = (V, E)$ taken from the unrestricted family of graphs is an undirected star.

UndirectedStar

- $X = \{0, 1\}$, $Y = \{0, 1\}$,
- $Q = \{(i, j) \mid i \in \{0, 1, 2\} \text{ and } j \in \{0, 1, 2\}^2\} \cup \{z\}$,
- $I(x) = (0, (0, 0))$, for all $x \in X$,
- $O(z) = 0$ and $O(q) = 1$, for all $q \in Q - \{z\}$,
- δ:

$$
\begin{aligned}
((0,(0,0)),(0,(0,0))) &\to ((1,(1,1)),(2,(0,0)))\\
((1,(i,j)),(0,(0,0))) &\to ((1,(0,0)),(2,(i,j+1))),\\
&\quad \text{if } i \in \{0,1\} \text{ and } j \in \{0,1\} \text{ or } i = 2 \text{ and } j = 0\\
&\to ((1,(0,0)),(2,(i,j))), \text{ if } i = 1 \text{ and } j = 2\\
&\to (z,z), \text{ if } i = 2 \text{ and } j = 1\\
((2,(l,k)),(0,(0,0))) &\to ((2,(0,0)),(1,(l+1,k))),\\
&\quad \text{if } k \in \{0,1\} \text{ and } l \in \{0,1\} \text{ or } k = 2 \text{ and } l = 0\\
&\to ((2,(0,0)),(1,(l,k))), \text{ if } k = 1 \text{ and } l = 2\\
&\to (z,z), \text{ if } k = 2 \text{ and } l = 1\\
((1,(i,j)),(2,(l,k))) &\to ((1,(2,j+k)),(2,(0,0))), \text{ if } j+k<2 \text{ and } i+l \geq 2\\
&\to ((1,(i+l,2)),(2,(0,0))), \text{ if } i+l<2 \text{ and } j+k \geq 2\\
&\to ((1,(i+l,j+k)),(2,(0,0))), \text{ if } i+l<2 \text{ and } j+k<2\\
&\to (z,z), \text{ if } i+l \geq 2 \text{ and } j+k \geq 2\\
((1,(i,j)),(2,(0,0))) &\to ((1,(0,0)),(2,(i,j)))\\
((1,(0,0)),(2,(l,k))) &\to ((1,(l,k)),(2,(0,0)))\\
((1,(i,j)),(1,(l,k))) &\to (z,z)\\
((2,(i,j)),(2,(l,k))) &\to (z,z)\\
(z,x) &\to (z,z)
\end{aligned}
$$

Note that in the transition $\delta((1,(i,j)),(2,(l,k)))$ we assume that $(i,j) \neq (0,0)$ and $(l,k) \neq (0,0)$.

Definition 5. *An* undirected star *of order n ("n-star") is a tree on n vertices with one vertex having degree $n-1$ and $n-1$ vertices having degree* 1.

Lemma 1. *A connected undirected graph $G=(V,E)$, with $|V|=n\geq 3$, is an undirected star if and only if there is at most one $u\in V$ where $d(u)\geq 2$ (i.e. at most one vertex of degree at least 2).*

Proof. For the only if part, Definition 5 states that an undirected star has only one vertex of degree at least 2. For the other direction, first we note that since G is connected it must have at least $n-1$ edges. Any cycle should contain at least two vertices of degree 2, so G is acyclic. Since G is acyclic and connected it is a tree and therefore it has exactly $n-1$ edges. The latter, together with the fact that each $v\in V$ has $d(v)\geq 1$, but at most one $u\in V$ has $d(u)\geq 2$ implies that $d(u)=n-1$ and $d(v)=1$, for all $v\in V-\{u\}$, and, according to Definition 5, this completes the proof. □

Corollary 1. *A connected undirected graph $G=(V,E)$, with $|V|=n\geq 3$, is not an undirected star if and only if there are at least two vertices $u,v\in V$ where $d(u)\geq 2$ and $d(v)\geq 2$ (i.e. at least two vertices of degree at least 2).*

So, generally speaking, an algorithm that decides if a connected graph is an undirected star could only check if there is at most one vertex of degree at least 2 (if $n\geq 3$).

Remark 1. Any simple connected graph with only two vertices is an undirected star.

This remark fills the gap of the assumption in Lemma 1 that $n\geq 3$. Note that $n=1$ is meaningless, since in a graph with a unique vertex no computation can take place (there is not even a single pair of agents to interact).

We can think of states $(i,j)\in Q$ as consisting of two components i and j. We will call i the *basic component* and j the *counter component*. If the interacting pair consists of two agents in the initial state, the protocol assigns basic component 1 to one agent and basic component 2 to the other. Moreover, an agent in basic component 1 gives an agent that is in the initial state basic component 2 and an agent in basic component 2 gives an agent that is in the initial state basic component 1, while the pairs (1,2) and (2,1) do nothing w.r.t. the basic components. Clearly, if

the topology is a star and if w.l.o.g. the central vertex (vertex of degree $n-1$) gets basic component 2, then all peripheral vertices will eventually get basic component 1 and the protocol will output that the topology is a star.

If the topology is not a star, then the protocol must detect that at least two vertices are of degree at least 2. Note, also, that the agents never change their basic component except for the case when they get the reject state z.

