Labelled Graph Strategic Rewriting for Social Networks

Maribel Fernandez, Hélène Kirchner, Bruno Pinaud, Jason Vallet

To cite this version:

Maribel Fernandez, Hélène Kirchner, Bruno Pinaud, Jason Vallet. Labelled Graph Strategic Rewriting for Social Networks. Journal of Logical and Algebraic Methods in Programming, Elsevier, 2018, 96 (C), pp.12–40. 10.1016/j.jlamp.2017.12.005 . hal-01664593

HAL Id: hal-01664593

https://hal.archives-ouvertes.fr/hal-01664593

Submitted on 15 Dec 2017

Labelled Graph Strategic Rewriting for Social Networks

Maribel Fernández[1], Hélène Kirchner[2], Bruno Pinaud[3], and Jason Vallet[3]

[1] King's College London, UK maribel.fernandez@kcl.ac.uk
[2] Inria, France helene.kirchner@inria.fr
[3] University of Bordeaux, CNRS UMR5800 LaBRI, France
firstname.lastname@u-bordeaux.fr

Abstract. We propose an algebraic and logical approach to the study of social networks, where network components and processes are directly defined by labelled port graph strategic rewriting. Data structures attached to graph elements (nodes, ports and edges) model local and global knowledge in the network, rewrite rules express elementary and local transformations, and strategies control the global evolution of the network. We show how this approach can be used to generate random networks, simulate existing propagation and dissemination mechanisms, and design new, improved algorithms.

Paper accepted for publication in the Journal of Logical and Algebraic Methods in Programming (JLAMP). This is the author's version of the paper.

1 Introduction

Social networks, representing users and their relationships, have been intensively studied in the last years [11, 37, 43]. Their analysis raises several questions, in particular regarding their construction and evolution. Network *propagation* mechanisms, replicating real-world social network phenomena such as communications between users (e.g., announcing events), and the related concept of *dissemination* mechanism, whose goal is to transmit specific information across the nodes of a network, have applications in various domains, ranging from sociology [27] to epidemiology [16, 8] or even viral marketing and product placement [14]. Dissemination algorithms have also applications outside the social network domain, for example in computer networks routing protocols [45] or to model cache poisoning attacks in DNS servers [44]. To gain a better understanding of these phenomena, we need to model and analyse such systems, dealing with features that are complex (since they involve quantities of highly heterogeneous data), dynamic (due to interactions, time, external or internal evolutions), and distributed.

To address these challenges we use: **Labelled Graphs** to represent networks of data or objects, **Rules** to deal with local transformations, and **Strategies** to control the application of rules (including probabilistic application) and to focus on points of interest. The dynamic evolution of data is generally modelled by simple transformations, possibly applied in parallel and triggered by events or time. However, such transformations may be controlled by some laws that induce a global behaviour. Modelling may reveal conflicts, which can be detected by computing overlaps of rules and solved, for instance, by using precedence. Thus, the ability to define strategies is also essential, including mechanisms to deal with backtracking and history through notions of derivations or traces. Preliminary results obtained by applying this approach to the study of propagation phenomena and network generation are described in [46, 22], respectively.

In this paper, we expand the study of network generation and propagation algorithms, and we also consider examples of dissemination algorithms. We start by presenting a framework, based on port graph rewriting, to define social network models and specify their dynamic behaviour. Port graph rewriting systems have been used to model systems in other domains (e.g., biochemistry, interaction nets, games; see [46, 1, 19, 20]). Here, we adapt to the specific domain of social network modelling the general port graph rewriting notions given in [21], mainly by using oriented edges. This framework is then used to specify an algorithm to generate networks that have the characteristics of real-world social networks, and to study the processes of propagation (triggered by users wanting to pass information to their neighbours) and dissemination (performed by an algorithm according to predefined rules). In the latter, we consider two propagation algorithms –the *Independent Cascade* model **IC** [30] and the *Linear Threshold* model **LT** [26]–, which have both already been specified using strategic rewriting in [46], and the Riposte dissemination model **RP** [24] which, up to our knowledge, has not been specified using graph rewriting transformations yet. Finally, to show the resilience of our approach, we develop a new dissemination algorithm, which we call **RP-LT**, based on **RP** but incorporating characteristics of **LT** as well. This model is proposed as an example to illustrate how the port graph strategic rewriting framework can help in the specification of new models by simply enriching the graph data structure with new attributes and adapting the strategy.

All algorithms have been implemented in Porgy, an interactive port graph rewriting tool [39]. Thanks to the formal semantics of Porgy's language, we are able to provide proofs of expected properties for the algorithms implemented.

Related Work Several definitions of graph rewriting are available, using different kinds of graphs and rewrite rules (see, for instance, [5, 33, 6, 15, 40, 28, 2, 3]). To model social networks, we use *directed port graphs with attributes* and the general notion of *port graph rewriting* presented in [21]. An alternative solution using undirected edges with port labels encoding direction ("In" and "Out") was previously developed [46], however, having oriented edges as a primitive concept makes it easier to represent social relationships that are naturally asymmetric.

Although many data sets, extracted from various real-world social networks, are publicly available,[4] in order to test new ideas, demonstrate the generality of a new technique, or design and experiment with stochastic algorithms on a

[4] For instance from `http://snap.stanford.edu` or `http://konect.uni-koblenz.de/`

sufficiently large sample of network topologies, it is sometimes convenient to use randomly generated networks as they can be fine-tuned to produce graphs with specific properties (number of nodes, edge density, edge distribution, ...). Many generative models of random networks are available (e.g., [18, 49, 4, 47, 7] to cite only a few). Some, like the Erdös–Rényi (ER) model [18], do not guarantee any specific property regarding their final topology, whereas others can show small-world characteristics [49], distinctive (scale-free) edge distributions [4] or both at the same time [47]. In this paper, we show how to generate such models using labelled port graphs, rules and strategies. Moreover, we take advantage of the visual and statistical features available in PORGY to tune the algorithms: our experimental results guide and validate the parameter choices made in the generation algorithms, ensuring the generated networks satisfy the required properties. Afterwards, we implement two propagation and two dissemination algorithms. While three of them are based on existing models described in previous work [30, 26, 24], the fourth model is original, specifically built to incorporate the influence mechanisms proposed in [26] into the privacy-preserving dissemination model described in [24].

The paper is organised as follows. Sect. 2 introduces the modelling concepts: labelled port graphs, rewriting, derivation tree, strategy and strategic graph programs. We develop social network generation algorithms in Sect. 3, propagation algorithms in Sect. 4 and dissemination algorithms in Sect. 5. Sect. 6 briefly describes a framework for designing and experimenting with social network models. Finally, we conclude in Sect. 7.

2 Labelled Graph Rewriting for Social Networks

A social network [10] is usually described as a graph where nodes represent users and edges represent their relationships. Some real-world social relations involve mutual recognition (e.g., friendship), whereas others present an asymmetric model of acknowledgement (e.g., Twitter, where one of the users is a *follower* while the other is a *followee*). It is thus natural to represent such relations using directed graphs. In this paper, we model social networks using labelled directed port graphs, as defined below.

2.1 Directed Port Graphs for Social Networks

Roughly speaking, a port graph is a graph where edges are attached to nodes at specific points, called *ports*. Nodes, ports and edges are labelled using records.

A record r is a set of pairs $\{a_1 := v_1, \ldots, a_n := v_n\}$, where a_i, called *attribute*, is a constant in a set $\mathcal{A}$ or a variable in a set $\mathcal{X}_{\mathcal{A}}$, and v_i is the value of a_i, denoted by $r \cdot a_i$; the elements a_i are pairwise distinct.

The function *Atts* applies to records and returns all the attributes:

$$Atts(r) = \{a_1, \ldots, a_n\} \quad \text{if } r = \{a_1 := v_1, \ldots, a_n := v_n\}.$$

Each record $r = \{a_1 := v_1, \ldots, a_n := v_n\}$ contains one pair where $a_i = Name$. The attribute $Name$ defines the type of the record in the following sense: for all r_1, r_2, $Atts(r_1) = Atts(r_2)$ if $r_1.Name = r_2.Name$.

Values in records can be concrete (numbers, Booleans, etc.), or can be terms built on a signature $\Sigma = (\mathcal{S}, \mathcal{O}p)$ of an abstract data type and a set $\mathcal{X}_\mathcal{S}$ of variables of sorts $\mathcal{S}$. We denote by $T(\Sigma, \mathcal{X}_S)$ the set of terms over Σ and $\mathcal{X}_S$.

Records with abstract values (i.e., expressions $v_i \in T(\Sigma, \mathcal{X}_S)$ that may contain variables), allow us to define generic patterns in rewrite rules: abstract values in left-hand sides of rewrite rules are matched against concrete data in the graphs to be rewritten. We use variables not only in values but also to denote generic attributes and generic records in port graph rewrite rules.

Port graphs are now defined as an algebra (sets and functions defined on these sets) in the following way:

Definition 1 (Attributed port graph). *An* attributed port graph $G = (V, P, E, D)_\mathcal{F}$ *is given by a tuple* (V, P, E, D) *where:*

- $V \subseteq \mathscr{N}$ *is a finite set of nodes;* $n, n_1, \ldots$ *range over nodes;*
- $P \subseteq \mathscr{P}$ *is a finite set of ports;* $p, p_1, \ldots$ *range over ports;*
- $E \subseteq \mathscr{E}$ *is a finite set of edges between ports;* $e, e_1, \ldots$ *range over edges; two ports may be connected by more than one edge;*
- D *is a set of records;*

and a set $\mathcal{F}$ *of functions* $Connect$, $Attach$ *and* $\mathcal{L}$ *such that:*

- *for each edge* $e \in E$, $Connect(e)$ *is the pair* (p_1, p_2) *of ports connected by* e;
- *for each port* $p \in P$, $Attach(p)$ *is the node* n *to which the port belongs;*
- $\mathcal{L} : V \cup P \cup E \mapsto D$ *is a labelling function that returns a record for each element in* $V \cup P \cup E$.

Moreover, we assume that for each node $n \in V$, $\mathcal{L}(n)$ *contains an attribute* $Interface$ *whose value is the list of names of its ports, that is,* $\mathcal{L}(n) \cdot Interface = [\mathcal{L}(p_i) \cdot Name \mid Attach(p_i) = n]$ *such as the following constraint is satisfied:*

$$\mathcal{L}(n_1) \cdot Name = \mathcal{L}(n_2) \cdot Name \Rightarrow \mathcal{L}(n_1) \cdot Interface = \mathcal{L}(n_2) \cdot Interface.$$

By definition of record, nodes/ports/edges with same name (i.e., the same value for the attribute $Name$) have the same attributes, but could have different values. This type constraint is stronger for nodes: Def 1 forces nodes with the same name to have the same port names (i.e., the same interface) although other attribute values may be different.

We present in Fig. 1 an example of attributed port graph. In this example, nodes have attributes *State*, *Marked*, and *Tau* (used in the algorithms given in the next sections), as well as an attribute *Colour*, for visual purposes.

If an edge $e \in E$ goes from n to n', we say that n' is *adjacent* to n (not conversely) or that n' is a neighbour of n. The set of nodes adjacent to a subgraph F in G consists of all the nodes in G *outside* F and adjacent to any node in F. $Ngb(n)$ is used to denote the set of neighbours of the node n.

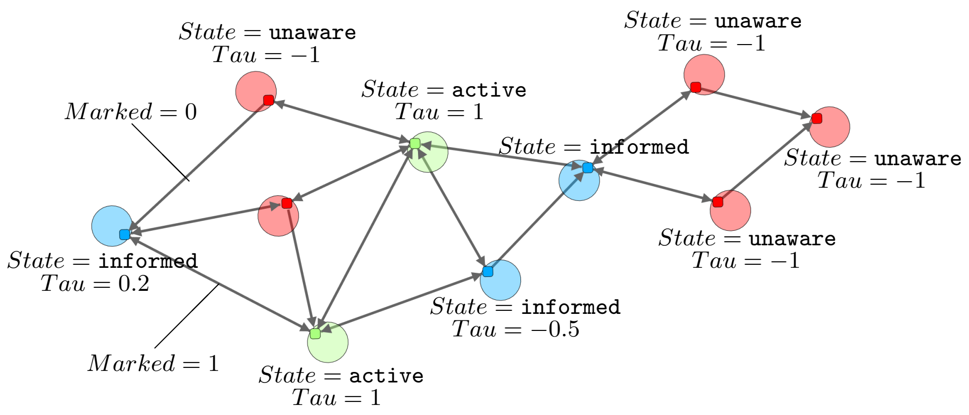


Fig. 1. Example of port graph for a toy social network

In the social network models used in this paper, nodes representing users have only one port gathering directed connections and edges are directed. This simplifies in particular drawings and visualisation of big networks. While this is sufficient in many cases, when dealing with real social networks, multiple ports are useful, either to connect users according to the nature of their relation (e.g., friend, parent, co-worker, ...) or to model situations where a user is connected to friends via different social networks. In such cases, the advantage of using port graphs rather than plain graphs is to allow us to express in a more structured and explicit way the properties of the connections, since ports represent the connecting points between edges and nodes. The full power of port graphs is indeed necessary in multi-layer networks [32] where edges are assigned to different layers and where nodes are shared. In that case, different ports are related to different layers, which can improve modularity of design, readability and matching efficiency through various heuristics. This is however another topic left for future work.

2.2 Located Rewriting

A *port graph rewrite rule* is itself a port graph consisting of two subgraphs L and R together with an *arrow* node that links them. Each rule is characterised by its arrow node, which has a unique name (the rule's label), a condition restricting the rule's application at matching time, and ports to control the rewiring operations at replacement time.

We use here a simple version of port graph rewrite rule suitable for the context of social networks. The full definition implemented in Porgy is given in [21].

Definition 2 (Simple port graph rewrite rule). *A port graph rewrite rule, denoted $L \Rightarrow R$, is a port graph consisting of:*

- *two port graphs L and R, called* left-hand side *and* right-hand side*, respectively, such that all variables in R occur in L;*
- *an* arrow *node $\Rightarrow$ with a set of edges that each connects a port of the arrow node to a port in L and a port in R. The arrow node has an attribute $Name$ with value lab which is unique; an attribute $Where := C$ where C is a Boolean expression such that all variables in C occur in L; and a number of ports corresponding to each connection from a port in L and a port R.*

The $Where$ attribute in the arrow node has a value of the form

$$saturated(p_1) \wedge ... \wedge saturated(p_n) \wedge B$$

where $p_1, \ldots, p_n$ are the ports in L not linked by an edge to the arrow node, saturated is a special predicate whose role is explained below, and B is an optional user-defined Boolean expression involving elements of L (edges, nodes, ports and their attributes).

