Mobile Agent Algorithms versus Message Passing Algorithms

J. Chalopin[1], E. Godard[2], Y. Métivier[1] and R. Ossamy[1]

[1] LaBRI UMR 5800
ENSEIRB - Université Bordeaux 1
351 Cours de la Libération
33405 - Talence France
{chalopin,metivier,ossamy}@labri.fr
[2] LIF UMR 6166
Université de Provence
39 rue Joliot-Curie
13453 Marseille France
{egodard@cmi.univ-mrs.fr}

1 Introduction

The mobile agent paradigm has been developed to solve problems in dynamic and heterogeneous environment [8]. The agent model of this paper is quite general. It is based on the concepts of agents, communication links and places. An agent is an entity which executes an algorithm: it can move from place to place (with some data and its algorithm) through communication links and it can make local computations on a place (a place provides tools for local computations: data, memories and process). Thus a mobile agent system is defined:

- by a network or equivalently by an undirected labelled graph (the vertices correspond to the places) with a port numbering function,
- by a set of agents, and
- by an initial placement of the agents on the graph.

As a particular case of our model the network and the mobile agents may be anonymous: identities are available neither for the vertices of the network nor for the agents. A mobile agent (based) algorithm is defined by a mobile agent system where each agent is endowed with its proper algorithm.

Classical problems for mobile agents include: election, naming, locating agents, rendez-vous, stabilisation, termination detection of agents, exploring, topology recognition. There exist a lot of results for these problems assuming different properties of the environment [2, 4–6, 10–12].

A message passing system is a set of processes and a communication subsystem. It corresponds to the standard models given in [1, 16]. The communication model is a point-to-point communication network which is represented as a simple connected undirected graph where the vertices represent processes and two

vertices are linked by an edge if the corresponding processes have a direct communication link. Processes communicate by message passing, and each process knows from which channel it receives a message or it sends a message. We consider the asynchronous message passing model: processes cannot access a global clock and a message sent from a process to a neighbor arrives within some finite but unpredictable time. A message passing algorithm is a collection of local algorithms, one for each process. The anonymous case corresponds to the case where all local algorithms are the same.

In this paper we are interested in the computational power of a mobile agent system and, more particularly, in the comparison with a message passing system. It is easy to verify that a message passing system can simulate a mobile agent system (Section 4); it has been already indicated in [3]. But, given a mobile agent system, can mobile agents be used to implement arbitrary message passing algorithms? We give a positive answer in Section 5; and we obtain a theorem stating the equivalency of the two models (Theorem 10).

Our result establishes a useful bridge between message passing systems and mobile agent systems. In Section 6, we give an example of consequence of this result with a translation of known results in message passing computing for the election problem into the mobile agents setting. In [10], it is proved that a leader can be elected among k agents on a graph having n vertices if k and n are coprime; this result becomes a consequence of the characterisation given in [17] and of Theorem 10. Theorem 2 and Theorem 3 of [5] become corollaries of results presented in [13]. The same method can be applied for other classical problems such as the naming, the topology recognition, the spanning tree construction, etc.

2 Graphs and Labelled Graphs

We consider finite undirected connected graphs. A graph $G = (V(G), E(G))$ (or $G = (V, E)$ for short) is defined by a set $V(G)$ of vertices and a set $E(G)$ of edges; in this paper graphs are without multiple edges or self-loop. Two vertices u and v are said to be adjacent or neighbors if $\{u, v\}$ is an edge of G (thus u and v are necessarily distinct since no self-loop is admitted) and $N_G(v)$ will stand for the set of neighbors of v; u and v are the endvertices of e. An edge e is incident to a vertex v if $v \in e$ and $I_G(v)$ will stand for the set of all the edges incident to v. A homomorphism between graphs G and H is a mapping $\gamma\colon V(G) \to V(H)$ such that if $\{u, v\} \in E(G)$ then $\{\gamma(u), \gamma(v)\} \in E(H)$. We say that γ is an isomorphism if γ is bijective and γ^{-1} is a homomorphism.

Throughout the rest of this paper we will consider graphs whose vertices are labelled with labels from a recursive set L. A graph labelled over L will be denoted by (G, λ), where G is a graph and $\lambda\colon V(G) \to L$ is the vertex labelling function. The graph G is called the underlying graph and the mapping λ is a labelling of G. Labelled graphs will be designated by bold letters like $\mathbf{G}$, $\mathbf{H}, \ldots$ If $\mathbf{G}$ is a labelled graph, then G denotes the underlying graph.

A mapping $\gamma\colon V(G) \to V(G')$ is a homomorphism from (G, λ) to (G', λ') if γ is a graph homomorphism from G to G' which preserves the labelling, i.e., such that $\lambda'(\gamma(v)) = \lambda(v)$ holds for every $v \in V(G)$.

3 The Formal Models

3.1 The Message Passing Model

Definitions given in this subsection follow [16] (p. 45-47) or [1] (p. 10-12).

Message Passing System. A message passing system (P, C) consists of a collection P of processes and a communication subsystem C. It is described by a simple connected undirected graph $G = (V, E)$, where the vertices represent the processes and the edges represent the bidirectional channels. The system is asynchronous: no global time is available; messages can arrive at arbitrary times and processes can take steps at arbitrary speeds. Processes communicate by asynchronous message passing and each process knows from which channel it receives a message or it sends a message: an edge between two processes p_1 and p_2 (or vertices v_1 and v_2) represents a channel connecting a port i of p_1 (or v_1) to a port j of p_2 (or v_2). Let δ be the port numbering function, we assume that for each vertex u (or process p) and each adjacent vertex v (or process q), $\delta_u(v)$ (or $\delta_p(q)$) is a unique integer belonging to $[1, deg(u)]$. Finally, the communication subsystem is described by $C = (V, E, \delta)$. Each process has an initial state defined by a labelling function λ. Thus the message passing system is defined by (P, C, λ) or equivalently by (V, E, δ, λ).

