HAL
open science

Labelled Graph Rewriting Meets Social Networks

Maribel Fernández, Hélène Kirchner, Bruno Pinaud, Jason Vallet

To cite this version:

Maribel Fernández, Hélène Kirchner, Bruno Pinaud, Jason Vallet. Labelled Graph Rewriting Meets Social Networks. Rewriting Logic and Its Applications, WRLA 2016, Apr 2016, Eindhoven, Netherlands. pp.1–25, 10.1007/978-3-319-44802-2_1 . hal-01347355

HAL Id: hal-01347355

https://inria.hal.science/hal-01347355

Submitted on 21 Jul 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Labelled Graph Rewriting Meets 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
bpinaud@labri.fr,jvallet@labri.fr

Abstract. The intense development of computing techniques and the increasing volumes of produced data raise many modelling and analysis challenges. There is a need to represent and analyse information that is: complex –due to the presence of massive and highly heterogeneous data–, dynamic –due to interactions, time, external and internal evolutions–, connected and distributed in networks. We argue in this work that relevant concepts to address these challenges are provided by three ingredients: labelled graphs to represent networks of data or objects; rewrite rules to deal with concurrent local transformations; strategies to express control versus autonomy and to focus on points of interests. To illustrate the use of these concepts, we choose to focus our interest on social networks analysis, and more precisely in this paper on random network generation. Labelled graph strategic rewriting provides a formalism in which different models can be generated and compared. Conversely, the study of social networks, with their size and complexity, stimulates the search for structure and efficiency in graph rewriting. It also motivated the design of new or more general kinds of graphs, rules and strategies (for instance, to define positions in graphs), which are illustrated here. This opens the way to further theoretical and practical questions for the rewriting community.

1 Introduction

With the intense development of computing techniques, the last decades have seen an increasing complexity of models needed to study phenomena of the physical world and, at the same time, increasing volumes of data produced by observations and computations. New paradigms of data science and data exploration have emerged and opened the way to analytic approaches, such as data-driven algorithms, analysis and mining (also called data analytics). Social and human sciences are also impacted by this evolution and provide interesting research problems for computer scientists. To illustrate these concepts, we choose to focus our interest on social networks, which have been intensively studied in the last years [12,31,36]. The analysis of social networks, used to represent users

and their relations with one another, raises several questions concerning their possible construction and evolutions. Among these questions, the study of network propagation phenomena has initiated a sustained interest in the research community, offering applications in various domains, ranging from sociology [23] to epidemiology [9, 17] or even viral marketing and product placement [15]. To solve these problems we need to model and analyse systems that are complex, since they involve data that are massive and highly heterogeneous, dynamic, due to interactions, time, external or internal evolutions, connected and distributed in networks.

We argue in this paper that relevant concepts to address these challenges are: **Labelled Graphs** to represent networks of data or objects, **Rules** to deal with concurrent local transformations, and **Strategies** to express control versus autonomy and to focus on points of interests. Indeed, modelling social networks raises many questions we have to address. First, large networks are involved for which an efficient search of patterns is needed, along with capability of focusing on points of interest and defining appropriate views. Since data are often corrupted or imprecise, one should also deal with uncertainty, which implies that we need to address probabilistic or stochastic issues in the models. The dynamic evolution of data is generally modelled by simple transformations, applied in parallel and triggered by events or time. However, such models should also take into account controlled versus autonomous behaviour. Modelling may reveal conflicts that have to be detected (for instance through overlapping rules) and solved (using precedence, choices, i.e., strategic issues). Memory and backtracking must be provided, through notions of computation history or traces. Last but not least, visualisation is important at all levels: for data analysis, program engineering, program debugging, tests and verification (for instance to provide proof intuition).

In [37], we focused on propagation phenomena and showed how some popular models can be expressed using labelled graph and rewriting. In the current paper, we use this previous work to illustrate our computing model and introduce a generative model for social networks. Indeed, many data sets, extracted from various social networks, are publicly available.[4] However, in order to demonstrate the generality of a new approach, or to design and experiment with stochastic algorithms on a sufficiently large sample of network topologies, it is more convenient to use randomly generated networks. Several generative models of random networks are available to work with (e.g., [5, 8, 18, 39]). Some, like the Erdös–Rényi (ER) model [18], do not guarantee any specific property regarding their final topology, whereas others can be characterised as small-world or scale-free networks. This paper shows how to generate such models using labelled graphs, rules and strategies.

Port graph rewriting systems have been used to model systems in a wide variety of domains, such as biochemistry, interaction nets, games or social networks (e.g., [1, 20, 21, 37]). In the following, we reuse from [19] the formal definitions of port graphs with attributes, rewrite rule and rewriting step, the concept of

[4] For instance from `http://snap.stanford.edu`

strategic graph program, as well as the definition of the strategy language and its operational semantics, and enrich them in order to achieve a more complete and generic definition. Most notably, the refined definitions permit the use of oriented edges and conditional existence matching, reminiscent of similar solutions found in ELAN [10] and GP [35]. We use the PORGY environment which supports interactive modelling using port graph rewriting; more details concerning the rewriting platform can be found in [33].

Summarising, our contributions are twofold: we present a general modelling framework, based on strategic port graph rewriting, that facilitates the analysis of complex systems, and we illustrate its power by focusing on social networks (more precisely, their generation). For this application, the visual high-level modelling features of port graph rewriting are particularly relevant. Concepts of port graphs, rules and strategies are illustrated on this specific domain. Conversely, the study of social networks, with their size and complexity, stimulates the search for structure and efficiency in graph rewriting. We identify open problems and questions that arise when studying social networks.

The paper is organised as follows. Section 2 introduces the modelling concepts we propose to use: port graphs, morphism, rewriting, derivation tree, strategy and strategic graph programs are defined in their full generality, while illustrated on the special case of social networks. In Section 3, we focus on social network behaviour simulation, more precisely on social network generation. In Section 4, we conclude by synthesising the lessons learned from this study and giving perspectives for future work.

2 Labelled Graph Rewriting

Several definitions of graph rewriting are available, using different kinds of graphs and rewrite rules (see, for instance, [6,7,16,24,28,34]). In this paper we consider *port graphs* with *attributes* associated with nodes, ports and edges, generalising the notion of port graph introduced in [2, 3]. The following definitions, based on [19], have been generalised to use indistinctly either directed or undirected edges. We present first the intuitive ideas, followed by the formal definition of port graph rewriting.

2.1 Port graphs

Intuitively, a port graph is a graph where nodes have explicit connection points called *ports*, to which edges are attached. Nodes, ports and edges are labelled by records listing their attributes.

A *signature* ∇ used to label the graph is composed of:

- $\nabla_{\mathscr{A}}$, a set of attributes;
- $\mathcal{X}_{\mathscr{A}}$, a set of attribute variables;
- $\nabla_{\mathscr{V}}$, a set of values;
- $\mathcal{X}_{\mathscr{V}}$, a set of value variables.

where $\nabla_{\mathscr{A}}$, $\mathcal{X}_{\mathscr{A}}$, $\nabla_{\mathscr{V}}$ and $\mathcal{X}_{\mathscr{V}}$ are pairwise disjoint. $\nabla_{\mathscr{A}}$ contains distinguished elements *Name*, *(In/Out)Arity*, *Connect*, *Attach*, *Interface*. Values in $\nabla_{\mathscr{V}}$ are assumed to be of basic data types such as *strings*, *int*, *bool*,... or to be well-typed computable expressions built using ∇ and basic types.