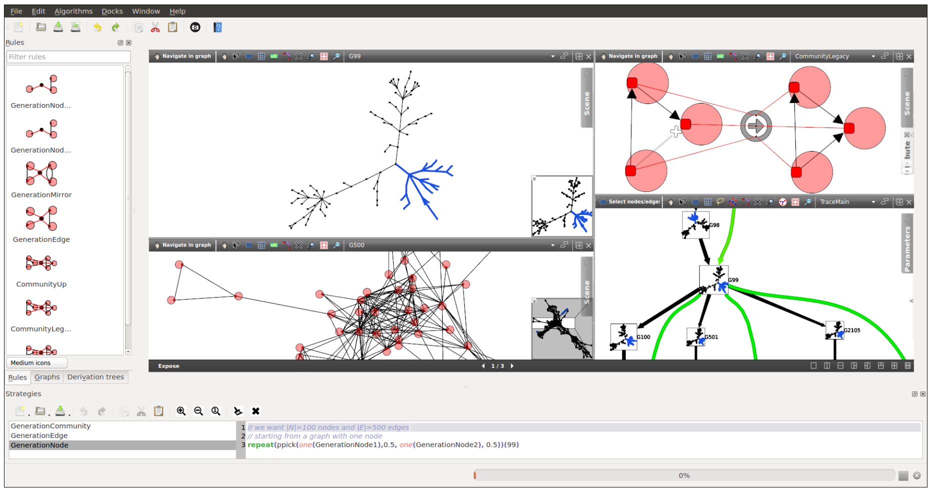
The introduction of the *Where* attribute is inspired from the GP programming system [41] and by a more general definition given in Elan [9]. Its value is a Boolean expression in which B is used in our examples to specify the absence of certain edges. For instance, a condition `where not Edge(n,n')` requires that no edge exists between n and n'; this condition is checked at matching time. The condition involving the *saturated* predicate is automatically generated in Porgy for every port in L not connected to the arrow node and also checked during matching.

The edges connecting the arrow node with L and R and the *saturated* predicate are used to control the rewiring that occurs during a rewriting step, ensuring no dangling edges [17] arise during rewriting [21]. Figure 2 (top right-hand side corner) shows an example of a simple port graph rewrite rule, which is used in the generation algorithm given in the next section (we give details below).

Let us now briefly recall the notion of port graph morphism, fully defined in [21]: if G and H are two port graphs, a *port graph morphism* $f\colon G \mapsto H$ maps nodes, ports and edges of G to those of H such that the attachment of ports to nodes and edge connections are preserved, as well as their data values. In other words, G and $f(G)$ have the same structure, and each corresponding pair of nodes, ports and edges in G and H have the same set of attributes and associated values, except at positions where there are variables.

Variables are useful to specify rules where some attributes of the left-hand side are not relevant for the application of the transformation. Intuitively, the morphism identifies a subgraph of H that is equal to G except at positions where G has variables (at those positions, H could have any instance).

Definition 3 (Match). *Let $L \Rightarrow R$ be a simple port graph rewrite rule and G a port graph without variables (i.e., a* ground *port graph). A* match *$g(L)$ of the left-hand side (also called a* redex*) is found in G if there is a total port graph morphism g, injective on graph items (ports, nodes, edges), called* matching morphism*, from L to G such that if the arrow node has an attribute* Where



```
// we want |N|=100 nodes and |E|=500 edges
// starting from a graph with one node
repeat(ppick(one(GenerationNode1),0.5, one(GenerationNode2), 0.5))(99)
```

Fig. 2. A screenshot of PORGY in action.

with value C, then $g(C)$ is true in $g(L)$. The atom saturated$(g(p))$ is true if there are no edges between $g(p)$ and ports outside $g(L)$ in G.

Several injective matching morphisms g from L to G may exist, leading to different rewriting steps.

Definition 4 (Rewriting step). *A* rewriting step *on G using a simple port graph rule $L \Rightarrow R$ and a matching morphism $g : L \mapsto G$, written $G \to^{g}_{L\Rightarrow R} G'$, transforms G into a new graph G' obtained from G by performing the following operations in three phases:*

- *In the* build *phase, after a redex $g(L)$ is found in G, a copy $R_c = g(R)$ (i.e., an instantiated copy of the port graph R) is added to G.*
- *The* rewiring *phase then redirects edges from G to R_c as follows:*
  For each port p in the arrow node: if $p_L \in L$ is connected to p, for each port $p^i_R \in R$ connected to p, find all the ports p^k_G in G that are connected to $g(p_L)$ and are not in $g(L)$, and redirect each edge connecting p^k_G and $g(p_L)$ to connect p^k_G and $p^i_{R_c} = g(p^i_R)$.
- *The* deletion *phase simply deletes $g(L)$. This creates the final graph G'.*

In [21], we show that attributed port graphs are attributed graph structures [35]; in a simple port graph rewrite rule, the arrow node defines a partial morphism between the left and right-hand side of the rule; a rewriting step is the pushout defined by the arrow node morphism and the matching morphism; simple port graph rewrite rules define a rewriting relation that corresponds exactly to the single pushout semantics and can be translated to the double pushout framework.

To facilitate the specification of graph transformations by defining explicitly the focus of the transformation and the forbidden subgraph if any, we use the concept of *located graph* from [21].

Definition 5 (Located graph). *A* located graph G_P^Q *consists of a port graph G and two distinguished subgraphs P and Q of G, called respectively the* position subgraph*, or simply* position*, and the* banned subgraph*.*

In a located graph G_P^Q, P is the subgraph of G under study (the focus of the transformations), and Q is a protected subgraph, where transformations are forbidden. Below, where the located graph G_P^Q is clear from the context, we refer to P as the *current position.*

When applying a port graph rewrite rule, not only the underlying graph G but also the position and banned subgraphs may change. A *located rewrite rule*, defined below, specifies two disjoint subgraphs M and M' of the right-hand side R that are respectively used to update the position and banned subgraphs. If M (resp. M') is not specified, R (resp. the empty graph $\emptyset$) is used as default. Below, we use the operators $\cup, \cap, \setminus$ to denote union, intersection and complement of port graphs. These operators are defined on port graphs from the usual set operations on sets of nodes, ports and edges, except for $\setminus$ where edges attached to ports are dropped when the ports are not in the difference to avoid dandling edges.

Definition 6 (Located rewrite rule). *A* located rewrite rule *is given by a (simple) port graph rewrite rule $L \Rightarrow R$, with two disjoint subgraphs M and M' of R and optionally, a subgraph W of L. It is denoted $L_W \Rightarrow R_M^{M'}$.*

We write $G_P^Q \rightarrow^g_{L_W \Rightarrow R_M^{M'}} G'^{Q'}_{P'}$ and say that the located graph G_P^Q rewrites *to $G'^{Q'}_{P'}$* using *$L_W \Rightarrow R_M^{M'}$* at position *P* avoiding *Q, if $G \rightarrow_{L \Rightarrow R} G'$ with a morphism g such that $g(L) \cap P = g(W)$ or simply $g(L) \cap P \neq \emptyset$ if W is not provided, and $g(L) \cap Q = \emptyset$. The new position subgraph P' and banned subgraph Q' are defined as $P' = (P \setminus g(L)) \cup g(M)$, and $Q' = (Q \cup g(M')$; if M (resp. M') are not provided then we assume $M = R$ (resp. $M' = \emptyset$).*

Sections 3, 4 and 5 provide several examples of located graphs and rewriting. For instance, in influence propagation, carefully managed position and banned subgraphs are used to avoid several consecutive activations of the same neighbours. Another usage is to select a specific community in the social network where the propagation should take place.

In general, for a given located rule $L_W \Rightarrow R_M^{M'}$ and located graph G_P^Q, several rewriting steps at P avoiding Q may be possible. Thus, the application of the rule at P avoiding Q may produce several located graphs. A *derivation*, or computation, is a sequence of rewriting steps. If all derivations are finite, the system is said to be *terminating.* A *derivation tree* from G is made of all possible computations (including possibly infinite ones). *Strategies* are used to specify the rewriting steps of interest, by selecting branches in the derivation tree. See Fig. 2 (bottom right-hand side corner) for an example of a derivation tree. Black arrows are for rewrite steps and green ones for strategy steps.

2.3 Strategic graph programs

A *strategic graph program* consists of a *located graph*, a set of located rewrite rules, and a strategy expression that combines applications of located rules and focusing constructs. A full description of the language describing strategy expressions (abstract syntax, semantics) can be found in [21]. The user manual describing how to write a working strategy (concrete syntax) can be found in [23]. Below, we remind constructs used in this paper and their abstract syntax. The only slight difference is the use of oriented graphs.

A strategy expression S combines applications of located rewrite rules T and position updates using focusing expressions F.

The primary construct is a located rewrite rule (or *transformation* for short), T, which can only be applied to a located graph G_P^Q if at least a part of the redex is in P, and does not involve Q. A probabilistic choice of the rule to apply is possible with `ppick`$(T_1, \ldots, T_n, \Pi)$ which picks one of the transformations for application, according to the probability distribution Π. `one`(T) computes only one of the possible applications of the transformation T on the current located graph at the current position and ignores the others; more precisely, it makes an equiprobable choice between all possible applications. Respectively, `all`(T) denotes all possible applications of the transformation T, thus creating a new located graph for each application. In the derivation tree, this creates as many children as there are possible applications.

In this paper, F (position at which a rule can be applied or not) is defined using the following elements:

- `crtGraph`, `crtPos` and `crtBan`: applied to a located graph G_P^Q, return respectively the whole graph G, the position subgraph P and the banned subgraph Q.
- `property`$(F, Elem, Expr)$ returns a subgraph G' of G (defined by F) that satisfies the decidable property $Expr$. Depending on the value of $Elem$, the property is evaluated on nodes, ports, or edges. If $Expr$ is not specified, all designed graph elements are selected.
- `ngb`$(F, Elem, Expr)$ returns a subset of the neighbours (i.e., adjacent nodes) of (the graph defined by) F that satisfy the decidable property $Expr$. Depending on the value of $Elem$, the property is evaluated on nodes, ports, or edges. To emphasise edge direction we also introduce `ngbOut`$(F, Elem, Expr)$ and its counterpart `ngbIn`$(F, Elem, Expr)$.

Similar constructs as `one(T)` and `all(T)` exist for focusing expressions, which are used to define positions for rewriting in a graph, or to define positions where rewriting is not allowed: `one`(F) returns one node in the subgraph defined by F and `all`(F) returns the full F.

Let D be `one`(F) or `all`(F), then `setPos`(D) (resp. `setBan`(D)) sets the position subgraph P (resp. Q) to be the graph resulting from the expression D.

The following constructs are used to build strategies:

- S;S' represents sequential application of S followed by S'.

- `if`(S)`then`(S')`else`(S'') checks if the application of S on (a copy of) G_P^Q succeeded, in which case S' is applied to (the original) G_P^Q, otherwise S'' is applied to the original G_P^Q. The `else`(S'') part is optional.
- `repeat`(S)[max n] simply iterates the application of S until it fails; if max n is specified, then the number of repetitions cannot exceed n.
- (S)`orelse`(S') applies S if possible, otherwise applies S'. It fails if both S and S' fail.
- `try`(S) always succeeds even if S fails.
- `ppick`$(S_1, \ldots, S_k, \Pi)$ picks one of the strategies for application, according to the given probability distribution Π.

Probabilistic features of the strategy language, through the use of the `ppick`() construct, are used in Sect 3 for social network generation. The propagation models described in Sect. 4 show how record expressions are used to compute attribute values and how these are updated through application of rules.

3 Social network generation

In this section we address the problem of generating graphs with a *small-world property* as defined in [49]. Such graphs are characterised by a small diameter –the average distance between any pair of nodes is short– and strong local clustering –for any pair of connected nodes, both tend to be connected to the same neighbouring nodes, thus creating densely linked groups of nodes called *communities*, whose interest has been stressed in [42] for instance. Popularised by Milgram in [36], small-world graphs are a perfect case study for information propagation in social networks due to their small diameter allowing a quick and efficient spreading of information.

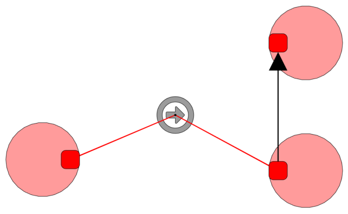
Our goal is to design an algorithm to generate small-world graphs of a given size, that is, for which the number of nodes $|N|$ and directed edges $|E|$ are given *a priori*. Moreover, the graphs generated should satisfy the following conditions: they must have only one connected component, thus $|E| \geq |N| - 1$; they should be simple, that is, any ordered pair of nodes (n, n') can only be linked once, thus the maximum number of edges is $|E|_{max} = |N| \times (|N| - 1)$; finally, the number of communities should be randomly decided during the generation process.

A few previous works have explored the idea of using rules to generate networks. In [29], the authors define and study probabilistic inductive classes of graphs generated by rules which model spread of knowledge, dynamics of acquaintanceship and emergence of communities. Below we present a new algorithm for social network generation that follows a similar approach, however, we have adjusted its generative rules to cope with directed edges and ensure the creation of a graph with a single connected component. This is achieved by performing the generation through local additive transformations, each only creating new elements connected to the sole component, thus increasingly making the graph larger and more intricate.

Starting from one node, the generation is divided into three phases imitating the process followed by real-world social networks. Whenever new users first join the social network, their number of connections is very limited, mostly to the other users who have introduced them to the social network (Sect. 3.1). During the second phase, these new users can reach the people they already know personally, thus creating new connections within the network (Sect. 3.2). Finally, the users get to know the people with whom they are sharing friends in the network, potentially leading to the creation of new connections (Sect. 3.3).

3.1 Generation of a directed acyclic port graph

The first step towards the construction of a directed port graph uses the two rules shown in Figures 3(a) and 3(b). Both rules apply to a single node and generate two linked nodes (thus each application increases by one the number of nodes and also the number of edges). The difference between these two rules lies in the edge orientation as Rule 3(a) creates an outgoing edge on the initiating node, while Rule 3(b) creates an incoming edge.


(a) Rule *GenerationNode1*.

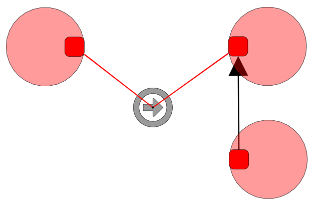

(b) Rule *GenerationNode2*.

Fig. 3. Rules used for generating and re-attaching nodes to pre-existing node with a directed edge going from the pre-existing node to the newly added node in (a) or oriented in the opposite direction in (b).

Strategy 1: *Node generation*: Creating a directed acyclic graph of size N

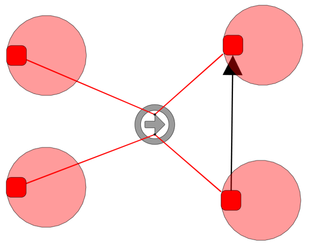
```
//equiprobabilistic application of the two rules used for generating nodes
repeat(
  ppick(one($GenerationNode1$),
            one($GenerationNode2$),
            $\{0.5, 0.5\}$)
)($|N| - 1$) // Generation of $N$ nodes
```

The whole node generation is achieved during this first phase and managed using Strategy 1. It repeatedly applies the generative rules $|N| - 1$ times so that the graph reaches the appropriate number of nodes. As mentioned earlier, each rule application also generates a new edge, which means that once executed, Strategy 1 produces a graph with exactly $|N|$ nodes and $|N| - 1$ edges. The orientation of each edge varies depending of the rule applied (either 3(a) or 3(b)), moreover, their application using the `ppick`() construct ensures an equiprobable choice between the two rules.