Remark 1. The labelling λ of processes may encode anonymous network (all the vertices have the same label) or any initial process knowledge. Examples of such knowledge include: (a bound on) the number of processes, (a bound on) the diameter of the communication subsystem, the topology of the communication subsystem, identities or partial identities of processes, distinguished processes, sense of direction.

Message Passing Algorithm. To each process is associated a transition system which can interact with the communication subsystem. The events which are associated with a process are internal events, send events and receive events. In a send (resp. receive) event a message is produced (resp. consumed). This definition contains the particular case where every processor executes the same algorithm.

Remark 2. In general, names are not available to the processes themselves. Nevertheless, for ease of exposition, a message m in transit is denoted by (p, m, p') where p is the sending process and p' is the receiving process.

Let $\mathcal{M}$ be the set of possible messages. Let p be a process. The local algorithm of the process p, denoted by $\mathcal{D}_p$, is defined by:

- the (recursive) set Q of possible states of p,
- the subset I of Q of initial states,
- the initial state of p equals to $\lambda(p)$,
- a relation $\vdash_p$ of events (internal events, send events or receive events).

Let p be a process. Let M be the multiset of messages in transit (initially M is empty). The state associated to p is denoted by $\texttt{state}(p)$. The transition associated to the process p is denoted by:

$$(c, in, m) \vdash_p (d, out, m'),$$

(where c and d are states, in and out are integers, and m and m' are messages) means that:

- if $in = out = 0$ then $m = m' = \perp$ (m and m' are undefined): it is an internal event, the new state of the process p is d (it was c before);
- if $in \neq 0$ then $out = 0$ and $m' = \perp$ (m' is undefined): it is a receive event, the state of the process p was c, p has received the message m through the port in and its new state is d; an occurence of m (of the form (p', m, p) where $\delta_p(p') = in$) is removed from M;
- if $out \neq 0$ then $in = 0$ and $m = \perp$ (m is undefined): it is a send event, initially the state of the process p is equal to c; after the transition it is equal to d and the message m' is sent via the port out; an occurence of m' (of the form (p, m', p') where $\delta_p(p') = out$) is added to M.

A message passing algorithm $\mathcal{D}$ for the message passing system (P, C, λ) is a collection of local algorithms $\mathcal{D}_p$, one for each process $p \in P$. It is denoted by $\mathcal{D} = (\mathcal{D}_p)_{p \in P}$. An event of the message passing algorithm is defined by an event on a process.

Execution of a Message Passing Algorithm. An execution $\mathcal{E}$ of the message passing algorithm is defined by a sequence $(\texttt{state}_0, M_0), (\texttt{state}_1, M_1), \ldots,$ $(\texttt{state}_i, M_i), \ldots$ such that:

- for each i, M_i is the multiset of messages in transit,
- $M_0 = \emptyset$,
- for each i and for each process p, $\texttt{state}_i(p)$ denotes the state of the process p,
- for each process p, $\texttt{state}_0(p) = \lambda(p) \in I$,
- for each i, there exists a unique process p such that:
  - if $p' \neq p$ then $\texttt{state}_{i+1}(p') = \texttt{state}_i(p')$,
  - $\texttt{state}_{i+1}(p)$ and M_{i+1} are obtained from $\texttt{state}_i(p)$ and M_i by an event on the process p.

By definition, $(\texttt{state}_i, M_i)$ is a configuration. The execution $\mathcal{E}$ of the message passing algorithm is defined by:

$$\mathcal{E} = (\texttt{state}_i, M_i)_{i \geq 0}.$$

A terminal configuration is a configuration for which no more event is applicable. We can note that in a terminal configuration the multiset M of messages in transit is empty. In this case the length of the execution is the length of the sequence.

Termination Detection. Definitions given in this section are in [16] (Chapter 8). A state q of a process p is active if an internal or send event of p is applicable in q, and passive otherwise. In a passive state q only receipts are applicable, or no event is applicable at all, in which case q is a terminal state of p. Process p is said to be active if it is in an active state, and process p is said to be passive otherwise.

As it is explained in [16] (p. 270), some assumptions are made in order to simplify the description (any process can be easily modified to satisfy these assumptions) : an active process becomes passive only in an internal event, a process always becomes active when a message is received, the internal events in which p becomes passive are the only internal events of p (internal events in which p moves from one active state to another active state are ignored).

A message passing algorithm has terminated when all processes are passive and no message is in transit. In this case, termination is said to be implicit if the processes are not aware that the algorithm has terminated.

Termination is said to be explicit if at least one process detects the termination of the message passing algorithm in the sense that all processes have computed their final values. It then can call an algorithm which floods a termination message to all processes. Only after explicit termination can the result of a computation be regarded as final and variables used in the computation discarded.

3.2 The Mobile Agent Model

Mobile Agent System. A mobile agent system consists of :

- a collection $\mathbb{P}$ of execution places (or places for short),
- a navigation subsystem $\mathbb{S}$,
- a collection $\mathbb{A}$ of mobile agents,
- an injection $\pi_0 : \mathbb{A} \longrightarrow \mathbb{P}$ describing the initial placement of the agents,
- an initial labelling λ of the places and the agents.