Definition 1 (Record). *A record r over the signature ∇ is a set of pairs $\{(a_1, v_1), \ldots, (a_n, v_n)\}$, where*

$a_i \in \nabla_{\mathscr{A}} \cup \mathcal{X}_{\mathscr{A}}$ for $1 \leq i \leq n$, called attributes; each a_i occurs only once in r, and there is one distinguished attribute Name.

$v_i \in \nabla_{\mathscr{V}}$ for $1 \leq i \leq n$, called values.

The function Atts applies to records and returns all their attributes:

$$Atts(r) = \{a_1, \ldots, a_n\}$$

if $r = \{(a_1, v_1), \ldots, (a_n, v_n)\}$. As usual, $r.a_i$ denotes the value v_i of the attribute a_i in r.

The attribute Name identifies the record in the following sense: For all r_1, r_2, $Atts(r_1) = Atts(r_2)$ if $r_1.Name = r_2.Name$.

Definition 2 ((Directed) Port graph). *Given sets $\mathscr{N}, \mathscr{P}, \mathscr{E}$ of nodes, ports and edges, a* port graph *over a* signature *∇ is a tuple $G = (N, P, E, \mathcal{L})$ where*

- *$N \subseteq \mathscr{N}$ is a finite set of nodes; $n, n', n_1, \ldots$ range over nodes.*
- *$P \subseteq \mathscr{P}$ is a finite set of ports; $p, p', p_1, \ldots$ range over ports.*
- *$E \subseteq \mathscr{E}$ is a finite set of edges between ports; $e, e', e_1, \ldots$ range over edges. Edges can be directed and two ports may be connected by more than one edge.*
- *$\mathcal{L}$ is a labelling function that returns, for each element in $N \cup P \cup E$, a record such that:*
  - *for each edge $e \in E$, $\mathcal{L}(e)$ contains an attribute Connect whose value is the ordered pair (p_1, p_2) of ports connected by e.*
  - *for each port $p \in P$, $\mathcal{L}(p)$ contains an attribute Attach whose value is the node n which the port belongs to, and an attribute Arity whose value is the number of edges connected to this port. When edges are directed, ports have instead two attributes, InArity and OutArity, whose respective values are the number of edges directed to and from this port.*
  - *For each node $n \in N$, $\mathcal{L}(n)$ contains an attribute Interface whose value is the set of names of ports in the node: $\{\mathcal{L}(p_i).Name \mid \mathcal{L}(p_i).Attach = n\}$. We assume that $\mathcal{L}$ satisfies the following constraint:*

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

By definition 2, nodes with the same name (i.e., the same value for the attribute *Name*) have the same set of port names (i.e., the same interface), with the same attributes but possibly with different values. Variables may be used to denote any value.

Two nodes $n, n' \in N$ connected by an undirected edge are said to be adjacent and each other neighbours. However, for a directed edge $(n, n') \in E$ going from

n to n', only n' is said *adjacent* to n (not conversely) and is called 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. $N(n)$ denotes the set of neighbours of the node n.

The advantage of using port graphs rather than plain graphs is that they 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. However, the counterpart is that the implementation, rules and matching operations are more complex. So, whenever possible, it is simpler and more efficient to keep the number of ports for each node to a minimum.

Example 1 (Social Network). A social network [11] is commonly described as a graph $G = (N, E)$ built from a set of nodes (the users) N and a set of edges $E \subseteq N \times N$ linking users. Although in most real-world social relations, two persons relate to each other with a mutual recognition, some social networks present an asymmetric model of acknowledgement, the most popular of them being Twitter, classifying one of the users as a *follower* while the other is a *followee*. Such relations can be very simply represented by orienting edges, thus transforming our initial graph in a directed graph.

In this paper, we model a social network as a port graph, where nodes represent users and edges are connections between them. Edges are directed to reflect the relation between users (e.g., follower/followee) and store the attributes of their relation (e.g., influence level, threshold value...). An alternative solution would be to use undirected edges and nodes with two ports called "In" and "Out" for instance, as in [37], to simulate edge direction. In this paper, the nodes representing users have only one port gathering directed connections. While this is sufficient for simple cases, when facing real social networks, multiple ports are useful, either to connect users according to the nature of their relation (e.g., friends, family, co-workers...) or to model situations where a user is connected to friends via different social networks. The full power of port graphs is indeed necessary in multi-layer networks [27] 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 a topic left for future work.

Example 2 (Propagation). Propagation in a network can be seen as follows: when users perform a specific action (announcing an event, spreading a gossip, sharing a video clip, *etc.*), they become *active*. They inform their neighbours of their state change, giving them the possibility to become active themselves if they perform the same action. Such process reiterates as the newly active neighbours share the information with their own neighbours. The activation can thus propagate from peer to peer across the whole network.

To replicate this phenomena observed in real-world networks, some models opt for entirely probabilistic activations (e.g., [14, 42]) where the presence of only one active neighbour is enough to allow the propagation to occur. Other models use threshold values (e.g., [22,26,40]) building up during the propagation.

Such values represent the influence of one user on his neighbours or his tolerance towards performing a given action (the more solicited a user is, the more inclined he becomes to either activate or utterly resist).

To express propagation conditions (e.g., a probabilistic model for node activation, or activation after reaching a predefined threshold), it is natural to make use of records with expressions, i.e., include specific attributes whose values are numerical expressions. More specifically:

- Each node n has an attribute *Active* that indicates whether it contributes to the propagation or not. It is coupled with the *Colour* attribute, which takes accordingly green or red values. The node n has also a *Sigma* attribute that measures the maximum influence withstood by n from its active neighbours at the time being.
- An edge e that connects two ports p' and p of the respective nodes n' and n has an attribute *Influence* which indicates the influence of n' (i.e., $\mathcal{L}(p').Attach$) on n (i.e., $\mathcal{L}(p).Attach$). The edge e has also a Boolean attribute *Marked*, initially false, which becomes true when n is inactive, n' is active and n' has tried to influence n.

2.2 Rewriting

We see a *port graph rewrite rule* $L \Rightarrow R$ as a port graph consisting of two subgraphs L and R together with a special node (called *arrow* node) that encodes the correspondence between the ports of L and the ports of R. Each of the ports attached to the arrow node has an attribute $Type \in \nabla_{\mathscr{A}}$, which can have three different values: *bridge*, *wire* and *blackhole*. The value indicates how a rewriting step using this rule should affect the edges that connect the redex to the rest of the graph. We give details below.