3.2 Creating complementary connections

During this phase, we either create seemingly random connections between the network users or reciprocate already existing single-sided connections.

We use two rules to link existing nodes, thus creating a new additional edge with each application. The first rule (Fig. 4(a)) simply considers two nodes and adds an edge between them to emulate the creation of a (one-sided) connection between two users. The second rule (Fig. 4(b)) reciprocates an existing connection between a pair of users: for two nodes $n, n' \in N$ connected with an oriented edge (n', n), a new oriented edge (n, n') is created; it is used to represent the mutual appreciation of users in the social network. Note that, because each node is randomly chosen among the possible matches, we do not need to create alternative versions of these rules with reversed oriented edges.


(a) Rule *GenerationEdge*.

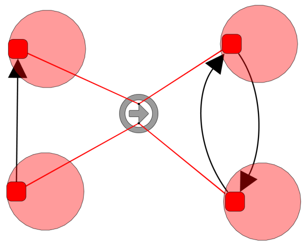

(b) Rule *GenerationMirror*.

Fig. 4. Rules generating additional connections: (a) between two previously unrelated nodes, (b) by reciprocating a pre-existing connection.

In both rules, the existence of edges between the nodes on which the rule applies should be taken into account: the rules should not create an edge if a similar one already exists (since we aim at creating a simple graph rather than a multi-graph). This can be achieved by adding a condition "`where not Edge(n,n')`" (see Definition 2), or by using position constructs to restrict the elements to be considered during matching. We use the latter solution here.

In Strategy 2, we first filter the elements to consider during the matching. We randomly select one node among the nodes whose outgoing arity (*OutArity*) is lower than the maximal possible value (i.e., $|N| - 1$), and we ban all its outgoing neighbours as they cannot be considered as potential matching elements. Then, Rule 4(a) or Rule 4(b) are equiprobably applied to add a new edge from the selected node. By banning neighbours, we ensure that future applications of the rule will not use those nodes, that is, the rule will only apply on pairs of nodes not already connected. This ensures that the graph is kept simple (i.e., only one edge per direction between two nodes).

Strategy 2: *Edge generation*: addition of $|E'|$ edges if possible.

```
repeat(
//select one node with an appropriate number of neighbours
  setPos(one(property(crtGraph, node, $OutArity < |N| - 1$)));
//for this node, forbid rule applications on its outgoing neighbours
  setBan(all(ngbOut(crtPos, node, $true$)));
//equiprobable application of the edge generation rules
  ppick((one($GenerationEdge$))orelse(one($GenerationMirror$)),
           (one($GenerationMirror$))orelse(one($GenerationEdge$)),
           $\{0.5, 0.5\}$)
)($|E'|$)
```

In this phase, we create $|E'|$ edges, where $|E'| < (|E| - |N| + 1)$ to keep the number of edges below $|E|$. The use of the `orelse` construct allows testing all possible rule application combinations, thus, if one of the rules can be applied, it is found. If no rule can be applied, the maximum number of edges in the graph has been reached, i.e., the graph is complete. If the value of $|E'|$ is not too high, we are left with $(|E| - |E'| - |N| + 1)$ remaining edges to create in the next step for enforcing communities within G.

3.3 Construction of communities

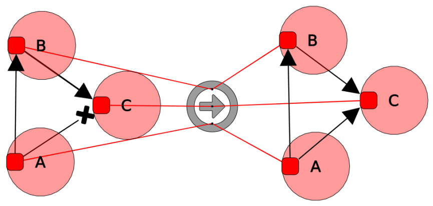
To create a realistic social network, we now add communities. For this, we need to ensure that the links between users follow certain patterns. Based on ideas advanced in several previous works (e.g., [12, 29, 34, 38]), we focus on triad configurations (i.e., groups formed by three users linked together). Our community generation algorithm uses three rewrite rules, introduced in Figure 5.

The first triad rule (Fig. 5(a)) considers how a first user (A) influences a second user (B) who influences in turn a third user (C)[5]. The second rule (Fig. 5(b)) shows two users (B and C) being influenced by a third user (A)[6]. The last rule (Fig. 5(c)) depicts one user (B) being influenced by two other users (A and C)[7].

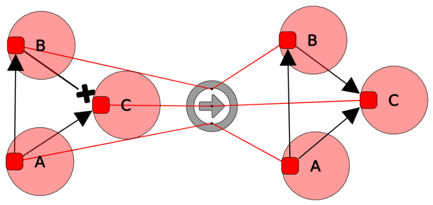
[5] This situation can produce some sort of transitivity as "the idol of my idol is my idol", meaning that A is much likely to influence C. We use here the term "idol" instead of the more classical "friend" because we only consider single-sided relations.

[6] When in this position, the users B and C might start exchanging (similar connections, common interests...), thus creating a link between the two of them.

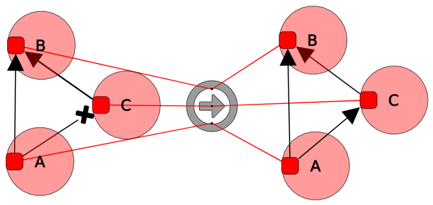
[7] This case can happen when A and C are well-versed about a common subject of interest which is of importance to B. A link is thus created between the two influential users.



(a) Rule *CommunityLegacy*.



(b) Rule *CommunityDown*.



(c) Rule *CommunityUp*.

Fig. 5. Generation of additional connections based on triads. Two distinctive edge types are used: standard arrow edges for representing existing connections and cross-shaped headed edges for indicating edges which should not exist during the matching phase.

The three rules use a `where not Edge(n,n')` condition to forbid the existence of an edge between two matching nodes.

Strategy 3 is used to drive the three rules. Like the previous strategy, this one aims at equiprobably testing all possible rule combinations.

3.4 Resulting network generation

For the sake of simplicity, the strategies presented above make equiprobable choices between rules. The probabilities may of course be modified to take into account specific conditions present in the modelled system. Whatever the chosen probabilities are, the following result holds.

Proposition 1. *Given three positive integer parameters* $|N|$, $|E|$, $|E'|$, *such that* $|N|-1 \leq |E| \leq |N| \times (|N|-1)$ *and* $|E'| \leq |E|-|N|+1$, *let the strategy* $S_{|N|,|E|,|E'|}$ *be the sequential composition of the strategies* Node generation, Edge generation *and* Community generation *described above, and* G_0 *be a port graph composed of one node with one port. The strategic graph program* $[S, G_0]$ *terminates with a simple and weakly-connected directed port graph* G *with* $|N|$ *nodes and* $|E|$ *edges.*

Proof. The termination property is a consequence of the fact that the three composed strategies have only one command which could generate an infinite derivation (the repeat loop) but in the three cases, there is a limit on the number of iterations (i.e., it is a bounded repeat).

Strategy 3: *Community generation*: creating edges to strengthen communities

```
repeat(
 ppick(
  (one($CommunityDown$))orelse(
   ppick(
    (one($CommunityUp$))orelse(one($CommunityLegacy$)),
    (one($CommunityLegacy$))orelse(one($CommunityUp$)),
    $\{0.5, 0.5\}$)),
  (one($CommunityUp$))orelse(
   ppick(
    (one($CommunityLegacy$))orelse(one($CommunityDown$)),
    (one($CommunityDown$))orelse(one($CommunityLegacy$)),
    $\{0.5, 0.5\}$)),
  (one($CommunityLegacy$))orelse(
   ppick(
    (one($CommunityDown$))orelse(one($CommunityUp$)),
    (one($CommunityUp$))orelse(one($CommunityDown$)),
    $\{0.5, 0.5\}$)),
    $\{1/3, 1/3, 1/3\}$)
)($|E| - |E'| - |N| + 1$)
```

Since the program terminates, we can use induction on the number of rewriting steps to prove that the generated port graphs are directed, simple (at most one edge in each direction between any two nodes) and weakly connected (connected when direction of edges is ignored). This is trivially true for G_0 and each rewrite step preserves these three properties, thanks to the positioning strategy that controls the out degree in *Edge generation* (Strategy 2) and the forbidden edges in the rules for *Community generation* (Figure 5). As the strategic program never fails, since a repeat strategy cannot fail, this means that a finite number of rules has been applied and the three properties hold by induction.

It remains to prove that the number of nodes and edges is as stated. Observe that by construction, the strategy *Node generation* creates a new node and a new edge at each step of the repeat loop, exactly $|N|-1$, and is the only strategy that creates new nodes. Hence, after applying the *Node generation* strategy, the graph created has exactly $|N|$ nodes and $|N|-1$ edges. The strategies *Edge generation* and *Community generation* create a new edge at each step of the repeat loop, so respectively $|E'|$ and $|E| - |E'| - |N| + 1$. As a result, when the strategy S terminates, the number of edges created is equal to $\big(|N| - 1\big) + \big(|E'|\big) + \big(|E| - |E'| - |N| + 1\big) = |E|$.

The method presented above can easily be extended to create graphs with more than one component. One has to use a number of starting nodes equal to the number of desired connected components and ensure that no edge is created between nodes from different components. The generative rules and strategies can then be applied on each component iteratively or in parallel (parallel application of rules is possible but beyond the scope of this paper).

3.5 Implementation and Experimental Validation

We use the Porgy system [39] to experiment with our generative model. The latest version of the rewriting platform[8] is available either as source code or binaries for MacOS and Windows machines.

Figures 6 and 7 are two examples of social networks generated using a sequential composition of the previous strategies. Although both graphs have the same number of nodes and edges ($|N| = 100$ and $|E| = 500$), they have been generated with different $|E'|$, respectively $|E'| = 50$ for Fig. 6 and $|E'| = 0$ for Fig. 7. This changes the number of purely random edges created in the resulting graph and explains why the first graph seems to visually present less structure than the other one. Conversely, a graph with only randomly assigned edges could be generated with $|E'| = |E| - |N| + 1$.

To ensure that our constructions present characteristics of real-world social networks, we have performed several generations using different parameters and

[8] Porgy website: `http://porgy.labri.fr`

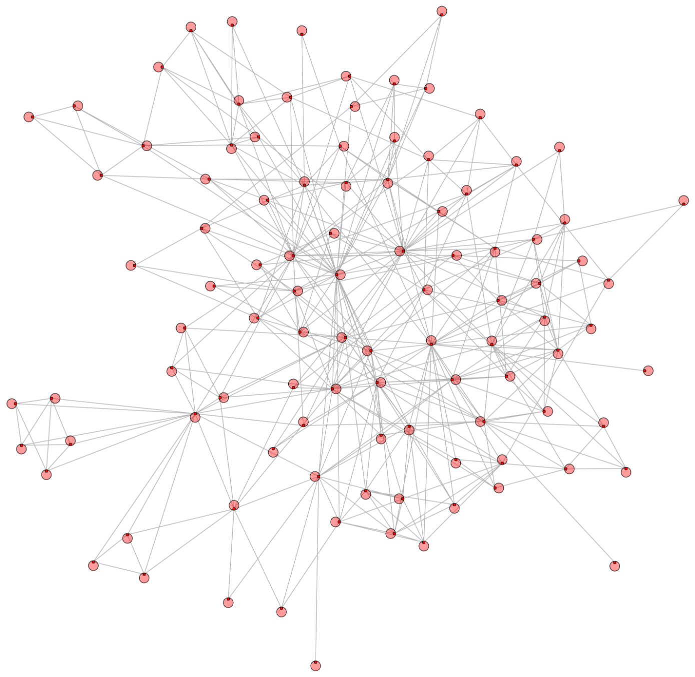


Fig. 6. A generated social network. $|N| = 100$ nodes, $|E| = 500$ edges and $|E'| = 50$. With these parameters, the average characteristic path length is $L \simeq 2.563$ and the average clustering coefficient is $C \simeq 0.426$.

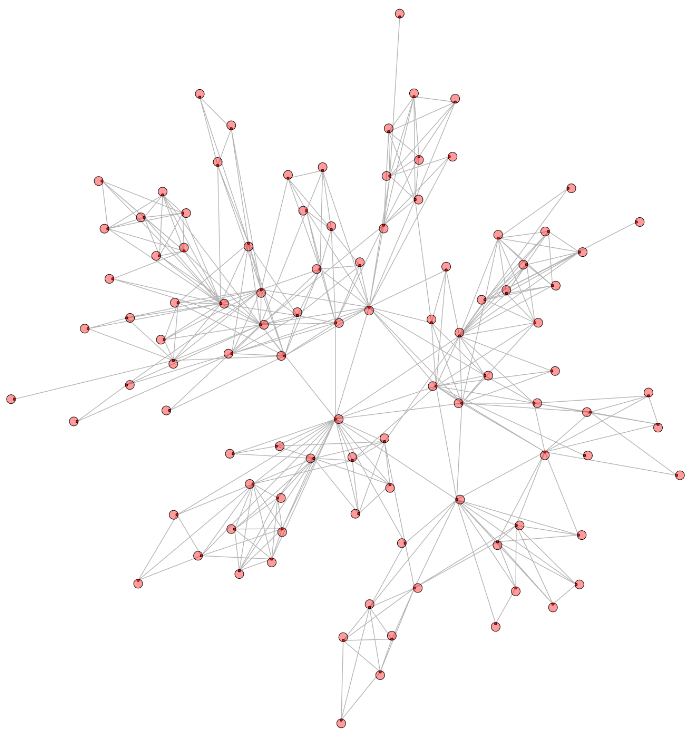


Fig. 7. A generated social network. $|N| = 100$ nodes, $|E| = 500$ edges and $|E'| = 0$. With these parameters, the average characteristic path length is $L \simeq 3.372$ and the average clustering coefficient is $C \simeq 0.596$.

measured the *characteristic path length* (the average number of edges in the shortest path between any two nodes in the graph) and the *clustering coefficient* (how many neighbours of a node n are also connected with each other) as defined in [49]. In a typical random graph, e.g., a graph generated using the Erdös–Rényi model [18] or using our method with the parameters $|N| = 100$ nodes, $|E| = 500$ edges and $|E'| = |E| - |N| + 1 = 401$, the average characteristic path length is very short ($L \simeq 2.274$), allowing information to go quickly from one node to another, but the clustering coefficient is low ($C \simeq 0.101$), implying the lack of well-developed communities. However, with the parameters used in Figure 6 (respectively, Figure 7), we retain a short characteristic path length $L \simeq 2.563$ (resp. $L \simeq 3.372$) while increasing the clustering coefficient $C \simeq 0.426$ (resp. $C \simeq 0.596$), thus matching the characteristics of small-world graphs: a small diameter and strong local clustering [49].

The graphs generated using our method can be subsequently used as any randomly generated network. For instance, we have used such graphs in [46] to study the evolution of different information propagation models.

4 Propagation in social networks

In social networks, propagation occurs when users perform a specific action (such as relaying information, announcing an event, spreading gossip, sharing a video clip), thus becoming *active*. Their neighbours are then informed of their state change, and are offered the possibility to become active themselves if they perform the same action. The process then reiterates as newly active neighbours share the information with their own neighbours, propagating the activation from peer to peer throughout the whole network.