Lemma 2. *If G is not a star, then protocol UndirectedStar will eventually create one of the following two situations:*

- *Two neighboring agents with the same basic component, or*
- *at least two agents in basic component 1 and at least two agents in basic component 2.*

Proof. Assume that it won't. Then any neighboring agents will have different basic components and at most one of the basic components 1 and 2 will appear in at least two different agents. So w.l.o.g. $n-1$ agents will be in component 1 and only one in component 2, since both components always exist in any computation (except before the first interaction and sometime after rejection). But since G is connected it must have at least $n-1$ edges. In fact, if it has more than $n-1$ edges it will contain a cycle with at least two vertices (agents) in basic component 1 which will violate the fact that any neighboring vertices will have different basic components and thus G must have exactly $n-1$ edges. But the latter implies that the $n-1$ vertices in basic component 1 are directly connected to the unique vertex in basic component 2, which in turn implies that G must be an undirected star, a fact that contradicts the fundamental assumption of the lemma. We could also have proven the statement by contradicting the fact that G not being an undirected star must have at least two vertices $u, v \in V$, where $d(u) \geq 2$ and $d(v) \geq 2$. □

In fact, the protocol always rejects if it finds two neighboring agents in the same basic component and the same does if it finds out that there are at the same time at least two agents in basic component 1 and at least two agents in basic component 2 in the population. The latter is done by the counter component of the states. Thus, according to the preceding lemma, the protocol will eventually reject if G is not a star, because it will provably fall in a situation that leads it to rejection.

Theorem 2. *UndirectedStar stably computes property Undirected Star.*

Proof. The correctness of the statement should be clear after the above discussion. The protocol always reaches an output-stable configuration and at that point any agent outputs the correct answer for the property Undirected Star. □

3 The Mediated Population Protocol model

In [13] the authors considered the following question: "Is there a way to extend the population protocol model and obtain a stronger model, without violating the uniformity and anonymity properties"? As we shall, in this section, see, the answer to this question is "Yes". Although the idea is simple, it provides us with a model with significantly more computational power and extra capabilities in comparison to the population protocol model. The main modification is to allow the edges of the communication graph to store states from a finite set, whose cardinality is independent of the population size. Two interacting agents read the corresponding edge's state and update it, according to a global transition function, by also taking into account their own states.

3.1 Formal Definition

Definition 6. *A* mediated population protocol *(MPP) is a 12-tuple* $(X, Y, Q, I, O, S, \iota, \omega, r, K, c, \delta)$*, where* X*,* Y*,* Q*,* S*, and* K *are all finite sets and*

1. X *is the* input alphabet*,*
2. Y *is the* output alphabet*,*
3. Q *is the set of* agent states*,*
4. $I : X \to Q$ *is the* agent input function*,*
5. $O : Q \to Y$ *is the* agent output function*,*
6. S *is the set of* edge states*,*
7. $\iota : X \to S$ *is the* edge input function*,*
8. $\omega : S \to Y$ *is the* edge output function*,*
9. r *is the* output instruction *(informing the output-viewer how to interpret the output of the protocol),*
10. K *is the totally ordered* cost set*,*
11. $c : E \to K$ *is the* cost function
12. $\delta : Q \times Q \times K \times S \to Q \times Q \times K \times S$ *is the* transition function*.*

We assume that the cost remains the same after applying δ *and so we omit specifying an output cost. If* $\delta(q_i, q_j, x, s) = (q_i', q_j', s')$ *(which, according*

to our assumption, is equivalent to $\delta(q_i, q_j, x, s) = (q_i', q_j', x, s'))$*, we call* $(q_i, q_j, x, s) \to (q_i', q_j', s')$ *a* transition*, and we define* $\delta_1(q_i, q_j, x, s) = q_i'$*,* $\delta_2(q_i, q_j, x, s) = q_j'$ *and* $\delta_3(q_i, q_j, x, s) = s'$*. We call* δ_1 *the* initiator's acquisition*,* δ_2 *the* responder's acquisition*, and* δ_3 *the* edge acquisition *(after the corresponding interaction).*

In most cases we assume that $K \subset \mathbb{Z}^+$ and that $c_{max} = \max_{w \in K} \{w\} = \mathcal{O}(1)$. Generally, if $c_{max} = \max_{w \in K} \{|w|\} = \mathcal{O}(1)$ then any agent is capable of storing at most k cumulative costs (at most the value kc_{max}), for some $k = \mathcal{O}(1)$, and we say that the cost function is *useful* (note that a cost range that depends on the population size could make the agents incapable for even a single cost storage and any kind of optimization would be impossible).

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 communication 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

$$\begin{aligned} C'(u) &= \delta_1(C(u), C(v), x, C(e)) \\ C'(v) &= \delta_2(C(u), C(v), x, C(e)) \\ C'(e) &= \delta_3(C(u), C(v), x, C(e)) \\ C'(z) &= C(z), \text{ for all } z \in (V - \{u, v\}) \cup (E - e). \end{aligned}$$

The definitions of *execution* and *computation* are the same as in the population protocol model but concern network configurations. Note that the mediated population protocol model *preserves both uniformity and anonymity* properties. As a result, any MPP's *code* is of *constant size* and, thus, can be stored in each agent (device) of the population.