Definition 3 (Port graph rewrite rule). *A port graph rewrite rule is a port graph consisting of:*

- *two port graphs L and R over the signature ∇, respectively called* left-hand side *and* right-hand side*, such that all the variables in R occur in L, and R may contain records with expressions;*
- *an* arrow *node with a set of edges that each connect a port of the arrow node to ports in L or R.*

The arrow node has for $Name$ $\Rightarrow$. Each port in the arrow node has an attribute Type*, which can be of value:* bridge*,* blackhole *or* wire*, satisfying the following conditions:*

1. *A port of type* bridge *must have edges connecting it to L and to R (one edge to L and one or more to R).*
2. *A port of type* blackhole *must have edges connecting it only to L (at least one edge).*
3. *A port of type* wire *must have exactly two edges connecting to L and no edge connecting to R.*

The arrow node has an optional attribute Where *whose value is a Boolean expression involving the predicate* `Edge`*, applied to node and port names, and Boolean operators.*

When modelling rumour propagation, the rules never suppress nor add new nodes. Moreover, when there is only one port per node, there is no ambiguity on the rewiring between left and right-hand sides. In that case indeed, the structure and visualisation of the arrow node is much simpler. However, this only holds when the network's structure does not change.

The introduction of the *Where* attribute is inspired from the GP programming system [35] (and from ELAN [10] with a more general definition), in which a rule may have a condition introduced by the keyword `where`. For instance, a condition `where not Edge(n,n')` requires that no edge exists between the nodes n and n'. This condition is checked at matching time.

Let us first recall the notion of port graph morphism [19]. Let G and H be two port graphs over the same signature ∇. A *port graph morphism* $f : G \to H$ maps nodes, ports and (directed) edges of G to those of H such that the attachment of ports and the (directed) edges connections are preserved, all attributes and values are preserved except for variables in G, which must be instantiated in H. 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 4 (Match). *Let $L \Rightarrow R$ be a port graph rewrite rule and G a port graph. We say a* match *$g(L)$ of the left-hand side (also called a* redex*) is found if:*

- *There is a port graph morphism g from L to G; hence $g(L)$ is a subgraph of G.*
- *If the arrow node has an attribute* Where *with value C, C must be true of $g(L)$.*
- *For each port in L that is not connected to the arrow node, its corresponding port in $g(L)$* must not *be an extremity in the set of edges of $G - g(L)$.*

This last point ensures that ports in L that are not connected to the arrow node are mapped to ports in $g(L)$ that have no edges connecting them with ports outside the redex, to avoid dangling edges in rewriting steps.

Several injective morphisms g from L to G may exist (leading to different rewriting steps); they are computed as solutions of a *matching* problem from L to (a subgraph of) G.

Definition 5 (Rewriting step). *According to [19], a* rewriting step *on G using a rule $L \Rightarrow R$ (*`where` *C) and a morphism $g : L \to G$ (satisfying C), 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 is a bridge port and $p_L \in L$ is connected to p:*
    for each port $p_R^i \in R$ connected to p,
    find all the ports p_G^k in G that are connected to $g(p_L)$ and are not in $g(L)$, and redirect each edge connecting p_G^k and $g(p_L)$ to connect p_G^k and $p_{R_c}^i$.
  - *If p is a wire port connected to two ports p_1 and p_2 in L, then take all the ports outside $g(L)$ that are connected to $g(p_1)$ in G and connect each of them to each port outside $g(L)$ connected by an edge to $g(p_2)$.*
  - *If p is a blackhole: for each port $p_L \in L$ connected to p, destroy all the edges connected to $g(p_L)$ in G.*
– *The* deletion *phase simply deletes $g(L)$. This creates the final graph G'.*

Example 3 (Propagation). Figure 1 shows two rules used for propagation. Active nodes are depicted in green and visited nodes in purple. Red nodes are in an inactive state (however, they may have been visited already). Rule *R1* in Figure 1(a) indicates that when an activated node n is connected to an inactive node $\overline{n}$, it tries to influence it. If it succeeds, a second rule, Rule *R2* in Figure 1(b), makes this node active.

In a social network $G = (N, E)$, let n and $\overline{n}$ be two nodes ($n, \overline{n} \in N$) connected via an edge $e = (n, \overline{n}) \in E$. The node's attribute $\mathcal{L}(\overline{n}).Sigma$, giving the influence withstood by $\overline{n}$ and initially set to 0, is updated such as:

$$\mathcal{L}(\overline{n}).Sigma = \max\left(\frac{\mathcal{L}(e).Influence}{r}, \mathcal{L}(\overline{n}).Sigma\right)$$

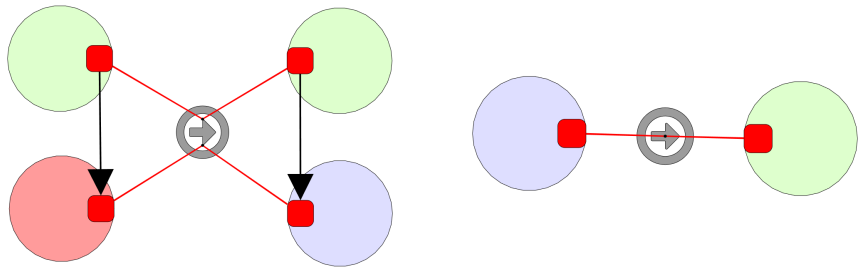
where r is a random number between 0 and 1 and $\mathcal{L}(e).Influence$ is the influence of n on $\overline{n}$. The formula is stored as a node attribute in the right-hand side of Rule *R1* in Figure 1(a) and each corresponding rewriting performs the update. More details are given in [37].

Given a finite set $\mathcal{R}$ of rules, a port graph G *rewrites* to G', denoted by $G \to_{\mathcal{R}} G'$, if there is a rule r in $\mathcal{R}$ and a morphism g such that $G \to_r^g G'$. This induces a reflexive and transitive relation on port graphs, called *the rewriting relation*, denoted by $\to_{\mathcal{R}}^*$. A port graph on which no rule is applicable is *irreducible*.

A *derivation*, or computation, is a sequence $G \to_{\mathcal{R}}^* G'$ of rewriting steps. Each rewriting step involves the application of a rule at a specific position in the graph. A *derivation tree* from G represents all possible computations (with possibly infinite ones) and *strategies* are used to specify the rewriting steps of interest, by selecting branches in the derivation tree.

2.3 Strategic graph programs

In this section, we recall the concept of *strategic graph program*, consisting of a *located graph* (a port graph with two distinguished subgraphs that specify the locations where rewriting is enabled/disabled), a set of rewriting rules, and a



(a) *R1*: Influence trial. An active neighbour (green) influences an inactive node (red) by visiting it (transformation into a blue node).

(b) *R2*: Node activation. A visited node (blue) sufficiently influenced is activated (transformation into a green node).

Fig. 1. Rules used to express a propagation model. For both rules, we use two specific node's *attributes* –*active* and *visited*– to manage the matching performed, the different colours being visual cues helping users identifying the node state at a glance. Green nodes, or active nodes, must have their attributes *active* equal to 1 and *visited* equal to 0; red nodes, or inactive nodes, must have their attributes *active* equal to 0 and *visited* equal to 0; finally, blue nodes, or visited nodes, must have their attributes *active* equal to 0 and *visited* equal to 1.

strategy expression. We then recall the strategy language presented in [19] to define strategy expressions. In addition to the well-known constructs to select rewrite rules, the strategy language provides position primitives to select or ban specific positions in the graph for rewriting. The latter is useful to program graph traversals in a concise and natural way, and is a distinctive feature of the language. In the context of social networks, the position primitives are also convenient to restrict the application of rules to specific parts of the graph.

