Algorithmic Verification of Population Protocols*

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

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

Abstract. In this work, we study the Population Protocol model of Angluin *et al.* from the perspective of protocol verification. In particular, we are interested in algorithmically solving the problem of determining whether a given population protocol conforms to its specifications. Since this is the first work on verification of population protocols, we redefine most notions of population protocols in order to make them suitable for algorithmic verification. Moreover, we formally define the general verification problem and some interesting special cases. All these problems are shown to be NP-hard. We next propose some first algorithmic solutions for a natural special case. Finally, we conduct experiments and algorithmic engineering in order to improve our verifiers' running times.

1 Introduction

Pervasive environments of tomorrow will consist of populations of tiny possibly mobile artifacts that will interact with each other. Such systems will play an important role in our everyday life and should be correct, reliable and robust. To achieve these goals, it is necessary to verify the correctness of future systems. Formal specification helps to obtain not only a better (more modular) description, but also a clear understanding and an abstract view of the system [4]. Given the increasing sophistication of algorithms for pervasive systems and the difficulty of modifying an algorithm once the network is deployed, there is a clear need to use formal methods to validate system performance and functionality prior to implementing such algorithms [20]. Formal analysis requires the use of models, trusted to behave like a real system. It is therefore critical to find the correct abstraction layer for the models and to verify the models.

Model checking is an exhaustive state space exploration technique that is used to validate formally specified system requirements with respect to a formal system description [14]. Such a system is verified for a fixed configuration; so, in most cases, no general system correctness can be obtained. Using some high-level formal modelling language, automatically an underlying state space can be derived, be it implicitly or symbolically. The system requirements are

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

specified using some logical language, like LTL, CTL or extensions thereof [19]. Well-known and widely applied model checking tools are SPIN [18], Uppaal [6] (for timed systems), and PRISM [17] (for probabilistic systems). The system specification language can, e.g., be based on process algebra, automata or Petri nets. However, model checking suffers from the so-called state explosion problem, meaning that the state space of a specified system grows exponentially with respect to its number of components. The main challenge for model checking lies in modelling large-scale dynamic systems.

Towards providing a concrete and realistic model for future sensor networks, Angluin *et al.* [1] introduced the notion of a computation by a population protocol. In their model, individual agents are extremely limited and can be represented as finite-state machines. The computation is carried out by a collection of agents, each of which receives a piece of the input. Information can be exchanged between two agents whenever they come sufficiently close to each other. The goal is to ensure that every agent can eventually output the value that is to be computed. The critical assumption that diversifies the population protocol model from traditional distributed systems is that the protocol descriptions are independent of the population size, which is known as the *uniformity* property of population protocols. Moreover, population protocols are *anonymous* since there is no room in the state of an agent to store a unique identifier. See also [11, 10, 16, 5, 13, 7] for population protocol relevant literature. For the interested reader, [3, 12] constitute introductions to the area.

In this work, we provide a tool for computer-aided *verification* of population protocols. Our tool can detect errors in the design that are not so easily found using emulation or testing, and they can be used to establish the correctness of the design. A very interesting property of population protocols is protocol composition; one may reduce a protocol into two (or more) protocols of reduced state space that maintain the same correctness and efficiency properties.

Section 2 provides all necessary definitions. In particular, several population protocols' definitions are modified in order to become suitable for algorithmic verification. Then the verification problems that we study throughout this work are formally defined. In Section 3, we prove that all verification problems under consideration are NP-hard. In Section 4, we focus on a particular special case of the general population protocol verification problem, called $BPVER$, in which the population size on which the protocol runs is provided as part of the verifier's input. In particular, we devise three verifiers, two non-complete and one complete. The complete one is slower but provably guarantees to always provide the correct answer. We have implemented our verifiers in C++ by building a new tool named *bp-ver*. As far as we are concerned, this is the first verification tool for population protocols. In Section 5, we conduct experiments concerning our verifiers' running times. It turns out that constructing the transition graph is a dominating factor. We then improve our verifiers by building all the reachable subgraphs of the transition graph one after the other and not all at once. In this manner, the running time is greatly improved most of the time and the new construction is easily parallelizable.

2 Necessary Definitions

2.1 Population Protocols

A *Population Protocol* (PP) $\mathcal{A}$ is a 6-tuple (X, Y, Q, I, O, δ), where X, Y, and Q are all finite sets and X is the *input alphabet*, Y is the *output alphabet*, Q is the set of *states*, $I : X \to Q$ is the *input function*, $O : Q \to Y$ is the *output function*, and $\delta : Q \times Q \to Q \times Q$ is the *transition function.* If $\delta(q_i, q_j) = (q_l, q_t)$, then when an agent in state q_i interacts as the *initiator* with an agent in state q_j (which is the *responder* in this interaction) they update their states deterministically to $\delta_1(q_i, q_j) = q_l$ and $\delta_2(q_i, q_j) = q_t$, respectively. δ can also be treated as a relation $\Delta \subseteq Q^4$, defined as $(q_i, q_j, q_l, q_t) \in \Delta$ iff $\delta(q_i, q_j) = (q_l, q_t)$.

A population protocol runs on the nodes-agents of a *communication graph* $G = (V, E)$. In this work, we always assume that the communication graph is a complete digraph, without self-loops and multiple edges (this corresponds to the *basic* population protocol model [1]). We denote by G^k the complete communication graph of k nodes.