A configuration C is called *r-stable* if one of the following conditions holds:

- If the problem concerns a subgraph to be found, then C should fix a subgraph that will not change in any C' reachable from C.
- If the problem concerns a function to be computed by the agents, then an r-stable configuration drops down to an output-stable configuration.

We say that a protocol $\mathcal{A}$ *stably solves* a problem Π, if for every instance I of Π and every computation of $\mathcal{A}$ on I, the network reaches an r-stable configuration C that gives the correct solution for I if interpreted

according to the output instruction r. If instead of a problem Π we have a function f to be computed, we say that $\mathcal{A}$ *stably computes* f.

In the special case where Π is an optimization problem, a protocol that stably solves Π is called an *optimizing population protocol* for problem Π.

3.2 An Optimizing Population Protocol

We now give an optimizing population protocol, named *SRLpath*, for the problem of finding the shortest path connecting the root of a directed arborescence to one of its leaves. Formally the problem is the following.

Problem 2. (*Shortest Root-Leaf Path*) Given that the communication graph $G = (V, E)$ is a directed arborescence and a useful cost function $c : E \to K$ on the set of edges, design a protocol that finds the minimum cost path of the (nonempty) set $P = \{p \mid p \text{ is a path from the root to a leaf and } c(p) = O(1)\}$, where $c(p)$ is simply another way to write $\sum_{e \in p} c(e)$.

We assume that the greatest value that any agent is capable of storing is kc_{max}, where both k and $c_{max} = max_{e \in E} c(e)$, are fixed and independent of the size of the population $|V| = n$, given a nonnegative integer-valued cost function $c : E \to K$ (i.e. $K \subset \mathbb{Z}^+$).

If there is at least one path p where $c(p) = \sum_{e \in P} c(e) < kc_{max}$ ($c(p)$ denotes the path length or the total cost of the path), then *SRLpath* will eventually return the shortest path connecting the root to one of the leaves, otherwise it will just return one of the paths with cost at least kc_{max} (one of all root-leaf paths), without guaranteeing that it will be the shortest one, but the output of the root will be 0 indicating that there was no such path.

SRLpath

- $X = \{0, 1\}$,
- $Y = \{0, 1\} \cup Q$,
- $Q = \{q_0, q_s\} \cup \{(i, j) \mid i \in \{q_1, q_2, q_3, q_s\}$ and $j \in \{0, 1, 2, \ldots, kc_{max}\}$,
- $I(x) = q_0$, for all $x \in X$,
- $O(q_1, kc_{max}) = 0$, $O(q) = q$, for all $q \in Q - \{(q_1, kc_{max})\}$,
- $S = \{0, 1\}$,
- $\iota(x) = 0$, for all $x \in X$,
- $\omega(s) = s$, for all $s \in S$,
- r: "*If the root outputs 0, fail, else start from the root and follow every edge with output 1, until you reach a leaf*",

– δ:

$$
\begin{aligned}
(q_0, q_0, c, 0) &\to ((q_1, c), q_0, 1)\\
(q_0, (q_1, c_1), c, 0) &\to ((q_1, kc_{max}), (q_1, c_1), 1), \text{ if } c_1 + c > kc_{max}\\
&\to ((q_1, c_1 + c), (q_1, c_1), 1), \text{ otherwise}\\
((q_1, c_1), (q_1, c_2), c, 1) &\to ((q_1, kc_{max}), (q_1, c_2), 1), \text{ if } c_2 + c > kc_{max}\\
&\to ((q_1, c_2 + c), (q_1, c_2), 1), \text{ otherwise}\\
((q_1, c_1), (q_1, c_2), c, 0) &\to ((q_2, c_2 + c), (q_s, c_2), 1), \text{ if } c_2 + c < c_1\\
((q_2, c_1), (q_i, c_2), c, 1) &\to ((q_3, c_1), (q_i, c_2), 0), \text{ for } i \in \{1, 2, 3\}\\
((q_3, c_1), (q_s, c_2), c, 1) &\to ((q_1, c_1), (q_1, c_2), 1)\\
((q_1, c_1), q_0, c, 0) &\to ((q_2, c), q_s, 1), \text{ if } c < c_1\\
((q_2, c_1), q_0, c, 1) &\to ((q_3, c_1), q_0, 0)\\
((q_3, c_1), q_s, c, 1) &\to ((q_1, c_1), q_0, 1)
\end{aligned}
$$

Theorem 3. *If there is at least one root-leaf path p, where $c(p) < kc_{max}$, then SRLpath is an optimizing population protocol for Problem 2. Otherwise, the root outputs 0, indicating that there is no such path.*

Proof. The proof is by induction on the number of nodes of the directed arborescence T. Let $\mathcal{T}_i = \{T \mid T \text{ is a directed arborescence of } i \text{ nodes}\}$, i.e. the family of all directed arborescences of i nodes. There is only one directed arborescence in $\mathcal{T}_1$, and only one in $\mathcal{T}_2$, the one that consists of two nodes connected by a directed edge. Obviously, *SRLpath* always finds the shortest root-leaf path for every directed arborescence of at most 2 nodes (finds it trivially). Assume that for every directed arborescence of at most n nodes, *SRLpath* always finds the shortest root-leaf path. Let T_{n+1} be any directed arborescence of $(n + 1)$ nodes. By ignoring the root of T_{n+1} and the corresponding edges we get at least one directed arborescence of at most n nodes. On any such subtree we know by the inductive hypothesis that *SRLpath* always finds the shortest root-leaf path. Moreover, it is easy to see that the protocol always keeps in the root of the tree the cost of the selected root-leaf path. Let, u be the removed root, and v_j, where $j = \{1, 2, \ldots, t\}$, its t children. Each child v_j has eventually marked the shortest root-leaf path in the subtree in which v_j is the root and eventually contains the cost of this path in its state, $c(p_j)$. So, eventually u will select the child v_j, for which $\min_{v_j}\{c(p_j) + c(u, v_j)\}$ holds, which will be the shortest root-leaf path of T_{n+1}. Note that if

