Distributed Graph Traversals by Relabelling Systems with Applications

Bilel Derbel [1]

LaBRI, Université Bordeaux 1
351, cours de la libération
33405 Bordeaux, France

Mohamed Mosbah [2]

LaBRI, Université Bordeaux 1
351, cours de la libération
33405 Bordeaux, France

Abstract

Graph traversals are in the basis of many distributed algorithms. In this paper, we use graph relabelling systems to encode two basic graph traversals which are the broadcast and the convergecast. This encoding allows us to derive formal, modular and simple encoding for many distributed graph algorithms. We illustrate this method by investigating the distributed computation of a breadth-first spanning tree and the distributed computation of a minimum spanning tree. Our formalism allows to focus on the correctness of a distributed algorithm rather than on the implementation and the communication details.

Key words: Distributed algorithms, Graph traversals, Relabelling systems.

1 Introduction

1.1 Motivation and contribution

Distributed algorithms are designed to run on networks consisting of interconnected autonomous entities of computations (processes) cooperating to solve given problems. Many of these algorithms, mainly dealing with the traversal of the network, appear as the compositions of some basic tasks. These basic tasks include the

[1] Email: `derbel@labri.fr`
[2] Email: `mosbah@labri.fr`

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

broadcasting or the propagation of information and the echo or convergecast algorithm [4,15,10,14]. For instance, in the message-passing scheme, a distributed computation of a spanning tree can be performed by broadcasting a message from an initial node to all other nodes of the graph; each node propagates the message to its neighbors upon receiving it. This simple algorithm may be described in some other way depending on the distributed model. In fact, in the distributed setting, many distributed algorithms are inherently dependent on the model one considers i.e., message-passing, shared memory, synchronous, asynchronous etc. A distributed algorithm which is designed and implemented in a given model becomes in general obsolete in another model. Even though it is possible, one has often to re-adapt or to re-encode the algorithm depending on the model specification.

In this context, graph relabelling systems and local computations [8,7,9] can be viewed as a tool which allows to encode distributed algorithms in a formal and unified way. In fact, a graph relabelling system is based on a set of relabelling rules which are executed locally and concurrently. These rules are described using mathematical and logic formulas which enables to derive formal and rigorous mathematical proofs of their correctness and by the same way to prove the correctness of an algorithm on a distributed system.

In this paper, we are interested in a high level encoding of some basic *Wave and graph traversal algorithms* which are in the basis of many sophisticated distributed algorithms. In particular, we show that by expressing the broadcast and the convergecast by graph relabelling systems, a large class of graph traversals can in turn be expressed by graph relabelling systems. The high-level encoding of such algorithms in form of graph relabelling systems allows to encode them and to prove them in a unified and simple way. Furthermore, we show that it is possible to combine these two subroutines to give a formal encoding for some basic applications which illustrate our approach.

First, we show how to encode the classical distributed layered breadth-first spanning (BFS for short) construction [5,15,13]. This algorithm involves the encoding of many iterations of a classical technique in distributed computing which is known as the “Propagation of Information with Feedback” (PIF for short) [14]. Even though our encoding is given in the special case of the BFS tree construction, it gives a general idea about how to design sophisticated algorithms based on the PIF technique.

Second, by using the convergecast as a building block, we give a general method to encode with relabelling system distributed algorithms for computing some particular (commutative and associative) global functions.

These two basic applications are then combined to derive the graph relabelling system that encodes the classical Prim’s distributed algorithm for computing a minimum spanning tree (MST for short) [5,15,13]. This example aims to show how to combine the basic graph traversals we have encoded in order to obtain a simple and formal encoding of more advanced algorithms.

1.2 Graph model and notations

In this section, we illustrate, in an intuitive way, the notion of graph relabelling systems by showing how some algorithms on networks of processors may be encoded within this framework [9]. As usual, such a network is represented by a graph $G = (V, E)$ whose nodes stand for processors and edges for (bidirectional) links between processors. We only consider undirected connected graphs without multiple edges and self-loops. At every time, each node and each edge is in some particular state encoded by a node or an edge label. According to its own state and to the states of its neighbors, each node may decide to realize an elementary *computation step*. After this step, the states of this node, of its neighbors and of the corresponding edges may have changed according to some specific *computation rules*. Let us recall that graph relabelling systems satisfy the following requirements:

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

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

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

A precise description and definition of local computations can be found in [7]. We recall here only the description of local computations and we explain the convention under which we will describe graph relabelling systems later. A relabelling system is a triple $\mathcal{R} = (\mathcal{L}, \mathcal{I}, \mathcal{P})$ where $\mathcal{L} = \mathcal{L}_v \cup \mathcal{L}_e$ a set of labels, $\mathcal{L}_v$ a set of node labels, $\mathcal{L}_v$ a set of edge labels, $\mathcal{I}$ a subset of $\mathcal{L}$ called the set of initial labels and P a finite set of relabelling rules.

