Universal Semi-local Election Protocol Using Forward Links *

Dobiesław Wróblewski†

Institute of Computer Science
Polish Academy of Sciences
ul. Ordona 21, 01-237 Warsaw
wrobldob@ipipan.waw.pl

Abstract. We consider finite connected undirected graphs as a model for anonymous computer networks. In this framework we show a general purpose distributed election protocol, which uses forward links over the standard communication channels between processors. The forward links are represented in the form of structured labels, so the algorithm is in fact a graph relabelling system. However, its transformations are not local in the classical sense. For this particular algorithm we define a new notion of semi-locality. We claim that semi-local computations are as powerful as global ones, while still conforming to the intuitive meaning of the locality term.

1. Introduction

A network of computers is quite naturally modelled by a finite connected graph. The nodes of such a graph represent the computers, while the edges stand for the communication links. The nodes may have unique labels called identities (ordered or not). Otherwise the network/graph is called anonymous.

In anonymous networks certain problems (including election) have no solution if we use local methods. In this paper we present a model of computations, which has the power of global approach while maintaining advantages of the local methodology. The model is called semi-local and we present an election algorithm using semi-local calculations.

The paper is organised as follows. In Section 2 we formulate the election problem and discuss related results. Section 3 introduces preliminary notions used to formulate our election protocol, which is defined and proved correct in Section 4. In Section 5 we show that the power of semi-local and global

*The author acknowledges support from the Ministry of Scientific Research and Information Technology grant No 3 T11C 006 27 for the years 2004-2005.
†Address for correspondence: ICS PAS, ul. Ordonia 21, 01-237 Warsaw, Poland
computations is identical. Then come the conclusions, including a discussion on implementation of the defined algorithm and prospects for further research.

2. The Problem and Related Results

There are two types of protocols that can act on computer networks - centralised processes having global knowledge of the whole network\footnote{By knowledge we mean awareness of the structure of the network and states of its processors.}, and - on the other hand - distributed computations that base on the knowledge of some fragment of the network and possibly change states of the processors - within this fragment only.

The first group of protocols is proved to - generally - have more computational power and their performance is trivially better for tasks that come down to network recognition or solving global problems (like election which is defined below). On the other hand, the local protocols are generally more efficient in tasks requiring occasional short-distance communication. They are decentralised, adaptive and failsafe\footnote{In the global model, a failure of the central server causes malfunction of the whole network, whereas in the local model the network remains operational as long as it remains connected.}, thus much more practical to be implemented on computer networks, especially on large ones or on those, whose structure dynamically changes.

In [1], D. Angluin presented a model of local computations, the basic ideas of which are used in our paper. In the model, the processors with the same number of communication ports are regarded identical. The computation starts with setting some uniform initial state for all the processors in the network and after that, the processors execute the same protocol, whose behaviour depends on their states and the number of neighbours (i.e. used communication ports). The author has investigated limitations of such a model and found a connection between the notion of covering (known from topological graph theory) and the limitations of the model defined. Besides, the author distinguished a class of networks called ‘centred’, which is of our particular interest.

The ‘centred’ network has exactly one processor, which is called a ‘centre’ or a ‘leader’. Note that computation in such a network still can be distributed and local. Besides, it can be fail-safe and not dependent on any particular node as long as any processor can be the leader. The network may be designed in such a way that the leadership rotates through the network (either periodically or as a response to a change of the network’s structure, e.g. a failure of the leader processor).

The process of establishing a leader is called election. The election problem is particularly interesting due to the following facts:

- the power of local computations in networks with a distinguished leader is the same as the power of global protocols;
- the election task is conceptually simpler than its equivalent problems (e.g. enumeration or finding a spanning tree);
- no universal solution to the problem exists for the general case of anonymous graphs of arbitrary structure.

Several specific solutions to the election problem already exist. The interested reader is referred to [2] and [11].
One group of protocols uses randomisation - an example of such a solution can be found in [3]. Another group of solutions is based on the assumption that the underlying network’s nodes have unique and linearly ordered identities.

All the solutions for anonymous networks assume specific structure of the network and include protocols for complete graphs and trees [1], graphs called mosaics [7] and covering-minimal graphs with a priori known size [8, 9].