Located graphs and rewrite rules First, we recall that, in graph theory, a subgraph of a graph $G = (N_G, E_G)$ is a graph $H = (N_H, E_H)$ contained in G, that is, $N_H \subseteq N_G$ and $E_H \subseteq E_G$. The definition extends to directed port graphs in the natural way: let $G = (N_G, P_G, E_G, \mathcal{L}_G)$ and $H = (N_H, P_H, E_H, \mathcal{L}_H)$ be port graphs over the signature ∇. H is a subgraph of G if $N_H \subseteq N_G$, $P_H \subseteq P_G$, $E_H \subseteq E_G$, $\mathcal{L}_H = \mathcal{L}_G|_{N_H \cup P_H \cup E_H}$, that is, $\mathcal{L}_H$ is the restriction to H of the labelling function of G.

Definition 6 (Located graph). *According to [19], 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 represents the subgraph of G where rewriting steps may take place (i.e., P is the focus of the rewriting) and Q represents the subgraph of G where rewriting steps are forbidden. We give a precise definition

below; the intuition is that subgraphs of G that overlap with P may be rewritten, if they are outside Q.

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 in the natural way on port graphs considered as sets of nodes, ports and edges.

Definition 7 (Located rewrite rule). *A* located rewrite rule *is given by a port graph rewrite rule $L \Rightarrow R$, and, optionally, a subgraph W of L and two disjoint subgraphs M and M' of R. 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 $P' = (P \setminus g(L)) \cup g(M)$, $Q' = Q \cup g(M')$; if M (resp. M') are not provided then we assume $M = R$ (resp. $M' = \emptyset$).*

In general, for a given located rule $L_W \Rightarrow R_M^{M'}$ and located graph G_P^Q, more than one morphism g, such that $g(L) \cap P = g(W)$ and $g(L) \cap Q$ is empty, may exist (i.e., several rewriting steps at P avoiding Q may be possible). Thus, the application of the rule at P avoiding Q produces a *set of located graphs*.

Example 4. In influence propagation, banned subgraphs are used to avoid several activations of the same neighbours. Another usage is to select a specific community in the social network where the propagation should take place.

2.4 Strategies

To control the application of the rules, a strategy language is presented in [19]. We recall it in Table 1, including some additional constructs that are needed to deal with directed edges.

Strategy expressions are generated by the grammar rules from the non-terminal S. A strategy expression combines applications of located rewrite rules, generated by the non-terminal A, and *position updates*, generated by the non-terminal U, using *focusing expressions*, generated by F. Subgraphs of a given graph can be defined by specifying simple properties, expressed with attributes of nodes, edges and ports. The strategy constructs, generated by S, are used to compose strategies and are strongly inspired from term rewriting languages such as Elan [10], Stratego [38] and Tom [4].

We briefly explain below the constructs used in this paper. A full description of the language can be found in [19].

Let L, R be port graphs; M, M' subgraphs of R; W a subgraph of L;
$k \in \mathbb{N}$; $\pi_{i=1\ldots k} \in [0,1]$; $\sum_{i=1}^{k} \pi_i = 1$; let *attribute* be an attribute label in $\nabla_{\mathscr{A}}$;
$v \in \nabla_{\mathscr{V}}$ a valid expression without variables;

Rules	**(Transformations)**	$T ::= L_W \Rightarrow R_M^{M'} \mid (T \parallel T)$
		$\mid \texttt{ppick}(T_1, \pi_1, \ldots, T_k, \pi_k)$
	(Applications)	$A ::= \texttt{all}(T) \mid \texttt{one}(T)$
Positions	**(Focusing)**	$F ::= \texttt{crtGraph} \mid \texttt{crtPos} \mid \texttt{crtBan}$
		$\mid F \cup F \mid F \cap F \mid F \setminus F \mid (F) \mid \emptyset$
		$\mid \texttt{ppick}(F_1, \pi_1, \ldots, F_k, \pi_k)$
		$\mid \texttt{property}(F, \rho) \mid \texttt{ngb}(F, \rho)$
		$\mid \texttt{ngbOut}(F, \rho) \mid \texttt{ngbIn}(F, \rho)$
	(Determine)	$D ::= \texttt{all}(F) \mid \texttt{one}(F)$
	(Update)	$U ::= \texttt{setPos}(D) \mid \texttt{setBan}(D)$
		$\mid \texttt{update}(function\{parameters_list\})$
Properties	**(Properties)**	$\rho := Elem, Expr$
		$Elem := \texttt{node} \mid \texttt{edge} \mid \texttt{port}$
		$Expr := attribute\ Relop\ v \mid \texttt{true}$
		$Relop := == \mid \texttt{!=} \mid > \mid <$
		$\mid >= \mid <= \mid =\sim$
Compositions	**(Comparison)**	$C ::= F = F \mid F \texttt{ != } F \mid F \subset F \mid \texttt{isEmpty}(F)$
		$\mid \texttt{match}(T)$
	(Strategies)	$S ::= \mathsf{id} \mid \mathsf{fail} \mid A \mid U \mid C \mid S; S$
		$\mid \texttt{if}(S)\texttt{then}(S)\texttt{else}(S) \mid (S)\texttt{orelse}(S)$
		$\mid \texttt{repeat}(S)[(k)] \mid \texttt{while}(S)[(k)]\texttt{do}(S)$
		$\mid \texttt{ppick}(S_1, \pi_1, \ldots, S_k, \pi_k)$
		$\mid \texttt{try}(S) \mid \texttt{not}(S)$

Table 1. Syntax of the Strategy Language.

The primary construct is a located rule, 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. When probabilities $\pi_1, \ldots, \pi_k \in [0,1]$ are associated to rules $T_1, \ldots, T_k$ such that $\pi_1 + \cdots + \pi_k = 1$, the strategy $\texttt{ppick}(T_1, \pi_1, \ldots, T_k, \pi_k)$ picks one of the rules for application, according to the given probabilities.

$\texttt{all}(T)$ denotes all possible applications of the transformation T on the located graph at the current position, creating a new located graph for each application. In the derivation tree, this creates as many children as there are possible applications.

$\texttt{one}(T)$ computes only one of the possible applications of the transformation and ignores the others; more precisely, it makes an equiprobable choice between all possible applications.

Similar constructs exist for positions focusing: $\texttt{one}(F)$ returns one node in F and $\texttt{all}(F)$ returns the full F. In the remaining of this paper, when not specified, F stands for $\texttt{all}(F)$.