For each relabelling rule, we will consider a generic star-graph of generic center v_0 and of generic set of nodes $B(v_0, 1)$ (star of radius 1 centered at v_0) and we will refer to a node $v \neq v_0$ of the star graph by writing $v \in B(v_0, 1)$. Within these conventions, each relabelling rule is described by its precondition and relabelling. If $\lambda(v)$ is the label of v in the precondition, then $\lambda'(v)$ will be its label in the relabelling. We will omit in the relabelling the description of labels that are not modified by the rule. This means that if $\lambda(v)$ is a label such that $\lambda'(v)$ is not explicitly described in the rule for a given v, then $\lambda'(v) = \lambda(v)$. The label of a node can be composed of k components (with k a given integer). In this case, we denote by $\mathcal{L}_v = \{L_1 \times L_2 \times ... \times L_k\}$ the set of labels where L_i ($1 \leq i \leq k$) is a set of possible values of the i^{th} component. For every node v, we denote by $\lambda(v).L_i$ the i^{th} component of the label of v. We adopt the same notations for edge labels. For instance, consider a node $v \in B(v_0, 1)$, then $\lambda(v_0, v)$ refers to the labels of the edge $e = (v_0, v)$ in the precondition and $\lambda'(v_0, v)$ will be its label in the relabelling. The preconditions and the relabelling are written as logic formulas. We use the logic symbols $\wedge$, $\vee$, $\exists$, $\exists!$ and $\forall$ to denote respectively the logic operators "and", "or", "it exist", "it exists a unique" and "for all". In the case of a weighted graph

$G_{\mathcal{W}}$, if $e = (u, v)$ is an edge then we will denote by $\mathcal{W}(u, v)$ the weight of e.

1.3 Summary

The paper is organized as follows. In Section 2, we give an encoding of the broadcast and convergecast process using graph relabelling systems. In Section 3, we give two basic applications which are the distributed layered construction of a BFS tree and the computation of global functions. As a combination of these two applications, in Section 4, we give a formal and detailed algorithm for computing a minimum spanning tree. Finally, in Section 5, we give some concluding remarks.

2 Building blocks

Many basic distributed algorithms can be described as a combination of many couples of broadcast and convergecast. The broadcast is usually used to deliver a given information (*e.g*: the value of a variable, the beginning of a new step in the algorithm) to all the nodes of the network. The convergecast is in general used to collect some information into one single node. This information is for example used to activate some treatment. In the next two subsections, we give the relabelling systems corresponding to the broadcast and convergecast operations. In this section, we do not care about the information to be broadcast or collected. We only give the intuitive method to do it using relabelling systems.

2.1 The broadcast technique

The broadcast operation can be defined as *the dissemination of some information from a source node to all nodes in the network*. It can be encoded with the relabelling system $\mathcal{R}_b = (\mathcal{L}_b, \mathcal{I}_b, \mathcal{P}_b)$ defined by: $\mathcal{L}_b = \{E\} \cup \{0, 1\}$, where $E \in \{A, S, O\}$, $\mathcal{I}_b = \{A, O\} \cup \{0\}$ and $\mathcal{P}_b = \{R_b^1, R_b^2\}$. Initially, one source node from whom the broadcast is initiated is labelled A and all other nodes are labelled O. All the edges of the graph are initially labelled 0. The label S encodes the fact that the broadcast process has reached some node.

R_b^1 : **Broadcast : initial step**

<u>*Precondition :*</u>
- $\lambda(v_0).E = A$

<u>*Relabelling :*</u>
- $\forall v \in B(v_0, 1)\ (\lambda(v).E = O \implies (\lambda'(v) := S, \lambda'(v, v_0) := 1))$

R_b^2 : **Broadcast**

<u>*Precondition :*</u>
- $\lambda(v_0).E = S$
- $\exists v \in B(v_0, 1)(\lambda(v).E = O)$

<u>*Relabelling :*</u>
- $\forall v \in B(v_0, 1)\ (\lambda(v).E = O \implies (\lambda'(v) := S, \lambda'(v, v_0) := 1))$

Rule R_b^1 (resp. R_b^2) can be applied as follows. If a star center v_0 is labelled A (resp. S) then for each node v in the star $B(v_0, 1)$, if v is labelled O then it becomes labelled S and the edge (v_0, v) becomes labelled 1. Note that as a basic application of the relabelling system $\mathcal{R}_b$, once the broadcast is terminated, we obtain a spanning tree by considering the edges with labels 1. Figure 1 shows an example of the broadcast algorithm using the relabelling system $\mathcal{R}_b$. Note that the rules can be applied by many nodes on distinct balls as far as their precondition states are at the same time satisfied.

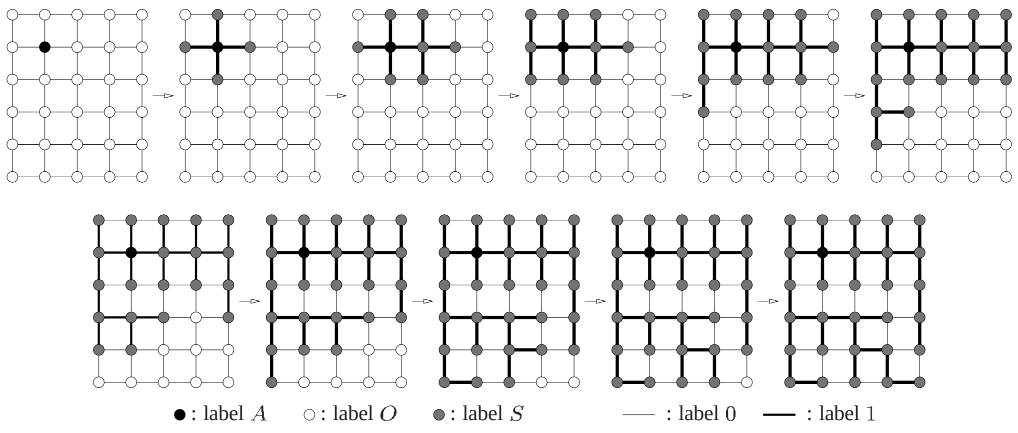

Fig. 1. An example of a broadcast using the relabelling system $\mathcal{R}_b$.