Generally, all solutions - disregarding those using randomisation - use some global knowledge about the underlying graph (e.g. unique comparable identities, size, specific structure). Thus, they can be seen as transferring one kind of global knowledge to another by local means. We are interested in a purely local solution that really computes some global knowledge strictly from local information. Thus, we choose a model with no global assumptions made - anonymous graphs with size and structure unknown to the local protocol.

The authors of [4] present necessary and sufficient conditions for families of graphs, in which distributed election is possible. This work seems to end any further discussion. According to the authors some global knowledge is essential.

We show that a slight change of the assumptions about locality makes any global knowledge unnecessary. Let us see what assumptions of the authors of [4] are:

“Graph relabelling systems and more generally local computations satisfy the following constraints which seem to be natural when describing distributed computations with a decentralized control:

(C1) they do not change the underlying graph but only the labelling of its components (edges and/or vertices), the final labelling being the result of the computation,

(C2) they are local, that is, each relabelling step changes only a connected subgraph of a fixed size in the underlying graph,

(C3) they are locally generated, that is, the application condition of the relabelling only depends on the local context of the relabelled graph.”

Our research reported in [12, 13] shows that breaking some of the above postulates leads to a nontrivial extension of the power of computation, however, the locality of the presented protocols is questionable. This paper is a third attempt and we claim that the very idea of the local approach is now preserved.

Although C2 is quite a traditional assumption, we believe that the fixed size of the area of a single computation is not an essence of a distributed computing. Notice that many distributed algorithms consist in gathering the global knowledge by local means. This includes the knowledge about the graph’s structure. Why not use the once recognised subgraphs as new regions for subsequent distributed steps?

Typical locality regions are balls of radius 1 (a node with its neighbours). Our idea is that in addition to performing a step of some protocol, a node draws forward links between its neighbours, thus merges its surrounding balls into one. This process preserves what we regard as the essence a local computation - it starts with small local units while the merging steps act in small subgraphs, too.

Our result is a discussion with [4]. For that reason we use the same formalism - the graph relabelling system. We introduce an election protocol that works for every connected network without any global assumptions. It satisfies C1, C3, and partially C2, without the restriction of fixed size/diameter of the locality region. Without this restriction the election is possible in any connected graph.

\[3\text{In fact, the authors probably mean subgraphs with fixed diameter or balls with fixed radius, rather than size.}\]
3. Preliminaries

Standard mathematical notation (including graph theory terms) is used through all the paper. \( N \) is the set of all natural numbers with 0. We consider connected networks (graphs) with reliable communications. The class of all connected graphs is denoted as \( \mathcal{G} \), while \( \mathcal{G}_L \) stands for the class of all connected \( L \)-labelled graphs. Labelled graphs are denoted with bold fonts, newly defined terms are in bold italics. We assume that a process running on a node \( v \in V \) can distinguish between \( v \)'s neighbours, thus also enumerate them using some order. We denote this order by \( <_v \).

3.1. Graph relabelling systems

We define a local computation in such a way that it satisfies postulates C1, C2 and C3, then we modify the definition along with our suggestions presented in Section 2.

Let \( R_H \subseteq \mathcal{G}_L \times \mathcal{G}_L \) denote a relabelling relation such that if \((G, G') \in R_H\), then \( R_H \) modifies labels only in some connected subgraph \( H \subseteq G \), using only the knowledge stored in \( H \). \( H \) is called a locality region of \( R_H \). Let \( R \) be a relation satisfying: \((G, G') \in R \) iff there exists \( H \subseteq G \) such that \((G, G') \in R_H \). The sequence \( G_1 R_H G_2 R_H \ldots \) is called \( R \)-relabelling chain. By \( R^* \) we mean the reflexive and transitive closure of \( R \). \( R \) is called noetherian, if all its relabelling chains are finite.

Definition 3.1. (local transformation)
\( R \subseteq \mathcal{G}_L \times \mathcal{G}_L \) is a local transformation iff there exists such \( k \in N \) that for each \( R \)-relabelling chain \( G_1 R_H G_2 R_H \ldots \) and for all \( i \in N \), \( H_i \) are connected subgraphs satisfying:
\[
H_i \subseteq B_G(v_i, k)
\]
for some \( v_i \in V \), where \( G_i \in \mathcal{G}_L \), \( G_i = (G, \lambda_i) \), \( \lambda_i \) is an \( L \)-labelling, and \( G = (V, E) \) is a graph.