Focusing expressions are used to define positions for rewriting in a graph, or to define positions where rewriting is not allowed. They denote functions used in strategy expressions to change the positions P and Q in the current located graph. In this paper, we use:

- `crtGraph`, `crtPos` and `crtBan`, applied to a located graph G_P^Q, return respectively the whole graph G, P and Q.
- $\texttt{property}(F, \rho)$ is used to select elements of a given graph that satisfy a certain property, specified by ρ. It can be seen as a filtering construct: if the expression F generates a subgraph G' then $\texttt{property}(F, \rho)$ returns only the nodes and/or edges from G that satisfy the decidable property $\rho = Elem, Expr$. Depending on the value of $Elem$, the property is evaluated on nodes, ports, or edges.
- $\texttt{ngb}(F, \rho)$ returns a subset of the neighbours (i.e., adjacent nodes) of F according to ρ. Note that the direction of the edge is taken into account; to emphasise it, we introduce $\texttt{ngbOut}(F, \rho)$ and its counterpart $\texttt{ngbIn}(F, \rho)$. If `edge` is used, i.e., if we write $\texttt{ngb}(F, \texttt{edge}, Expr)$, it returns all the neighbours of F connected to F via edges which satisfy the expression $Expr$.
- $\texttt{setPos}(D)$ (resp. $\texttt{setBan}(D)$) sets the position subgraph P (resp. Q) to be the graph resulting from the expression D. It always succeeds (i.e., returns `id`).

The following constructs are also used:

- $S;S'$ represents sequential application of S followed by S'.
- $\texttt{repeat}(S)[\texttt{max}\ n]$ simply iterates the application of S until it fails, but, if $\texttt{max}\ n$ is specified, then the number of repetitions cannot exceed n.
- $(S)\texttt{orelse}(S')$ applies S if possible, otherwise applies S'. It fails if both S and S' fail.
- When probabilities $\pi_1, \ldots, \pi_k \in [0, 1]$ are associated to strategies $S_1, \ldots, S_k$ such that $\pi_1 + \cdots + \pi_k = 1$, the strategy $\texttt{ppick}(S_1, \pi_1, \ldots, S_k, \pi_k)$ picks one of the strategies for application, according to the given probabilities. This construct generalises the probabilistic constructs on rules and positions.

Example 5 (Propagation). (Example 3 cont'd) To illustrate the strategy language, let us come back to the propagation model in social networks and to the two rules described in Figure 1. When Rule *R1* in Figure 1(a) is applied on a pair of nodes active(n)/non active($\overline{n}$) (green/red): *a)* we generate a random number $r \in]0, 1]$; *b)* we store in the attribute $\mathcal{L}(\overline{n}).Sigma$ the new value of *Sigma* for $\overline{n}$ computed with the previously given formula; and *c)* using the *Marked* attribute, we mark the edge e linking n to $\overline{n}$ to prevent the selection of this particular

pair configuration in the next pattern matching searches. This ensures that the active node n will not be able to try to influence the same node $\overline{n}$ over and over.

Once every pair of active/inactive neighbours has been tried, if $\overline{n}$ is sufficiently influenced (i.e., $\mathcal{L}(\overline{n}).Sigma \geq 1$), Rule *R2* in Figure 1(b) is applied and $\overline{n}$ becomes active. This behaviour is expressed with the following strategy:

Strategy 1: Influence propagation in social network.

```
1 repeat(R1);
2 setPos(property(crtGraph, node, Sigma ≥ "1"));
3 repeat(R2)
```

This example illustrates how record expressions may be used to compute attribute values and how they are updated through application of rules.

Probabilistic features of the PORGY strategy language, through the use of the `ppick`() construct, are illustrated in Section 3 for social network generation.

A more complete formal definition of strategic graph programs and their semantics can be found in [19]. Correctness and completeness of strategic port graph rewriting are stated and imply in particular that the derivation tree in which each rewrite step is performed according to the strategy –let us call it the *strategic derivation tree*– is actually a subtree of the derivation tree of the rewrite system without strategy. The strategic derivation tree is a valuable concept because it records the history of the transformations and provides access to generated models. It is, by itself, a source of challenging questions, such as detecting isomorphic models and folding the tree, finding equivalent paths and defining the "best ones", abstracting a sequence of steps by a composition strategy, or managing the complexity of the tree and its visualisation.

From now on, the paper focuses on social networks generation using the introduced labelled graph rewriting concepts and the PORGY environment.

3 Social network generation

We focus in the following on generating graphs with a small-world property as defined in [41]. Such graphs are characterised by a small diameter –the average distance between any pair of nodes is short– and strong local clustering –any pair of connected nodes tend to both be connected to the same neighbour nodes thus creating densely linked groups of nodes, also called *communities*. Popularised by Milgram in [30], 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 among the users. Furthermore, the graph $G = (N, E)$ produced by the generation process satisfies the following requirements: the number of nodes $|N|$ and directed edges $|E|$ are given *a priori*; G is formed of a sole connected component thus $|E|$ should at least be equal to $|N| - 1$;

any ordered pair of nodes (n, n') can only be linked once, thus maximising the possible number of edges in G to $|E|_{max} = |N| \times (|N| - 1)$; finally, the definitive number of communities is left to be randomly decided during the generative operations.

A few previous works have explored the idea of using rules to generate networks. In [25], 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. The model presented below follows a rather 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, more intricate and more interesting to study.

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. Then comes the second phase where the new users reach the people they already know personally, thus creating new connections within the network, which may seem random for any spectator only aware of the present social network. Finally, the new users start to get to know the people with whom they are sharing friends in the network, potentially leading to the creation of new connections.

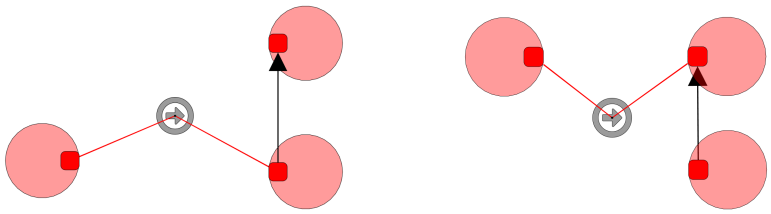
The method presented below can easily be extended to create graphs with more than one component. One has to use a number of starting nodes equal to the number of desired connected components and ensure that no edge is created between nodes from different components. The generative rules and strategies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

The first step (Sect. 3.1) generates a simple directed acyclic graph representing an initial simple network evolving as new users join it. It is then complemented with additional edges in the second step (Sect. 3.2), as users "import" their pre-existing connections into the social network. Finally, the third and final step (Sect. 3.3) focuses on creating communities as users connect with the friends of their friends within the network.

3.1 Generation of a directed acyclic graph

The first step toward the construction of the directed graph $G = (N, E)$ uses the two rules shown in Figures 2(a) and 2(b). Both rewriting operations start with a single node and transform it to generate a second node linked to the first one (thus creating a new node and a new edge with each application). The difference between those two rules lies in the edge orientation as Rule 2(a) creates an outgoing edge on the initiating node, while Rule 2(b) creates an incoming edge.

We can notice the left hand-sides of both rules require the existence of a node prior to their application, thus imposing the starting graph upon which the rules



(a) Rule *GenerationNode1*. (b) Rule *GenerationNode2*.