2.2 *The convergecast technique*

The convergecast operation consists in *collecting information upwards on a tree*. The most fundamental example is to let a source node, that has broadcast some information, detects that the broadcast has terminated. In fact, in order to detect the "broadcast termination", a convergecast process can be performed as follows. First, each leaf of the tree which has been reached by the broadcast sends an acknowledgment to its parent. Upon receipt of an acknowledgment from all its children, a node sends an acknowledgment to its parent and so on. When the source node receives an acknowledgment from all its children then the source node knows that the broadcast has reached all the nodes of the graph. This example can be generalized when we want to collect some other information in some root node.

We assume that we have a precomputed rooted spanning tree (Recall that the relabelling system $\mathcal{R}_b$ enables us to construct such a tree). Then, the convergecast operation can be encoded using the relabelling system $\mathcal{R}_c = (\mathcal{L}_c, \mathcal{I}_c, \mathcal{P}_c)$ defined by: $\mathcal{L}_c = \{E\} \cup \{0,1\}$, where $E \in \{A, S, F, T\}$, $\mathcal{I}_c = \{A, S\} \cup \{0,1\}$ and $\mathcal{P}_c = \{R_c^1, R_c^2\}$. Initially, the source node is labelled A, all other nodes are labelled S, an edge belonging to the spanning tree is labelled 1 and all other edges are labelled 0. If a node becomes labelled F then it has finished the convergecast. When the source node A becomes labelled T then the convergecast process is terminated. Figure 2 shows an example of the execution of the convergecast algorithm using the relabelling system $\mathcal{R}_c$.

R_c^1 : **A node becomes a leaf**

Precondition :
- $\lambda(v_0).E = S$
- $\exists!\, v_1 \in B(v_0, 1)\ ((\lambda(v_1).E = S \vee \lambda(v_1).E = A) \wedge \lambda(v_0, v_1) = 1)$

Relabelling :
- $\lambda'(v_0).E := F$

R_c^2 : **Termination detection**

Precondition :
- $\lambda(v_0).E = A$
- $\forall v \in B(v_0, 1)\ (\lambda(v).E = F)$

Relabelling :
- $\lambda'(v_0).E := T$

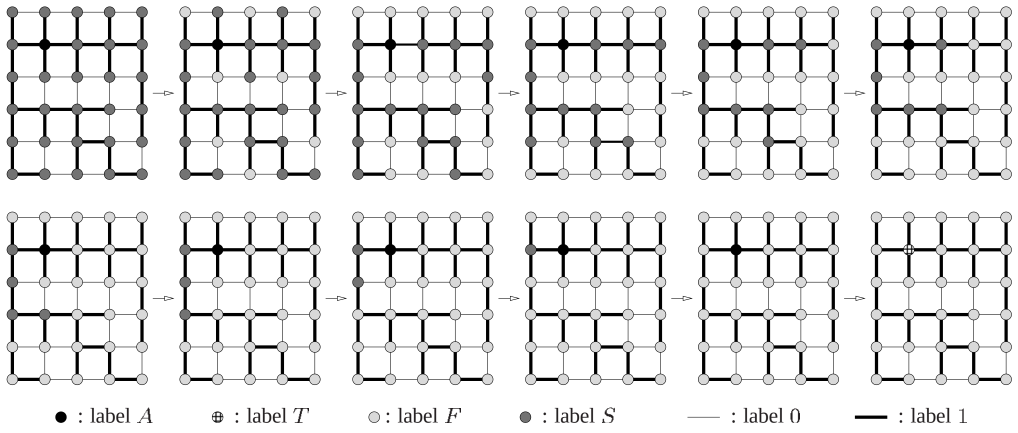

Fig. 2. An example of a convergecast using the relabelling system $\mathcal{R}_c$.

3 Two basic applications

3.1 Layered BFS tree construction

Many distributed algorithms can be implemented by performing as many as necessary phases of broadcast and convergecast. Each phase corresponds to a propagation of some information with feedback (PIF). The broadcast can be viewed as the beginning of a new stage of the algorithm and the convergecast corresponds to the termination of that stage. This technique is fundamental when designing distributed algorithms because, in the distributed setting, there is no centralized entity which supervises in a global way the execution of an algorithm. In the following, our main goal is to show how to encode by graph relabelling systems the Dijkstra's layered BFS tree [5,15,13] algorithm which is based on the PIF operation.

Recall that a BFS tree of a graph G with respect to a root node r is a spanning tree with the property that for every node v, the path leading from the root r to v in the tree is of the minimum (unweighted) length possible. One classical technique to construct a BFS tree begins by growing the tree from one pre-distinguished node. Then, it proceeds in many iterations by constructing the tree in a layered fashion beginning from the root downwards. At each iteration, the unprocessed nodes which are adjacent to some node marked as part of the tree are added. Once a layer is added, the construction of a new layer can begin. The main difficulty here is to begin adding the next layer only when the previous layer has been completely added.