Definition 3.2. (semi-local transformation)
\( R \subseteq \mathcal{G}_L \times \mathcal{G}_L \) is a semi-local transformation iff there exists such \( k \in N \) that for each \( R \)-relabelling chain \( G_1 R_H G_2 R_H \ldots \) and for all \( i \in N \), \( H_i \) are connected subgraphs satisfying:
\[
H_i \subseteq B_G(v_i, k), \text{ or } H_i \subseteq H_1 \cup \ldots \cup H_{i-1} \cup B_G(v_i, k),
\]
for some \( v_i \in V \), where \( G_i \in \mathcal{G}_L \), \( G_i = (G, \lambda_i) \), \( \lambda_i \) is an \( L \)-labelling, and \( G = (V, E) \) is a graph.

If \( R \) is a local (semi-local, respectively) transformation, then \( R^* \) is called a local (semi-local) protocol / algorithm and any \( R \)-relabelling chain is a run of the protocol.

\footnote{The relation \( <_v \) is a local order, defined only in the neighbourhood of \( v \). This assumption is quite natural - in practice computer data is never organised in unordered sets, the basic reason being that it is kept in memory addressed by natural numbers.}

\footnote{No such \( k \) exists for global transformation, because the graphs \( G_1, G_2, \ldots \) can be of arbitrary size. This remark concerns both definitions.
4. The Algorithm

In this section we define our election algorithm. We start with named forward links - the very idea that lies behind our solution. We define the forward links algebra, formulate our algorithm and show its properties. Then we represent forward links by graph labelling and replace the forward algebra with a graph relabelling system.

4.1. Named forward links

Let $G \in \mathcal{G}$, $G = (V,E)$. Let $P(G)$ be a set of all finite paths in $G$.

We define a binary equivalence relation $\equiv$ such that for every $p, q \in P(G)$:

$$ p \equiv q \iff q = p \lor [p = (v_0, \ldots, v_k) \land q = (v_k, \ldots, v_0)], \text{for some } v_0, \ldots, v_k \in V. $$

For each $p \in P(G)$, the equivalence class of $p$ in the $\equiv$ relation is denoted as $[p]$.

Let us define the set $F(G) := \{(a, [q]) : a \in N - \{0\}, q \in P(G)\}$. The elements of $F(G)$ are called named forward links (forwards or f-links for short). Intuitively, a forward is a labelled path without the sense of direction.

For every $f \in F(G), f = (a, [q])$, we define its name $n(f) = a$ and base path $p(f) = [q]$. We also say that $f$ is based on $q$. If $q = (v_0, \ldots, v_k)$, for some $v_0, \ldots, v_k \in V$, we say that $v_0$ and $v_k$ are endpoints of $f$ or that $f$ ends in $v_0$ and $v_k$. We also say that $f$ crosses the nodes $\{v_1, \ldots, v_{k-1}\}$. The set of endpoints of $f$ is denoted as $V \perp (f)$, while the set of nodes crossed by $f$ is denoted as $V \rightarrow (f)$. The set $V(f) := \{v_0, \ldots, v_k\}$ is the set of nodes of $f$. We also say that $f$ is incident with all its nodes. Two forwards are incident with each other iff there exists a node incident with both of them (called a common node).

A graph with locally identifiable forward links (an f-graph for short) is a pair $(G, F)$ such that:

(i) $G \in \mathcal{G}$,
(ii) $F \subseteq F(G)$,
(iii) $(f, g \in F, p(f) \neq p(g), f$ is incident with $g) \Rightarrow (n(f) \neq n(g))$

The class of all f-graphs is denoted as $\mathcal{G}_F$.

Graphically, we represent forward links with straight (breaking at the graph nodes) bold gray lines, drawn along their base paths, with names in square tags. The sample f-graph is depicted below. It has two incident forward links named 4 and 7.

![Figure 1. The forward links named 4 and 7. The common node is pointed out](image)

Let $(G, F)$ be an f-graph. We get the following corollary directly from (iii):
Corollary 4.1. If a node \( v \in V \) is incident with some forward links \( f \) and \( g \) having the same name \( n \), then \( f = g \).

Notice that according to Corollary 4.1 the name is a unique local identifier of an f-link. This is why we call the forward links in an f-graph locally identifiable.

4.1.1. Forward links algebra

The forward links can be created, extended, merged, cloned or removed.