Let $k \equiv |V|$ denote the *population size.* An *input assignment* x is a mapping from $V = [k]$ to X (where $[l]$, for $l \in \mathbb{Z}^{\geq 1}$, denotes the set $\{1, \ldots, l\}$), assigning an input symbol to each agent of the population. Since the communication graph is complete, due to symmetry, we can, equivalently, think of an input assignment as a $|X|$-vector of integers $x = (x_i)_{i \in [|X|]}$, where x_i is nonnegative and equal to the number of agents that receive the symbol $\sigma_i \in X$, assuming an ordering on the input symbols. We denote by $\mathcal{X}$ the set of all possible input assignments. Note that for all $x \in \mathcal{X}$ it holds that $\sum_{i=1}^{|X|} x_i = k$.

A state $q \in Q$ is called *initial* if $I(\sigma) = q$ for some $\sigma \in X$. A *configuration* c is a mapping from $[k]$ to Q, so, again, it is a $|Q|$-vector of nonnegative integers $c = (c_i)_{i \in [|Q|]}$ such that $\sum_{i=1}^{|Q|} c_i = k$ holds. Each input assignment corresponds to an *initial configuration* which is indicated by the input function I. In particular, input assignment x corresponds to the initial configuration $c(x) = (c_i(x))_{i \in [|Q|]}$, where $c_i(x)$ is equal to the number of agents that get some input symbols σ_j for which $I(\sigma_j) = q_i$ (q_i is the ith state in Q if we assume the existence of an ordering on the set of states Q). More formally, $c_i(x) = \sum_{j: I(\sigma_j) = q_i} x_j$ for all $i \in [|Q|]$. By extending I to a mapping from input assignments to configurations we can write $I(x) = c$ to denote that c is the initial configuration corresponding to input assignment x. Let $\mathcal{C} = \{(c_i)_{i \in [|Q|]} \mid c_i \in \mathbb{Z}^{\geq 0} \text{ and } \sum_{i=1}^{|Q|} c_i = k\}$ denote the set of all possible configurations given the population protocol $\mathcal{A}$ and G^k. Moreover, let $C_I = \{c \in \mathcal{C} \mid I(x) = c \text{ for some } x \in \mathcal{X}\}$ denote the set of all possible initial configurations. Any $r \in \Delta$ has four components which are elements from Q and we denote by r_i, where $i \in [4]$, the i-th component (i.e. state) of r. $r \in Q^4$ belongs to Δ iff $\delta(r_1, r_2) = (r_3, r_4)$. We say that a configuration c can go in one step to a configuration c' via transition $r \in \Delta$, and write $c \xrightarrow{r} c'$, if

- $c_i \geq r_{1,2}(i)$, for all $i \in [|Q|]$ for which $q_i \in \{r_1, r_2\}$,
- $c'_i = c_i - r_{1,2}(i) + r_{3,4}(i)$, for all $i \in [|Q|]$ for which $q_i \in \{r_1, r_2, r_3, r_4\}$, and
- $c'_j = c_j$, for all $j \in [|Q|]$ for which $q_j \in Q - \{r_1, r_2, r_3, r_4\}$,

where $r_{l,t}(i)$ denotes the number of times state q_i appears in (r_l, r_t). Moreover, we say that a configuration c can go in one step to a configuration c', and write $c \rightarrow c'$ if $c \xrightarrow{r} c'$ for some $r \in \Delta$. We say that a configuration c' is reachable from a configuration c, denoted $c \xrightarrow{*} c'$ 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, $0 \leq i < t$, where c^i denotes the $(i+1)$th configuration of an execution (and not the ith component of configuration c which is denoted c_i). An *execution* is a finite or infinite sequence of configurations $c^0, c^1, \ldots$, so that $c^i \rightarrow c^{i+1}$. An execution is *fair* if for all configurations c, c' such that $c \rightarrow c'$, if c appears infinitely often then so does c'. A *computation* is an infinite fair execution. A predicate $p : \mathcal{X} \rightarrow \{0, 1\}$ is said to be *stably computable* by a PP $\mathcal{A}$ if, for any input assignment x, any computation of $\mathcal{A}$ contains an *output stable configuration* in which all agents output $p(x)$. A configuration c is called *output stable* if $O(c) = O(c')$, for all c' reachable from c (where O, here, is an extended version of the output function from configurations to output assignments in Y^k). We denote by $C_F = \{c \in \mathcal{C} \mid c \rightarrow c' \Rightarrow c' = c\}$ the set of all *final configurations*. We can further extend the output function O to a mapping from configurations to $\{-1, 0, 1\}$, defined as $O(c) = 0$ if $O(c(u)) = 0$ for all $u \in V$, $O(c) = 1$ if $O(c(u)) = 1$ for all $u \in V$, and $O(c) = -1$ if $\exists u, v \in V$ s.t. $O(c(u)) \neq O(c(v))$.

It is known [1, 2] that a predicate is stably computable by the PP model iff it can be defined as a first-order logical formula in *Presburger arithmetic*. Let ϕ be such a formula. There exists some PP that stably computes ϕ, thus ϕ constitutes, in fact, the specifications of that protocol. For example, consider the formula $\phi = (N_a \geq 2N_b)$. ϕ partitions the set of all input assignments, $\mathcal{X}$, to those input assignments that satisfy the predicate (that is, the number of *a*s assigned is at least two times the number of *b*s assigned) and to those that do not. Moreover, ϕ can be further extended to a mapping from C_I to $\{-1, 0, 1\}$. In this case, ϕ is defined as $\phi(c) = 0$ if $\phi(x) = 0$ for all $x \in I^{-1}(c)$, $\phi(c) = 1$ if $\phi(x) = 1$ for all $x \in I^{-1}(c)$, and $\phi(c) = -1$ if $\exists x, x' \in I^{-1}(c)$ s.t. $\phi(x) \neq \phi(x')$, where $I^{-1}(c)$ denotes the set of all $x \in \mathcal{X}$ for which $I(x) = c$ holds (the *preimage* of c).