The classical layered BFS tree construction can be encoded using the graph relabelling system $\mathcal{R}_t = (\mathcal{L}_t, \mathcal{I}_t, \mathcal{P}_t)$ defined by: $\mathcal{L}_t = \{E \times i\} \cup \{0,1\}$, where $E \in \{A, S, F, T, O\}$ and $i \in \{-1, 1\}$; $\mathcal{I}_t = \{(O,-1),(A,-1)\} \cup \{0\}$ and $\mathcal{P}_t = \{R_t^1, R_t^2, R_t^3, R_t^4, R_t^5, R_t^6, R_t^7\}$.

In the remainder, by BFS tree, we mean the fragment which is being enlarged. Initially, a pre-distinguished node (the root) is labelled $(A,-1)$ i.e., active. All other nodes are labelled $(O,-1)$ i.e., outside the BFS tree. All edges are labelled 0. If an edge becomes labelled 1 then it is part of the tree. The A-labelled node acts as the initiator of a new iteration of the algorithm i.e., the construction of a new layer. The label $(S,1)$ indicates that a node which is inside the fragment tree must broadcast some information (i.e., the construction of a new layer). In contrast, when a node is labelled $(S,-1)$ then it is waiting for the acknowledgment of its children. A node labelled F is a node that has finished the convergecast and is waiting for an order from its parent. Finally, when a node is labelled T then this node has locally terminated i.e., it can not contribute any more in the tree construction.

Rule R_t^1 initiates the computation of the BFS tree by adding the first layer. It also encodes the beginning of a new iteration of the algorithm. In fact, when the neighbors of the A-node become F (or T) labelled, then the A-node knows that a new layer has been added and the construction of a new layer can begin. Thus, the labels of all F-neighbors are set to $(S,1)$ in order to begin the broadcast of this information up to the leaves of the BFS tree.

R_t^1 : **Beginning the construction of a new layer**

Precondition :
- $\lambda(v_0).E = A$ `/*the root node*/`
- $\forall v \in B(v_0,1)\ (\lambda(v).E \neq S)$ `/*broadcast-convergecast finished*/`
- $\exists v_1 \in B(v_0,1)\ (\lambda(v_1).E \neq T)$ `/*the computation is not over */`

Relabelling :
- $\forall v \in B(v_0,1)\ (\lambda(v).E \neq T \implies (\lambda'(v) := (S,1), \lambda'(v_0,v) := 1))$

If an $(S,1)$-labelled node u is in the interior of the BFS tree, then it just informs its children by setting their labels to $(S,1)$ and it becomes $(S,-1)$ labelled (Rule R_t^2). Otherwise, if a u is a leaf, then either there exist some not yet marked O-neighbors in which case these nodes are added to the tree and u becomes F-labelled (Rule R_t^3), or there are no new nodes to add. In this case, u becomes T-labelled: terminated state (Rule R_t^4).

R_t^2 : **Broadcast**

Precondition :
- $\lambda(v_0) = (S, 1)$ `/*broadcast in progress*/`
- $\exists v_1 \in B(v_0, 1)$ $(\lambda(v_0, v_1) = 1 \wedge \lambda(v_1).E = F)$ `/*children are not informed*/`

Relabelling :
- $\lambda'(v_0) := (S, -1)$
- $\forall v \in B(v_0, 1)$ $((\lambda(v).E = F \wedge \lambda(v_0, v) = 1) \implies \lambda'(v) := (S, 1))$

R_t^3 : **Construction of the next layer**

Precondition :
- $\lambda(v_0) = (S, 1)$
- $\exists v_1 \in B(v_0, 1)$ $(\lambda(v_1).E = O)$ `/*some neighbors are not in the tree*/`

Relabelling :
- $\lambda'(v_0) := (F, -1)$
- $\forall v \in B(v_0, 1)$ $(\lambda(v).E := O \implies (\lambda(v) := (F, -1), \lambda'(v_0, v) := 1))$

R_t^4 : **No nodes to add to the next layer**

Precondition :
- $\lambda(v_0) = (S, 1)$
- $\forall v \in B(v_0, 1)(\lambda(v).E \neq O)$ `/*all neighbors are in the tree*/`
- $\exists!\, v_1 \in B(v_0, 1)$ $(\lambda(v_0, v_1) = 1)$ `/*`v_0` is a leaf*/`

Relabelling :
- $\lambda'(v_0) := (T, -1)$

After the broadcast step in Rule R_t^2, a node u which becomes $(S, -1)$-labelled waits for the acknowledgment of its children. When these children become F-labelled, then u knows that the new layer that corresponds to the last broadcast has been added by the leaves of the subtree rooted at it. Thus, it becomes $(F, -1)$-labelled in order to inform its parent (Rule R_t^5). Note that if all the children of u become T-labelled, then u knows that no new nodes can be added in the subtree rooted at it. Thus, it becomes T-labelled (Rule R_t^6).