Remark 3. The labelling λ of the places and of the agents may encode anonymous places (all the places have the same label), anonymous agents (all the agents have the same label) or any initial agent knowledge. Examples of such knowledge include: (a bound on) the number of places, (a bound on) the number of agents, (a bound on) the diameter of the navigation subsystem, the topology of the navigation subsystem, the topology of the placement of the agents, identities or partial identities of places or agents, distinguished places or agents, sense of direction.

The navigation subsystem is described by a simple undirected connected graph $G = (V, E)$, where the vertices V represent execution places and the edges represent bidirectionnal navigation channels operating between them. In the sequel, we identify the places and the corresponding vertices, and we identify the edges and the corresponding channels.

Each agent which migrates from a place to another place knows through which channel it migrates, that is, for each place port numbers are assigned to its ports : let u be a vertex, let δ_u be the port numbering function which assigns to each adjacent vertex v of u a unique integer $\delta_u(v)$ belonging to $[1, deg(u)]$. Thus the navigation subsystem is defined by $\mathbb{S} = (G, \delta)$.

The system is asynchronous: no global clock is accessible; a migration is asynchronous: an agent which migrates arrives within some finite but unpredictable time on a place.

Mobile Agent Algorithm. Given a mobile agent system, we define a mobile agent algorithm. To each mobile agent is associated a transition system that can interact with the execution places and the navigation subsystem.

Let $Q_{\mathbb{P}}$ be a (recursive) set of states associated to the execution places, and let $Q_{\mathbb{A}}$ be a (recursive) set of states associated to the mobile agents. The initial state of each mobile agent $\mathbf{a}$ is $\lambda(\mathbf{a})$ and the initial state of each execution place $\mathbf{p}$ is $\lambda(\mathbf{p})$.

Let $\mathbf{p}$ be a place. We denote by $\mathtt{state}(\mathbf{p})$ the state associated to $\mathbf{p}$. Let $\mathbf{a}$ be an agent. We denote by $\mathtt{state}(\mathbf{a})$ the state associated to $\mathbf{a}$.

The transition associated to the mobile agent $\mathbf{a}$ in the state s on the place $\mathbf{p}$ in the state q, transforms s into s', q into q' and either $\mathbf{a}$ does not move or it migrates on an adjacent place through the port *out*. We denote the transition by :

$$(s, q, in) \vdash_{\mathbf{p}}^{\mathbf{a}} (s', q', out),$$

it means that the mobile agent $\mathbf{a}$ has migrated on the place $\mathbf{p}$ through the port *in* or after the transition it leaves the place $\mathbf{p}$ through the port *out*, with the convention that if the agent was already on the place and it does not move after the transition then $in = 0$ and $out = 0$; furthermore *in* and *out* cannot be simultaneously different from 0.

Remark 4. In general, names are not available to the places themselves. Nevertheless, for ease of exposition, an agent $\mathbf{a}$ in transit is denoted by $(p, \mathbf{a}, p')$ where p is the place of departure and p' is the place of arrival.

A configuration of the mobile agent system consists of the state of each place, the state of each agent, the collection $\mathbb{M}$ of agents in transit (initially $\mathbb{M}$ is empty) and a mapping π describing the placement of the agents which are not in a channel (several agents can be on the same place).

An event in the mobile agent system is defined by a transition associated to an agent $\mathbf{a}$ on a place $\mathbf{p}$, it has the form :

$$(s, q, in) \vdash_{\mathbf{p}}^{\mathbf{a}} (s', q', out),$$

the state of each agent different from $\mathbf{a}$ is not affected, the state of each place different from $\mathbf{p}$ is not affected, the new state of $\mathbf{a}$ is s' (it was s before the event), the new state of $\mathbf{p}$ is q' (it was q before the event), and:

- if $in = 0$ and $out = 0$ then π and $\mathbb{M}$ are not affected by the event,

– if $in = 0$ and $out \neq 0$ then the set of agents in transit after the event is $\mathbb{M} \cup \{(\mathbf{p}, \mathbf{a}, \mathbf{p}')\}$ (where $\mathbf{p}'$ is the adjacent place of $\mathbf{p}$ corresponding to the port out,) and π is no more defined for $\mathbf{a}$ and unchanged for the other agents,
– if $in \neq 0$ and $out = 0$ then the set of agents in transit after the event is $\mathbb{M} \setminus \{(\mathbf{p}', \mathbf{a}, \mathbf{p})\}$ (where $\mathbf{p}'$ is the adjacent place of $\mathbf{p}$ corresponding to the port in), $\pi(\mathbf{a}) = \mathbf{p}$ and π is unchanged for the other agents.