To replicate this phenomena, some propagation models opt for entirely probabilistic activations (e.g., [30, 13, 50]), where the presence of only one active neighbour is often enough to allow the propagation to occur, while others (e.g., [26, 31, 48]) use threshold values, building up during the propagation. Such values represent the *influence* of one user on his neighbours or the tolerance towards performing a given action (the more requests a user gets, the more inclined he becomes to either activate or utterly resist). In general, several propagations may happen in one network at the same time, but most propagation models focus only on one action (e.g., relaying a specific information) as the other propagations are likely to be about entirely different subjects, thus creating little if any interference.

In [46], two basic propagation models were specified using strategic rewriting: the *independent cascade* model **IC** [30] and the *linear threshold* model **LT** [26]. In this section, we recall the definition of these two models.

In order to make it easier to compare them, we first extract common features that are used in our specifications:

– We assume that, at any given time, each node is in a precise *state*, which determines its involvement in the current spreading of information. States are represented by one of the following values: *unaware* for those who have not (yet) heard of the action, *informed* to describe those who have been informed of the action/influenced by their neighbours, or *active* to qualify those who have been successfully influenced and decided to propagate the action themselves. We encode this information on each node using an attribute *State*, which can take one of these three values as a string of characters: `unaware`, `informed`, or `active`. For visualisation purposes, an attribute *Colour* is associated to *State* to colour the nodes in `red`, `blue`, or `green`, respectively.
– The rules we use to express the models describe how the nodes' states evolve. An `unaware` node becomes `informed` when at least one of its `active` neighbours tries to influence it, and an `informed` node becomes `active` when its influence level is sufficiently high. These two distinct steps correspond to the two basic *State* transformations we need to represent using the rewrite rules. We name the first step the *influence trial*, during which an `active` node n tries to influence an inactive neighbour n' (where n' is either `unaware` or just `informed`). The following step is the *activation* of n', where the node becomes `active` once it has been successfully influenced.

– For each model, we use an attribute called *Tau* to store the influence level of the `informed` nodes. Computed/updated during the *Influence trial* step, this attribute is by default initialised to -1 and can take a numerical value in $[-1, 1]$.

With each model, we introduce visual representations of the rules applied to perform the rewriting operations. We mention in their left-hand sides the attributes that are used in the matching process, and in their right-hand sides the attributes whose values are modified during the rewriting step. The specifications are detailed hereafter.

4.1 The independent cascade model (IC)

We first describe a basic form of the **IC** model as introduced in [30]. This model has several variants (*e.g.* [25, 48]) allowing, for instance, to simulate the propagation of diverging opinions in a social network [13].

Quoting from [30], the model is described as follows: "We again start with an initial set of active nodes A_0, and the process unfolds in discrete steps according to the following randomised rule. When node v first becomes active in step t, it is given a single chance to activate each currently inactive neighbour w; it succeeds with a probability $p_{v,w}$ (a parameter of the system) independently of the history thus far. (If w has multiple newly activated neighbours, their attempts are sequenced in an arbitrary order.) If v succeeds, then w will become active in step $t + 1$; but whether or not v succeeds, it cannot make any further attempts to activate w in subsequent rounds. Again, the process runs until no more activations are possible."

Studying this description, we identify the subsequent properties which must be satisfied at each step t where an active node v is selected:

IC.1 v is given a single chance to activate each inactive neighbour w
IC.2 v succeeds in activating w with a probability $p_{v,w}$
IC.3 attempts of v to activate its inactive neighbours are performed in arbitrary order
IC.4 if v succeeds in activating w at step t, w must be considered as an active node in step $t + 1$
IC.5 the process ends if no more activations are possible.

We now present an implementation of the **IC** model using our formalism, and show that it complies with the properties stated above. First, we introduce the notations and main ideas: let us assume that for each pair of adjacent nodes (n, n'), the influence probability from n on n' is given; it is denoted $p_{n,n'}$ where $0 \leq p_{n,n'} \leq 1$. Note that $p_{n,n'}$ is history independent (its value is fixed regardless of the operations performed beforehand), and non symmetric, i.e., $p_{n,n'}$ does not have to be equal to $p_{n',n}$.

Let $N_0 \subset N$ be the subset of nodes initially active, N_k be the set of active nodes at step k, and ξ_k be the set of ordered pairs (n, n') subjected to a propagation from n (active) towards n' (inactive).

The set N_k of nodes is computed from N_{k-1} by adding nodes as follows.

- We consider an active node $n \in N_{k-1}$ and an inactive node n' ($\notin N_{k-1}$) adjacent to n but whom n has not tried to influence yet: $n' \in Ngb(n) \setminus N_{k-1}$, and $(n, n') \notin \xi_{k-1}$. A given node n is only offered one chance to influence each of its neighbours, and it succeeds with a probability $p_{n,n'}$; thus we add the pair (n, n') to ξ_k to avoid repeating the same propagation.
- If the adjacent node n' is successfully activated, it is added to the set of active nodes N_k.

This process continues until no more activations can be performed, that is when ξ_k contains all the possible pairs (n, n') where n belongs to the current set of active nodes and n' is an inactive neighbour. The order used to choose the nodes n and their neighbours during the propagation is arbitrary.

Attributes To take into account the specificities of **IC**, we need a few additional attributes. First, two attributes are needed for each edge going from n to n': *Influence*, ranging on $[0, 1]$, which gives the influence probability from n on n' (i.e., $p_{n,n'}$), and *Marked*, taking for value 0 or 1, which is used to indicate whether the given pair (n, n') has already been considered, thus avoiding multiple influence tentatives; $Marked$ is equal to 1 if $(n, n') \in \xi$, and 0 otherwise.

The attribute *Tau*, ranging on $[-1, 1]$, is used to measure how influenced a given node is. Initially, the few preset `active` nodes have their attribute $\mathit{Tau} = 1$, while `unaware` ones see their attribute *Tau* set to -1. During the propagation, the value of the attribute *Tau* is updated by a first rewrite rule, called *IC influence trial*, in order to reflect the influence probability $p_{n,n'}$, stored in the *Influence* attribute:

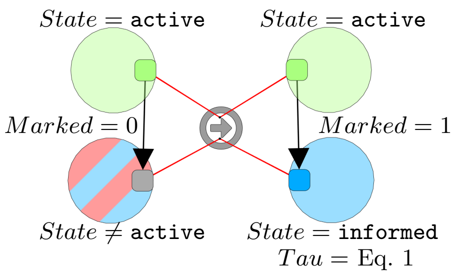
$$\mathit{Tau} = \mathit{Influence} - \mathit{random}(0, 1) \tag{1}$$

where $\mathit{random}(0, 1)$ is a random number in $[0, 1[$. We design the Equation 1 such that when a node is successfully influenced and ready to become active, the value of its attribute *Tau* is greater or equal to 0 ($\mathit{Tau} \geq 0$). This is because n' has a probability $p_{n,n'}$ of becoming active (where $p_{n,n'}$ is given as the value of the attribute *Influence*). A random number $\mathit{random}(0, 1)$ is thus chosen in an equi-probabilistic way and compared to the value of *Influence*. As a result, *Influence* is greater than or equal to $\mathit{random}(0, 1)$ in $p_{n,n'}$% of cases, so $\mathit{Tau} = \mathit{Influence} - \mathit{random}(0, 1)$ is greater or equal to 0 in $p_{n,n'}$% of cases.

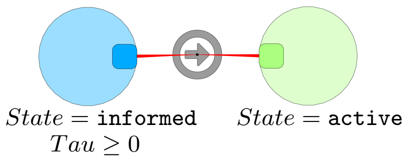
Rewrite rules The rewrite rules used to represent the **IC** model are given in Figure 8. The first one, Rule *IC influence trial* (Fig. 8(a)), shows a pair of connected nodes in the left-hand side and their corresponding replacements in the right-hand side. The `active` node n (in `green`) is connected to the node n', initially `unaware` (in `red`), or already `informed` (in `blue`) by another neighbour, through an *unmarked* edge (its attribute *Marked* is equal to 0). In the right-hand side, n remains unchanged while n' becomes or stays `blue` to visually indicate that it has been *influenced* by n and `informed` of the propagation. The updated influence level Tau of n' in the right-hand side is set according to Equation 1.

Furthermore, the directed edge linking the two port nodes is *marked*, by setting to 1 the attribute *Marked*.

Rule *IC activate* in Figure 8(b) is then applied on a single node n. If n has been sufficiently influenced, i.e., if its attribute *Tau* is greater than 0, then its state is changed, going from `informed` (`blue`) to `active` (`green`).



(a) *IC influence trial*: influence from an `active` neighbour on an *inactive* node (either `unaware` or just `informed`).



(b) *IC activate*: an `informed` node becomes `active` when sufficiently influenced.

Fig. 8. Rules used to express the Independent Cascade model (**IC**): `active` nodes are depicted in `green`, `informed` nodes in `blue` and `unaware` nodes in `red`. A bi-colour `red`/`blue` node can be matched to either of the two corresponding states (`unaware` or `informed`).

Strategy Application of the rules describing **IC** is controlled by Strategy 4.

Strategy 4: Strategy progressive IC propagation

```
1 setPos(all(property(crtGraph, node, $State == active$)));
2 repeat(
3   one($IC\ influence\ trial$);
4   try(one($IC\ activate$))
5 )
```

The first instruction exclusively selects all the nodes whose *State* is `active` and adds them to the position P (see Definition 5). The first instruction in the body of the repeat command (line 3) then performs a located rewriting operation[9] (see Definition 6). An `active` node is used as a mandatory element

[9] We recall that a rule can only be applied if the matching subgraph contains at least one node belonging to the position P, and no element belonging to the banned set Q.

from P when calling the *IC influence trial* rule to rewrite a pair of active/inactive neighbours. To comply with the original model, the attempts of activation are performed in an arbitrary order (Lemma 1) and can only occur once between each possible pair of active/inactive nodes (Lemma 2).[10]

Lemma 1 (IC.3). *Attempts of an active node n to activate its inactive neighbours n' are performed in arbitrary order.*

Proof. Because the *IC influence trial* rule is applied using the construct `one`(), for each rule application, the elements corresponding to the left-hand side are chosen arbitrarily among the matching possibilities.

Lemma 2 (IC.1). *Each active node n is given a single chance to activate its inactive neighbour n'.*

Proof. The pair (n, n') can only be chosen by the *IC influence trial* rule if the directed edge going from n to n' is unmarked (*Marked* is equal to 0). As the rule application results in the marking of the directed edge between n and n' (*Marked* = 1), it also limits to one the number of influence attempts for each pair of active-inactive neighbours since no other rule resets the marked edge.

The application of the rule *IC influence trial* results in the `active` node remaining unchanged while the inactive node becomes `informed`. Additionally, all the nodes in the right-hand side of the rule follow the default behaviour described in Definition 6 and are consequently added to the current position subgraph P.

The strategy proceeds and the *IC activate* rule (Fig. 8(b)) is then immediately applied (line 4) to try to activate an influenced node in P. According to the semantics of the `try` command, if there exists one `informed` node in P where $Tau \geq 0$ then the *IC activate* rule is applied and the node becomes `active`, otherwise, if no proper candidate is identified as match, the rule cannot apply and no new node becomes `active`, but the strategy does not fail. As discussed earlier, the activation condition reliably respect the initial model (Lemma 3) and the newly `active` nodes are immediately considered in the next loop iteration (Lemma 4).

Lemma 3 (IC.2). *Each active node n succeeds in activating its inactive neighbour n' with a probability $p_{n,n'}$.*

Proof. Rule *IC activate* can only be applied on n' once the node has been successfully influenced in rule *IC influence trial.* This occurs when the value of the attribute *Tau* is greater than 0, a result effectively happening with a probability $p_{n,n'}$: see the computation of Tau defined above (Equation 1 in section 4.1).

Lemma 4 (IC.4). *If the active node n succeeds in activating its neighbour n' at step k, n' must be considered as an active node at step $k + 1$.*

[10] Note that inactive nodes may still be influenced several times but only when selected by *different* active neighbours.

Proof. All nodes in the right-hand side of the rules are put in P by default, including the newly influenced or active nodes. Considering the repeat loop, as the *IC influence trial* rule is applied directly after the *IC activate* rule with no modification of the position set occurring in-between, if the influenced node n' becomes `active` through rule *IC activate*, then the now `active` node is added to P. Thus, it is an eligible candidate for the `active` node in rule *IC influence trial* during the next iteration of the loop.

With the repeat loop closing after the *IC activate* rule, the whole process is then repeated until the propagation phenomenon comes to an end (Lemma 5). As all the eligible edges are marked and all the possible influences and activations have been performed, the rule applications can no longer find suitable candidates, the repeat loop stops and the strategy terminates as further detailed in Proposition 2.

Lemma 5 (IC.5). *The process ends if there exists no pair of adjacent nodes n, n' such that n is active, n' is inactive and n has not tried to activate n'.*

Proof. The semantics of the repeat loop guarantees that if a command inside the body fails, the loop is terminated. The command `one`($IC\ influence\ trial$) fails when no unmarked pair of nodes (active, inactive) exists in the current graph. Then the repeat loop stops and the program terminates.

Proposition 2 (IC termination). *If the network is finite, the strategic rewrite program given by the rules in Figure 8 and Strategy 4 terminates.*

Proof. If the initial set of active nodes is empty, the strategic program immediately terminates without changing the graph. Otherwise the repeat loop starts with a non-empty position subgraph P containing all the active nodes (line 1 in Strategy 4), P represents the set N_0. Termination is a consequence of the iterative construction of sets N_k and ξ_k: at each completed iteration of the repeat loop, the set ξ_k of marked pairs of nodes (active, inactive) strictly increases, thanks to *IC influence trial* whereas the set of active nodes N_k increases or remains constant, thanks to *IC activate.*

Since no edge is added to the graph in the process, if the initial network is finite then rule *IC influence trial* eventually fails (the set of unmarked edges is strictly decreasing in size at each iteration since $|\xi_k| < |\xi_{k+1}|$) causing the repeat loop to end. Thus the program terminates.

In the following proposition, we summarise and prove the properties of our strategic rewrite program.

Proposition 3 (IC properties). *The propagation process defined by the rules in Figure 8 and Strategy 4 proceeds by iteration such that:*

1. *each active node n is given a single chance to activate its inactive neighbours n' and these attempts are performed in arbitrary order;*
2. *each active node n succeeds in activating its inactive neighbour n' with a probability $p_{n,n'}$;*

3. *if the active node n succeeds in activating its neighbour n' at step k, n' is considered as an active node at step $k+1$;*
4. *the process ends if and only if there exists no pair of adjacent nodes n, n' such that n is active, n' is inactive and n has not tried to activate n'.*

Proof. Let us prove each point in turn. Point 1 is proved by Lemmas 2 and 1, Point 2 by Lemma 3, Point 3 by Lemma 3. The 'if' part of Point 4 is proved in Lemma 5. Conversely, we can show that if the process has ended then all pairs of nodes (active, inactive) in the network have been considered: assume by contradiction that one such pair remains, there would then be an unmarked pair on which `one`(*IC influence trial*) succeeds, contradicting the assumption that the process has ended.