We now define the *transition graph*, which is similar to that defined in [1], except for the fact that it contains only those configurations that are reachable from some initial configuration in C_I. Specifically, given a population protocol $\mathcal{A}$ and an integer $k \geq 2$ we can define the transition graph of the pair $(\mathcal{A}, k)$ as $G_{\mathcal{A},k} = (C_r, E_r)$, where the node set $C_r = C_I \cup \{c \in \mathcal{C} \mid c' \xrightarrow{*} c \text{ for some } c' \in C_I\}$ of G_r (we use G_r as a shorthand of $G_{\mathcal{A},k}$) is the subset of $\mathcal{C}$ containing all initial configurations and all configurations that are reachable from some initial one, and the edge (or arc) set $E_r = \{(c, c') \mid c, c' \in C_r \text{ and } c \rightarrow c'\}$ of G_r contains a directed edge (c, c') for any two (not necessarily distinct) configurations c and c' of C_r for which it holds that c can go in one step to c'. Note that G_r is a directed (weakly) connected graph with possible self-loops. It was shown in [1] that, given a computation Ξ, the configurations that appear infinitely often in Ξ form a *final strongly connected component* of G_r. We denote by S the collection of all strongly connected components of G_r. Note that each $B \in S$ is simply a set of configurations. Moreover, given $B, B' \in S$ we say the B can go in one step to B', and write $B \to B'$, if $c \to c'$ for $c \in B$ and $c' \in B'$. $B \xrightarrow{*} B'$ is defined as in the case of configurations. We denote by $I_S = \{B \in S \mid \text{such that } B \cap C_I \neq \emptyset\}$ those components that contain at least one initial configuration, and by $F_S = \{B \in S \mid \text{such that } B \to B' \Rightarrow B' = B\}$ the final ones. We can now extend ϕ to a mapping from I_S to $\{-1, 0, 1\}$ defined as $\phi(B) = 0$ if $\phi(c) = 0$ for all $c \in B \cap C_I$, $\phi(B) = 1$ if $\phi(c) = 1$ for all $c \in B \cap C_I$, and $\phi(B) = -1$ if $\exists c, c' \in B \cap C_I$ s.t. $\phi(c) \neq \phi(c')$, and O to a mapping from F_S to $\{-1, 0, 1\}$ defined as $O(B) = 0$ if $O(c) = 0$ for all $c \in B$, $O(B) = 1$ if $O(c) = 1$ for all $c \in B$, and $O(B) = -1$ otherwise.

2.2 Problems' Definitions

We begin by defining the most interesting and natural version of the problem of algorithmically verifying basic population protocols. We call it $GBPVER$ ('G' standing for "General', 'B' for "Basic", and 'P' for "Predicate") and its complement $\overline{GBPVER}$ is defined as follows:

Problem 1 ($\overline{GBPVER}$). Given a population protocol $\mathcal{A}$ for the basic model whose output alphabet $Y_{\mathcal{A}}$ is binary (i.e. $Y_{\mathcal{A}} = \{0, 1\}$) and a first-order logical formula ϕ in Presburger arithmetic representing the specifications of $\mathcal{A}$, determine whether there exists some integer $k \geq 2$ and some legal input assignment x for the complete communication graph of k nodes, G^k, for which not all computations of $\mathcal{A}$ on G^k beginning from the initial configuration corresponding to x stabilize to the correct output w.r.t. ϕ.

A special case of $GBPVER$ is $BPVER$ (its non-general version as revealed by the missing 'G'), and is defined as follows.

Problem 2 ($BPVER$). Given a population protocol $\mathcal{A}$ for the basic model whose output alphabet $Y_{\mathcal{A}}$ is binary (i.e. $Y_{\mathcal{A}} = \{0, 1\}$), a first-order logical formula ϕ in Presburger arithmetic representing the specifications of $\mathcal{A}$, and an integer $k \geq 2$ (in binary) determine whether $\mathcal{A}$ conforms to its specifications on G^k.

"Conforms to ϕ" here means that for any legal input assignment x, which is a $|X_{\mathcal{A}}|$-vector with nonnegative integer entries that sum up to k, and any computation beginning from the initial configuration corresponding to x on G^k, the population stabilizes to a configuration in which all agents output the value $\phi(x) \in \{0, 1\}$.

Problem 3 ($BBPVER$). $BBPVER$ (the additional 'B' is from "Binary input alphabet") is $BPVER$ with $\mathcal{A}$'s input alphabet restricted to $\{0, 1\}$.