Execution. An execution of the mobile agent algorithm is defined by a sequence $(\mathtt{state}_0, \mathbb{M}_0, \pi_0), (\mathtt{state}_1, \mathbb{M}_1, \pi_1), ..., (\mathtt{state}_i, \mathbb{M}_i, \pi_i), ...$ such that :

– $\mathbb{M}_0 = \emptyset$,
– for each agent $\mathbf{a}$, $\mathtt{state}_0(\mathbf{a}) = \lambda(\mathbf{a})$ is an initial state,
– for each place $\mathbf{p}$, $\mathtt{state}_0(\mathbf{p}) = \lambda(\mathbf{p})$ is an initial state,
– π_0 is the initial placement of agents,
– for each i there exists a unique place $\mathbf{p}$ and a unique agent $\mathbf{a}$ such that:
  • if $\mathbf{p}' \neq \mathbf{p}$ then $\mathtt{state}_{i+1}(\mathbf{p}') = \mathtt{state}_i(\mathbf{p}')$,
  • if $\mathbf{a}' \neq \mathbf{a}$ then $\mathtt{state}_{i+1}(\mathbf{a}') = \mathtt{state}_i(\mathbf{a}')$,
  • $(\mathtt{state}_{i+1}(\mathbf{a}), \mathtt{state}_{i+1}(\mathbf{p}), \mathbb{M}_{i+1}, \pi_{i+1})$ is obtained from $(\mathtt{state}_i(\mathbf{a}), \mathtt{state}_i(\mathbf{p}), \mathbb{M}_i, \pi_i)$ by an event of the form :

$$(s, q, in) \vdash_{\mathbf{p}}^{\mathbf{a}} (s', q', out).$$

A configuration is defined by $(\mathtt{state}_i, \mathbb{M}_i, \pi_i)$. A terminal configuration is a configuration for which no more event can appear; we can note that in this case the collection of agents in transit is empty. By definition, the length of the sequence is the length of the execution.

A place which is the initial place of an agent is called an homebase. The initial placement of the agents can be encoded in the state of the place (it can be the first action of each mobile agent) thus we assume that the state of a place enables to know whether it is a homebase or not. Nevertheless, in general, an agent cannot know whether a homebase is its own homebase.

Finally, the mobile agent system is defined by:

$$(\mathbb{A}, \mathbb{P}, \mathbb{S}, \pi_0, \lambda),$$

the mobile agent algorithm is defined by:

$$\mathcal{A} = (\vdash_{\mathbf{p}}^{\mathbf{a}})_{\mathbf{a} \in \mathbb{A}, \mathbf{p} \in \mathbb{P},}$$

and an execution $\mathcal{E}$ is defined by :

$$\mathcal{E} = (\mathtt{state}_i, \mathbb{M}_i, \pi_i)_{i \geq 0}.$$

Termination Detection. An agent $\mathbf{a}$ in state s on the place $\mathbf{p}$ in state q is said passive if no transition is associated to this configuration. In a terminal configuration all the agents are passive. Termination is said implicit if no agent is aware that the mobile agent algorithm has terminated. Termination is said explicit if at least one agent detects the termination of the mobile agent algorithm in the sense that all places have their final values. If at least one agent detects the termination then a termination annoucement algorithm using the agents which have detected termination can be activated.

3.3 Equivalent Executions

We consider mobile agent algorithms and message passing algorithms such that the graph corresponding to the navigation subsystem and the graph corresponding to the communication subsystem are equal.

Various kinds of equivalences between mobile agent algorithms and message passing algorithms can be defined. In this work, we consider algorithms which always terminate.

A mobile agent algorithm and a message passing algorithm are equivalent for the terminal configurations if the set of graphs corresponding to the navigation subsystem labelled by the final states of places and the set of graphs corresponding to the communication subsystem labelled by the final states of processes are equal and if the termination of the two algorithms is implicit or the termination of the two algorithms is explicit.

4 Simulating a Mobile Agent Algorithm through a Message Passing Algorithm

The purpose of this section is to verify that, given a mobile agent algorithm, it is possible to implement the same algorithm through a message passing algorithm. To reach this goal, we intend to prove that all the agents basic computation steps can be effectively simulated in the asynchronous message passing system. The main idea of this proof appears in [3].

Let $(\mathbb{A}, \mathbb{P}, \mathbb{S}, \pi_0, \lambda)$ be a mobile agent system and let $\mathcal{A} = (\vdash_{\mathbf{p}}^{\mathbf{a}})_{\mathbf{a}\in\mathbb{A},\mathbf{p}\in\mathbb{P}}$ be a mobile agent algorithm implemented on this system. Let $\mathbb{S} = (V, E, \delta)$ be the corresponding navigation subsystem. We assume that the system contains k agents. We define an additional labelling χ_{π_0} of the vertices of G such that $\chi_{\pi_0}(v) = 1$ if there exists an agent $\mathbf{a}$ such that $\pi_0(\mathbf{a}) = v$, and $\chi_{\pi_0}(v) = 0$ otherwise. Starting from the mobile agent system we build up a message passing system $(P, C, \lambda') = (V, E, \delta, \lambda')$. On each vertex v which corresponds to an execution place $\mathbf{p}$ we install a process p. Let $\sharp$ be a new label. The labelling function λ' encodes on each process p the label of the corresponding place $\mathbf{p}$, whether the place is a homebase and, in the case of a homebase, the label of the corresponding mobile agent $\mathbf{a}$, i.e., if v corresponds to the homebase of $\mathbf{a}$ $\lambda'(v) = (\lambda(v), 1, \lambda(\mathbf{a}))$ and if not $\lambda'(v) = (\lambda(v), 0, \sharp))$.

Now we build a message passing algorithm $\mathcal{D}$ such that each execution $\mathcal{E}$ of $\mathcal{A}$ can be simulated by an execution $\mathcal{E}'$ of $\mathcal{D}$. A state of a process is defined by: - the state of the corresponding place, - the presence of a mobile agent is encoded by a token and the state of the corresponding mobile agent is encoded by the value of the token. Finally, the set of possible states of the processes is the set of possible states defined by the places, the presence of the token and if there is a token by the state of the corresponding agent.