4.2 The linear threshold model (LT)

In the second propagation model, the **LT** model, the node activation process takes into account the neighbours' combined influence and threshold values to determine whether an informed node can become active or not. While [30] also explores the threshold model and cites publications describing such models, the definition considered in this section is based on a generalised version described in [26].[11]

Quoting the general model description from [26]: "At a given timestamp, each node is either active (an adopter of the innovation, or a customer which already purchased the product) or inactive, and each node's tendency to become active increases monotonically as more of its neighbours become active. Time unfolds deterministically in discrete steps. As time unfolds, more and more of neighbours of an inactive node u may become active, eventually making u become active, and u's activation may in turn trigger further activations by nodes to which u is connected. In the General Threshold Model each node u has a monotone activation function $f_u : 2^{N(u)} \to [0,1]$, from the set of neighbours N of u, to real numbers in $[0,1]$, and a threshold θ_u, chosen independently and uniformly at random from the interval $[0,1]$. A node u becomes active at time $t+1$ if $f_u(S) \geq \theta_u$, where S is the set of neighbours of u that are active at time t."

As previously we can identify and rephrase the following properties. At each step t where a node u is selected:

LT.1 The node u has a monotone activation function $f_u(S)$ computing its *active* neighbours' joint influence value.
LT.2 An inactive node u becomes *active* at step $t+1$ if its neighbours' joint influence ($f_u(S)$) exceeds its threshold value (θ_u).
LT.3 When u becomes active, its influence must be considered on its inactive neighbours.
LT.4 The process ends if no more activations are possible. [12]

[11] While the authors propose several alternative versions of their generalised **LT** model, we only consider one of the depicted instances, namely, the first one.
[12] Note that this characteristic is not explicitly mentioned in [26].

The **LT** model is more difficult to understand than the **IC** model as the description proposed above is less precise than the one we excerpted from the **IC** model. Nevertheless, it is possible to follow a similar approach to implement this description in our formalism and notations. Here again, two different operations are used to perform the propagation: for each inactive node n', we compute the joint influence of its active neighbours, then, if the influence n' is subjected to exceeds a threshold value, the node becomes active.

Let $p_{n,n'}$ be the influence probability of n on n' ($0 \leq p_{n,n'} \leq 1$) and $\theta_{n'}$ the *threshold* value of n', i.e., the resistance of n' to its neighbours' influence, chosen independently from n' and randomly in $[0, 1[$. Let also $S_{n'}(k)$ denote the set of nodes currently active at step k and adjacent to n', and $p_{n'}\big(S_{n'}(k)\big)$ the joint influence on n' of its active neighbours at step k. In our specification, the function $p_{n'}\big(S_{n'}(k)\big)$ corresponds to the monotone activation function $f_u : 2^{N(u)} \rightarrow [0, 1]$ described in [26]. The **LT** propagation thus operates as follows: let $N_0 \subset N$ be the subset of nodes initially active, N_k be the set of active nodes at step k, and ξ_k be the set of ordered pairs (n, n') subjected to a propagation from n (active) towards n' (inactive). The set N_k of nodes is computed from N_{k-1}, by adding nodes as follows. Let us consider an active node $n \in N_{k-1}$ and an inactive node n' ($\notin N_{k-1}$) adjacent to n but whom n has not tried to influence yet:

– The inactive node $n' \notin N_{k-1}$ has its *active* neighbours' joint influence value computed using the formula: $p_{n'}\big(S_{n'}(k)\big) = 1 - \prod_{n \in S_{n'}(k)}(1 - p_{n,n'})$ where $S_{n'}(k) = Ngb(n') \cap N_{k-1}$ (the active neighbours of n').
– The inactive node n' becomes *active* at step k when its neighbours' joint influence exceeds the threshold value, i.e., $p_{n'}\big(S_{n'}(k)\big) \geq \theta_{n'}$, leading n' to be added to N_k.

To simplify the following mathematical formulas and considering we only deal with transformation occurring at the most recent step k at all time, we use the notation $S_{n'}$ instead of $S_{n'}(k)$. This process continues until, for all the joint influences up-to-date, no more activation can be performed.

As for **IC**, the **LT** propagation takes place in two phases: influence computation followed by activation. Before presenting the corresponding rules, we need to specify more precisely the properties of the intended propagation model from [26], as the authors present several propagation models with multiple definitions of the influence and joint influence probability of n over n' (i.e., $p_{n,n'}$ and $p_{n'}(S_{n'})$). In this paper, we are implementing the static propagation model where $p_{n,n'}$ is expressed as a constant value. Because the activation of a specific node n' is dependent of the influence probabilities coming from each of its active neighbours, we need to update their joint influence $p_{n'}(S_{n'})$ whenever one of the previously inactive neighbours of n' activates. This operation is performed using the formula $p_{n'}\big(S_{n'} \cup \{n\}\big)$, introduced in the original paper.

$$p_{n'}\big(S_{n'} \cup \{n\}\big) = p_{n'}(S_{n'}) + \big(1 - p_{n'}(S_{n'})\big) \times p_{n,n'} \tag{2}$$

This equation adds the influence of n among the other active nodes adjacent to n' (where $n \notin S_{n'}$).

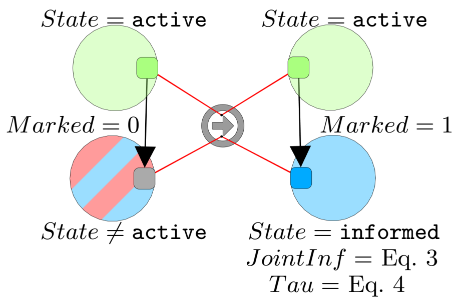
Attributes In order to take into account these specificities, two new attributes are needed in addition to the ones introduced earlier in **IC** (*State*, *Colour*, *Influence*, *Marked* and *Tau*). Each node is now also provided with a threshold value, stored in the attribute *Theta*, whose value is in $[0, 1]$. The joint influence probability, measuring the influence level an inactive node is subjected to, is stored using the attribute *JointInf*. Initially, the `active` nodes have their attributes $JointInf = 1$, while `unaware` ones have $JointInf = 0$. During the influence step, the value of the attribute *JointInf* on the node being `informed` is updated as specified by Equation 2, which is translated to the following formula when using the appropriate attributes:

$$JointInf = JointInf_{\text{OLD}} + (1 - JointInf_{\text{OLD}}) \times Influence \qquad (3)$$

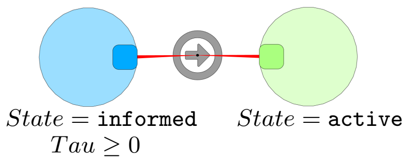
We can then compare this updated joint influence value for a node n' with its threshold value, stored in *Theta* and assign the result to the attribute *Tau*:

$$Tau = JointInf - Theta \qquad (4)$$

If, for an `informed` node n', $Tau \geq 0$, then the joint influence of its neighbours ($JointInf$) exceeds its threshold value ($Theta$), thus leading n' to endorse the propagation subject and to activate to spread this very information to all of its neighbours.



(a) *LT influence trial*: Joint influence computation from an `active` neighbour on an *inactive* node (either `unaware` or just `informed`).



(b) *LT activate*: an `informed` node becomes `active` when sufficiently influenced.

Fig. 9. Rules used to express the Linear Threshold model **LT**. Colours have the same meaning as previously: `active` nodes are `green`, `informed` nodes are `blue` and `unaware` nodes are `red`. A bi-colour `red/blue` node can be in either of the two states `unaware` or `informed`.

Rewrite rules The rules for the **LT** model are quite similar to those introduced in the **IC** model. The first rule, *LT influence trial* (Figure 9(a)), is applied on a connected pair of active-inactive nodes (respectively `green` and `red`/`blue`). During its application, the rule transforms an inactive node n' into an `informed` node as its `active` neighbour n tries to influence it. Two computations are performed in order to update the attributes of the inactive node n'. The first one updates the joint influence of n' by adding the influence of the `active` node n to the total influence using Equation 3. This operation respects the property of the **LT** model as shown in Lemma 6. Once *JointInf* is revised, *Tau* is calculated by comparing n' joint influence (*JointInf*) and its threshold value (*Theta*).

Lemma 6 (LT.1). *An inactive node n has a monotone activation function computing its* active *neighbours' joint influence value.*

Proof. The value of *JointInf* is changed by the rule *LT influence trial* using the new *Influence* value to consider to update the previous joint influence probability measure; this is given by Equation 3, according to which $\mathit{JointInf} \geq \mathit{JointInf}_{\text{OLD}}$, as $\mathit{JointInf}_{\text{OLD}}$ and *Influence* are both defined in $[0, 1]$. Consequently, the activation function is monotone.

The second rule, named *LT activate* (Figure 9(b)), is identical to the *IC activate* rule shown in Figure 8(b). A successfully influenced node, identified by the positive value of its *Tau* attribute, has simply its *State* attribute value changed from `informed` (`blue`) to `active` (`green`).

Strategy We use the rewriting Strategy 5 to manage the rules application similarly to the **IC** model. Overall, the two strategies (used for **IC** and **LT**) follow the same design and only vary by applying different rules.

Strategy 5: Strategy LT propagation

```
1 setPos(all(property(crtGraph, node, $State == active$)));
2 repeat(
3   one($LT\ influence\ trial$);
4   try(one($LT\ activate$))
5 )
```

As in the previous model, we start by defining a position P which gathers all the `active` nodes (line 1), then a repeat command (line 2) to compute the propagation as many times as possible. One of the active nodes is considered and we apply the *LT influence trial* rule (line 3) on it and one of its inactive neighbours. At the end of the rewriting operation, these two nodes follow the default behaviour of the right-hand side elements and are added to P.

We then try to apply the *LT activate* rule (line 4) on an informed node whose *Tau* attribute value exceeds or equals 0. If there exists no node which has been successfully influenced (whose *Tau* attribute value is lower than 0), the *LT activate* rule is not applied. Otherwise, the activation follows the conditions expressed in the model description as shown in the next Lemma.

Lemma 7 (LT.2). *An inactive node n becomes* active *if its neighbours' joint influence exceeds its threshold value.*

Proof. The attribute Tau *is used to stock the comparison result between the attributes* JointInf *and* Theta *as described in Equation 4. As the rule* LT activate *is only applied when* JointInf *is greater or equal to* Theta*, the inactive node can only become* `active` *if this condition is verified.*

The newly active node is then added to the subgraph P and can be considered to start influencing its own neighbours in the next iteration of the repeat loop.

Lemma 8 (LT.3). *When n becomes active, its influence must be considered by its inactive neighbours*

Proof. Once a node n is `active`, it can be considered as a candidate in the rule *LT influence trial* with one of its inactive neighbour n' which has been selected to be influenced. During the application, the edge between the two nodes is marked to avoid successive influence trials from n to n', and, since no other rule puts the mark back to 0 once it has been set to 1, the rule can only be applied once on this pair. As the rule *LT influence trial* is the only one which can stop the strategy, it is applied as many times as possible, thus considering all the possible pairs of active-inactive nodes and successfully guaranteeing that each `active` node influences all its inactive neighbours once.

Although the original model does not specify when the propagation comes to an end, we consider that once the `active` nodes have tried to influence all their existing inactive neighbours, no more changes can occur in the network and the propagation can no longer continue. This is expressed in the next Lemma, while the corresponding termination of the graph rewrite program is defined in Proposition 4.

Lemma 9 (LT.4). *The process ends if there exists no pair of adjacent nodes n, n' such that n is active, n' is inactive and sufficiently influenced.*

Proof. Same proof as Lemma 5 just by changing the rules' names.

Proposition 4 (LT termination). *If the network is finite, the strategic rewrite program given by the rules in Figure 9 and Strategy 5 terminates.*

Proof. Same proof as Proposition 2 just by changing the rules' names.

The following proposition summarises the properties of our strategic rewrite program.

Proposition 5 (LT properties). *The propagation process proceeds by iteration in discrete steps. For any pair of adjacent nodes n, n' such that at some step k during the propagation n is active and n' is inactive:*

1. *An inactive node n has a monotone activation function $(p_{n'}(S_{n'}(k)))$ computing its* active *neighbours' joint influence value.*

2. *An inactive node n becomes* active *if its neighbours' joint influence exceeds its threshold value.*
3. *When n becomes active, its influence must be considered on its inactive neighbours.*
4. *The process ends if and only if there exists no pair of adjacent nodes n, n' such that n is active, n' is inactive and n has not tried to activate n'.*

Proof. Let us prove each point in turn. Point 1 is proved by Lemma 6, Point 2 by Lemma 7, Point 3 by Lemma 8. The 'if' part of Point 4 is proved in Lemma 9. Conversely, we can show that if the process has ended, then all pairs of nodes (active, inactive) in the network have been considered; assume by contradiction that one such pair remains, there would then be an unmarked pair on which `one`(*LT influence trial*) could be applied, thus contradicting the assumption that the process has ended.

5 Dissemination in Networks

We now turn our attention to dissemination algorithms, which spread information within a network (not necessarily a social network) in an automatic way. Unlike propagation models, dissemination models do not aim at replicating social behaviours but instead spread the information automatically.

In this section we consider a dissemination model, called *Riposte* (**RP**), described in [24].

In this model, it is considered that an information deemed interesting by a sufficiently large fraction of the population is more likely to appeal widely to other individuals, whereas an information that only a few people consider interesting will not engage others beyond the set of users initially exposed to it. Moreover, when observing the information dissemination process (more precisely, the users' re-posts), one cannot determine with sufficient confidence the opinion of any single user concerning the information that is disseminated.

In this section, we first implement the **RP** diffusion algorithm as a strategic graph program. Then, we show how to easily develop a new dissemination model incorporating features of **LT** and **RP**.

5.1 Riposte (RP): a privacy preserving dissemination model

RP differs from the two models seen previously as it is not a propagation model. However, as a diffusion model, it still follows the characteristic principle of randomly driven activations encountered in **IC**, while introducing some key variations. First of all, its activation and spreading mechanisms are not directly linked: both active or inactive users can be considered as starting point to transmit information, and active users are not automatically assumed to spread information to their neighbours. These features confer to **RP** the property of *plausible deniability*, which is essential to preserve the users' privacy. Indeed, independently of the user's opinions and consent concerning the information, **RP** will sometimes disseminate information to the user's neighbours. The user's opinion influences the probability of sharing in order to favour topics deemed interesting by most people, but with this model, witnesses observing the exchanges within the network can now no longer precisely pinpoint which users have supported the diffusion and have intended to share the information with their neighbours. Finally, conversely to **LT**, **RP** does not take into account the *influence* from one user upon another, but considers instead the personal *interest* a given user has in the information.