$\min_{v_j}\{c(p_j) + c(u, v_j)\} < kc_{max}$, then at least one such path can be selected by the root (which will never give 0 as output). On the other hand, if there is no such path, then $\min_{v_j}\{c(p_j) + c(u, v_j)\} \geq kc_{max}$ and u can only store kc_{max} which combined with q_1 gives always output 0, indicating that no such path exists. □

3.3 Approximation Protocols

Consider now the following problem:

Problem 3. (*Maximal matching*) Given an undirected communication graph $G = (V, E)$, find a maximal matching, i.e., a set $E' \subseteq E$ such that no two members of E' share a common end point in V and, moreover, there is no $e \in E - E'$ such that e shares no common end point with every member of E'.

A simple protocol to solve this problem is the following: Initially all agents are in state q_0 and all edges in state s_0. When two agents in q_0 interact via an edge in s_0, they both go to q_1 to indicate that they are endpoints of an edge that belongs to the matching formed so far, and the edge goes to s_1 to indicate that it has been put in the matching. All the other transitions have no effect. It is easy to see that eventually the edges in s_1 will form a maximal matching. Moreover, $\omega(s_1) = O(q_1) = 1$ and $\omega(s_0) = O(q_0) = 0$. An appropriate instruction r could be: "*Get each $e \in E$ for which $\omega(s_e) = 1$ (where s_e is the state of e)*", which simply informs the user how to interpret the output of the protocol to get the correct answer (i.e. which edges form the matching).

Definition 7. *Let Π be a minimization problem, and let δ be a positive real number, $\delta \geq 1$. A protocol $\mathcal{A}$ is said to be a* δ factor approximation protocol *for Π if for each instance I of Π, and every computation of $\mathcal{A}$ on I, the network reaches an r-stable configuration C, that, if interpreted according to the output instruction r of $\mathcal{A}$, gives a feasible solution s for I such that*

$$f_{\Pi}(I, s) \leq \delta \cdot OPT(I),$$

where $f_{\Pi}(I, s)$ denotes the objective function value of solution s of instance I, and $\mathrm{OPT}(I)$ denotes the objective function value of an optimal solution of instance I.

Now, consider the well-known minimum vertex cover problem defined as follows:

Problem 4. (*Minimum vertex cover*) Given an undirected communication graph $G = (V, E)$, find a minimum cardinality *vertex cover*, i.e., a set $V' \subseteq V$ such that every edge has at least one end point incident at V'.

Let $VertexCover$ be a MPP that agrees on everything to the one already described for Maximal matching, except for the output instruction r, which is now r: "*Get each* $v \in V$ *for which* $O(q_v) = 1$ *(where* q_v *is the state of* v*)*". Intuitively, we now collect all agents incident to an edge in the maximal mathcing M (for all $e \in M$ we collect the end points of e).

Theorem 4. *$VertexCover$ is a 2 approximation protocol for the minimum vertex cover problem.*

Proof. According to the previous discussion, the edges collected form a maximal matching on G. Moreover, the set C formed by the end points of the edges of M is a minimum vertex cover where $|C| \leq 2 \cdot OPT$, according to the analysis in the introductory chapter of [16]. Thus, $VertexCover$ is a 2 approximation protocol for the minimum vertex cover problem. □

3.4 Computational Power

It is easy to see that the population protocol model is a special case of the mediated population protocol model. In [13] it was proven that there exists a MPP protocol that stably computes the non-semilinear predicate $N_c = N_a \cdot N_b$. In words, it eventually decides whether the number of c's in the input assignment is equal to the product of the number of a's and the number of b's. To do so, the authors stated a composition theorem, that simplifies the proof of existence of MPP protocols. Here we also provide a complete proof of that theorem.

Definition 8. *A MPP $\mathcal{A}$ has* stabilizing states *if in any computation of $\mathcal{A}$, after a finite number of interactions, the states of all agents stop changing.*

Definition 9. *We say that a predicate is* strongly stably computable *by the MPP model, if it is stably computable with the predicate output convention, that is, all agents eventually agree on the correct output value.*

Theorem 5. *Any mediated population protocol $\mathcal{A}$, that stably computes a predicate p with stabilizing states in some family of directed and connected communication graphs $\mathcal{G}$, containing an instruction r that defines a semilinear predicate t on multisets of $\mathcal{A}$'s agent states, can be composed with a provably existing mediated population protocol $\mathcal{B}$, that strongly stably*

computes t with stabilizing inputs in $\mathcal{G}$, to give a new mediated population protocol $\mathcal{C}$ satisfying the following properties:

- *$\mathcal{C}$ is formed by the composition of $\mathcal{A}$ and $\mathcal{B}$,*
- *its input is $\mathcal{A}$'s input,*
- *its output is $\mathcal{B}$'s output, and*
- *$\mathcal{C}$ strongly stably computes p (i.e. all agents agree on the correct output) in $\mathcal{G}$.*

Proof. Protocol $\mathcal{A}$ has stabilizing states and an instruction r that defines a semilinear predicate t on multisets of $\mathcal{A}$'s states. Let $X_{\mathcal{A}}$ be the input alphabet of $\mathcal{A}$, $Q_{\mathcal{A}}$ the set of $\mathcal{A}$'s states, $\delta_{\mathcal{A}}$ the transition function of $\mathcal{A}$, and similarly for any other component of $\mathcal{A}$. We will use the indexes $\mathcal{B}$ and $\mathcal{C}$, for the corresponding components of the other two protocols.

Since predicate t is semilinear, according to a result in [1], there is a population protocol $\mathcal{B}'$ that stably computes t with stabilizing inputs in the *unrestricted family of graphs* (denoted by $\mathcal{G}^d_{Unr}$, and consisting of all possible directed and connected communication graphs). Note that $\mathcal{G}$ is a subset of $\mathcal{G}^d_{Unr}$ ($\mathcal{G} \subseteq \mathcal{G}^d_{Unr}$), so any predicate stably computable (both with or without stabilizing inputs) in $\mathcal{G}^d_{Unr}$ is also stably computable in $\mathcal{G}$, since it is stably computable in any possible communication graph. So, $\mathcal{B}'$ stably computes t with stabilizing inputs in $\mathcal{G}$. Moreover, there also exists a mediated population protocol $\mathcal{B}$ (the one that is the same as $\mathcal{B}'$ but simply ignores the additional components of the new model) that strongly stably computes t with stabilizing inputs in $\mathcal{G}$. Note that the input alphabet of $\mathcal{B}$ is $X_{\mathcal{B}} = Q_{\mathcal{A}}$, and its transition function is of the form $\delta_{\mathcal{B}} : (Q_{\mathcal{A}} \times Q_{\mathcal{B}}) \times (Q_{\mathcal{A}} \times Q_{\mathcal{B}}) \to Q_{\mathcal{B}} \times Q_{\mathcal{B}}$, since there is no need to specify edge states (formally we should, but the protocol ignores them). $Q_{\mathcal{A}}$ is the set of $\mathcal{A}$'s agent states and $\mathcal{B}$'s inputs that eventually stabilize.

We define a mediated population protocol $\mathcal{C}$ as follows: $X_{\mathcal{C}} = X_{\mathcal{A}}$, $Y_{\mathcal{C}} = Y_{\mathcal{B}} = \{0, 1\}$, $Q_{\mathcal{C}} = Q_{\mathcal{A}} \times Q_{\mathcal{B}}$, $I_{\mathcal{C}} : X_{\mathcal{A}} \to Q_{\mathcal{C}}$ defined as $I_{\mathcal{C}}(x) = (I_{\mathcal{A}}(x), i_{\mathcal{B}})$, for all $x \in Q_{\mathcal{C}}$ where $i_{\mathcal{B}} \in Q_{\mathcal{B}}$ is the initial state of protocol $\mathcal{B}$, $S_{\mathcal{C}} = S_{\mathcal{A}}$, $\iota_{\mathcal{C}} : X_{\mathcal{C}} \to S_{\mathcal{C}}$, that is, $\iota_{\mathcal{C}}(x) = \iota_{\mathcal{A}}(x)$, for all $x \in X_{\mathcal{C}}$, $O_{\mathcal{C}}(a, b) = O_{\mathcal{B}}(b)$, for all $q = (a, b) \in Q_{\mathcal{C}}$, and finally its transition function $\delta_{\mathcal{C}} : Q_{\mathcal{C}} \times Q_{\mathcal{C}} \times S_{\mathcal{C}} \to Q_{\mathcal{C}} \times Q_{\mathcal{C}} \times S_{\mathcal{C}}$ (we omit specifying costs since there is no need for them) is defined as

$$\begin{aligned}\delta_{\mathcal{C}}((a,b),(a',b'),s) = ((&\delta_{\mathcal{A}_1}(a,a',s), \delta_{\mathcal{B}_1}((a,b),(a',b'))),\\ &(\delta_{\mathcal{A}_2}(a,a',s), \delta_{\mathcal{B}_2}((a,b),(a',b'))),\\ &\delta_{\mathcal{A}_3}(a,a',s)),\end{aligned}$$

where for $\delta_{\mathcal{A}}(x, y, z) = (x', y', z')$ (in $\mathcal{A}$'s transition function), we have that $\delta_{\mathcal{A}_1}(x, y, z) = x'$, $\delta_{\mathcal{A}_2}(x, y, z) = y'$, $\delta_{\mathcal{A}_3}(x, y, z) = z'$, and similarly for $\delta_{\mathcal{B}}$.