Fig. 2. Rules used for generating and re-attaching nodes to the social network. For both rules, a new node is created in the right-hand side and connected to the pre-existing node. The main difference between the two rules resides in the generated edge orientation: going from the pre-existing node (belonging to the social network) to the newly added node in Rule (a) or oriented in the opposite direction in Rule (b).

will be applied to have at least one node. As we also seek to ensure that only one connected component exists prior to any transformation, we use a single node as the starting graph.

Strategy 2: *Node generation*: Creating a directed acyclic graph of size N

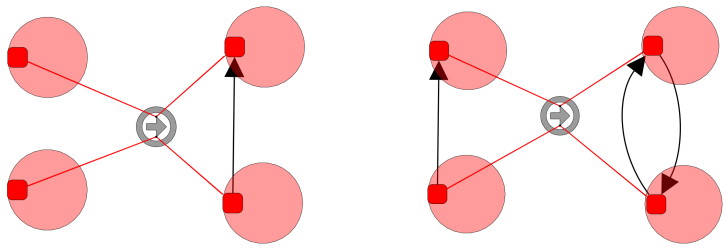
```
//equiprobabilistic application of the two rules used for generating nodes
repeat(
  ppick(one(GenerationNode1), 0.5,
        one(GenerationNode2), 0.5)
)(|N| − 1) // Generation of N nodes
```

The whole node generation is achieved during this first phase and managed using Strategy 2. 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 2 produces a graph with exactly $|N|$ nodes and $|N| - 1$ edges. The orientation of each edge varies depending of the rule applied (either 2(a) or 2(b)), moreover, their application using the `ppick`() construct allows us to ensure an equiprobable choice between the two rules. We focus next on generating additional edges.

3.2 Creating complementary connections

We still need to generate $(|E|-|N|+1)$ additional edges in the graph G. However, because we want to ensure the creation of communities during the last phase, we do not wish to create all the remaining edges just now. Depending on how we balance the number of edges created during this phase and the next one, the final graphs will present different characteristics (see Figures 5 and 6). During

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



(a) Rule *GenerationEdge*. (b) Rule *GenerationMirror*.

Fig. 3. First set of rules used to generate additional connections within the social network. Rule (a) is used to create a new connection between two previously created and unrelated nodes, while Rule (b) is applied on a pair of connected nodes and generates a new edge reciprocating the pre-existing connection.

We use two rules to link existing nodes thus creating a new additional edge with each application. The first rule (Fig. 3(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. 3(b)) reciprocates an existing connection between a pair of users: for two nodes $n, n' \in N$ connected with an edge (n', n), a new 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.

In both rules, the existence of edges between the nodes on which the rule applies should be taken into account. Though the rules visual representations do not explicitly indicate it, any edge (n, n') created by either rule cannot already exist in the network, thus forbidding the rules to apply in such case. This requirement can be taken into account by adding a condition "`where not Edge(n,n')`" introduced in Definition 3. It can also be handled through positions for limiting the elements to be considered during matching. We use the latter solution here. Strategy 3 presents how we proceed. First, we 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 banish all of its outgoing neighbours as they cannot be considered as potential matching elements. Then, Rule 3(a) or Rule 3(b) are equiprobably applied to add a new edge from the selected node. Previously banishing neighbours allows only considering pair of nodes not already connected. This ensures that the graph is kept simple (i.e., only one edge per direction between two nodes).

We aim to create $|E'|$ more edges, where $|E'| < (|E| - |N| + 1)$ to keep the number of edges below $|E|$. The use of the ( )`orelse`() construct allows testing

Strategy 3: *Edge generation*: addition of $|E'|$ edges if possible.

```
1 repeat(
2 //select one node with an appropriate number of neighbours
3   setPos(one(property(crtGraph, node, OutArity < |N| − 1)));
4 //for this node, forbid rule applications on its outgoing neighbours
5   setBan(all(ngbOut(crtPos, node, true)));
6 //equiprobable application of the edge generation rules
7   ppick((one(GenerationEdge))orelse(one(GenerationMirror)), 0.5,
8         (one(GenerationMirror))orelse(one(GenerationEdge)), 0.5);
9 )(|E'|)
```

all possible rule application combinations, thus, if one of the rules can be applied, it is found. If neither rule can be applied, the maximum number of edges in the graph has been reached, i.e., the graph is complete. The values given for the number of edges $|E'|$ is too high to create a simple graph. If the strategy went well, we are left with $(|E| - |E'| - |N| + 1)$ remaining edges to create in the next step for enforcing communities within G.

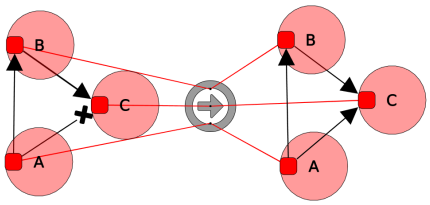
3.3 Construction of communities

To create a realistic social network, we want to add communities. Thus, some of the links between users have to follow certain patterns. Based on ideas advanced in several previous works (e.g., [13, 25, 29, 32]), we focus our interest on triad configurations (i.e., groups formed by three users linked together) to generate and extend communities via the three rewrite rules introduced in Figure 4.

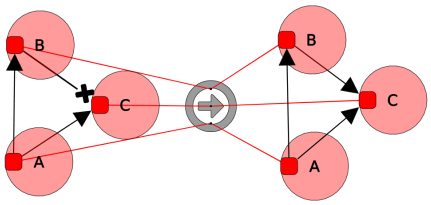
The first triad rule (Fig. 4(a)) considers how a first user (A) influences a second user (B) who influences in turn a third user (C). This situation can produce some sort of transitivity as "the idol of my idol is my idol", meaning that A is much likely to influence C. We use here the term "idol" instead of the more classical "friend" because we only consider single-sided relations as a base for the transformation. The second rule (Fig. 4(b)) shows two users (B and C) being influenced by a third user (A). When in this position, the users B and C might start exchanging (similar connections, common interests. . .), thus creating a relation between the two of them (either from B to C or the opposite). The last rule (Fig. 4(c)) depicts one user (B) being influenced by two other users (A and C). This case can happen when A and C are well-versed about a common subject of interest which is of importance to B. An exchange can thus appear between the two influential users (from A to C for instance).

The three rules use a `where not Edge(n,n')` condition to forbid the existence of an edge between two matching nodes. The condition is visually encoded using a cross-shaped headed edge to indicate which edge should be verified as non-existent during the matching operations.

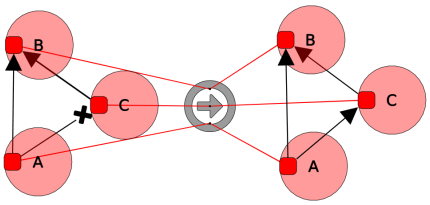
Strategy 4 is used to drive the three rules. Like the previous strategy, this one aims at equiprobably testing all possible combinations between the rules.



(a) Rule *CommunityLegacy*.



(b) Rule *CommunityDown*.



(c) Rule *CommunityUp*.