Quoting the model description given in [24]: "Let G denote the (directed) graph modeling the social network, and n be the total number of users, and suppose that some (small) initial set of users learn an information item t. For each user u that learns t, Riposte decides to either repost t, to all u's outgoing neighbours in G, or to not repost t, to anyone. The decision is randomised and depends on the user's (private) opinion on the information, and the number of the user's neighbours that have not received the information yet. Precisely, if u likes t, then t is reposted with probability λ/s_u , and if u does not like t, then t is reposted with a (smaller) probability δ/s_u, where $0 < \delta < 1 < \lambda$ are global parameters of the dissemination mechanism, and s_u is an upper bound on the number of u's outgoing neighbours that have not received t yet. [...] The process either finishes after a finite number of steps, when no individuals are left [*to be informed*], or continues forever."

Rephrasing this description, we can isolate the following characteristics:

RP.1 For each user u that learns an information item, the Riposte algorithm either reposts it to all u's outgoing neighbours, or it does not repost it to any of them.

RP.2 If u likes the information item, it is reposted to all of u's neighbours with a probability λ/s_u; if u does not like it, the information is reposted with a (smaller) probability δ/s_u.

RP.3 The process either terminates after a finite number of steps, when no more diffusion is possible, or continues forever.

To implement this description in our formalism and notations, new parameters are needed to reflect these characteristics. First, let p_n be the probability given for a specific information to be re-posted by the node n. The value of p_n can be seen as a measure of how interesting the information is to n. Then, in order to prevent revealing the opinion of individual users, some randomness concerning the information diffusion is incorporated. Let δ and λ be the dissemination model global parameters where $0 < \delta < 1 < \lambda$. We define $\overline{S_n}$ as the set of nodes currently unaware of the information and adjacent to n. After being informed by one of its neighbours, two different behaviours are possible. If n wishes to diffuse the information (that is, n becomes active), then either all its neighbours are informed of it with a probability $\lambda/\overline{S_n}$. Alternatively, if n does not wish to spread the information (thus, n remains "simply" informed), then the information can still be passed to all its neighbours, but this time with a weaker probability $\delta/\overline{S_n}$.

Let $D_k \subseteq N$ the set of nodes aware of the information being diffused at step k, with D_0 being the set of nodes used as a source for the dissemination process. We define over D_k the set $M_k \subseteq D_k$ which contains the nodes having been considered by the algorithm to try to spread the information to their neighbours up to step k; as no node is initially considered, M_0 starts empty. For each new step k, the set D_k and M_k are computed incrementally from D_{k-1} and M_{k-1} as follows:

– a node $n \in D_{k-1} \setminus M_{k-1}$, who has been informed but have not yet been considered by the diffusion algorithm, is selected and is proposed to endorse the information according to its interest with a probability p_n. Having been selected, n is added to the set M_k ($M_k = M_{k-1} \cup \{n\}$).
– If n finds the information worthy, it becomes active, then all of its neighbours are informed about the item being diffused with a probability $\lambda/\overline{S_n}$ and are added to D_k. Otherwise, n remains inactive, but all its neighbours are can still be informed with a probability $\delta/\overline{S_n}$ and are consequently also added to D_k.
– This process continues until all the informed nodes have been considered by the algorithm to try to diffuse the information to their neighbours, that is, when $D_k = M_k$.

As one can see, the diffusion probability depends on both the user's opinion concerning the information (p_n) and the number of neighbours unaware of it ($\overline{S_n}$). In the original definition [24], the value $\overline{S_n}$ is an upper bound on the number of n's outgoing neighbours that have no knowledge yet of the information. However, a variant of the algorithm for systems where users are unable to know whether their neighbours have already heard of the information or not was also proposed. For such instance of application, which is our case, the probability is computed using the *total* number of n's outgoing neighbours instead of considering the upper bound of unaware neighbours.

Attributes We naturally make use of the generic *State* and *Colour* node attributes already described in the previous models, as well as *Marked* on directed edges. But here we also need to flag nodes that have already attempted to spread the information (regardless of their activation status). This information is reflected by a new node attribute called *MarkedN*, which is used to indicate which elements belong to the set M_k.

In addition to these, we introduce a few other new attributes to model the specificities of **RP**. First, the attribute *Interest* records each node's interest for an information, namely the probability p_n for an information to be re-posted by n. Then the attribute *Tau* is used to store the result of the activation decision, computed as

$$\mathit{Tau} = \mathit{Interest} - \mathit{random}(0,1) \tag{5}$$

where $\mathit{random}(0,1)$ is a number uniformly and randomly chosen in $[0,1[$. An `informed` node becomes `active` when $\mathit{Tau} \geq 0$. Initially, *Tau* is set to -1 on all the nodes before the diffusion begins and, as in the previous models, *Tau* is still the key attribute to enable node activation. This time however, *Tau* is computed using the *Interest* attribute instead of the *Influence* attribute as in **IC** and **LT**.

To perform the dissemination according to the given parameters λ and δ of the **RP** model, an additional attribute *Share* is used to store the likeliness of n to share (i.e., spread) the information. Its value is computed as follows:

$$Share = \frac{isActive(\lambda - \delta) + \delta}{OutArity} \tag{6}$$

where *isActive* is an integer set to 1 when the attribute $State =$ `active` (and set to 0 otherwise), and *OutArity* is the cardinality of the set of outgoing neighbours. Some explanations are in order:

1. let *OutArity* be the number of outgoing edges from n;
2. if n has no neighbour to transmit the information to, then *Share* does not need to be computed; we address this issue by having *OutArity* returning -1 in such case instead of 0 to avoid errors;
3. we formulate *Share* as a single expression using λ or δ depending on the value of *isActive* (otherwise, two different rules should be used with only a small variation in the computation of *Share*).

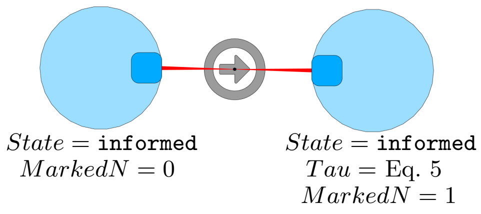
Finally, another attribute named *Sigma* is used to store the result of the sharing decision, in a way similar to *Tau*, and is computed as

$$Sigma = Share - random(0, 1) \tag{7}$$

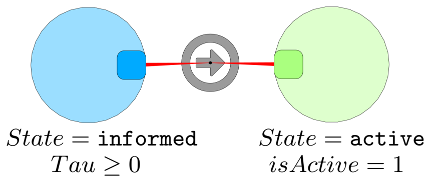
where $random(0, 1)$ is a random number chosen in $[0, 1[$. Initially, *Sigma* is set to -1 on all the nodes The information diffusion from n to all its neighbours is performed when the attribute *Sigma* of n is greater than or equal to 0. This attribute allows us to separate the activation process from the sharing process.

Although all these attributes are needed to emulate the dissemination process, it is important to note that, in real-world applications of the **RP** algorithm, the only visible information to an external observer is whether a node has heard of the information or not, i.e., if the node belongs to D_k or not. This translates to the *State* attribute marking a node as `unaware` or aware, without any distinction (such as *Colour*) between `informed` and `active` nodes.

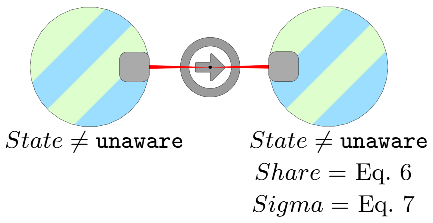
Rewrite rules Following the formal definition of the **RP** model, we can define the following steps in the dissemination mechanism given by the rules presented in Figure 10. The first rule, *RP initialisation* (Fig. 10(a)), is an opening step used to prepare the freshly `informed` nodes who did not yet try to spread the information (i.e., unmarked nodes). A node is offered the possibility to be interested in the information, with *Tau* computed accordingly (see Equation 5), and sees its *Marked* attribute set to 1. This means that the node is soon to be considered for activation and as a candidate for diffusing the information. We then keep the same `informed` node and tentatively apply the rule *RP activate* (Fig. 10(b)) on it. Depending of the previously obtained *Tau* value for n, and



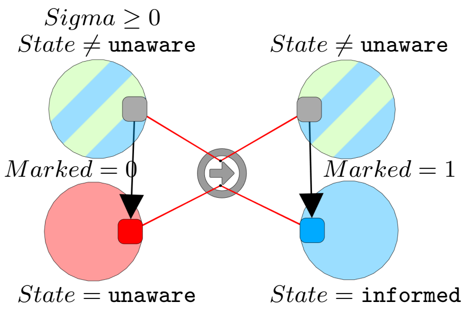
(a) *RP initialisation*: this rule is used to initialise an `informed` node aware of the information being spread.



(b) *RP activate*: an `informed` node becomes `active` if its *Tau* attribute is greater or equal to 0.



(c) *RP share trial*: whether a node is `active` or `informed`, the **RP** model can decide to use it to spread the information to others.



(d) *RP inform*: a node aware of the information (`active` or `informed`), and selected to share its knowledge, informs an `unaware` neighbour.

Fig. 10. Rules used to express the Riposte model **RP**. Colours keep their meaning from the previous propagation models: `active` nodes are `green`, `informed` nodes are `blue` and `unaware` nodes are `red`. A bi-colour `blue/green` node can be either `informed` or `active`.

more precisely if $Tau \geq 0$, its *activation* can take place, thus putting n in an `active` state and setting *isActive* to 1.

The *RP share trial* rule, shown in Figure 10(c), computes the *Share* and *Sigma* attributes of the previously considered, and either `informed` or `active`, node n. The *Share* computation, performed following Equation 6, uses *isActive* to change the probability result depending on n's current *State*. *Sigma* then reuses the *Share* value (see Equation 7) to randomly decide whether n must share the information with its neighbours. The transmission of information to the neighbours is performed by the last rule *RP inform*, depicted in Figure 10(d). An `active` or simply `informed` node n who has been selected to transmit the information (whose attribute $Sigma \geq 0$) informs an `unaware` neighbour n'. As a result, the `unaware` node becomes `informed`, leading it to be considered as a new potential information spreading source in the next dissemination step. The newly `informed` node has its *MarkedN* attribute untouched, thus still equal to its default value (0), and ready to be subjected to the *RP initialisation* rule.

Strategy The strategy used in this model is given below in Strategy 6. Much like the previous models, we use a repeat loop (line 1) in the **RP** strategy to control the rewriting steps. Recall that initially, all nodes have their attribute *MarkedN* set to 0. We initiate the strategy by choosing the node which is the focus of rewriting in the initial step. We select one which has never been considered to spread the information, that is, its *MarkedN* attribute is still equal to its default value (line 2).

Strategy 6: Riposte dissemination model **RP**

```
1 repeat(
2   setPos(one(property(crtGraph, node, State == informed && MarkedN ==
      0)));
3   one(RP initialisation);
4   try(one(RP activate));
5   one(RP share trial);
6   repeat(one(RP inform))
7 )
```

The first rule, *RP initialisation* (Fig. 10(a)), is then applied. In case not a single candidate satisfying the aforementioned conditions has been found, i.e., there is no `informed` node or all have already been considered before (with *MarkedN* = 1), then the rule application fails and the dissemination comes to an end (line 3). However, if a matching node n exists in position P, the rule is applied on it and its *Tau* attribute is computed according to Equation 5. The rewritten node is then inserted in P and ready for the next rule application.

The candidate node in P may endorse the subject being diffused and activate thanks to the *RP activate* rule (line 4). As shown in Figure 10(b), in addition to *State* as matching attribute, *Tau* is the real filtering condition to decide whether the selected `informed` node can become `active`. This operation is optional as, in **RP**, the activation and information spreading are distinct mechanisms. Thanks to the `try` construct, this rule application cannot cause the strategy to fail. We use the *isActive* attribute to store the result of the activation trial and add the rewritten node to P.

We then apply *RP share trial* (Fig. 10(c)) on the node n (line 5) which can either be `active` or just `informed` if it did not satisfy the matching conditions of *RP activate*. The transformation computes new values for n's *Share* (Eq. 6) and *Sigma* (Eq. 7) attributes while keeping the rewritten node n in P. These values indicate to the **RP** model whether to use n as a starting point to spread the information to its neighbours.

This leads us to the nested repeat loop applying *RP inform* (Fig. 10(d)) to all of n's neighbours (line 6). If the (indifferently `informed` or `active`) node has been selected to inform its `unaware` neighbours (n'), then its *Sigma* attribute is greater or equal to 0. The rule application sets the *State* attribute of n' to `informed` and, through its *Marked* attribute, marks the edge connecting the two nodes to avoid multiple applications of the rule on the same pair of nodes. While all elements of the right-hand side are added to the subgraph P by default, n is the only node whose attribute *Sigma* is greater or equal to 0; it is thus reselected for each application of *RP inform* in the loop. All the newly `informed` nodes are now eligible to be subjected to a dissemination step themselves as their *MarkedN* attributes are still equal to their default value ($\mathit{MarkedN} = 0$).

The specific application order of these rules allows us to verify the properties extracted from the original model.

Lemma 10 (RP.1). *For each node n that learns an information item, the Riposte algorithm either reposts it to all n's outgoing neighbours, or does not repost it to any of them.*

Proof. As n is informed, it is subjected to rule *RP share trial* in which a value for the attribute *Sigma* is computed. If *Sigma*'s value for n is greater or equal to 0, then rule *RP inform* is applied as many times as possible, changing the *State* of all of n's neighbours to `informed`, thus reposting the information to them. Otherwise, when *Sigma*'s value is lower than 0, nothing happens, thus the information is not reposted to anyone.

Lemma 11 (RP.2). *If n likes the information item, it is reposted to all of n's neighbours with a probability $\lambda/\overline{S_n}$; if n does not like it, the information is reposted with a (smaller) probability $\delta/\overline{S_n}$.*

Proof. A node n reposts an information when $\mathit{Sigma} \geq 0$ (see rule *RP inform*, Fig. 10(d)). However, *Sigma*'s value has a probability *Share* of being greater or equal to 0, with *Share*'s value itself ultimately depending of the attribute *IsActive* (see Equations 6 and 7), where *IsActive* indicates if n likes the information. When n likes the information item, *IsActive* is equal to 1 and *Share* is equal to $\frac{\lambda}{\mathit{OutArity}}$. Conversely, if n does not like the information, then *IsActive* is equal to 0 and *Share* is equal to $\frac{\delta}{\mathit{OutArity}}$. As expressed before, *OutArity* (the number of edges outgoing from n) is used to approximate $\overline{S_n}$ (the upper bound on the number of n's outgoing neighbours that have yet no knowledge of the information).

As in the previous models, we finally take a closer look at the termination of the process. While Lemma 12 mentions the possibility of infinite computation, if the network is finite, the computation does not go on forever as shown in Proposition 6.

Lemma 12 (RP.3). *The process either terminates after a finite number of steps, when no more diffusion is possible, or continues forever.*

Proof. Each iteration of the main repeat loop in Strategy 6 corresponds to a dissemination step k. If there is no informed node which is not marked ($MarkedN = 0$) in $D_k \setminus M_k$, the set P is empty and the process stops. Otherwise, *RP initialisation* marks the node. The strategy `repeat`(`one`($RP\ inform$)) fails if the chosen node has no unaware neighbour (then *RP inform* fails). No more diffusion is possible then and $D_k = M_k$. But during the dissemination step, new informed nodes may be added by *RP inform* which have to be taken into account in the next iteration of the main loop. So the process can go forever.