R_t^5 : **Convergecast: Waiting for the next broadcast**

Precondition :
- $\lambda(v_0) = (S, -1)$
- $\exists!\, v_1 \in B(v_0, 1)$ $((\lambda(v_1).E = S \vee \lambda(v_1).E = A) \wedge \lambda(v_0, v_1) = 1)$
  `/*all children have received an acknowledgment*/`
- $\exists v_2 \in B(v_0, 1)$ $(\lambda(v_2).E = F \wedge \lambda(v_0, v_2) = 1)$
  `/*some children have not yet terminated the algorithm*/`

Relabelling :
- $\lambda'(v_0) := (F, -1)$

R_t^6 : **Convergecast: The tree construction is locally finished**

Precondition :
- $\lambda(v_0) = (S, -1)$
- $\exists!\, v_1 \in B(v_0, 1)\ ((\lambda(v_1).E = S \vee \lambda(v_1).E = A) \wedge \lambda(v_0, v_1) = 1)$
- $\forall v \in B(v_0, 1)\ ((v \neq v_1 \wedge \lambda(v_0, v) = 1) \implies \lambda(v).E = T)$

Relabelling :
- $\lambda'(v_0) := (T, -1)$

The construction of the BFS tree is terminated when all the neighbors of the A-labelled node become T-labelled. In fact, this means that there is no new layer to add: all the nodes of the graph are in the tree. In this case, the A-labelled node becomes T-labelled (Rule R_t^7). Note that at this stage of the algorithm, only the A-labelled node detects the global termination of the algorithm.

R_t^7 : **Termination detection**

Precondition :
- $\lambda(v_0).E = A$
- $\forall v \in B(v_0, 1)\ (\lambda(v).E = T)$

Relabelling :
- $\lambda'(v_0).E = T$

3.2 Global function computation

In many distributed algorithms, the convergecast and the broadcast are used in order to compute some functions of the graph. Suppose for instance that we want a source node to compute a global function $f(X_{v_1}, X_{v_2}, ..., X_{v_n})$ where X_v is an input stored in each node v. Suppose that f verifies the following properties (we adopt the same notations as in [13] page 36) :

- f is well-defined for any subset of the inputs.
- f is associative and commutative.

In the following, we assume that we have a precomputed spanning tree (obtained for example by using the relabelling system $\mathcal{R}_t$). Such a function f, also called *semigroup function*, can be computed in a distributed manner by performing a convergecast process. In fact, $f(X_{v_1}, X_{v_2}, ..., X_{v_n})$ can be computed using the relabelling system $\mathcal{R}_f = (\mathcal{L}_f, \mathcal{I}_f, \mathcal{P}_f)$ defined by: $\mathcal{L}_f = \{E \times X \times Y\} \cup \{0, 1\}$, where $E \in \{S, F, T\}$, X an input of the function f, $\mathcal{I}_f = \{S\} \cup \{0, 1\}$ and $\mathcal{P}_f = \{R_f^1, R_f^2\}$. Initially, all the nodes are labelled S and an edge with label 1 is part of the precomputed spanning tree.

By *local value of* f in rule R_f^1, we mean the value of f computed on the subtree T_v rooted at the node v that executes the rule R_f^1. The variable Y_1 in rule R_f^1 contains the value of f applied to all the entries X_{v_i} with $v_i \in T_v$ and $v_i \neq v$. The local value of f computed by v will enable the parent u of v to compute its own local value of f. Each time a node applies Rule R_f^1 it becomes F-labelled which means that it has finished to compute the local value of f. At the end of the process, only one node with label S remains. This node then applies rule R_f^2 and it computes the global value $f(X_{v_1}, X_{v_2}, ..., X_{v_n})$. Note that the convergecast process used here does not end at a pre-distinguished node but at some node which is elected at random depending on the algorithm execution. However, the two rules must be executed on disjoint stars that do not overlapp.

R_f^1 : **Convergecast: Computation of the local value of** f

Precondition :
- $\lambda(v_0).E = S$
- $\exists!\, v_1 \in B(v_0, 1)\ (\lambda(v_1).E = S \wedge \lambda(v_0, v_1) = 1)$

Relabelling :
- $Y_1 := f(\cup_{v_i \in B(v_0,1)(\lambda(v_i).E = F \wedge \lambda(v_0,v_i)=1)} \lambda(v_i).Y)$.
- $\lambda'(v_0).E := F$
- $\lambda'(v_0).Y := f(\lambda(v_0).X, Y_1)$

R_f^2 : **Convergecast: Global computation of** f **and termination detection**

Precondition :
- $\lambda(v_0).E = S$
- $\forall v \in B(v_0, 1)\ (\lambda(v).E = F)$

Relabelling :
- $Y_1 := f(\cup_{v_i \in B(v_0,1)(\lambda(v_0,v_i)=1)} \lambda(v_i).Y)$.
- $\lambda'(v_0).E := T$
- $\lambda'(v_0).Y := f(\lambda(v_0).X, Y_1)$

Using these two simple rules, we can encode in a formal way some classical distributed algorithms. For example, to compute the maximum of values stored by the network nodes, we just take $f := max$; to compute the sum over the node inputs, we take $f := +$. This method can also be used to derive distributed algorithms for computing some logical functions. For example, let X_v be a variable set to 1 if some predicate $Pred(v) = true$ and 0 otherwise. Suppose that we want to design a distributed algorithm to determine if the predicate $\exists v Pred(v)$ holds in the network (i.e., : there exists some node v with $Pred(v) = true$). This can be done by letting f be the logical or operator. The predicate $\forall v Pred(v)$ can also be computed distributively be setting $f := and$.