3 Hardness Results

Theorem 1. *$BPVER$ is coNP-hard.*

Proof. We shall present a polynomial-time reduction from $HAMPATH = \{\langle D, s, t\rangle \mid D$ is a directed graph with a Hamiltonian path from s to t $\}$ to $\overline{BPVER}$. In other words, we will present a procedure that given an instance $\langle D, s, t\rangle$ of $HAMPATH$ returns in polynomial time an instance $\langle \mathcal{A}, \phi, k\rangle$ of $\overline{BPVER}$, such that $\langle D, s, t\rangle \in HAMPATH$ iff $\langle \mathcal{A}, \phi, k\rangle \in \overline{BPVER}$. If there is a hamiltonian path from s to t in D we will return a population protocol $\mathcal{A}$ that for some computation on the complete graph of k nodes fails to conform to its specification ϕ, and if there is no such path all computations will conform to ϕ.

We assume that all nodes in $V(D) - \{s, t\}$ are named $q_1, \ldots, q_{n-2}$, where n denotes the number of nodes of D (be careful: n does not denote the size of the population, but the number of nodes of the graph D in $HAMPATH$'s instance). We now construct the protocol $\mathcal{A} = (X, Y, Q, I, O, \delta)$. The output alphabet Y is $\{0, 1\}$ by definition. The input alphabet X is $E(D) - (\{(\cdot, s)\} \cup \{t, \cdot\})$, that is, consists of all edges of D except for those leading into s and those going out of t. The set of states Q is equal to $X \cup T \cup \{r\}$, where $T = \{(s, q_i, q_j, l) \mid 1 \leq i, j \leq n-2$ and $1 \leq l \leq n-1\}$ and its usefulness will be explained later. r can be thought of as being the "reject" state, since we will define it to be the only state giving the output value 0. Notice that $|Q| = \mathcal{O}(n^3)$. The input function $I : X \to Q$ is defined as $I(x) = x$, for all $x \in X$, and for the output function $O : Q \to \{0, 1\}$ we have $O(r) = 0$ and $O(q) = 1$ for all $q \in Q - \{r\}$. That is, all input symbols are mapped to themselves, while all states are mapped to the output value 1, except for r which is the only state giving 0 as output. Thinking of the transition function δ as a transition matrix Δ it is easy to see that Δ is a $|Q| \times |Q|$ matrix whose entries are elements from $Q \times Q$. Each entry $\Delta_{q,q'}$ corresponds to the rhs of a rule $(q, q') \to (z, z')$ in δ. Clearly, Δ consists of $\mathcal{O}(n^6)$ entries, which is again polynomial in n.

We shall postpone for a while the definition of Δ to first define the remaining parameters ϕ and k of $\overline{BPVER}$'s instance. We define formula ϕ to be a trivial first-order Presburger arithmetic logical formula that is always false. For example, in the natural nontrivial case where $X \neq \emptyset$ (that is, D has at least one edge that is not leading into s and not going out of t) we can pick any $x \in X$ and set $\phi = (N_x < 0)$ which, for N_x denoting the number of xs appearing in the input assignment, is obviously always false. It is useful to notice that the only configuration that gives the correct output w.r.t. ϕ is the one in which all agents are in state r. ϕ being always false means that in a correct protocol all computations must stabilize to the all-zero output, and r is the only state giving output 0. On the other hand for $\mathcal{A}$ not to be correct w.r.t. ϕ it suffices to show that there exists some computation in which r cannot appear. Moreover, we set k equal to $n - 1$, that is, the communication graph on which $\mathcal{A}$'s correctness has to be checked by the verifier is the complete digraph of $n - 1$ nodes (or, equivalently, agents).

To complete the reduction, it remains to construct the transition function δ:

- $(r, \cdot) \to (r, r)$ and $(\cdot, r) \to (r, r)$ (so r is a propagating state, meaning that once it appears it eventually becomes the state of every agent in the population)

– $((q_i, q_j), (q_i, q_j)) \rightarrow (r, r)$ (if two agents get the same edge of D then the protocol rejects)
– $((q_i, q_j), (q_i, q_l)) \rightarrow (r, r)$ (if two agents get edges of D with adjacent tails then the protocol rejects)
– $((q_j, q_i), (q_l, q_i)) \rightarrow (r, r)$ (if two agents get edges of D with adjacent heads then the protocol rejects - it also holds if one of q_j and q_l is s)
– $((q_i, t), (q_j, t) \rightarrow (r, r)$ (the latter also holds for the sink t)
– $((s, \cdots), (s, \cdots)) \rightarrow (r, r)$ (if two agents have both s as the first component of their states then the protocol rejects)
– $((s, q_i), (q_i, q_j)) \rightarrow ((s, q_i, q_j, 2), (q_i, q_j))$ (when s meets an agent v that contains a successor edge it keeps q_j to remember the head of v's successor edge and releases a counter set to 2 - it counts the number of edges encountered so far on the path trying to reach t from s)
– $((s, q_i, q_j, i), (q_j, q_l)) \rightarrow ((s, q_i, q_l, i + 1), (q_j, q_l))$, for $i < n - 2$
– $((s, q_i, q_j, i), (q_j, t)) \rightarrow (r, r)$, for $i < n - 2$ (the protocol rejects if s is connected to t through a directed path with less than $n - 1$ edges)
– All the transitions not appearing above are identity rules (i.e. they do nothing)

Now we prove that the above, obviously polynomial-time, construction is in fact the desired reduction. If D contains some hamiltonian path from s to t, then the $n - 1$ edges of that path form a possible input assignment to protocol $\mathcal{A}$ (since its input symbols are the edges and the population consists of $n - 1$ agents). When $\mathcal{A}$ gets that input it cannot reject (r cannot appear) for the following reasons:

– no two agents get the same edge of D
– no two agents get edges of D with adjacent tails
– no two agents get edges of D with adjacent heads
– only one $(s, \cdots)$ exists
– s cannot count less than $n - 1$ edges from itself to t

So, when $\mathcal{A}$ gets the input alluded to above, it cannot reach state r, thus, it cannot reject, which implies that $\mathcal{A}$ for that input always stabilizes to the wrong output w.r.t. ϕ (which always requires the "reject" output) when runs on the G^{n-1}. So, in this case $\langle \mathcal{A}, \phi, k \rangle$ consists of a protocol $\mathcal{A}$ that, when runs on G^k, where $k = n - 1$, for a specific input it does not conform to its specifications as described by ϕ, so clearly it belongs to $\overline{BPVER}$.