Fig. 4. Generation of additional connections based on triads. Rule (a) is used to identify influence chains: when C is influenced by B, itself influenced by A, the rule creates a new connection from A to C. Rule (b) focuses on triads where two users B and C are influenced by a third person A: this common characteristic can lead B and C to develop a relation. Rule (c) is somewhat the opposite of Rule (b): two users A and C influence a third user B, creating a connection between them (from A to C). 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.

3.4 Resulting network generation

Once the last strategy execution is completed, the social network generation is achieved. For the sake of simplicity, the strategies presented above aim at making equiprobable choices between rules. The probabilities may of course be modified to take into account any specific condition present in the modelled system, moreover, 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 port graph* G *with* $|N|$ *nodes and* $|E|$ *edges, which is simple, directed and weakly-connected.*

Proof. Let us prove by induction 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

Strategy 4: *Community generation*: remaining edges creation to strengthen communities

```
repeat(
 ppick(
  (one(CommunityDown))orelse(
   ppick(
    (one(CommunityUp))orelse(one(CommunityLegacy)), 0.5,
    (one(CommunityLegacy))orelse(one(CommunityUp)), 0.5)
   ), 1/3,
  (one(CommunityUp))orelse(
   ppick(
    (one(CommunityLegacy))orelse(one(CommunityDown)), 0.5,
    (one(CommunityDown))orelse(one(CommunityLegacy)), 0.5)
   ), 1/3,
  (one(CommunityLegacy))orelse(
   ppick(
    (one(CommunityDown))orelse(one(CommunityUp)), 0.5,
    (one(CommunityUp))orelse(one(CommunityDown)), 0.5)
   ), 1/3)
)(|E| − |E'| − |N| + 1)
```

for G_0 and each rewrite step preserves these three properties, thanks to the positioning strategy that controls the outdegree in *Edge generation* (Strategy 3) and the forbidden edges in the rules for *Community generation* (Figure 4). 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 rewriting induction. Then 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. From here, G 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 $|N|-1+|E'|+|E|-|E'|-|N|+1=|E|$. □

3.5 Implementation, Experimentation and Visualisation

We use the Porgy system [33] to experiment with our generative model. The latest version of the rewriting platform[5] is available either as source code or binaries for MacOS and Windows machines.

Figures 5 and 6 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. 5 and $|E'|=0$ for

[5] Porgy website: `http://tulip.labri.fr/TulipDrupal/?q=porgy`

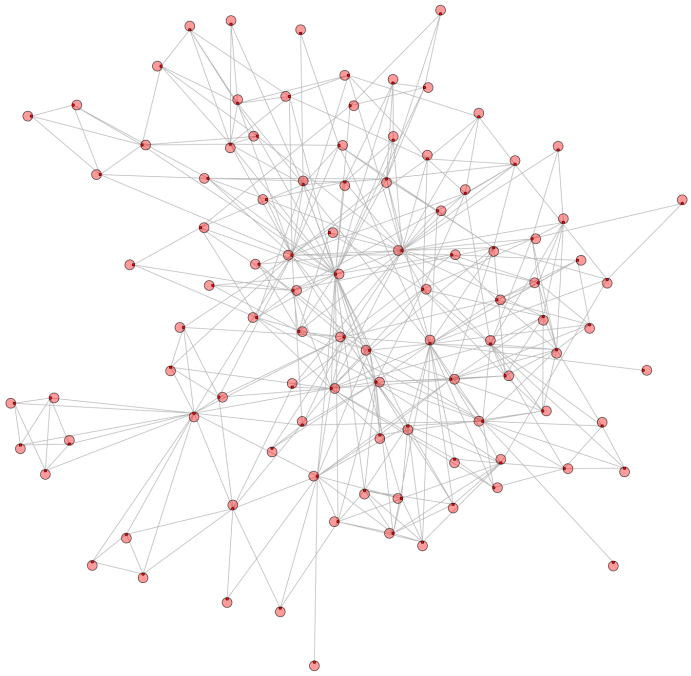


Fig. 5. A generated social network. $|N| = 100$ nodes, $|E| = 500$ edges and $|E'| = 50$. With these parameters, the average characteristic path length is $L \simeq 2.563$ and the average clustering coefficient is $C \simeq 0.426$.

Fig. 6. 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 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 [41]. 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 5 (respectively, Figure 6), 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.

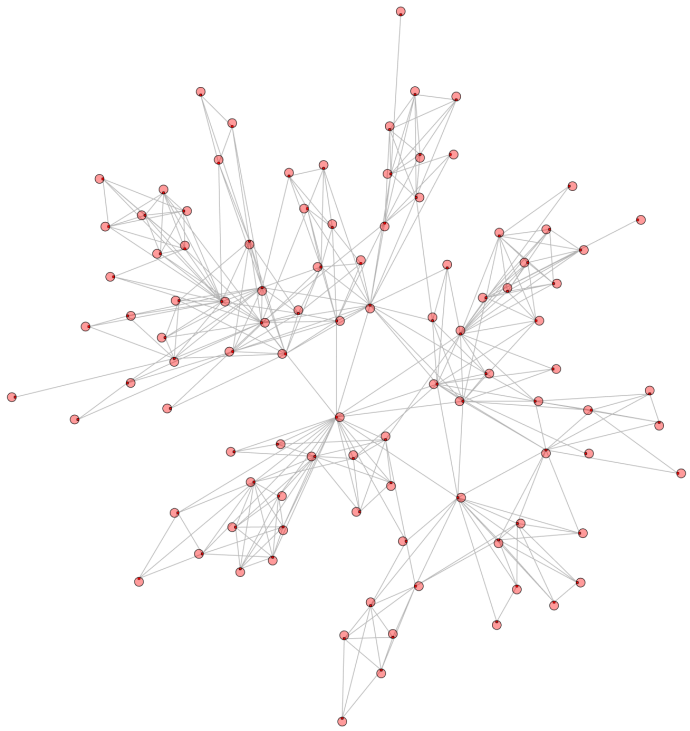


Fig. 6. A generated social network. $|N| = 100$ nodes, $|E| = 500$ edges and $|E'| = 0$. With these parameters, the average characteristic path length is $L \simeq 3.372$ and the average clustering coefficient is $C \simeq 0.596$.

The graphs generated using our method can be subsequently used as any randomly generated network. For instance, we have used such graphs in [37] to study the evolution of different information propagation models. PORGY was used in this work to run several propagation scenarios and analyse the resulting outputs with its visualisation tools.

4 Conclusion

Our first experiments and results on generation and propagation in social networks, obtained in [37] and in this work, illustrate how labelled port graph strategic rewriting provides a common formalism in which different mathematical models can be expressed and compared. The ultimate goal is to provide a simulation environment helpful for making decisions, such as choosing good parameters, detecting and preventing unwanted situations, or looking for a better diffusion strategy.

As a first approach to this ambitious challenge, we focused on social networks that already offer a big variety of situations and problems. Several lessons

and research directions can be drawn from this study, both for the rewriting community and for the social network community.