4 Distributed minimum spanning tree: Prim's algorithm

4.1 Preliminaries

In this section, we focus on the distributed construction of a minimum spanning tree. This problem is of special interest because the classical distributed algorithms for solving it use the basic techniques that we have described in previous sections as basic procedures. Our main motivation is to show that by using relabelling systems, we can design such a sophisticated algorithm in a detailed and comprehensive way.

```
Input: a weighted graph $G_{\mathcal{W}} = (V, E)$.
    Step 1: Initially, set $E_T$ (a set of edges) empty.
            Set $V_T$ (a set of nodes) empty.
    Step 2: Select an arbitrary node in $V$, and add it to $V_T$.
    Step 3: Find the lowest weight edge $e = (u, v)$ such that
            $u \in V_T$ but $v \notin V_T$. Add $v$ to $V_T$, and $e$ to $E_T$.
    Step 4: Repeat $Step3$ until $V_T$ equals $V$.
Output: A minimum spanning tree $T = (V, E_T)$.
```

Fig. 3. Prim's Algorithm

Recall that given a weighted graph $G_{\mathcal{W}}$, the MST problem consists in computing a spanning tree T such that the sum of the weights of the edges of T is the minimum over all possible spanning trees of $G_{\mathcal{W}}$. The problem has been heavily studied and many features of the MST problem, such as the distributed computability of such a tree or the time complexity for constructing it, were studied under many assumptions in past works. In this paper, we assume that the edge weights are unique, real and positive. Under this assumption, it is well known that there exists a unique minimum spanning tree of $G_{\mathcal{W}}$ (see [5,15,13] and references there).

One of the most basic algorithms for computing such a MST is the Prim's algorithm [5,15,13] (see Figure 3). Starting from one node, this algorithm consists in growing a fragment by adding at each iteration the minimum outgoing edge (MOE for short) to this fragment. The correctness of the algorithm relies on the fact that, at each iteration, the constructed fragment is part of the MST.

The classical distributed implementation of this algorithm consists of many phases, each one consists of two stages. In the first stage, the nodes in a fragment cooperate to compute the weight of the MOE. This is performed using a convergecast in the already computed fragment. The second stage consists in adding the MOE which is performed by broadcasting the weight of the MOE to all nodes in the fragment. When learning about the weight of the MOE, a node either adds the new edge to the fragment (if the MOE is incident to it) or re-initialize its state in order to begin another phase. The main difficulty here is to combine many broadcast and convergecast operations with the MOE computation. In the following, we give the relabelling system which encodes this MST algorithm.

By combining $\mathcal{R}_t$ and $\mathcal{R}_{f=min}$, Prim's algorithm can be encoded by the graph relabelling system $\mathcal{R}_m$=($\mathcal{L}_m$,$\mathcal{I}_m$,$\mathcal{P}_m$) defined by: $\mathcal{L}_m$=$\{E \times w_{subtree} \times w_{local} \times i\} \cup \{0, 1\}$ where $E \in \{S, F, T, O\}$, $i \in \{-1, 1\}$ and $(w_{subtree}, w_{local}) \in \mathbb{R}^2_+ \cup \bot$; $\mathcal{I}_m = \{(O, \bot, \bot, -1), (S, \bot, \bot, -1)\} \cup \{0\}$ and $\mathcal{P}_m$=$\{R^1_m, R^2_m, R^3_m, R^4_m, R^5_m\}$. Note that, if the value of attribute $w_{subtree}$ (or w_{local}) is equal to $\bot$, then this value has not been set yet.

Initially, there is a distinguished node with label $(S, \bot, \bot, -1)$ which is the first node in the fragment. All other nodes are labelled $(O, \bot, \bot, -1)$. As for the relabelling system $\mathcal{R}_t$, if the value of the attribute i is equal to 1 (resp. -1) then an S-labelled node knows that it is in the broadcast (resp. convergecast) stage. At the beginning, all edges are labelled 0. If an edge becomes labelled 1 then it is part of the tree.

4.2 Computing the weight of the MOE: convergecast

The nodes of the fragment have to cooperate in order to compute the MOE i.e., convergecast from the leaves of the fragment up to an elected node (rule R_m^1). Each node must compute the attributes $w_{subtree}$ which is the weight of the minimum outgoing edge of the subtree rooted at it. Note that, during the convergecast, each node also stores the attribute w_{local} which is the weight of incident edges that connect it to nodes with label O. This will serve in the broadcast stage to find and to add the MOE of the whole fragment.