For the other direction, if $\langle \mathcal{A}, \phi, k \rangle \in \overline{BPVER}$ then obviously there exists some computation of $\mathcal{A}$ on the complete graph of $k = n - 1$ nodes in which r does not appear at all (if it had appeared once then, due to fairness, the population would have stabilized to the all-r configuration, resulting to a computation conforming to ϕ). It is helpful to keep in mind that most arguments here hold because of the fairness condition. Since r cannot appear, every agent (of the $n - 1$ in total) must have been assigned a different edge of D. Moreover, no two of them contain edges with common tails or common heads in D. Note that there is only one agent with state $(s, \cdots)$ because if there were two of them they would have rejected when interacted with each other, and if no $(s, \cdots)$ appeared then two agents would have edges with common tails because there are $n-1$ edges for $n-2$ candidate initiating points (we have not allowed t to be an initiating point) and the pigeonhole principle applies (and by symmetric arguments only one with state $(\cdots, t)$). So, in the induced graph formed by the edges that have been assigned to the agents, s has outdegree 1 and indegree 0, t has indegree 1 and outdegree 0 and all remaining nodes have indegree at most 1 and outdegree at most 1. This implies that all nodes except for s and t must have indegree equal to 1 and outdegree equal to 1. If, for example, some node had indegree 0, then the total indegree could not have been $n-1$ because $n-3$ other nodes have indegree at most 1, t has indegree 1, and s has 0 (the same holds for outdegrees). Additionally, there is some path initiating from s and ending to t. This holds because the path initiating from s (s has outdegree 1) cannot fold upon itself (this would result in a node with indegree greater than 1) and cannot end to any other node different from t because this would result to some node other than t with outdegree equal to 0. Finally, that path has at least $n-1$ edges (in fact, precisely $n-1$ edges), since if it had less the protocol would have rejected. Thus, it must be clear after the above discussion that in this case there must have been a hamiltonian path from s to t in D, implying that $\langle D, s, t\rangle \in HAMPATH$. □

The following theorem captures the hardness of the other two problems.

Theorem 2. *$BBPVER$ and $GBPVER$ are coNP-hard.*

Proof. Due to spave restrictions we prove only the second statement. We will prove the statement by presenting a polynomial-time reduction from $\overline{BPVER'}$ to $\overline{GBPVER}$. Keep in mind that the input to the machine computing the reduction is $\langle \mathcal{A}, \phi, k\rangle$. Let $X_{\mathcal{A}}$ be the input alphabet of $\mathcal{A}$. Clearly, $\phi'' = \neg(\sum_{x \in X_{\mathcal{A}}} N_x = k)$ is a semilinear predicate if k is treated as a constant (N_x denotes the number of agents with input x). Thus, there exists a population protocol $\mathcal{A}''$ for the basic model that stably computes ϕ''. The population protocol $\mathcal{A}''$ can be constructed efficiently. Its input alphabet $X_{\mathcal{A}''}$ is equal to $X_{\mathcal{A}}$. The construction of the protocol can be found in [1] (in fact they present there a more general protocol for any linear combination of variables corresponding to a semilinear predicate). When the number of nodes of the communication graph is equal to k, $\mathcal{A}''$ always stabilizes to the all-zero output (all agents output the value 0) and when it is not equal to k, then $\mathcal{A}''$ always stabilizes to the all-one output.

We want to construct an instance $\langle \mathcal{A}', \phi'\rangle$ of $\overline{GBPVER}$. We set $\phi' = \phi \vee \phi''$. Moreover, $\mathcal{A}'$ is constructed to be the composition of $\mathcal{A}$ and $\mathcal{A}''$. Obviously, $Q_{\mathcal{A}'} = Q_{\mathcal{A}} \times Q_{\mathcal{A}''}$. We define its output to be the union of its components' outputs, that is, $O(q_{\mathcal{A}}, q_{\mathcal{A}''}) = 1$ iff at least one of $O(q_{\mathcal{A}})$ and $O(q_{\mathcal{A}''})$ is equal to 1. It is easy to see that the above reduction can be computed in polynomial time.

We first prove that if $\langle \mathcal{A}, \phi, k\rangle \in \overline{BPVER'}$ then $\langle \mathcal{A}', \phi'\rangle \in \overline{GBPVER}$. When $\mathcal{A}'$ runs on the complete graph of k nodes, the components of its states corresponding to $\mathcal{A}''$ stabilize to the all-zero output, independently of the initial configuration. Clearly, $\mathcal{A}'$ in this case outputs whatever $\mathcal{A}$ outputs. Moreover, for this communication graph, ϕ' is true iff ϕ is true (because $\phi'' = \neg(\sum_{x \in X_{\mathcal{A}}} N_x = k)$ is false, and $\phi' = \phi \vee \phi''$). But there exists some input for which $\mathcal{A}$ does not give the correct output with respect to ϕ (e.g. ϕ is true for some input but $\mathcal{A}$ for some computation does not stabilize to the all-one output). Since ϕ' expects the same output as ϕ and $\mathcal{A}'$ gives the same output as $\mathcal{A}$ we conclude that there exists some erroneous computation of $\mathcal{A}'$ w.r.t. ϕ', and the first direction has been proven.