The presence of an agent $\mathbf{a}$ at a given vertex u is represented by the token $t(\mathbf{a})$ located at u. Each token has a *homebase* that corresponds to the initial location of the corresponding mobile agent. To each token is associated a state: the current state of the token $t(\mathbf{a})$ is equal to the state of the agent $\mathbf{a}$.

The translation of the event:

$$(s, q, in) \vdash_{\mathbf{p}}^{\mathbf{a}} (s', q', out)$$

of the mobile agent algorithm into an event of the message passing algorithm is done according to the following rules.

The state of each token different from the token which is associated to $\mathbf{a}$ is not affected, the state of each process different from p is not affected, the new state of the token associated to $\mathbf{a}$, i.e., $t(\mathbf{a})$, is s' (it was s before the event), the new state of p is q' (it was q before the event), and

- if $in = 0$ and $out = 0$ then the token $t(\mathbf{a})$ does not move,
- if $in = 0$ and $out \neq 0$ then the token $t(\mathbf{a})$ is sent via the port out,
- if $in \neq 0$ and $out = 0$ then the token $t(\mathbf{a})$ is received by the process p via the port in.

Let $\mathcal{D}_p$ be the algorithm induced by this construction on the process p. Let $\mathcal{D} = (\mathcal{D}_p)_{p \in P}$. By an induction on the length of the executions, we prove that if the mobile agent algorithm has the termination property then the message passing algorithm defined above has also the termination property. The message passing algorithm $\mathcal{D}$ terminates explicitly if and only if $\mathcal{A}$ terminates explicitly. A graph corresponding to the navigation subsystem labelled by the final states of places is obtained with $\mathcal{A}$ if and only if it can be obtained as a graph corresponding to the communication subsystem labelled by the final states of processes with $\mathcal{D}$. Finally:

Proposition 5. *Let $(\mathbb{A}, \mathbb{P}, \mathbb{S}, \pi_0, \lambda)$ be a mobile agent system.*
Let $\mathcal{A} = (\vdash_{\mathbf{p}}^{\mathbf{a}})_{\mathbf{a} \in \mathbb{A}, \mathbf{p} \in \mathbb{P}}$ be a mobile agent algorithm implemented on this system. Let (P, C, λ') be the message passing system built above. Let $\mathcal{D} = (\mathcal{D}_p)_{p \in P}$ be the message passing algorithm defined above. Then the executions of $\mathcal{D}$ are equivalent to the executions of $\mathcal{A}$.

5 Simulating a Message Passing Algorithm through a Mobile Agent Algorithm

As it is mentionned by Tel ([16] p. 46), the transition systems serve as a theoretical model and algorithms are not necessarily described by an enumeration of their states and events but by means of variables and a convenient pseudocode (see [16], Appendix A).

In this section we turn our attention to the presentation of a procedure that, given a message passing algorithm $\mathcal{D}$ over the message passing system (V, E, δ, λ) and a number $k \geq 1$, generates an equivalent mobile agent algorithm $\mathcal{A}$ with k agents over a mobile agent system.

Our procedure works as follows. The navigation subsystem corresponds to (V, E, δ). The state of the place $\mathbf{p}$ which corresponds to the process p and (which is identified to the vertex v) is defined by the state of the process p (with the same initialisation) and by the values of the variables defined below. The states and the algorithms associated to the mobile agents are defined in the sequel.

Procedure 1

Step 1: *On wake-up, an agent* $\mathbf{a}$ *constructs a tree* $T_{\mathbf{a}}$ *using a partial traversal of the graph. This leads to a spanning forest of* k *trees where each tree is constructed by exactly one agent.*

Step 2: *The agent* $\mathbf{a}$ *executes the algorithm* $\mathcal{D}$ *on the vertices of* $T_{\mathbf{a}}$*. This execution is performed in rounds, where, in each round,* $T_{\mathbf{a}}$ *is traversed and, if possible,* d $(d \geq 1)$ *computation steps of* $\mathcal{D}$ *are executed at each vertex of* $T_{\mathbf{a}}$*. Thus,* $\mathbf{a}$ *is responsible for the computation steps performed on the vertices that belong to its constructed tree.*

Due to the fact that the vertices and the agents are possibly anonymous and because of a lack of global orientation in the network, it may be difficult for an agent to find its way through the graph. To overcome this problem, the agents have to make use of the local edge labelling informations in order to keep track of their way. Therefore, we must keep in mind that each edge $e = \{u, v\}$ has two labels $\delta_u(v)$ and $\delta_v(u)$ that respectively correspond to the labels of e at u and v. Whenever an agent traverses the graph, it stores in its memory the ordered sequence of the labels of the traversed edges. In the rest of this section *ePath* (exploration path) will refer to this sequence. When the edge e is traversed by an agent $\mathbf{a}$ from u to v, then the label $\delta_v(u)$ is appended to the path associated to the agent $\mathbf{a}$. This enables $\mathbf{a}$ to return back to the previously visited vertex (i.e., u) whenever it wants to. When it does so, the label $\delta_v(u)$ is deleted from *ePath*. Thus, at any time during the computation, *ePath* contains the sequence of the labels of the edges that $\mathbf{a}$ has to traverse (in reverse order) to return to its homebase from the current vertex.