The creation of a forward link from a triple of different nodes \( x, y, z \in V \) such that \( \{x, y\} \in E \) and \( \{y, z\} \in E \) is an operation, whose result is a forward link \( f = (a, [(x, y, z)]) \) such that there is no other f-link named \( a \) incident with \( x, y \) or \( z \).

![Figure 2: The forward link named 1 is created from nodes pointed out with the arrows](image)

An extension of a forward link \( f = (a, [(v_0, \ldots, v_n)]) \) by a node \( v_{n+1} \in V \) (such that \( \{v_n, v_{n+1}\} \in E \)) is a forward link \( g = (b, [(v_0, \ldots, v_n, v_{n+1})]) \) such that \( a \neq b \) and there is no other forward named \( b \) incident with \( f \) or \( v_{n+1} \).

![Figure 3: The forward link named 1 is extended by a node that is pointed out](image)

A merge of an f-link \( f = (a, [(v_0, \ldots, v_n)]) \) with a different f-link \( g = (b, [(w_0, w_1, \ldots, w_j)]) \) is possible iff \( f \) and \( g \) have a common endpoint, say \( v_n = w_0 \). The result is a forward link \( h = (c, [(v_0, \ldots, v_n, w_1, \ldots, w_j)]) \) such \( a \neq c \), \( b \neq c \) \(^6\) and there is no other forward named \( c \) incident with \( f \) or \( g \). For an example, see the picture below.

![Figure 4: Forward named 1 is merged with forward 2 and the result is forward named 3](image)

A clone of a forward link \( f = (a, [(v_0, \ldots, v_n)]) \) is a forward link \( g = (b, [(v_0, \ldots, v_n)]) \) such that \( a \neq b \) and there is no other forward named \( b \) incident with \( f \).

\(^6\)Note that \( a \neq b \) by Corollary 4.1.
In result of the removal of a forward $f \in F$ from $(G, F)$ the f-graph becomes a pair $(G, F - \{f\})$. The figure below shows an example.

Below we extend some terms from graph theory to f-graphs.

Two different nodes $v, w \in V$ are said to be f-adjacent (or f-neighbours) iff they are adjacent or there exists a forward $f \in F$ such that $v$ and $w$ are endpoints of $f$. The set of f-neighbours of node $v \in V$ is its f-neighbourhood, denoted as $f-N_{(G,F)}(v)$. The union of the sets of nodes of all forwards ending in $v$ and $B_G(v)$ is called the f-ball centred at $v$ and is denoted as $f-B_{(G,F)}(v)$. The subscripts are omitted when no confusion is possible.

Any sequence of nodes $v_0, \ldots, v_k \in V$ such that for all $i \in \{0, \ldots, k - 1\}$ $v_i$ is f-adjacent to $v_{i+1}$, is called f-path joining $v_0$ with $v_k$. The set of nodes $W \subseteq V$ is f-connected iff for all $v, w \in V$ there exists f-path joining $v$ with $w$.

Once we have defined f-graphs algebra, we can formulate our algorithm.

4.2. The Election Protocol Defined in Terms of Forward Link Algebra

Let $(G, F)$ be an f-graph such that $F = \emptyset$.

Let us introduce the set of tags $T := \{\text{active}, \text{resigned}, \text{leader}\}$. $T$ is fixed from now on. Any function $t : V \rightarrow T$ is called a tagging function. If for some $v \in V$ and $t_0 \in T$ we have $t(v) = t_0$, we say that $v$ is tagged $t_0$ (or $v$ is $t_0$, for short).

Consider the uniform tagging $t^I$, in which all nodes are tagged active, and a family of taggings $\{t^F_v\}_{v \in V}$ such that for each $v \in V$ exactly one node $v$ is tagged leader and all the other ones are tagged resigned. Each tagging $t^F_v$ is a solution to the election problem, $v$ being the elected node. The tagging $t^I$ is the starting point of our algorithm. Our algorithm’s task is to stop in such a configuration that the tagging belongs to the $\{t^F_v\}_{v \in V}$ family.

The general idea of the protocol is very simple: the active nodes resign one by one and the last one becomes the leader. However, in an anonymous graph the fact of being the only one active node would

---

Although tagging is a kind of labelling, at this point we avoid the term ‘labelling’. The actual labelling will be defined further and will combine tagging and encoding of forward links together, using structured labels.
be impossible to be detected locally. Our protocol ensures that the set of active nodes is f-connected during the whole run and before any tagging is changed, all cyclic f-links are removed. Thus, detection of being the last active node is trivial - the last active node has no f-neighbours.