R_m^1 : **Computing the minimum outgoing edge**

```
  Precondition :
  · $\lambda(v_0) = (S, \perp, \perp, -1)$                    /*convergecast stage*/
  · $\forall v \in B(v_0,1)((\lambda(v).E = S \wedge \lambda(v_0,v) = 1) \Rightarrow \lambda(v).i = -1)$
  · $\exists!\, v_1 \in B(v_0,1)\ (\lambda(v_1).E = S \wedge \lambda(v_0,v_1) = 1)$
    /*all children have received an acknowledgment */
  Relabelling :
  · $w := min\{\{\mathcal{W}(v_0,v) \mid v \in B(v_0,1)(\lambda(v).E = O)\} \cup \{+\infty\}\}$ /*local MOE*/
  · $w_{min} := min\{\{\lambda(v).w_{subtree} \mid v \in B(v_0,1)(\lambda(v_0,v) = 1 \wedge \lambda(v).E = F)\} \cup \{w\}\}$
    /*the MOE of the subtree rooted at $v_0$*/
  · $(w_{min} = +\infty) \Rightarrow \lambda'(v_0).E := T$            /*local termination*/
  · $(w_{min} \neq +\infty) \Rightarrow \lambda'(v_0) := (F, w_{min}, w, -1)$
```

At the end of the convergecast, the weight w_{min} of the MOE is computed at some elected node (rule R_m^2). This node sets its label to $(S, w_{min}, w, 1)$ in order to begin the broadcast phase. (Note also that rule R_m^2 also enables to initialize the MST construction).

R_m^2 : **Election of a node**

```
  Precondition :
  · $\lambda(v_0) = (S, \perp, \perp, -1)$
  · $\forall v \in B(v_0,1)(\lambda(v_0,v) = 1 \implies \lambda(v).E \neq S)$
  Relabelling :
  · $w := min\{\{\mathcal{W}(v_0,v) \mid v \in B(v_0,1)(\lambda(v).E = O)\} \cup \{+\infty\}\}$
  · $w_{min} := min\{\{\lambda(v).w_{subtree} \mid v \in B(v_0,1)(\lambda(v_0,v) = 1 \wedge \lambda(v).E = F)\} \cup \{w\}\}$
  · $(w_{min} = +\infty) \Rightarrow \lambda'(v_0).E := T$
  · $(w_{min} \neq +\infty) \Rightarrow \lambda'(v_0) := (S, w_{min}, w, 1)$
```

Rules R_m^1 and R_m^2 also allow to detect the termination of the MST construction. In fact, if the weight of the MOE is equal to $+\infty$ then there is no node with label O at the frontier of the fragment and thus all the nodes of the graph are in the fragment.

4.3 *Finding and adding the MST: broadcast*

At the end of the convergecast process (rule R_m^2), there is an elected node with label $(S, w, w', 1)$ that begins the broadcast (attribute i is equal to 1). Thus, a node u with label $(S, w, w', 1)$ first compares w and w'. If w=w' then the MOE is incident to u itself. Thus, the minimum edge is added and u sets its variable i to -1 in order to reinitialize the computation of another minimum edge (Rule R_m^4). Otherwise, if $w \neq w'$ then there must exist a neighbor with label $(F, w, w'', -1)$ from whom u has inherited its w value and the MOE must be in the subtree rooted at that neighbor. Thus, the F-labelled neighbor becomes $(S, w, w'', 1)$-labelled (Rule R_m^3). The other F-labelled children become $(S, \perp, \perp, 1)$-labelled in order re-initialize the computation of a new MOE (Rule R_m^5).

R_m^3 : **Broadcast the weight of the MOE**

<u>*Precondition :*</u>
- $\lambda(v_0) = (S, w, w', 1)$
- $w \neq w' \wedge (w \neq +\infty) \wedge (w \neq \perp)$
- $\exists!\, v_1 \in B(v_0, 1)\ (\lambda(v_0, v_1) = 1 \wedge \lambda(v_1).w_{subtree} = w)$
  `/*weights are unique*/`

<u>*Relabelling :*</u>
- $\lambda'(v_0) := (S, \perp, \perp, -1)$
- $\lambda'(v_1).E := S, \lambda'(v_1).i := 1$
- $\forall v \in B(v_0, 1)\ ((v \neq v_1 \wedge \lambda(v_0, v) = 1 \wedge \lambda(v).E = F) \Rightarrow \lambda'(v) := (S, \perp, \perp, 1))$

R_m^4 : **Adding the MOE**

<u>*Precondition :*</u>
- $\lambda(v_0) = (S, w, w, 1)$
- $(w \neq +\infty) \wedge (w \neq \perp)$
- $\exists!\, v_1 \in B(v_0, 1)\ (\lambda(v_0, v_1) = 1 \wedge \lambda(v_1).E = O \wedge \mathcal{W}(v_0, v_1) = w)$

<u>*Relabelling :*</u>
- $\lambda'(v_0) := (S, \perp, \perp, -1)$
- $\forall v \in B(v_0, 1)\ ((v \neq v_1 \wedge \lambda(v_0, v) = 1 \wedge \lambda(v).E = F) \Rightarrow \lambda'(v) := (S, \perp, \perp, 1))$
- $\lambda'(v_1) := (S, \perp, \perp, -1)$
- $\lambda'(v_0, v_1) := 1$

R_m^5 : **Reinitialization**

<u>*Precondition :*</u>
- $\lambda(v_0) = (S, \perp, \perp, 1)$ `/*the MOE is not in the subtree rooted at` v_0`*/`

<u>*Relabelling :*</u>
- $\lambda'(v_0) = (S, \perp, \perp, -1)$
- $\forall v \in B(v_0, 1)\ ((\lambda(v).E = F \wedge \lambda(v_0, v) = 1) \Rightarrow \lambda'(v) := (S, \perp, \perp, 1))$

5 Concluding Remarks

In this paper, we give a general technique that provides a modular construction of a large class of distributed computing algorithms. By exploiting this modular construction and the properties of graph relabelling systems, we obtain a general and a unified framework for expressing, proving and implementing distributed algorithms. The expressiveness has been clearly demonstrated by the numerous algorithms described in the paper. However, some other features concerning the proof techniques and the impelmentation issues remains to be studied in future work.