The Tree Computation by an Agent Each agent $\mathbf{a}$ computes its tree $T_{\mathbf{a}}$ by executing the following. Starting from its homebase, $\mathbf{a}$ performs a partial traversal of the graph. During this traversal, it marks all the unmarked vertices that are visited for the first time. At each vertex w marked by $\mathbf{a}$, $\mathbf{a}$ arbitrarily chooses an unexplored link incident to w and traverses it. This technique has been employed in [10]. Whenever $\mathbf{a}$ traverses an edge e to reach an unmarked vertex v, it marks v as *visited* and marks e as a T edge. On the other hand, when it reaches a vertex u that is already visited (the vertex u is already marked), the edge leading to u is marked as an NT edge and the agent immediately backtracks to the previous vertex (marked by it) and tries the other unexplored edges incident to that vertex. When there is no more unexplored edges, the agent backtracks to the previous visited vertex (by taking the last link in the $ePath$ sequence) and then tries to explore any unexplored link at that vertex. Finally, when the agent has returned to its homebase and there is no more unexplored link at the current vertex, it stops the tree computation.

Remark 6. A mark on an edge can be done with marks on the corresponding ports on the endvertices of the edge.

The tree computation procedure executed by each agent is given in Algorithm 1.

```
   Mark the homebase;
   Set ePath to empty;
3: while there is an unexplored edge e = (u, v) at the current vertex u, do
     traverse e to reach vertex v;
     if v is already marked or v contains an agent a_1, then
6:     return back to u and mark the link e with the label NT;
     else
       mark the link e as T and mark v as explored;
9:   end if
   end while
   if there are no more unexplored links at the current vertex, then
12:  if ePath is not empty, then
       remove the last link from ePath, traverse that link and go to line 3;
     else
15:    Stop the tree computation;
     end if
   end if
```

Algorithm 1: The tree construction by a mobile agent

Fact 1 *Every vertex in the graph is marked by exactly one agent. If two vertices u_1 and u_2 are marked by the same agent then there exists exactly one simple path of T edges joining them. If two vertices u_1 and u_2 are marked by two different agents, then each path joining them contains at least one NT edge. There is no cycle consisting of only T edges.*

Encoding Message Passing Actions Among the computation steps inherent to the message passing system, the operations *send a message via the port j* and *receive a message from the port j* are surely the most important. To encode these operations, we require that at each place there is a variable called *in-buf*, where received messages are stored. In this framework, the first part of a message always contains the port from which it was received.

Send message m via port j. Let $e = \{u, v\}$ and let **a** be an agent located at u with $\delta_u(v) = j$. The execution of the operation *Send message m via port j* by the agent **a** consists in traversing the edge e, writing the composed message $\langle \delta_v(u), m \rangle$ in the *in-buf* of v and backtracking the edge e.

Receive message m from port j. Let $e = \{u, v\}$ and let **a** be an agent located at u with $\delta_u(v) = j$. The execution of the operation *Receive message m from port j* by the agent **a** consists in looking for the first message arrived from port j in the *in-buf* of u (if the initial algorithm needs the FIFO property of channels, i.e., it requires that messages are received in the same order as they have been sent; if it is not the case then we take any message in *in-buf*). Once this message is found, it is deleted from the *in-buf* and stored in a temporary variable for purpose of computation.

Internal events. It suffices to apply the transformation corresponding to the transformation of the state of the process to the state of the place.

Remark 7. The execution of the *send* and *receive* operations allows an agent $\mathbf{a}$ to write informations in the *in-buf* of vertices that do not necessary belong to $T_{\mathbf{a}}$. Moreover, the simulation executed by Procedure 1 can be viewed as a distributed computation in a network whose processes are decomposed in clusters, and where processes, of the same cluster, execute their computation steps in turn.

Remark 8. Suppose that at each round of Step 2 of Procedure 1, we ask that each time a computation step can be performed on a vertex belonging to the tree of an agent, it simulates it within a finite number of rounds. Then any global state of the message passing system obtained by the message passing algorithm can be obtained by the mobile agents simulation.

Transforming a Terminating Message Passing Algorithm into a Mobile Agent Algorithm that Terminates We are now interested in showing that if $\mathcal{D}$ is a terminating algorithm, then the mobile agent algorithm $\mathcal{A}$ has also the termination property. For this reason, we have adapted the behavior of each agent to this new goal. In fact, we add on each place two variables *fatherLink* and *fState*. The *fatherLink* of a vertex u, contains the port number through which an agent $\mathbf{a}$, located at u, can reach the *father* of u in the tree that contains u. Let u be the homebase of the agent $\mathbf{a}$, the *fState* of u indicates to $\mathbf{a}$ that it has to perform one more computation round on the tree $T_{\mathbf{a}}$. The *fState* is either the token *Finished* or the token *NotFinished*. Initially the *fatherLink* of each homebase contains 0 and the *fState* of each homebase is set to *NotFinished*. The *fatherLink* of the other vertices is 0 and the *fState* of the other vertices is *Finished*. All these changes lead to the following adapted version of Procedure 1.