All nodes perform actions, which are listed below.

(A1) Cycle detection and removal
If an active node is an endpoint of two forward links with the same name \( a \in N \), it determines that \( a \) is a name of a cycle (by Corollary 4.1) and removes it. After performing the action it remains active.

(A2) Connecting active neighbours using forward links and resignation
This is the most intricate operation, performed by an active node that has at least one active neighbour or is an endpoint of at least one forward (if any of them is cyclic, A2 is not enabled). This action has three variants:

(A2.1) the node has only one active neighbour and is not an endpoint of any f-link:
\[ \rightarrow \text{the node becomes resigned}, \]

(A2.2) the node is an endpoint of exactly one f-link and has no active neighbours:
\[ \rightarrow \text{the node removes the f-link in question and becomes resigned}, \]

(A2.3) all other cases:
\[ \text{(the general idea is to assure the f-connectedness of the set of active neighbours and f-neighbours; this can be achieved in many ways; we choose the routine described below)} \]
\[ \rightarrow \text{let the node performing the action be called } v \]
\[ \circ \text{using the lexicographical order combined of } <_v \text{ and the order of forward names, edge by edge, forward by forward}\text{, the node } v \text{ makes a list of active neighbours and forward links ending in it;} \]
\[ \text{let this list be denoted as } (X_0, X_1, \ldots, X_n), n \in N, \text{ for each } i \in \{0, \ldots, n\} \text{ } X_i \text{ being either } v \text{'s neighbour or a forward link ending in } v, \]
\[ \circ \text{then } v \text{ processes the list and for each } i \in \{1, \ldots, n-1\} \text{, if } X_i \text{ is a forward link, it is cloned and the clone is inserted to the list immediately before } X_i; \]
\[ \text{the result list is } (X_0, X'_1, X_1, \ldots, X'_{n-1}, X_{n-1}, X_n), \text{ the primed symbols denote the clones or duplicates created in this substep,} \]
\[ \circ \text{now, } v \text{ processes the new list; this time, it considers pairs:} \]
\[ (X_0, X'_1), (X_1, X'_2), \ldots, (X_{n-2}, X'_{n-1}), (X_{n-1}, X_n); \]
\[ \text{each pair is of a form } (X, Y); \]
\[ \circ \text{- if both } X \text{ and } Y \text{ are nodes, } v \text{ creates a new forward link from nodes } X, v, Y; \]
\[ \circ \text{- if one element is a node, and the other a forward, } v \text{ extends the forward by the node.} \]
\[ \circ \text{- if both elements are forwards, } v \text{ merges them into one,} \]
\[ \circ \text{after that } v \text{ becomes resigned.} \]

\( ^8 \)The order is well-defined, because no cyclic f-link is incident with \( v \).

\( ^9 \)The first and the last element need not be cloned or duplicated, so they are not considered.

\( ^{10} \)Note that \( X, v, Y \) are pairwise different.
(A3) Taking a leadership

If an active node has no active f-neighbours (this includes neighbours), it becomes the leader.

Examples of the above described actions are depicted in Figures 7 - 11. The active nodes are white, the resigned ones are grey and the leader is black. The node performing the action is pointed with an arrow.

Figure 7. (A1) Cyclic forward link 2 is removed

Figure 8. (A2.1) Active node with one active neighbour resigns

Figure 9. (A2.2) Active node is an endpoint of one forward. It removes the forward and resigns

Figure 10. (A2.3) Active node is an endpoint of two forwards and has two active neighbours. Neighbours and links are considered clockwise, starting from the leftmost node in the picture. The picture does not show an intermediate step - cloning of the forward link 7. The clone gets name 1 (not shown), and then is extended by a node (the result is named 4)

Figure 11. (A3) An isolated active node becomes the leader
Any sequence of f-graphs and taggings starting from \((G, \emptyset, tI)\) and produced by repetitive application of the above actions is called a \textit{run}. Let \(\sigma\) be an arbitrary run, fixed till the end of this section.

Observe that each action changes tagging or f-links only in an f-ball centred in some node \(v\) that is performing the action. For every pair of consecutive elements \((\sigma_i, \sigma_{i+1})\) of \(\sigma\) we call such \(v\) the \textit{performer node} and denote it by \(v(\sigma_i, \sigma_{i+1})\).