Intuitively, $\mathcal{C}$ consists of $\mathcal{A}$ and $\mathcal{B}$ running in parallel. The state of each agent is a pair $c = (a, b)$, where $a \in Q_{\mathcal{A}}$, $b \in Q_{\mathcal{B}}$, and the state of each edge is a member of $S_{\mathcal{A}}$. Initially each agent senses an input x from $X_{\mathcal{A}}$ and this is transformed according to $I_{\mathcal{C}}$ to such a pair, where $a = I_{\mathcal{A}}(x)$ and b is always a special $\mathcal{B}$'s initial state $i_{\mathcal{B}} \in Q_{\mathcal{B}}$. When two agents in states (a, b) and (a', b') interact through an edge in state s, then protocol $\mathcal{A}$ updates the first components of the agent states, i.e. a and a', and the edge state s, as if $\mathcal{B}$ didn't exist. On the other hand, protocol $\mathcal{B}$ updates the second components by taking into account the first components that represent its separate input ports at which the current input symbol of each agent is available at every interaction ($\mathcal{B}$ takes $\mathcal{A}$'s states for agent input symbols that may change arbitrarily between any two computation steps, but the truth is that they change due to $\mathcal{A}$'s computation). Since the first components of $\mathcal{C}$'s agent states eventually stabilize as a result of $\mathcal{A}$'s stabilizing states, protocol $\mathcal{B}$ will eventually obtain stabilizing inputs, consequently will operate correctly, and will strongly stably compute t as if it had began computing on $\mathcal{A}$'s final (output) configuration. But, since t provides the correct answer for p if applied on $\mathcal{A}$'s final configuration, it is obvious that $\mathcal{C}$ must strongly stably compute p in $\mathcal{G}$, and the theorem follows. □

Since MPP strongly stably computes a non-semilinear predicate and PP is a special case of MPP, it follows that the class of stably computable predicates by MPP is a proper superset of the class of stably computable predicates by PP. In other words, *the MPP model is computationally stronger than the PP model.*

In [13] the authors also proved the following result: "Any predicate that is stably computable by the MPP model in any family of communication graphs belongs to the space complexity class $NSPACE(m)$". The idea is simple: By using the MPP that stably computes the predicate we construct a nondeterministic Turing machine that guesses in each step the next selection of the scheduler (thus the next configuration). The machine always replaces the current configuration with a new legal one, and, since any configuration can be represented explicitly with $\mathcal{O}(m)$ space, any branch uses $\mathcal{O}(m)$ space. The machine accepts if some branch reaches a configuration C that satisfies instruction r of the protocol, and if, moreover, no configuration reachable from C violates r (i.e. C must also be r-stable).

4 The GDM model

In [13] MPP's ability to decide graph languages was implicitly observed. In fact, the authors made a similar observation: "MPPs are able to locate subgraphs". Based on this observation, in [11] the authors considered a special case of the mediated population protocol model, the *graph decision mediated population protocol model*, or simply *GDM*. The purpose of GDM was to simplify the study of decidability capabilities of the MPP model.

4.1 Formal Definition

Definition 10. *A* GDM *is an 8-tuple* $(Y, Q, O, S, r, \delta, q_0, s_0)$*, where* Y*,* Q*, and* S *are all finite sets and*

1. $Y = \{0, 1\}$ *is the* binary output alphabet*,*
2. Q *is the set of* agent states*,*
3. $O : Q \to Y$ *is the* agent output function*,*
4. S *is the set of* edge states*,*
5. r *is the* output instruction*,*
6. $\delta : Q \times Q \times S \to Q \times Q \times S$ *is the* transition function*,*
7. $q_0 \in Q$ *is the* initial agent state*, and*
8. $s_0 \in S$ *is the* initial edge state*.*

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* $\delta_3(a, b, s) = s'$*.*

Let $\mathcal{U}$ be a graph universe. A *graph language* L is a subset of $\mathcal{U}$ containing communication graphs that share some common property. For example, a common graph universe is the set of all possible directed and weakly connected communication graphs, denoted by $\mathcal{G}$, and $L = \{G \in \mathcal{G} \mid G \text{ has an even number of edges}\}$ is a possible graph language w.r.t. $\mathcal{G}$.

A GDM protocol may run on any graph from a specified graph universe. The graph on which the protocol runs is considered as the *input graph* of the protocol. Note that GDM protocols have no sensed input. Instead, we require each agent in the population to be initially in the initial agent state q_0 and each edge of the communication graph to be initially in the initial edge state s_0. In other words, the initial network configuration, C_0, of any GDM is defined as $C_0(u) = q_0$, for all $u \in V$, and $C_0(e) = s_0$, for all $e \in E$, and any input graph $G = (V, E)$.

We say that a GDM $\mathcal{A}$ *accepts* an input graph G if in any computation of $\mathcal{A}$ on G after finitely many interactions all agents output the value 1 and continue doing so in all subsequent (infinite) computational steps. By replacing 1 with 0 we get the definition of the *reject* case.

Definition 11. *We say that a GDM* $\mathcal{A}$ decides *a graph language* $L \subseteq \mathcal{U}$ *if it accepts any* $G \in L$ *and rejects any* $G \notin L$.

Definition 12. *A graph language is said to be* decidable *if some GDM decides it.*

4.2 Weakly Connected Graphs

Decidability

The most meaningful graph universe is $\mathcal{G}$ containing all possible directed and weakly connected communication graphs, without self-loops or multiple edges, 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 we neither allow infinite graphs). Here the graph universe is $\mathcal{G}$ and, thus, a graph language can only be a subset of $\mathcal{G}$ (moreover, its elements must share some common property).