Now, for the other direction, assume that $\langle \mathcal{A}', \phi' \rangle \in \overline{GBPVER}$. For any communication graph having a number of nodes not equal to k, ϕ' is true and $\mathcal{A}'$ always stabilizes to the all-one output because of the $\mathcal{A}''$ component. This means that the erroneous computation of $\mathcal{A}'$ happens on the G^k. But for that graph, ϕ'' is always false and $\mathcal{A}''$ always stabilizes its corresponding component to the all-zero output. Now ϕ' is true iff ϕ is true and $\mathcal{A}'$ outputs whatever $\mathcal{A}$ outputs. But there exists some input and a computation for which $\mathcal{A}'$ does not stabilize to a configuration in which all agents give the output value that ϕ' requires which implies that $\mathcal{A}$ does not stabilize to a configuration in which all agents give the output value required by ϕ. Since the latter holds for G^k, the theorem follows. □

4 Algorithmic Solutions for *BPVER*

Our algorithms are search algorithms on the transition graph G_r. The general idea is that a protocol $\mathcal{A}$ does not conform to its specifications ϕ on k agents, if one of the following criteria is satisfied:

1. $\phi(c) = -1$ for some $c \in C_I$.
2. $\exists c, c' \in C_I$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq \phi(c')$.
3. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $O(c') = -1$.
4. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq O(c')$.
5. $\exists B' \in F_S$ such that $O(B') = -1$.
6. $\exists B \in I_S$ and $B' \in F_S$ such that $B \xrightarrow{*} B'$ and $\phi(B) \neq O(B')$ (possibly $B = B'$).

Note that any algorithm that correctly checks some of the above criteria is a possibly *non-complete verifier*. Such a verifier guarantees that it can discover an error of a specific kind, thus, we can always trust its "reject" answer. On the other hand, an "accept" answer is a weaker guarantee, in the sense that it only informs that the protocol does not have some error of this specific kind. Of course, it is possible that the protocol has other errors, violating criteria that are indetectable by this verifier. However, this is a first sign of *BPVER*'s *parallelizability*.

Theorem 3. *Any algorithm checking criteria 1, 5, and 6 decides BPVER.*

Algorithm 1 *SinkVER*

Input: A PP $\mathcal{A}$, a Presburger arithmetic formula ϕ, and an integer $k \geq 2$.
Output: ACCEPT if $\mathcal{A}$ is correct w.r.t. its specifications and the criteria 1, 2, 3, and 4 on G^k and REJECT otherwise.

```
 1: $C_I \leftarrow FindC_I(\mathcal{A}, k)$
 2: if there exists $c \in C_I$ such that $\phi(c) = -1$ then
 3:     return REJECT // Criterion 1 satisfied
 4: end if
 5: $G_r \leftarrow ConG_r(\mathcal{A}, k)$
 6: for all $c \in C_I$ do
 7:     Collect all $c'$ reachable from $c$ in $G_r$ by BFS or DFS.
 8:     while searching do
 9:         if one $c'$ is found such that $c' \in C_F$ and $(O(c') = -1$ or $\phi(c) \neq O(c'))$
            then
10:             return REJECT // Criterion 3 or 4 satisfied
11:         end if
12:         if one $c'$ is found such that $c' \in C_I$ and $\phi(c) \neq \phi(c')$ then
13:             return REJECT // Criterion 2 satisfied
14:         end if
15:     end while
16: end for
17: return ACCEPT // Tests for criteria 1,2,3, and 4 passed
```

4.1 Constructing the Transition Graph

Let $FindC_I(\mathcal{A}, k)$ be a function that given a PP $\mathcal{A}$ and an integer $k \geq 2$ returns the set C_I of all initial configurations. This is not so hard to be implemented. $FindC_I$ simply iterates over the set of all input assignments $\mathcal{X}$ and for each $x \in \mathcal{X}$ computes $I(x)$ and puts it in C_I. Alternatively, computing C_I is equivalent to finding all distributions of indistinguishable objects (agents) into distinguishable slots (initial states), and , thus, can be done by Fenichel's algorithm [15].

The transition graph G_r can be constructed by some procedure, call it $ConG_r$, which is a simple application of searching and, thus, we skip it. It takes as input a population protocol $\mathcal{A}$ and the population size k, and returns the transition graph G_r.

4.2 Non-complete Verifiers

We now present two non-complete verifiers, namely *SinkBFS* and *SinkDFS*, that check all criteria but the last two. Both are presented via procedure *SinkVER* (Algorithm 1) and the order in which configurations of G_r are visited determines whether BFS or DFS is used.