Procedure 2

Step 1: *On wake-up, each agent* $\mathbf{a}$ *constructs a tree* $T_{\mathbf{a}}$ *using a partial traversal of the graph. This leads to a spanning forest of* k *trees where each tree is constructed by exactly one agent. During this construction, the fatherLinks of all the vertices, other than the homebase, that were marked by* $\mathbf{a}$ *are actualized.*

Step 2: *The agent* $\mathbf{a}$ *executes the events of* $\mathcal{D}$ *on the vertices of* $T_{\mathbf{a}}$*. This execution is performed in rounds. The agent* $\mathbf{a}$ *is allowed to execute round* r *if and only if at the beginning of the round* r *the fState of its homebase has the value NotFinished and it has simulated all the possible steps in the former round otherwise it falls asleep. In each round,* $\mathbf{a}$ *sets the fState of its homebase to Finished, afterward* $T_{\mathbf{a}}$ *is traversed and, if possible,* d $(d \geq 1)$ *computation steps of* $\mathcal{D}$ *are executed at each vertex of* $T_{\mathbf{a}}$*. If an agent mimics the fact of sending a message to a vertex* u*, the agent takes advantage of the fatherLink to find the homebase* w *of the agent that has constructed the tree containing* u*. Once it arrives at* w*, it sets the fState of* w *to NotFinished (if there is a sleeping agent, it wakes it up) and goes back to* u*.*

One has to take notice of the fact that if the algorithm $\mathcal{D}$ terminates, then there exists a time t_1 such that no more computation step is performed after the time t_1 in $\mathcal{D}$. Let $S_{\mathcal{D}}^{t_1}$ be the state of the network at time t_1. Due to Lemma 9, there also exists a time t_2, during the execution of $\mathcal{A}$, such that $S_{\mathcal{D}}^{t_1} = S_{\mathcal{A}}^{t_2}$. It is then quite simple to see that there exists a time $t_3 \geq t_2$ such that at time t_3 the *fState* of any homebase is empty. Thus Procedure 2 and $\mathcal{A}$ stop. Furthermore, the algorithm $\mathcal{D}$ terminates explicitly if and only the mobile agent algorithm $\mathcal{A}$ constructed as defined by the Procedure 2 terminates explicitly. Finally:

Lemma 9. *The mobile agent algorithm $\mathcal{A}$, constructed as defined by Procedure 2, is equivalent to the algorithm $\mathcal{D}$.*

¿From these two kinds of equivalences, we can give our main theorem, stating the equivalency of the two models considered in this paper.

Theorem 10. *There exists a mobile agent algorithm $\mathcal{A}$ that solves a problem $\mathcal{P}$ on a mobile agent system (G, δ, λ) with an initial placement π_0 if and only if there exists a message passing algorithm $\mathcal{D}$ that solves the problem $\mathcal{P}$ on (G, δ, λ') (λ' is defined in Section 4).*

6 Applications

In this section, we will use our main theorem to give a characterisation of the mobile agent systems where we can solve two equivalent problems that are *election* and *rendez-vous*. The same method can be applied for other classical problems such as the topology recognition, the naming, the spanning tree construction, etc.

Election and Rendez-vous The election problem is one of the paradigms of the theory of distributed computing. It was first posed by LeLann [14]. A message algorithm solves the election problem if it always terminates and in the final configuration exactly one process is marked as *elected* and all the other processes are *non-elected*. Moreover, it is supposed that once a process becomes *elected* or *non-elected* then it remains in such a state until the end of the algorithm. Yamashita and Kameda [17] characterise the graphs for which there exists an election algorithm in the message passing model (see also [7, 9]). In the mobile agents setting, the aim of a mobile agent election algorithm is to elect one agent. The elected agent enters a final state *leader*, whereas all other agents enter a final state *follower*. Another important problem in this setting is the rendez-vous problem. The aim of a rendez-vous algorithm is to arrive in a configuration where all the mobile agents gather in a same vertex of the graph. Another interpretation of a rendez-vous algorithm $\mathcal{A}$ is that the aim of $\mathcal{A}$ is to elect a vertex of the network. These two problems are equivalent, since once an agent has been elected, all the agents can gather in the homebase of the elected agent. Reversely, once all the agents gather in some node, the first agent on this node becomes the leader, whereas all the others become followers. Agent election and rendez-vous have been studied in [5, 3, 10]. Consequently, from our

main theorem, there exists an algorithm that solves the rendez-vous and the mobile agent election in a mobile agent system $(\mathbb{A}, \mathbb{P}, \mathbb{S}, \pi_0, \lambda)$ with $\mathbb{S} = (V, E, \delta)$ if and only if there exists an election algorithm in the message passing system (V, E, δ, λ') (λ' is defined in Section 4).

From Message Passing Computations to Mobile Agent Computations The characterization of Yamashita and Kameda given in [17] is based on the notion of view. Given a labelled graph $\mathbf{G} = (V, E, \delta, \lambda)$ and an integer d, the d-view $\mathbf{T}^d_{\mathbf{G}}(v_0)$ of a node $v_0 \in V(G)$ is a tree of height d that can be defined recursively as follows.

- $\mathbf{T}^0_{\mathbf{G}}(v_0)$ is a single-vertex graph whose node is denoted x_0 and $\lambda'(x_0) = \lambda(v_0)$,
- To define $\mathbf{T}^{d+1}_{\mathbf{G}}(v_0)$, we take a copy of $\mathbf{T}^d_{\mathbf{G}}(v_i)$ for each neighbor v_i of v_0 in G. The root of the new tree is a vertex x_0 labelled by $\lambda(v_0)$ and there is an edge between x_0 and the root x_i of each tree $\mathbf{T}^d_{\mathbf{G}}(v_i)$, such that $\delta_{x_0}(x_i) = \delta_{v_0}(v_i)$ and $\delta_{x_i}(x_0) = \delta_{v_i}(v_0)$.