Proposition 6 (RP termination). *If the network is finite, the strategic rewrite program given by the rules in Figure 10 and Strategy 6 terminates.*

Proof. This follows from Lemma 12 and the fact that the sets M_k are always strictly growing but bounded by the size of the network.

The following proposition summarises the properties of our strategic rewrite program.

Proposition 7 (RP properties). *The dissemination process proceeds by iteration in discrete steps.*

1. *For each node n that learns an information item, the Riposte algorithm either reposts it to all n's outgoing neighbours, or does not repost it to any of them.*
2. *If n likes the information item, it is reposted to all of n's neighbours with a probability $\lambda/\overline{S_n}$; if n does not like it, the information is reposted with a (smaller) probability $\delta/\overline{S_n}$.*
3. *The process either terminates after a finite number of steps, when no more diffusion is possible, or continues forever.*

Proof. Each point follows respectively from Lemma 10, Lemma 11 and Lemma 12.

5.2 Adapting the Riposte model with linear thresholds

The strategic programs implementing **RP** and **IC** have some common features, but they differ in two aspects: first, the influence of neighbours is replaced by personal interest, second, the correlation between user activation and spread of information is mitigated through a sharing probability. Unlike **LT**, the dissemination algorithm **RP** completely ignores the influence of the neighbours, but allows users to influence the dissemination by either endorsing or rejecting the information, without exposing their true opinion to others.

While an individual may like a particular subject, he may also be influenced by friends on a topic he is not be familiar with. Therefore, we propose to develop a dissemination model merging elements from both **RP** and **LT**. This model, named *Riposte with Linear Thresholds* (**RP-LT**), hides the users' reaction (endorsement or reject) towards the information being diffused, while taking into account the influence from each user on its neighbours.

We now introduce the main elements of the new model, keeping the notations consistent with **LT** and **RP**. An inactive node n' is influenced by each of its active neighbours n according to the probability $p_{n,n'}$ and we note $p_{n'}\big(S_{n'}(k)\big)$ the joint influence endured by n' at step k from all its active neighbours $S_{n'}(k)$. The threshold value of n', or its resistance to activation, is defined as $\theta_{n'}$. Finally, λ and δ are global parameters ($0 < \delta < 1 < \lambda$), and $\overline{S_{n'}}$ is the set of unaware nodes adjacent to n'. The cardinal of this set is denoted $|\overline{S_{n'}}|$.

As in the **LT** model, a given node might need to be influenced multiple times before it becomes active. Let γ be the maximum number of times a node can be told an information before being asked to formulate his opinion. Thus, a node can be influenced at most γ times, but may decide to activate before.

Definition 7 (Dissemination process in RP-LT). *Let $p_{n,n'}$, $p_n\big(S_n(k)\big)$, θ_n, λ, δ, γ and $\overline{S_{n'}}$ be defined as above. Starting with a set of informed nodes, the model **RP-LT** disseminates information across the network such that:*

RP-LT.1 *For each user n that learns an information item, the **RP-LT** algorithm either reposts it to all n's outgoing neighbours, or does not repost it to any of them.*

RP-LT.2 *If n likes the information item, it is reposted to all of n's neighbours with a probability $\lambda/\overline{S_n}$; if n does not like it, the information is reposted with a (smaller) probability $\delta/\overline{S_n}$.*

RP-LT.3 *An inactive node is influenced at most γ times, and is thus given γ chances to endorse the information.*

RP-LT.4 *An inactive node n has a monotone activation function $(p_n\big(S_n(k)\big))$ computing its active neighbours' joint influence value.*

RP-LT.5 *An inactive node n becomes active if its neighbours' joint influence exceeds its threshold value, i.e., $p_n\big(S_n(k)\big) \geq \theta_n$.*

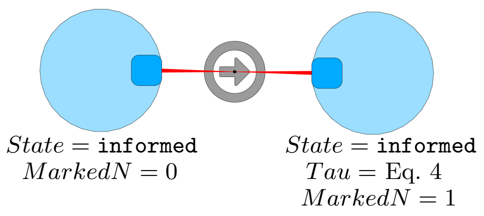
RP-LT.6 *The process terminates when no more diffusion is possible.*

Quite naturally, these properties are similar to the ones encountered in **RP** and **LT**. Only **RP-LT.3** is specific to this dissemination model. While the model description has similarities with **RP**'s (see Definition 5.1), the use of the influence ($p_{n,n'}$), joint influence ($p_n(S_n)$) and theta (θ_n) attributes distinguish the two models: where **RP** focuses on the users' interest to promote the information being disseminated, **RP-LT** looks at the influence users have on one another and their response to it.

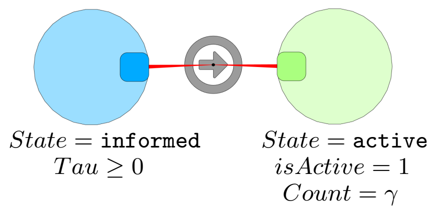
Attributes For the sake of completeness, we recall the different attributes already used in **RP** and **LT**. Obviously, we keep the general attributes: *State* and *Colour* to distinguish the nodes' states, *Marked* to mark the visited pairs of nodes, as well as *MarkedN* for the nodes previously considered for diffusion to their neighbours, and *Tau* to store the activation decision. We complete them with the attributes *Influence* to store $p_{n,n'}$; *Theta* for the threshold θ_n; *JointInf* for $p_n(S_n)$; *Share* to store the node's sharing probability according to its *State*; *isActive* to mark whether the node is active or not (used to compute *Share*); *OutArity* to request the number of outgoing neighbours; and *Sigma* (with initial value -1) to store the result of the sharing decision. The equations used to compute attributes *JointInf*, *Tau*, *Share*, and *Sigma* are given as previously in Equations 3, 4, 6, 7.

In addition, we introduce a new attribute *Count* to track the number of times a node has been informed of the information being diffused. All nodes have their *Count* attribute initialised to 0 and each node is given the same information at most γ times (from different neighbours).

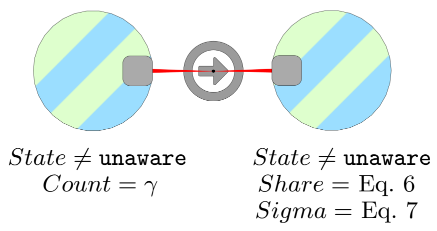
Rewrite rules The rewrite rules, given in Figure 11, are quite similar to the **RP** rules. The first rule *RP-LT initialisation* (Fig. 11(a)) updates the *Tau* attribute (according to Eq. 4) of an `informed` node. When rewritten, the node stays `informed` and its *MarkedN* attribute is set to 1.



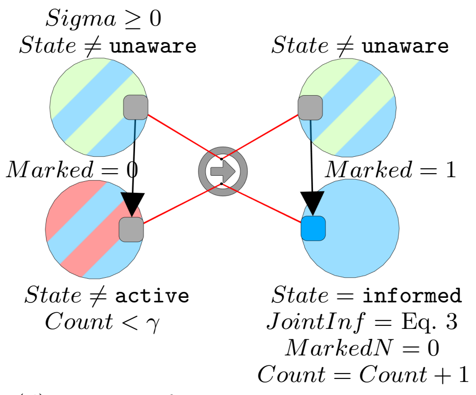
(a) *RP-LT initialisation*: the rule initialises an `informed` node, computes its *Tau* attribute and marks it using *MarkedN*.



(b) *RP-LT activate*: an `informed` node becomes `active` when its *Tau* attribute is positive.



(c) *RP-LT share trial*: whether the node is `informed` or `active`, if it has been informed γ times or made its decision, its sharing probability must be computed.



(d) *RP-LT inform*: an inactive user is informed by an `informed` or `active` neighbour. The rule computes the neighbour's *JointInf* value, resets its marker and increments its influence counter.

Fig. 11. Rules used to express the Riposte with Linear Threshold model **RP-LT**.

The second rule *RP-LT activate* (Fig. 11(b)) is in charge of the potential activations. When the *Tau* attribute indicates that the node n has been successfully influenced (when $Tau \geq 0$), then its *State* becomes `active` and the attribute *isActive* is accordingly updated to match n's current state. In case of activation, we also set the *Count* attribute to γ to indicate the node will no longer be responsive to influence.

Every node aware of the information being diffused, who either decided to activate, or who has been influenced γ times, is entitled to compute its *Share* (6) and *Sigma* (7) attribute values. This is achieved by using the *RP-LT share trial* rule (Fig. 11(c)), which applies only to nodes where *Count* equals γ.

The last rule is *RP-LT inform* (Fig. 11(d)). The `active` or `informed` node n, successfully selected to spread the information ($Sigma \geq 0$), shares it with its `unaware` or `informed` neighbours n'. To avoid multiple matching with the same pair of connected nodes, the edge between n and n' is marked. The joint influence probability of n' is updated using Equation 3 and the node is unmarked to indicate a change has happened ($MarkedN = 0$). The dissemination step is only targeting inactive nodes which have been influenced less than γ times since, after having been informed of the diffusion subject, n' should be able to form an opinion about it. When n' is rewritten, its influence counter is incremented to keep track of the operation ($Count = Count + 1$).

Strategy The rewriting operations are applied according to Strategy 7. Just like the strategies used for **IC** and **LT**, the ones defining the **RP** and **RP-LT** are very similar. For each rule application considered hereafter, we reinsert the newly rewritten elements in position P.

Strategy 7: Riposte with Linear thresholds dissemination model **RP-LT**

```
1 repeat(
2   setPos(one(property(crtGraph, node, State == informed && MarkedN ==
    0)));
3   one(RP-LT initialisation);
4   try(one(RP-LT activate));
5   try(one(RP-LT share trial); repeat(one(RP-LT inform)))
6 )
```

As for the previous models, we use a repeat loop (line 1) to perform as many dissemination steps as possible. We select an `informed` node which has not yet been subjected to an initialisation or which has since undergone changes (line 2).

By applying *RP-LT initialisation* (Fig. 11(a)), we mark the selected `informed` node ($MarkedN = 1$), compare the *JointInf* and *Theta* attributes and store the result in *Tau* (Eq. 4). This value is used when the strategy tries to apply the second rule *RP-LT activate* (Fig. 11(b)) to activate the node (line 3). Only successfully applied if *Tau* is positive or null, the rule transforms the `informed` node into an `active` one, respectively modifying *isActive* and *Count* values to reflect the node current *State* and indicate that its decision concerning the diffusion subject has been confirmed.

The *RP-LT share trial* rule successfully applies only when the *Count* attribute of the node at the selected position P is set to γ. In such case, the *Share* and *Sigma* attributes are computed using respectively Equations 6 and 7.

Depending on its *Sigma*'s value, the node n in the selected position P shares the information with its inactive neighbours n' which have been influenced less than γ times and have not yet been contacted by n. The *RP-LT inform* rule (Fig. 11(d)) then marks the connection between n and n' and updates the `informed` node n' attributes: *JointInf* is recomputed taking into account the new *Influence* of n on n' (Eq. 3), the influence counter *Count* is incremented to track the new influence, and the marker *MarkedN* is reset to its default value, indicating that some changes have been applied to the attributes of n'.

The *RP-LT share trial* and *RP-LT inform* rules are repeated as long as there are nodes which have either been influenced γ times or for which the joint influence is sufficient to persuade them to endorse the subject being disseminated.

Let us prove the expected properties of our strategic rewrite program for **RP-LT**.

Lemma 13 (RP-LT.1). *For each user n that learns an information item, the **RP-LT** algorithm either reposts it to all n's outgoing neighbours, or does not repost it to any of them.*

Proof. Similar to Lemma 10, if n is supposed to diffuse the information, *Sigma*'s value is greater or equal to 0, in which case rule *RP-LT inform* is applied as many times as possible on n and its neighbours with each neighbour being only considered once thanks to the attribute *Marked*. Only neighbours which are active or have their attribute *Count* greater than γ are not concerned but these nodes are already informed of the diffusion subject.

Lemma 14 (RP-LT.2). *If n likes the information item, it is reposted to all of n's neighbours with a probability $\lambda/\overline{S_n}$; if n does not like it, the information is reposted with a (smaller) probability $\delta/\overline{S_n}$.*

Proof. The proof is the same as for Lemma 11.

Lemma 15 (RP-LT.3). *An inactive node is influenced at most γ times, and is thus given γ chances to endorse the information.*

Proof. Each time a node is influenced, its attribute *Count* is incremented and its attribute *MarkedN* is reset. After being influenced γ times, rule *RP-LT inform* no longer authorises the node to be influenced, thus a node influenced γ times can still be considered one last time for diffusion, but once it is marked in *RP-LT initialisation*, it will either activate, disseminate the information, or remain unchanged.

Lemma 16 (RP-LT.4). *An inactive node n has a monotone activation function ($p_n\big(S_n(k)\big)$) computing its active neighbours' joint influence value.*

Proof. The proof is the same as for Lemma 6.

Lemma 17 (RP-LT.5). *An inactive node n becomes active if its neighbours' joint influence exceeds its threshold value, i.e.,* $p_n\big(S_n(k)\big) \geq \theta_n$.

Proof. The proof is the same as Lemma 7.

Lemma 18 (RP-LT.6). *The process terminates when no more diffusion is possible.*

Proof. Only the failed application of rule *RP-LT initialisation* can force Strategy 7 to come to an end. The condition can only occur when no `informed` nodes remain or when all the *informed* nodes are marked ($\mathit{MarkedN} = 1$).

Proposition 8 (RP-LT termination). *If the network is finite, the strategic rewrite program given by rules in Figure 11 and Strategy 7 terminates.*

Proof. We prove the termination of the graph rewrite program in the case of a finite graph, by showing that each iteration of the repeat loop strictly decreases an interpretation of the graph with respect to a well founded ordering (thus, there is no infinite descending chain). We now define the interpretation associated with the graph at step k.

Let INM_k denote the set of `informed` and non marked nodes ($\mathit{MarkedN} = 0$) at step k. Let $\mathcal{M}_k$ denote the multi-set of values ($\mathit{Gamma} - \mathit{Count}$) of all nodes in the graph at step k. Note that the values of ($\mathit{Gamma} - \mathit{Count}$) can never be negative, so $\mathcal{M}_k$ is a multiset of natural numbers.

The graph at step k is interpreted as a pair: $(\mathcal{M}_k, INM_k)$. We compare the interpretation at step k and step $k+1$ using a lexicographic order, where the first component of the pairs are compared using the multiset extension of the usual ordering $\geq$ on natural numbers, and the second components are compared using the usual superset ordering $\supseteq$.

In each iteration of the repeat loop, either *Count* increases for a non-empty set of nodes (hence the multiset of values $\mathit{Gamma} - \mathit{Count}$ is strictly decreasing) or the values of *Count* do not change but a node in INM_k becomes active (therefore $INM_k \supset INM_{k+1}$, and the second component of the interpretation is strictly decreasing). Thus, each iteration of the loop strictly decreases our interpretation, and we conclude that the process terminates.