Algorithm 2 *SolveBPVER*

Input: A PP $\mathcal{A}$, a Presburger arithmetic formula ϕ, and an integer $k \geq 2$.
Output: ACCEPT if the protocol is correct w.r.t. its specifications on G^k and REJECT otherwise.

```
 1: $C_I \leftarrow FindC_I(\mathcal{A}, k)$
 2: if there exists $c \in C_I$ such that $\phi(c) = -1$ then
 3:     return REJECT
 4: end if
 5: $G_r \leftarrow ConG_r(\mathcal{A}, k)$
 6: Run one of Tarjan's or Gabow's algorithms to compute the collection $S$ of all
    strongly connected components of the transition graph $G_r$.
 7: Compute the dag $D = (S, A)$, where $(B, B') \in A$ (where $B \neq B'$) if and only if
    $B \to B'$.
 8: Compute the collection $I_S \subseteq S$ of all connected components $B \in S$ that contain
    some initial configuration $c \in C_I$ and the collection $F_S \subseteq S$ of all connected
    components $B \in S$ that have no outgoing edges in $A$, that is, all final strongly
    connected components of $G_r$.
 9: for all $B \in F_S$ do
10:     if $O(B) = -1$ then
11:         return REJECT
12:     end if
13:     // Otherwise, all configurations $c \in B$ output the same value $O(B) \in \{0, 1\}$.
14: end for
15: for all $B \in I_S$ do
16:     if there exist initial configurations $c, c' \in B$ such that $\phi(c) \neq \phi(c')$ then
17:         return REJECT
18:     else
19:         // all initial configurations $c \in B$ expect the same output $\phi(B) \in \{0, 1\}$.
20:         Run BFS or DFS from $B$ in $D$ and collect all $B' \in F_S$ s.t. $B \xrightarrow{*} B'$
            (possibly including $B$ itself).
21:         if there exists some reachable $B' \in F_S$ for which $O(B') \neq \phi(B)$ then
22:             return REJECT
23:         end if
24:     end if
25: end for
26: return ACCEPT
```

4.3 *SolveBPVER*: A Complete Verifier

We now construct the procedure *SolveBPVER* (Algorithm 2) that checks criteria 1, 5, and 6 (and also 2 for some speedup) of Section 4, and, thus, according to Theorem 3, it correctly solves *BPVER* (i.e. it is a complete verifier). In particular, *SolveBPVER* takes as input a PP $\mathcal{A}$, its specifications ϕ and an integer $k \geq 2$, as outlined in the *BPVER* problem description, and returns "accept" if the protocol is correct w.r.t. its specifications on G^k and "reject" otherwise.

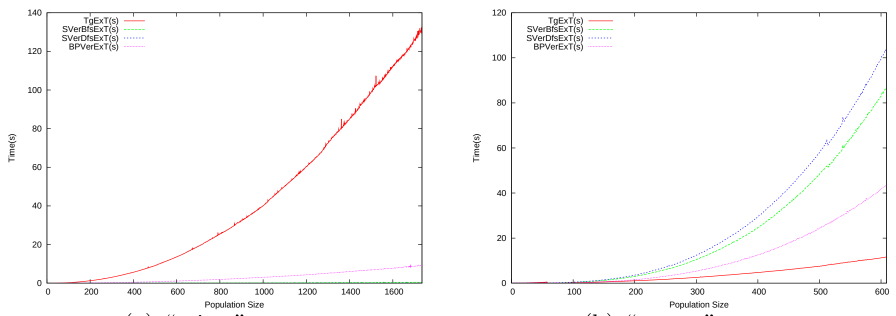
The idea is to use Tarjan's [21] or Gabow's (or any other) algorithm for finding the strongly connected components of G_r. In this manner, we obtain a collection S, where each $B \in S$ is a strongly connected component of G_r, that is, $B \subseteq C_r$. Given S we can easily compress G_r w.r.t. its strongly connected components as follows. The compression of G_r is a dag $D = (S, A)$, where $(B, B') \in A$ if and only if there exist $c \in B$ and $c' \in B'$ such that $c \rightarrow c'$ (that is, iff $B \rightarrow B'$). In words, the node set of D consists of the strongly connected components of G_r and there is a directed edge between two components of D if a configuration of the second component is reachable in one step from a configuration in the first one.

We have implemented our verifiers in C++. We have named our tool *bp-ver* and it can be downloaded from http://ru1.cti.gr/projects/BP-VER. Our implementation makes use of the *boost* graph library. In particular, we exploit boost to store and handle the transition graph and to find its strongly connected components. Boost uses Tarjan's algorithm [21] for the latter. We use Fenichel's algorithm [15] in order to find all possible initial configurations, and our implementation is based on Burkardt's FORTRAN code [8]. Protocols and formulas are stored in separate files and simple, natural syntax is used. Formulas are evaluated with Dijkstra's Shunting-yard algorithm for evaluating expressions.

5 Experiments & Algorithmic Engineering

As presented so far, all verifiers first construct the transition graph and then start searching on it, each with its own method, in order to detect some error. Note also that all algorithms halt when the first error is found, otherwise they halt when there is nothing left to search.

Our first suspicion was that the time to construct the transition graph must be a dominating factor in the "reject" case but not in the "accept" case. To see this for the "reject" case imagine that all verifiers find some error near the first initial configuration that they examine. For example, they may reach in a few steps another initial configuration that expects different output. But, even if all of them reject in a few steps, they still will have paid the construction of the whole transition graph first, which is usually huge. On the other hand, this must not be a problem in the "accept" case, because all verifiers have, more or less, to search the whole transition graph before accepting.



(a) “reject” case. (b) “accept” case.