The view $\mathbf{T}_{\mathbf{G}}(v_0)$ of a node v_0 is an infinite rooted labelled tree that can be defined recursively in the same way. The root of the tree is a vertex x_0 that corresponds to v_0 and is labelled by $\lambda(v_0)$. For each neighbor v_i of v_0 in G, there is an edge between x_0 and the root x_i of the tree $\mathbf{T}_{\mathbf{G}}(v_i)$, such that $\delta_{x_0}(x_i) = \delta_{v_0}(v_i)$ and $\delta_{x_i}(x_0) = \delta_{v_i}(v_0)$. The view of a vertex v in a graph $\mathbf{G}$ can also be obtained by considering all labelled walks in $\mathbf{G}$ starting from v. Clearly, the d-view of a vertex v is its view truncated at distance d. In [17], the following theorem is given:

Theorem 11 ([17]). *There exists an election algorithm over a graph (G, δ, λ) if and only if $\forall v, v' \in V(G)$ $(v \neq v')$, the labelled trees $\mathbf{T}_{\mathbf{G}}(v)$ and $\mathbf{T}_{\mathbf{G}}(v')$ are not isomorphic.*

Norris shows in [15] that $\mathbf{T}_{\mathbf{G}}(v)$ and $\mathbf{T}_{\mathbf{G}}(v')$ are isomorphic if and only if $\mathbf{T}^n_{\mathbf{G}}(v)$ and $\mathbf{T}^n_{\mathbf{G}}(v')$ are isomorphic, where $n = |V(G)|$. Each vertex can compute its $2n$-view, and then it will know all the n-views of the other vertices. Once each vertex knows the views of all the other nodes, the vertex with the weaker view is elected. ¿From our main result, we can therefore give the following corollary for the mobile agent election problem. Let λ' the labelling function defined at the beginning of Section 4.

Corollary 12. *There exists an agent election algorithm or a rendez-vous algorithm in a mobile agent system $(\mathbb{A}, \mathbb{P}, \mathbb{S}, \pi_0, \lambda)$ with $\mathbb{S} = (V, E, \delta)$ if and only if for all vertices $v, v' \in V(G)$, the labelled trees $\mathbf{T}^n_{\mathbf{G}'}(v)$ and $\mathbf{T}^n_{\mathbf{G}'}(v')$ are not isomorphic, where $\mathbf{G}' = (G, \delta, \lambda')$ and $n = |V(G)|$.*

In the same way, using the results of Flocchini et al. [13], we can obtain similar characterizations for these problems in the mobile agent system where there is a sense of direction, and the results of [5] become corollaries of these characterizations.

References

1. H. Attiya and J. Welch. *Distributed computing: fundamentals, simulations, and advanced topics*. McGraw-Hill, 1998.
2. B. Awerbuch, M. Betke, R. Rivest, and M. Singh. Piecemeal graph exploration by a mobile robot (extended abstract). In *Proc. of the eighth annual conference on Computational Learning Theory, COLT'95*, pages 321–328. ACM Press, 1995.
3. L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Can we elect if we cannot compare? In *Proc. of the fifteenth annual ACM Symposium on Parallel Algorithms and Architectures, SPAA'03*, pages 324–332. ACM Press, 2003.
4. L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Election and rendezvous in fully anonymous systems with sense of direction. In *Proc. of the 10th International Colloquium on Structural Information Complexity, SIROCCO'03*, volume 17, pages 17–32. Carleton Scientific, 2003.
5. L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Rendezvous and election of mobile agents: impact of sense of direction. *Theory of Computing Systems*, to appear.
6. M. Bender and D. Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. In *Proc. of the 35th annual Symposium on Foundations of Computer Science, FOCS'94*, pages 75–85, 1994.
7. P. Boldi, B. Codenotti, P. Gemmell, S. Shammah, J. Simon, and S. Vigna. Symmetry breaking in anonymous networks: Characterizations. In *Proc. 4th Israeli Symposium on Theory of Computing and Systems*, pages 16–26. IEEE Press, 1996.
8. P. Braun and W. Rossak. *Mobile agents: basic concepts, mobility models and the tracy toolkit*. Morgan Kaufman, 2005.
9. J. Chalopin and Y. Métivier. A bridge between the asynchronous message passing model and local computations in graphs (*extended abstract*). In *Proc. of Mathematical Foundations of computer science, MFCS'05*, volume 3618 of *LNCS*, pages 212–223, 2005.
10. S. Das, P. Flocchini, A.Nayak, and N. Santoro. Distributed exploration of an unknown graph. In *Proc. of the 12th international colloquium on Structural Information and Communication Complexity, SIROCCO'05*, volume 3499 of *LNCS*, pages 99–114, 2005.
11. X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment. i: the rectilinear case. *J. ACM*, 45(2):215–245, 1998.
12. A. Dessmark, P. Fraigniaud, and A. Pelc. Deterministic rendezvous in graphs. In *Proc. of the 11th annual European Symposium on Algorithms, ESA'03*, volume 2832 of *LNCS*, pages 184–195, 2003.
13. P. Flocchini, A. Roncato, and N. Santoro. Computing on anonymous networks with sense of direction. *Theoretical Computer Science*, 301:355–379, 2003.
14. G. LeLann. Distributed systems: Towards a formal approach. In B. Gilchrist, editor, *Information processing'77*, pages 155–160. North-Holland, 1977.
15. N. Norris. Universal covers of graphs: isomorphism to depth $n-1$ implies isomorphism to all depths. *Discrete Applied Math.*, 56:61–74, 1995.
16. G. Tel. *Introduction to distributed algorithms*. Cambridge University Press, 2000.
17. M. Yamashita and T. Kameda. Computing on anonymous networks: Part i - characterizing the solvable cases. *IEEE Transactions on parallel and distributed systems*, 7(1):69–89, 1996.