Proposition 9 (RP-LT properties). *The dissemination algorithm defined by the rules in Figure 11 and Strategy 7 implements the RP-LT model as specified in Definition 7.*

Proof. By Lemmas 13-18.

6 Discussion and Future Work: towards a graph rewriting based framework to study social networks

At this point, we may argue that we have the ingredients for a social network modelling framework based on graph rewriting:

- in Sect. 3, we have explained how to generate social network models tailored to various sizes, number of links and communities. The capability of generating arbitrary models is important to validate new methods or algorithms and check their behaviour.
- in Sections 4 and 5, we have formalised with labelled port graphs, rules and strategies, three known models of propagation and dissemination with different properties. We identified common basic features (attributes, rules, strategies) and simultaneously better understood what is different between these three approaches. We thus have grasped the existing properties of these models and identified the attributes and rules inducing them. This formalisation then helped us rearrange them to design a new dissemination model **RP-LT**, by combining the features of the **LT** and **RP** models.
- The formalism used, based on labelled port graphs, rewrite rules and strategies, provides the logical background necessary to understand and analyse the programs and their executions. For instance we have proved the termination of the different strategic rewrite programs for each propagation and dissemination model.
- Visualisation features provided by PORGY are indeed an important component of the framework. The prototyping aspect of rules and strategies is amplified by visualisation of results, especially for large graphs. For instance, the result of generating a social network according to different parameters is illustrated on examples in Sect. 3. Then, the behaviour of a new dissemination model like **RP-LT** can be checked and visualised on a generated network with a selected topology.
- The environment can for instance be used to compare two propagation models in the same line as done in [46]. Visualisation provides a first intuition for comparing two models applied to the same starting network, but indeed comparison needs to use measurement methods appropriate to propagation phenomena in social networks. This is left for future work.

Overall, and although PORGY has been used to perform rewrite operations on a variety of models, its first incursion in the territory of social networks has not been without challenges. Social networks come in very different sizes and shapes: from the smallest ones (e.g., 34 individuals in [52]) to very large ones (e.g. 64M nodes and 1G edges in one of the datasets studied in [51]); and the popularity of online social networks has produced expectations of large networks as soon as social networks are mentioned (e.g., Facebook which has recently reached two billion users[13]). It is obvious that our method is not suitable for generating or handling graphs on such a large scale, notably due to the overhead induced by the rewriting mechanisms. At the moment, creating graphs with several hundreds of elements can be achieved in a few minutes; for instance, both graphs given as examples at the end of Sect. 3 have been generated in less than two minutes on a standard workstation at the time of writing. Multiple benchmarks have already been performed on our rewriting platform to identify bottlenecks and

[13] `https://web.archive.org/web/20170905081244/https://newsroom.fb.com/news/2017/06/two-billion-people-coming-together-on-facebook/`

critical operations, however, the results are quite ordinary and tedious for now. It is important to note that Porgy was not originally designed to address such requirements, and therefore improvements are needed to start tackling graphs with several thousands or tens of thousands of elements in a fair amount of time. Consequently, a complete break down of Porgy's performances is left for future work, once the necessary modifications have been completed to enhance the platform performances.

7 Conclusion

Our first experiments and results on generation and propagation in social networks illustrate how labelled port graph strategic rewriting provides a suitable common formalism in which different mathematical models can be expressed and compared.

For the social network community, the rewrite rule approach is not quite surprising because some works such as [29] already use rules to generate social networks, although without claiming it. Expressing different models in the same formalism facilitates the comparison of algorithms and models. With the development of social networks analysis, there are many opportunities where simulations can indeed be of assistance during decision taking, for instance to prevent bad situations, to test counter-measures, or to look for an optimal diffusion strategy. Although we did not develop this aspect here, when modelling the evolution of a network, the derivation tree (also a port graph) provides support for history tracking, state comparison, state recovery and backtracking.

Overall, several issues still need to be addressed. Although graph rewriting has been largely studied, social network applications have only recently been developed, and require a drastic change of scale. Dealing for instance with millions of nodes and edges requires a great attention to size and complexity. As a consequence, there is room for improvement in data storage and retrieval (relevant for graph data bases), subgraph matching algorithms (either exact or approximate) for finding one or all solutions, parallel graph rewriting avoiding dangling edges, and probabilistic or stochastic issues for matching and rewriting, for instance, in the context of imprecise data or privacy constraints.

Also related to size, but even more to complexity of data, there is a need for data structuring and management, that may be carried on by abstraction pattern, focusing on points of interests, hierarchies and views (for instance, through multi-layer graphs). All these notions need a precise and logical definition that may be influenced by well-known programming language concepts.

Like programs, data need certification and validation tools and processes, not only at a single step but all along their evolution. The knowledge developed in the logic and rewriting community should be valuable in this context.

This study has also revealed the importance of visualisation and raises some challenges in this area. Visualisation is important, more widely, for data analysis, program engineering, program debugging, testing or verifying. However, the representation of dynamic or evolving data, such as social networks or rich graph structures, is yet an actual research topic for the visualisation community.

Acknowledgements We thank Guy Melançon (University of Bordeaux) and all the other members of the Porgy project.

References

1. O. Andrei, M. Fernández, H. Kirchner, G. Melançon, O. Namet, and B. Pinaud. PORGY: Strategy-Driven Interactive Transformation of Graphs. In R. Echahed, editor, 6^{th} *Int. Workshop on Computing with Terms and Graphs*, volume 48, pages 54–68, 2011.
2. O. Andrei and H. Kirchner. A Rewriting Calculus for Multigraphs with Ports. In *Proc. of RULE'07*, volume 219 of *Electronic Notes in Theoretical Computer Science*, pages 67–82, 2008.
3. O. Andrei and H. Kirchner. A Higher-Order Graph Calculus for Autonomic Computing. In *Graph Theory, Computational Intelligence and Thought. Golumbic Festschrift*, volume 5420 of *LNCS*, pages 15–26. Springer, 2009.
4. A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
5. H. Barendregt, M. van Eekelen, J. Glauert, J. R. Kennaway, M. Plasmeijer, and M. Sleep. Term graph rewriting. In *Proc. of PARLE, Parallel Architectures and Languages Europe*, number 259-II in LNCS, pages 141–158. Springer-Verlag, 1987.
6. K. Barthelmann. How to construct a hyperedge replacement system for a context-free set of hypergraphs. Technical report, Universität Mainz, Institut für Informatik, 1996.
7. V. Batagelj and U. Brandes. Efficient generation of large random networks. *Phys. Rev. E*, 71:036113, Mar 2005.
8. E. Bertuzzo, R. Casagrandi, M. Gatto, I. Rodriguez-Iturbe, and A. Rinaldo. On spatially explicit models of cholera epidemics. *Journal of The Royal Society Interface*, 7(43):321–333, 2010.
9. P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and C. Ringeissen. An overview of ELAN. *Electronic Notes in Theoretical Computer Science*, 15:55–70, 1998.
10. U. Brandes and D. Wagner. Analysis and visualization of social networks. In M. Jünger and P. Mutzel, editors, *Graph Drawing Software*, Mathematics and Visualization, pages 321–340. Springer Berlin Heidelberg, 2004.
11. P. Carrington, J. Scott, and S. Wasserman. *Models and Methods in Social Network Analysis.* Structural Analysis in the Social Sciences. Cambridge University Press, 2005.
12. D. Cartwright and F. Harary. Structural balance: a generalization of Heider's theory. *Psychological Review*, 63:277–293, 1956.
13. W. Chen, A. Collins, R. Cummings, T. Ke, Z. Liu, D. Rincón, X. Sun, Y. Wang, W. Wei, and Y. Yuan. Influence maximization in social networks when negative opinions may emerge and propagate. In *Proc. of the* 11^{th} *SIAM Int. Conf. on Data Mining, SDM 2011*, pages 379–390, 2011.
14. W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proc. of the* 16^{th} *ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '10, pages 1029–1038. ACM, 2010.

15. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation - part i: Basic concepts and double pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 163–246. World Scientific, 1997.
16. P. Dodds and D. Watts. A generalized model of social and biological contagion. *Journal of Theoretical Biology*, 232(4):587 – 604, 2005.
17. H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. Algebraic approaches to graph transformation - part II: Single pushout approach and comparison with double pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations, Chapter 4*, pages 247–312. World Scientific, 1997.
18. P. Erdős and A. Rényi. On the evolution of random graphs. In *Publication of the Mathematical Institute*, volume 5, pages 17–61. Hungarian Academy of Sciences, 1960.
19. M. Fernández, H. Kirchner, and O. Namet. A strategy language for graph rewriting. In G. Vidal, editor, *Logic-Based Program Synthesis and Transformation*, volume 7225 of *LNCS*, pages 173–188. Springer Berlin Heidelberg, 2012.
20. M. Fernández, H. Kirchner, and B. Pinaud. Strategic port graph rewriting: An interactive modelling and analysis framework. In D. Bosnacki, S. Edelkamp, A. Lluch-Lafuente, and A. Wijs, editors, *Proc. 3rd Workshop on GRAPH Inspection and Traversal Engineering, GRAPHITE 2014*, volume 159 of *EPTCS*, pages 15–29, 2014.
21. M. Fernández, H. Kirchner, and B. Pinaud. Strategic Port Graph Rewriting: an Interactive Modelling Framework. Research report, Inria ; LaBRI - Laboratoire Bordelais de Recherche en Informatique ; King's College London, 2017.
22. M. Fernández, H. Kirchner, B. Pinaud, and J. Vallet. Labelled Graph Rewriting Meets Social Networks. In D. Lucanu, editor, *Rewriting Logic and Its Applications, WRLA 2016*, volume 9942 of *LNCS*, pages 1–25, Eindhoven, Netherlands, Apr. 2016. Springer International Publishing Switzerland.
23. M. Fernández, H. Kirchner, B. Pinaud, and J. Vallet. Porgy Strategy Language: User Manual. Research report, Université de Bordeaux, LaBRI ; Inria Bordeaux Sud-Ouest ; King's College London, July 2017.
24. G. Giakkoupis, R. Guerraoui, A. Jégou, A.-M. Kermarrec, and N. Mittal. Privacy-Conscious Information Diffusion in Social Networks. In Y. Moses and M. Roy, editors, *DISC 2015*, volume LNCS 9363 of 29th *Int. Symp. on Distributed Computing*, Tokyo, Japan, Oct. 2015. Toshimitsu Masuzawa and Koichi Wada, Springer-Verlag Berlin Heidelberg.
25. M. Gomez-Rodriguez, J. Leskovec, and A. Krause. Inferring networks of diffusion and influence. In *Proc. of the 16th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '10, pages 1019–1028. ACM, 2010.
26. A. Goyal, F. Bonchi, and L. V. Lakshmanan. Learning influence probabilities in social networks. In *Web Search and Data Mining, Proc. of the 3rd ACM Int. Conf. on*, WSDM '10, pages 241–250. ACM, 2010.
27. M. Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83(6):1420, 1978.
28. A. Habel, J. Müller, and D. Plump. Double-pushout graph transformation revisited. *Mathematical Structures in Computer Science*, 11(5):637–688, 2001.
29. N. Kejžar, Z. Nikoloski, and V. Batagelj. Probabilistic inductive classes of graphs. *The Journal of Mathematical Sociology*, 32(2):85–109, 2008.

30. D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proc. of the 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '03, pages 137–146. ACM, 2003.
31. D. Kempe, J. Kleinberg, and É. Tardos. Influential nodes in a diffusion model for social networks. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Automata, Languages and Programming*, volume 3580 of *LNCS*, pages 1127–1138. Springer Berlin Heidelberg, 2005.
32. M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter. Multilayer networks. *Journal of Complex Networks*, 2(3):203–271, 2014.
33. Y. Lafont. Interaction nets. In *Proc. of the 17th ACM Symp. on Principles of Programming Languages (POPL'90)*, pages 95–108. ACM Press, 1990.
34. J. Leskovec, D. Huttenlocher, and J. Kleinberg. Signed networks in social media. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*, CHI '10, pages 1361–1370. ACM, 2010.
35. M. Löwe, M. Korff, and A. Wagner. An algebraic framework for the transformation of attributed graphs. In M. R. Sleep, M. J. Plasmeijer, and M. C. J. D. van Eekelen, editors, *Term Graph Rewriting*, pages 185–199. John Wiley and Sons Ltd., Chichester, UK, 1993.
36. S. Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.
37. M. Newman, A.-L. Barabási, and D. J. Watts. *The structure and dynamics of networks*. Princeton Studies in Complexity. Princeton University Press, 2006.
38. B. Nick, C. Lee, P. Cunningham, and U. Brandes. Simmelian backbones: Amplifying hidden homophily in facebook networks. In *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM Int. Conf. on*, pages 525–532, Aug 2013.
39. B. Pinaud, G. Melançon, and J. Dubois. PORGY: A Visual Graph Rewriting Environment for Complex Systems. *Computer Graphics Forum*, 31(3):1265–1274, 2012.
40. D. Plump. Term graph rewriting. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 2: Applications, Languages, and Tools*, pages 3–61. World Scientific, 1998.
41. D. Plump. The Graph Programming Language GP. In S. Bozapalidis and G. Rahonis, editors, *CAI*, volume 5725 of *LNCS*, pages 99–122. Springer, 2009.
42. A. Sallaberry, F. Zaidi, and G. Melançon. Model for Generating Artificial Social Networks having Community Structures with Small World and Scale Free Properties. *Social Network Analysis and Mining*, 3(597-609), Jan. 2013.
43. J. Scott and P. J. Carrington. *The SAGE Handbook of Social Network Analysis*. SAGE, 2011.
44. S. Son and V. Shmatikov. *The Hitchhiker's Guide to DNS Cache Poisoning*, pages 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
45. A. S. Tanenbaum and D. J. Wetherall. *Computer Network*. Pearson, 2010.
46. J. Vallet, H. Kirchner, B. Pinaud, and G. Melançon. A visual analytics approach to compare propagation models in social networks. In A. Rensink and E. Zambon, editors, *Proc. Graphs as Models, GaM 2015*, volume 181 of *EPTCS*, pages 65–79, 2015.
47. L. Wang, F. Du, H. P. Dai, and Y. X. Sun. Random pseudofractal scale-free networks with small-world effect. *The European Physical Journal B - Condensed Matter and Complex Systems*, 53(3):361–366, 2006.
48. D. J. Watts. A simple model of global cascades on random networks. *Proc. of the National Academy of Sciences*, 99(9):5766–5771, 2002.

49. D. J. Watts and S. H. Strogatz. Collective dynamics of ‘small-world’ networks. *Nature*, 393:440–442, 1998.
50. L. Wonyeol, K. Jinha, and Y. Hwanjo. CT-IC: Continuously activated and time-restricted independent cascade model for viral marketing. In *Data Mining (ICDM), 2012 IEEE* 12^{th} *Int. Conf. on*, pages 960–965, 2012.
51. J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *CoRR*, abs/1205.6233, 2012.
52. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.