In [11] it was proven that the class of decidable graph languages is *closed* under *complement*, *union* and *intersection* operations. Moreover, the authors provided protocols (and proved their correctness) that decide the following graph languages:

1. $N_{even} = \{G \in \mathcal{G} \mid |V(G)| \text{ is even}\}$,
2. $E_{even} = \{G \in \mathcal{G} \mid |E(G)| \text{ is even}\}$,
3. $N_k^{out} = \{G \in \mathcal{G} \mid G \text{ has some node with at least } k \text{ outgoing neighbors}\}$ for any $k = \mathcal{O}(1)$,
4. $K_k^{out} = \{G \in \mathcal{G} \mid \text{Any node in } G \text{ has at least } k \text{ outgoing neighbors}\}$ for any $k = \mathcal{O}(1)$,
5. $M_{out} = \{G \in \mathcal{G} \mid G \text{ has some node with more outgoing than incoming neighbors}\}$, and
6. $P_k = \{G \in \mathcal{G} \mid G \text{ has at least one directed path of at least } k \text{ edges}\}$ for any $k = \mathcal{O}(1)$.

So, all the above languages are decidable by the GDM model, and, by closure under complement, the same holds for their complements. For example, $\overline{N}_k^{out}$ contains all graphs that have no node with at least $k = \mathcal{O}(1)$ outgoing neighbors, and is decidable. In other words, GDM can decide if all nodes have less than k outgoing edges, which is simply the well-known bounded by k out-degree predicate.

To illustrate the formal description of GDM protocols we provide the code of the protocol *DirPath* that was proven in [11] to decide the language $P_k = \{G \in \mathcal{G} \mid G \text{ has at least one directed path of at least } k \text{ edges}\}$

for any $k = \mathcal{O}(1)$.

DirPath

- $Q = \{q_0, q_1, 1, \ldots, k\}$, $S = \{0, 1\}$,
- $O(k) = 1$, $O(q) = 0$, for all $q \in Q - \{k\}$,
- r: "*Get any $u \in V$ and read its output*",
- δ:

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

Undecidability

If we allow only GDMs with stabilizing states, i.e. GDMs that in any computation after finitely many interactions stop changing their states, then we can prove that a specific graph language w.r.t. $\mathcal{G}$ is *undecidable*. In particular, we can prove that there exists no GDM with stabilizing states to decide the graph language

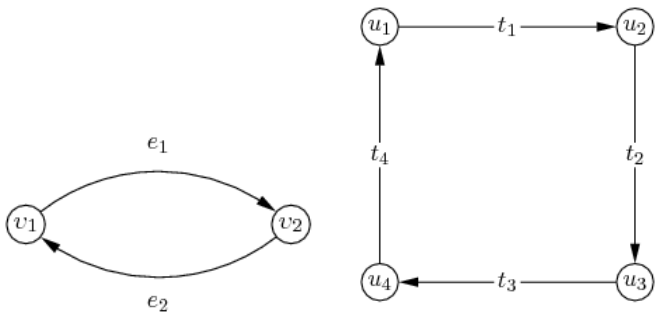
$$\begin{aligned}
2C = \{G \in \mathcal{G} \mid G \text{ has at least two nodes } u, v \text{ s.t. both } (u, v), (v, u) \\
\in E(G) \text{ (in other words, } G \text{ has at least one 2-cycle)}\}.
\end{aligned}$$

The proof is based on the following lemma.

Lemma 3. *For any GDM $\mathcal{A}$ and any computation $C_0, C_1, C_2, \ldots$ 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.*

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

for any finite $i \geq 0$.



(a) Graph G (b) Graph G'

Fig. 1. $G \in 2C$ and $G' \notin 2C$.

Proof. The proof is by induction on i.

Lemma 3 shows that if a GDM $\mathcal{A}$ with stabilizing states could decide $2C$ then there would exist a computation of $\mathcal{A}$ on G' forcing all agents to output incorrectly the value 1 in finitely many steps. But G' does not belong to $2C$, and, since $\mathcal{A}$ decides $2C$, all agents must correct their states to eventually output 0. By taking into account the fact that $\mathcal{A}$ has stabilizing states it is easy to reach a contradiction and prove that no GDM with stabilizing states can decide $2C$. Whether the graph language $2C$ is undecidable by the GDM model in the general case (not only by GDMs with stabilizing states) remains an interesting open problem.

4.3 All Possible Directed Graphs

In [11] it was, also, proven that if we allow the graph universe, $\mathcal{H}$, to contain also disconnected communication graphs, then in this case the GDM model is incapable of deciding even a single nontrivial graph language (we call a graph language L *nontrivial* if $L \neq \emptyset$ and $L \neq \mathcal{H}$). Here we assume the graph universe $\mathcal{H}$ consisting of all possible directed communication graphs, without self-loops or multiple edges of any finite number of nodes greater or equal to 2 (we now also allow graphs that are not even weakly connected). So, now, a graph language can only be a subset of $\mathcal{H}$. We will give the proof idea of that general impossibility result.

First we show that 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). If the statement

does not hold then any disconnected graph in L has all its components in L and any disconnected graph in $\overline{L}$ has all its components in $\overline{L}$.