First, dealing with this application domain led us to validate the concepts of labelled port graphs on a given signature, of rules that are themselves also labelled port graphs with variables from the given signature, and of strategy constructs added to define positions in graphs in a flexible way. When modelling the evolution of the studied network, the derivation tree (also a port graph) provides support for history tracking, state comparison, state recovery and backtracking. For the social network community, the rewrite rule approach is not quite surprising because some works such as [25] already use rules to generate social networks, although without claiming it. The fact that different models can be expressed in a common formalism provides a good argument for those who are interested to compare various algorithms and models. In such situations, simulations can indeed help for taking decision, for instance to prevent bad situations, or to look for optimal diffusion strategy.

Indeed several issues remain to address. For rewriting, although graph rewriting has been largely studied, addressing social network applications causes a drastic change of scale for the structures. Dealing with millions of nodes and edges requires great attention to size and complexity. There is also room for improvement in data storage and retrieval –in connection with 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 information 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.

As programs, data need certification and validation tools and process, not only at one 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 richer graph structures, is yet an actual research topic for the visualisation community.

In future work, we plan to address multi-layer networks, based on societal problems. An example is tracking criminal activities. The objective then is to build a new methodology for tracking, based on construction, manipulation and analysis of heterogeneous digital information coming from different sources: legal records of tribunal sentences, social networks coming from exchanges, meetings, phone calls, information on financial flows and even family relations. Beyond the modelisation challenge, in connection with jurists and social scientists, we expect

that our formalism of labelled port graphs, rules and strategy will provide an adequate framework for simulations and hypotheses testing.

Acknowledgements. We thank Guy Melançon (University of Bordeaux) and all the other members of the PORGY project. We also thank the anonymous reviewer for carefully reading this paper and making valuable suggestions for improvement.

References

1. Oana Andrei, Maribel Fernández, Hélène Kirchner, Guy Melançon, Olivier Namet, and Bruno Pinaud. PORGY: Strategy-Driven Interactive Transformation of Graphs. In R. Echahed, editor, 6th *Int. Workshop on Computing with Terms and Graphs*, volume 48, pages 54–68, 2011.
2. Oana Andrei and Hélène 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. Oana Andrei and Hélène 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. Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau, and Antoine Reilles. Tom: Piggybacking Rewriting on Java. In F. Baader, editor, *RTA*, volume 4533 of *LNCS*, pages 36–47. Springer, 2007.
5. Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
6. H.P. Barendregt, M.C.J.D. van Eekelen, J.R.W. Glauert, J. R. Kennaway, M.J. Plasmeijer, and M.R. Sleep. Term graph rewriting. In *Proc. of PARLE, Parallel Architectures and Languages Europe*, number 259-II in LNCS, pages 141–158. Springer-Verlag, 1987.
7. Klaus Barthelmann. How to construct a hyperedge replacement system for a context-free set of hypergraphs. Technical report, Universität Mainz, Institut für Informatik, 1996.
8. Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Phys. Rev. E*, 71:036113, Mar 2005.
9. 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.
10. Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Christophe Ringeissen. An overview of ELAN. *Electronic Notes in Theoretical Computer Science*, 15:55–70, 1998.
11. Ulrik Brandes and Dorothea 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.
12. P.J. Carrington, J. Scott, and S. Wasserman. *Models and Methods in Social Network Analysis*. Structural Analysis in the Social Sciences. Cambridge University Press, 2005.
13. Dorwin Cartwright and Frank Harary. Structural balance: a generalization of Heider's theory. *Psychological Review*, 63:277–293, 1956.

14. Wei Chen, Alex Collins, Rachel Cummings, Te Ke, Zhenming Liu, David Rincón, Xiaorui Sun, Yajun Wang, Wei Wei, and Yifei Yuan. Influence maximization in social networks when negative opinions may emerge and propagate. In *Proc. of the* 11th *SIAM Int. Conf. on Data Mining, SDM 2011*, pages 379–390, 2011.
15. Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proc. of the* 16th *ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '10, pages 1029–1038. ACM, 2010.
16. Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel, and Michael 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.
17. P.S. Dodds and D.J. Watts. A generalized model of social and biological contagion. *Journal of Theoretical Biology*, 232(4):587 – 604, 2005.
18. Paul Erdős and Alfréd Rényi. On the evolution of random graphs. In *Publication of the Mathematical Institute*, volume 5, pages 17–61. Hungarian Academy of Sciences, 1960.
19. Maribel Fernández, Héìène Kirchner, and Bruno Pinaud. Strategic Port Graph Rewriting: An Interactive Modelling and Analysis Framework. Research report, Inria, January 2016.
20. Maribel Fernández, Hélène Kirchner, and Olivier 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.
21. Maribel Fernández, Hélène Kirchner, and Bruno 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.
22. Amit Goyal, Francesco Bonchi, and Laks V.S. 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.
23. M. Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83(6):1420, 1978.
24. Annegret Habel, Jürgen Müller, and Detlef Plump. Double-pushout graph transformation revisited. *Mathematical Structures in Computer Science*, 11(5):637–688, 2001.
25. N. Kejžar, Z. Nikoloski, and V. Batagelj. Probabilistic inductive classes of graphs. *The Journal of Mathematical Sociology*, 32(2):85–109, 2008.
26. David Kempe, Jon Kleinberg, and Éva 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.
27. Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P. Gleeson, Yamir Moreno, and Mason A. Porter. Multilayer networks. *Journal of Complex Networks*, 2(3):203–271, 2014.
28. Yves Lafont. Interaction nets. In *Proc. of the* 17th *ACM Symp. on Principles of Programming Languages (POPL'90)*, pages 95–108. ACM Press, 1990.
29. Jure Leskovec, Daniel Huttenlocher, and Jon 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.

30. Stanley Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.
31. Mark Newman, Albert-László Barabási, and Duncan J. Watts. *The structure and dynamics of networks*. Princeton Studies in Complexity. Princeton University Press, 2006.
32. Bobo Nick, Conrad Lee, Pádraig Cunningham, and Ulrik 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.
33. Bruno Pinaud, Guy Melançon, and Jonathan Dubois. PORGY: A Visual Graph Rewriting Environment for Complex Systems. *Computer Graphics Forum*, 31(3):1265–1274, 2012.
34. Detlef 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.
35. Detlef Plump. The Graph Programming Language GP. In S. Bozapalidis and G. Rahonis, editors, *CAI*, volume 5725 of *LNCS*, pages 99–122. Springer, 2009.
36. John Scott and Peter J. Carrington. *The SAGE Handbook of Social Network Analysis*. SAGE, 2011.
37. Jason Vallet, Hélène Kirchner, Bruno Pinaud, and Guy 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.
38. Eelco Visser. Stratego: A language for program transformation based on rewriting strategies. System description of Stratego 0.5. In *Proc. of RTA'01*, volume 2051 of *LNCS*, pages 357–361. Springer-Verlag, 2001.
39. Lei 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.
40. Duncan J. Watts. A simple model of global cascades on random networks. *Proc. of the National Academy of Sciences*, 99(9):5766–5771, 2002.
41. Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.
42. Lee Wonyeol, Kim Jinha, and Yu Hwanjo. CT-IC: Continuously activated and time-restricted independent cascade model for viral marketing. In *Data Mining (ICDM), 2012 IEEE* 12th *Int. Conf. on*, pages 960–965, 2012.