We get the following results about \(\sigma\).

**Lemma 4.1. (stop property and complexity of the algorithm)**
The run \(\sigma\) is a finite sequence. Its length is bounded by \(n^3\), where \(n = |G|\).

**Proof:**
(sketch\(^{11}\)) The number of \(A_2\) and \(A_3\) transition is bounded by \(n\), the count of \(A_1\) transition is bounded by the number of forwards that can be produced by \(A_2\) actions, namely \((n^3 - n)/2\), which is bounded by \(n^3 - n\). \(\square\)

**Lemma 4.2. (active nodes are endpoints)**
In every f-graph in the run \(\sigma\), each endpoint of every forward link is active and each node the f-link crosses is resigned (in other words, if incident with any forwards, the active nodes can only be endpoints of forward links, whereas, the resigned ones can only be crossed by them).

**Proof:**
(sketch) We observe that this is true in the initial configuration and that while resigning \((A_2)\), the node stops being an endpoint (if it were one) and either it becomes crossed or no forwards are created. \(A_1\) and \(A_3\) actions are not significant because they do not produce f-links. \(\square\)

**Lemma 4.3. (active nodes are f-connected)**
In every f-graph in a sequence \(\sigma\) the set of active nodes is f-connected.

**Proof:**
(sketch) We notice that this is true in the initial f-graph. The \(A_1\) actions have no effect to the f-connectedness because they remove cyclic f-links that cross (by Lemma 4.2) only nodes that are not active. \(A_2\) transitions preserve f-connectedness of the active f-neighbours of the node that becomes resigned, and \(A_3\) actions do not modify any f-links. \(\square\)

**Theorem 4.1. (\(\sigma\) solves the election problem)**
The sequence of \(A\) transformations used in \(\sigma\) solves the election problem for \((G, F)\).

**Proof:**
(sketch) By Lemma 4.1, \(\sigma\) is finite. The number of active nodes decreases, so there is such a configuration that contains only one active node. Because of f-connectedness of the set of active nodes, this is the only configuration in which \(A_3\) can be executed (and no other action can). After executing \(A_3\) we have exactly one leader, all the other nodes are resigned and no further transition is possible. \(\square\)

Now we present the algorithm in terms of relabelling systems.

\(^{11}\)The full versions of the proofs in this section can be found in [14].
4.3. The election algorithm in terms of graph relabelling systems

First we start with defining the set of structured labels $L$, then we define the step relation $R$ on $G_L$, and finally we show that $R^*$ is a semi-local protocol that solves the election problem.

4.3.1. The label encoding

Now we define the set of labels that includes all information used by the algorithm defined in Section 4.2. Let $(G, F)$ be an $f$-graph, $t$ be a tagging, $v \in V, f \in F$, $p(f) = [(v_0, \ldots, v_k)], k \in N$. Then:

- the local edge numbering function for $v$ is the bijection
  $\text{edge}_v : \{1, \ldots, \sigma_G(v)\} \to \{e \in E : e = \{v, w\}, w \in V\}$, such that if $i', i'' \in N, e', e'' \in E$ are incident with $v$, $\text{edge}_v(i') = e'$ and $\text{edge}_v(i'') = e''$, then $i' < i'' \iff e' <_v e''$.

- the number of local instances of $f$ at $v$ is a value $\text{num}_v(f) := |\{i \in N : v = v_i\}|$,

- the set of labels $L$ is defined as follows: $L := (T \cup \{\text{none}\}) \times N^6$.

The label encoding of $((G, F), t)$ is any graph $G \in G_L$, $G = (G, \lambda)$ such that for each edge $e \in E$ $\lambda(e) = (\text{none}, \emptyset)$, and for each node $v \in V \lambda(v) = (t_v, F_v)$, where $t_v = t(v)$, and $F_v \subset N^6$ is a set of tuples $(i : a; se(si) - ge(gi))$, such that $(i : a; se(si) - ge(gi)) \in F_v$ iff $\exists g \in F$, satisfying:

- $i \in \{1, \ldots, \text{num}_v(g)\}$,
- $a = n(g)$,
- $|\{(i : a; b(c) - d(e)) \in F_v : b, c, d, e \in N\}| = 1$ (each instance of $g$ occurs once in $F_v$),
- $(se, si) < (ge, gi)$ in lexicographical order,
- $(se, si) = (0, 0)$ iff $v$ is an endpoint of $g$ (see Figure 12),
- if $(se, si) \neq (0, 0)$, then $se \neq 0, si \neq 0$, and $\exists w \in V, p \in N$ such that:
  - $\{v, w\} \in E$,
  - $\text{edge}_w(se) = \{v, w\}$,
  - $g$ is incident with $w$,
  - $si \leq \text{num}_w(g)$,
  - $\text{edge}_w(p) = \{v, w\}$,
  - $(si : a; p(i) - b(c)) \in F_w$ or $(si : a; b(c) - p(i)) \in F_w$, for some $b, c \in N$ (see Figure 13),

- $ge \neq 0, gi \neq 0$, and $\exists x \in V, q \in N$ such that:
  - $\{v, x\} \in E$,
  - $\text{edge}_v(ge) = \{v, x\}$,
  - $g$ is incident with $x$,
  - $gi \leq \text{num}_x(g)$,
  - $\text{edge}_x(q) = \{v, x\}$,
  - $(gi : a; p(i) - d(e)) \in F_x$ or $(gi : a; d(e) - p(i)) \in F_x$, for some $d, e \in N$ (see Figures 12 and 13).

The set of label encodings of $((G, F), t)$ is denoted as $G((G, F), t)$.

Figures 12 and 13 illustrate the definition of label encoding. All symbols used in the definition are preserved.
Remarks

- We use labelling of nodes only; the labels of edges are not needed, thus are all set to \((\text{none}, \emptyset)\).
- The number of encodings of \(((G, F), t)\) is equal to the number of different local instance numberings of forwards at \(G\)'s nodes. The encoding is unique iff every node is incident with at most one instance of any f-link.

Thus, for every configuration \(((G, F), t)\) we have a set \(G((G, F), t) \subseteq G_L\) such that for each \(G \in G((G, F), t)\), the \(L\)-labelling \(\lambda\) contains all the information needed to restore \(F\) and \(t\).

4.3.2. The algorithm

Take any run \(\sigma\).

\[
\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_k, \ldots) = (((G, F_0), t_0), ((G, F_1), t_1), \ldots, ((G, F_k), t_k), \ldots), k \in \mathbb{N},
\]

Let \(\sigma\) be an arbitrary sequence of \(L\)-labelled graphs:

\[
\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_k, \ldots) = (G_0, G_1, \ldots, G_k, \ldots)
\]

such that for all \(i \in \mathbb{N}\)

- \(G_i \in G(\sigma_i)\)
- if \((\sigma_i, \sigma_{i+1})\) is a pair of consecutive items of \(\sigma\) and if \(v = v(\sigma_i, \sigma_{i+1}), v \in V, G = (V, E)\), then the labelling of \(G_i\) and \(G_{i+1}\) is the same for all nodes outside the f-ball centred in \(v\) in \((G, F_i)\)\(^{12}\).

and let \(R \subseteq G_L \times G_L\) be a relation such that

\[
G_0RG_1R \ldots RG_KR \ldots
\]

Then \(R^*\) denotes a semi-local protocol that solves the election problem.

The correctness of the definition of \(R^*\) is expressed in the following theorem.

\(^{12}\)This restriction is possible, because the actions A1 \(\ldots\) A3 change taggings and f-links only in f-ball centred in \(v\); thanks to this restriction we get the semi-locality of \(R\).
Theorem 4.2. (correctness of $R^*$)

$R^*$ is a semi-local election protocol.

Proof:

(sketch\textsuperscript{13}) The fact that $R$ is noetherian and solves the election problem is straightforward from Theorem 4.1. Semi-locality of $R$ (with $k = 1$) follows from the conditions $A_1 \ldots A_3$ - each transformation is local and locally generated in some f-ball, the f-balls are initially balls of radius 1 and then they are extended either by balls of radius 1 or by other f-balls (i.e. previously defined locality regions). \hfill $\Box$

5. The Power of Semi-local Computations

As stated in Section 2, the power of local computations in the networks with a distinguished leader is identical to the power of fully global protocols. In the semi-local computation model we succeeded to solve the election problem, so the semi-local computations have the same possibilities as global algorithms. This is expressed in the following theorem.