In fact, in order to prove the correctness of a graph relabelling system, that is the correctness of the algorithm encoded by such a system, it is useful to exhibit *(i)* some *invariant properties* associated with the system (*i.e.*, some properties of the graph labelling that is satisfied by the initial labelling and that is preserved by the application of every relabelling rule) and *(ii)* some properties of irreducible graphs [11]. The correctness of the algorithms given in this paper can be formally proven using that technique. Nevertheless, because our algorithms are clearly expressed as a combination of some few basic procedures, proving these basic procedures allows us to get basic building blocks which can be used as a bottelneck for proving the more sophisticated algorithms. Our aim is to give more generic algorithms allowing to automatically combine the basic techniques presented in this paper by using some logical functions $\mathcal{F}$ to be formally defined in future work. This will allow to automatically derive the relabelling system $\mathcal{A}_{\mathcal{F}}$ corresponding to a distributed algorithm A expressed in term of some basic procedures (convergcast, broadcast, PIF, etc.). By the same way, by using the properties of $\mathcal{F}$ together with the proofs of these basic procedures, we hope to develop new modular techniques that help proving the correcteness of the relabelling system $\mathcal{A}_{\mathcal{F}}$.

In addition to be formal, provable and tractable, the relabelling systems given in this paper can be translated in practical distributed algorithms in the message passing model. In fact, a new language called *Lidia* has been developped in [12] in order to automatically transform a given relabelling system in an executable distributed program using the *Visidia* [6,1,2] platform (i.e., a software tool for the simulation and the visualization of distributed algorithms in the message passing model). Furthermore, the distributed model studied in [3] combined with ideas from this paper will enable to translate a distributed algorithm expressed in a message passing model in a more formal and theoritical framework. This will enable to verify and to debug existing sophisticated algorithms which are in general hard to validate.

Acknowledgement

We would like to thank J. Chalopin and Y. Métivier for helpful remarks and suggestions.

References

[1] Bauderon, M., Y. Métivier, M. Mosbah and A. Sellami, *From local computations to asynchronous message passing systems*, Technical Report RR-1271-02, LaBRI

(2002).
URL http://www.labri.fr/visidia/

[2] Bauderon, M. and M. Mosbah, *A unified framework for designing, implementing and visualizing distributed algorithms*, International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'02) (2002).
URL http://www.elsevier.nl/locate/entcs/volume72.html

[3] Chalopin, J. and Y. Métivier, *A bridge between the asynchronous message passing model and local computations in graphs*, 30th Int. Symp. on Mathematical Foundations of Computer Science **LNCS, to appear** (2005).

[4] Chang, E. J. H., *Echo algorithms: Depth parallel operations on general graphs.*, IEEE Trans. Software Eng. **8** (1982), pp. 391–401.

[5] Cormen, T. H., C. E. Leiserson and R. L. Rivest, "Introduction to Algorithms," MIT Press/McGraw-Hill, Cambridge, MA, 1990.

[6] Derbel, B. and M. Mosbah, *Distributing the execution of a distributed algorithm over a network*, 7th IEEE International Conference on Information Visualization, IV03-AGT. (16-18 July 2003, London), pp. 485–490.

[7] Godard, E., Y. Métivier and A. Muscholl, *Characterizations of classes of graphs recognizable by local computations*, Theory of Computing Systems **37:2** (2004), pp. 249–293.

[8] Litovsky, I., Y. Métivier and E. Sopena, *Different local controls for graph relabelling systems*, Math. Syst. Theory **28** (1995), pp. 41–65.

[9] Litovsky, I., Y. Métivier and E. Sopena, *Graph relabelling systems and distributed algorithms*, , **3**, H. Ehrig and H.J. Kreowski and U. Montanari and G. Rozenberg, World Scientific, 1999 pp. 1–56.

[10] Lynch, N. A., "Distributed Algorithms," Morgan Kaufmann Publishers, Inc., 1996.

[11] Métivier, Y., M. Mosbah and A. Sellami, *Proving distributed algorithms by graph relabeling systems: Examples of trees in networks with processor identities*, in: *Applied Graph Transformations*, Grenoble, 2002, pp. 45–57.

[12] Mosbah, M. and R. Ossamy, *A programming language for local computations in graphs: Computational completeness*, in: IEEE, editor, 5^{th} *Mexican Int. Conference in Computer Science Colima Mexico 20-24 September* (2004), pp. 12–19.

[13] Peleg, D., "Distributed Computing, A Locality-Sensitive Approach," SIAM Monographs on Discrete Mathematics and Applications, 2000.

[14] Segall, A., *Distributed network protocols.*, IEEE Transactions on Information Theory **29** (1983), pp. 23–34.

[15] Tel, G., "Introduction to distributed algorithms," Cambridge University Press, 2000.

(This page intentionally left blank)