Fig. 1. (a): Verifiers executed on erroneous version of *flock*$_2$ (see Protocol 3, where the corresponding stably computable predicate is $(N_1 \geq i)$) w.r.t. formula $(N_1 \geq 2)$. The dominating factor is the time needed to construct the transition graph. For all $n \geq 4$ all verifiers found some error. (b): Verifiers executed on the correct *flock*$_2$t w.r.t. formula $(N_1 \geq 2)$. *SinkDFS* and *SinkBFS* verifiers are clearly faster than *SolveBPVER* in this case. For all $n \geq 4$ all verifiers decided that the protocol is correct.

Protocol 3 *flock*$_i$

```
1: // $i$ must be at least 1
2: $X = Y = \{0, 1\}$, $Q = \{q_0, q_1, \ldots, q_i\}$,
3: $I(0) = q_0$ and $I(1) = q_1$,
4: $O(q_l) = 0$, for $0 \leq l \leq i - 1$, and $O(q_i) = 1$,
5: $\delta$:
```

$$\begin{aligned} (q_k, q_j) &\to (q_{k+j}, q_0), \text{ if } k + j < i \\ &\to (q_i, q_i), \text{ otherwise.} \end{aligned}$$

These speculations are confirmed by our first experiments whose findings are presented in Figure 1. In particular, we consider the “flock of birds” protocol that counts whether at least 2 birds in the flock were found infected. The corresponding Presburger formula is $(N_1 \geq 2)$. In Figure 1(a) we introduced a single error to the protocol’s code that all verifiers would detect. Then we counted and plotted the time to construct the transition graph and the execution (CPU) time of all verifiers (until they answer “reject”) for different population sizes. In Figure 1(b) we did the same for the correct version of the protocol. See Protocol 3 for the code of protocol *flock*$_i$.

The good news is that our verifiers’ running times can easily be improved. The idea is to take the initial configurations one after the other and search only in the subgraph of the transition graph that is reachable from the current initial configuration. In this manner, we usually avoid in the “reject” case to construct the whole transition graph. Especially in cases that we are lucky to detect an error in the first subgraph that we ever visit, and if this subgraph happens to be small, the running time is greatly improved.

References

1. 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. Also in *23rd Annual ACM Symposium on Principles of Distributed Computing* (*PODC*), pages 290-299, 2004.
2. D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4): 279-304, November 2007.
3. J. Aspnes and E. Ruppert. An introduction to population protocols. *Bulletin of the European Association for Theoretical Computer Science*, 93:98-117, Oct. 2007.
4. R. Bakhshi, F. Bonnet, W. Fokkink, B. Haverkort. Formal analysis techniques for gossiping protocols. In *ACM SIGOPS Operating Systems Review*, 41(5):28-36, *Special Issue on Gossip-Based Networking*, October, 2007.
5. 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.
6. G. Behrmann, A. David, and K. G. Larsen. A tutorial on uppaal. In *Formal Methods for the Design of Real-Time Systems: Proc. 4th Int. School on Formal Methods for the Design of Comput., Commun. and Software Syst. (SFM-RT 2004)*, number 3185 in LNCS, pages 200236. Springer, 2004.
7. O. Bournez, P. Chassaing, J. Cohen, L. Gerin, and X. Koegler. On the convergence of population protocols when population goes to infinity. In *Applied Mathematics and Computation*, 215(4):1340-1350, 2009.
8. http://people.sc.fsu.edu/∼burkardt/f_src/combo/combo.f90.
9. I. Chatzigiannakis, S. Dolev, S. P. Fekete, O. Michail, and P. G. Spirakis. Not all fair probabilistic schedulers are equivalent. In *13th International Conference On Principles Of DIstributed Systems (OPODIS)*, pages 33-47, 2009.
10. I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Brief Announcement: Decidable graph languages by mediated population protocols. In *23rd International Symposium on Distributed Computing (DISC)*, Elche, Spain, Sept. 2009.
11. I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Mediated population protocols. In *36th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 363-374, Rhodes, Greece, 2009.
12. I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Recent advances in population protocols. In *34th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, Aug. 2009.
13. I. Chatzigiannakis and P. G. Spirakis. The dynamics of probabilistic population protocols. In *Distributed Computing*, *22nd International Symposium*, *DISC*, volume 5218 of *LNCS*, pages 498-499, 2008.
14. E. M. Clarke, O. Grumberg, and D. A. Peled. Model checking. MIT Press, 2000.
15. R. Fenichel. Distribution of indistinguishable objects into distinguishable slots. *Communications of the ACM*, 11(6), page 430, June 1968.
16. R. Guerraoui and E. Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In *36th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 484-495, Greece, 2009.

17. A. Hinton, M. Z. Kwiatkowska, G. Norman, and D. Parker. Prism: A tool for automatic verification of probabilistic systems. In *Proc. 2nd Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS06)*, volume 3920 of LNCS, pages 441-444. Springer, 2006.
18. G. Holzmann. The Spin model checker, primer and reference manual. Addison-Wesley, 2003.
19. M. Huth and M. Ryan. Logic in Computer Science: Modelling and reasoning about systems. Cambridge University Press, Cambridge, UK, 2004.
20. P.C. Olveczky and S. Thorvaldsen. Formal modeling and analysis of wireless sensor network algorithms in Real-Time Maude. *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium International*, pp. 157, 2006.
21. R. Tarjan. Depth-first search and linear graph algorithms. In *SIAM Journal on Computing*, Vol. 1, No. 2, p. 146-160, 1972.