Theorem 5.1. (semi-local computations are as powerful as global ones)

Let $L$ be a set of labels and let $G, H \in G_L$.

For every (global) relabelling transformation $T$ there exists a graph relabelling relation $R$ such that:

- $R^*$ is a semi-local protocol,
- $(G, H) \in R^* \iff (G', H') \in T$, for every $G', H' \in G_L$ such that $G \cong G, H \cong H'$.

In other words, every (global) relabelling transformation (or algorithm in general) can be simulated by means of semi-local computations. By global relabelling transformations we mean the ones that have no restrictions about their scope.

Proof:

(sketch\textsuperscript{14}) We are designing a semi-local relabelling protocol $R^*$ that simulates application of $T$ in $G$.

In the initial labelling of $G$, each label contains the following information:

- the label $l$ that is to be transformed by $T$ to label $m$,
- some uniform encoding of $T$,
- some uniform encoding of tags and f-links to be used by the election protocol, the initial values are as defined for that protocol,
- the identifier, zero at the beginning,
- the stage indicator $s \in \{\text{started, enumerating, enumerated, finished}\}$, initially $s := \text{started}$,
- the model of a graph - initially empty.

We describe the algorithm of each node, depending on its stage and tag.

$\rightarrow$ For each node with $s = \text{started}$ the election protocol rules are applied. In addition, whenever a node is tagged $\text{resigned or leader}$, its stage changes to $\text{enumerating}$.

\textsuperscript{13}The full version of the proof can be found in [14].

\textsuperscript{14}Due to expected complexity, a full version of the proof will be presented in a separate paper.
Among all nodes with stage \( s = \text{enumerating} \), only the node tagged leader gets the active role - it sets its identifier to 1 and initiates distributed DFS (or BFS) protocol that ends with all the other nodes enumerated (this means that their identifiers are higher than 1 and unique), having uniform model reflecting the graph’s structure (including identifiers, links and the ‘l’ labels), and having stage set to enumerated. When this is accomplished, the model of the graph at the leader node is updated and its stage changes to enumerated, too. All the resigned nodes having stage \( s = \text{enumerating} \) act only as passive relays transmitting knowledge to and from the leader.

This active-passive schema is maintained for all nodes with stage \( s = \text{enumerated} \). The leader node starts with analysing the model and finding isomorphism between \( G \) and \( G' \) in order to apply \( T \). Then, using the encoding of \( T \) and the calculated isomorphism, it determines which nodes (identifiers) should get specific result ‘m’ labels. After that, it initiates the distributed message passing, and passes calculated labels to nodes with specific identifiers. The role of a resigned node is to transmit the messages to and from the leader node or to accept them (if they are recipients of the message). On acceptance, a recipient node changes its label \( l \) according to the received message, changes its stage \( s \) to finished and sends a confirmation message to the leader. In the end, on receiving confirmations from all resigned nodes, the leader changes its label \( l \) according to the previous calculation and sets its stage \( s \) to finished. The algorithm stops.

The presented algorithm \( R^* \) is semi-local, because it behaves semi-locally in the started stage, and fully locally in the next stages. The result ‘m’ labels are computed according to transformation \( T \). \( \square \)

6. Conclusions

In the semi-local model the static structure of locality regions is replaced by dynamic regions that expand step by step as the semi-local protocol advances.

The semi-local computations have the following advantages:

- they start with classical locality regions, e.g. with balls of radius 1,
- the locality regions are expanded by means of local methods,
- the power of semi-local computations is equal to the power of global computations.

This means that the defined model is still conforming to the intuitive meaning of local computations and, on the other hand, it has no limitations present in the strictly local model. It is clear that the limitations were caused by static structure of locality regions. Thanks to this model we could define a universal election algorithm that is not using any a priori global knowledge. In the algorithm, forward links were used for keeping information about growing locality regions.

Further research might include simplifying the algorithm (maybe some other simpler formalism might be found). It is also tempting to make the algorithm self-stabilising.

The bound on the algorithm’s complexity presented here is very rough and is based on the number of (possibly complex) steps. We have recently implemented the algorithm in order to check its practical behaviour and identify the real basic operations that have greatest impact on the computation time. The first results of experiments are encouraging - the number of steps seems to be \( O(|E|) \), where \( |E| \) denotes the number of network links. We are going to investigate the results in detail in a separate paper.
References