1. All connected graphs belong to L. Then $\overline{L}$ contains at least one disconnected graph (since it is nontrivial) that has all its components in L, which contradicts the fact that the components of any disconnected graph in $\overline{L}$ also belong to $\overline{L}$.
2. All connected graphs belong to $\overline{L}$. The contradiction is symmetric to the previous case.
3. L and $\overline{L}$ contain connected graphs G and G', respectively. Their disjoint union $U = (V \cup V', E \cup E')$ is disconnected, belongs to L or $\overline{L}$ but one of its components belongs to L and the other to $\overline{L}$. The latter contradicts the fact that both components should belong to the same language.

To obtain the impossibility result the reader should use the fact that the class of decidable graph languages is closed under complement and, also, simply notice that GDMs 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).

5 Further Research Directions

Since the Mediated Population Protocol model was proposed very recently, many important questions concerning it remain open and many directions emerging from it are yet unexplored. The first question that comes to one's mind is if there exists some achievable architecture that implements the proposed models. Is there some other notion of fairness (probably weaker) that would be more suitable for real-life applications? Can we give an exact characterization of the stably computable predicates by the Mediated Population Protocol model like the one already given for the Population Protocol model? If we ignore the sensing capabilities of the MPP model and focus on the GDM model, can we give an exact characterization of the class of decidable graph languages? Is there a general method for proving impossibility results that best suits the GDM model? In other words, can we avoid ad-hoc proofs of impossibility results and borrow or modify techniques from classical distributed computing by also taking into account the fact that our systems are uniform, anonymous, have constant protocol descriptions, and have some nondeterminism inherent in the interaction pattern? Can we devise some reliable simulation

platform or testbed to extensively test or/and verify our protocols before running them in real application scenarios? Since sensor networks are most of the time used in critical environments (fire detection is a classical example), it is reasonable to wonder whether there exists a unified theoretical framework for fast and reliable protocol verification. Since such protocols are usually small we are hoping for the existence of some fast verification process. Assuming a unique leader in the system has been always helpful in distributed computing (to get an idea of its usefulness in population protocols the reader is referred to [4]). It seems that in the models under consideration assuming a leader in the initial configuration is more helpful than letting the protocol elect one. In particular, it seems that an assumed leader provides us with even more computational power, especially in the case where the goal is the construction of some subgraph or the decision of some specific property of the communication graph. Finally, we believe that it is not so bad to assume that some or all agents have $\mathcal{O}(\log n)$ storage capacity (note that if the population consists of 1 billion agents, i.e. $n = 10^9$, then only about 30 bits of memory are required in each agent!), then it is possible to assume unique identifiers, population protocols are no longer anonymous, and it is of great interest studying this realizable scenario.

References

1. D. Angluin, J. Aspnes, M. Chan, M. J. Fischer, H. Jiang, and R. Peralta. Stably computable properties of network graphs. In Proc. Distributed Computing in Sensor Systems: 1st IEEE International Conference, pages 63-74, 2005.
2. D. Angluin, J. Aspnes, Z. Diamadi, M.J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. In *23rd Annual ACM Sympsium on Principles of Distributed Computing* (*PODC*), pages 290-299, New York, NY, USA, 2004. ACM.
3. D. Angluin, J. Aspnes, Z. Diamadi, M.J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4): 235-253, 2006.
4. D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3): 183-199, Sept. 2008.
5. D. Angluin, J. Aspnes, and D. Eisenstat. Stably computable predicates are semilinear. In *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing*, pages 292-299, 2006.
6. D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4): 279-304, November 2007.
7. J. Aspnes and E. Ruppert. An introduction to population protocols. *Bulletin of the European Association for Theoretical Computer Science*, 93:98-117, October 2007. Columns: *Distributed Computing*, Editor: M. Mavronicolas.

8. J. Beauquier, J. Clement, S. Messika, L. Rosaz, and B. Rozoy. Self-stabilizing counting in mobile sensor networks. Technical Report 1470, LRI, Université Paris-Sud 11, 2007.
9. O. Bournez, P. Chassaing, J. Cohen, L. Gerin, and X. Koegler. On the convergence of population protocols when population goes to infinity. To appear in *Applied Mathematics and Computation*, 2009.
10. I. Chatzigiannakis and P. G. Spirakis. The dynamics of probabilistic population protocols. In *Distributed Computing, 22nd International Symposium, DISC*, volume 5218 of *Lecture Notes in Computer Science*, pages 498-499, 2008.
11. I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Decidable Graph Languages by Mediated Population Protocols. FRONTS Technical Report FRONTS-TR-2009-16, http://fronts.cti.gr/aigaion/?TR=80, May 2009.
12. I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Experimental verification and performance study of extremely large sized population protocols. FRONTS Technical Report FRONTS-TR-2009-3, http://fronts.cti.gr/aigaion/?TR=61, Jan. 2009.
13. I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Mediated Population Protocols. FRONTS Technical Report FRONTS-TR-2009-8, http://fronts.cti.gr/aigaion/?TR=65, February 2009. To appear in *36th International Colloquium on Automata, Languages and Programming (ICALP)*, July 5-12, Rhodes, Greece.
14. C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and E. Ruppert. When birds die: Making population protocols fault-tolerant. In *Proc. 2nd IEEE International Conference on Distributed Computing in Sensor Systems*, pages 51-66, 2006.
15. S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285-296, 1966.
16. V. Vazirani. Approximation Algorithms. Springer Verlag, 2001.