Where Social Networks, Graph Rewriting and Visualisation Meet : Application to Network Generation and Information Diffusion

Jason Vallet

▶ **To cite this version:**

Jason Vallet. Where Social Networks, Graph Rewriting and Visualisation Meet : Application to Network Generation and Information Diffusion. Other [cs.OH]. Université de Bordeaux, 2017. English. NNT : 2017BORD0818 . tel-01691037

HAL Id: tel-01691037
https://theses.hal.science/tel-01691037

Submitted on 23 Jan 2018

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.

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

par **Jason Vallet**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Where Social Networks, Graph Rewriting and Visualisation Meet: Application to Network Generation and Information Diffusion

---

Date de soutenance : Jeudi 7 Décembre 2017

Devant la commission d'examen composée de :

Hélène Kirchner	DR, Inria, FRA	Présidente du Jury
Vladimir Batagelj	PR, IMFM, Univ. Ljubljana, SVN ..	Rapporteur
Benoît Otjacques	DR, LIST, LUX	Rapporteur
David Auber	MCF, LaBRI, Univ. Bordeaux, FRA	Examinateur
Tatiana von Landesberger	MCF, GRIS, TU Darmstadt, DEU ..	Examinatrice
Guy Melançon	PR, LaBRI, Univ. Bordeaux, FRA .	Directeur de Thèse
Bruno Pinaud	MCF, LaBRI, Univ. Bordeaux, FRA	Co-Directeur

– 2017 –

Abstract In this thesis, we present a collection of network generation and information diffusion models expressed using a specific formalism called strategic located graph rewriting, as well as a novel network layout algorithm to show the result of information diffusion in large social networks. Graphs are extremely versatile mathematical objects which can be used to represent a wide variety of high-level systems. They can be transformed in multiple ways (e.g., creating new elements, merging or altering existing ones), but such modifications must be controlled to avoid unwanted operations. To ensure this point, we use a specific formalism called strategic graph rewriting. In this work, a graph rewriting system operates on a single graph, which can then be transformed according to some transformation rules and a strategy to steer the transformation process. First, we adapt two social network generation algorithms in order to create new networks presenting small-world characteristics. Then, we translate different diffusion models to simulate information diffusion phenomena. By adapting the different models into a common formalism, we make their comparison much easier along with the adjustment of their parameters. Finally, we finish by presenting a novel compact layout method to display overviews of the results of our information diffusion method.

Titre Quand les réseaux sociaux, la réécriture de graphes et la visualisation se rencontrent : application à la génération de réseaux et à la diffusion d'information.

Résumé Dans cette thèse, nous présentons à la fois une collection de modèles de générations de réseaux et de diffusion d'information exprimés à l'aide d'un formalisme particulier appelé la réécriture de graphes, ainsi qu'une nouvelle méthode de représentation permettant la visualisation de la diffusion d'information dans des grands réseaux sociaux. Les graphes sont des objets mathématiques particulièrement versatiles qui peuvent être utilisés pour représenter une large variété de systèmes abstraits. Ces derniers peuvent être transformés de multiples façons (création, fusion ou altération de leur éléments), mais de telles modifications doivent être contrôlées afin d'éviter toute opération non souhaitée. Pour cela, nous faisons appel au formalisme particulier de la réécriture de graphes afin d'encadrer et de contrôler toutes les transformations. Dans notre travail, un système de réécriture de graphes opère sur un graphe, qui peut être transformé suivant un ensemble de règles, le tout piloté par une stratégie. Nous commençons tout d'abord par utiliser la réécriture en adaptant deux algorithmes de génération de réseaux, ces derniers permettant la création de réseaux aux caractéristiques petit monde. Nous traduisons ensuite vers le formalisme de réécriture différents modèles de diffusion d'information dans les réseaux sociaux. En énonçant à l'aide d'un formalisme commun différents algorithmes, nous pouvons plus facilement les comparer, ou ajuster leurs paramètres. Finalement, nous concluons par la présentation d'un nouvel algorithme de dessin compact de grands réseaux sociaux pour illustrer nos méthodes de propagation d'information.

Keywords Graph Rewriting, Network Generation, Information Diffusion, Network Visualisation

Mots-clés Réécriture de graphes, Génération de réseaux, Diffusion d'information, Visualisation de réseaux

Laboratoire d'accueil Unité Mixte de Recherche CNRS (UMR 5800) 351, cours de la Libération F-33405 Talence cedex

Acknowledgments

Like every thesis, all the work leading to the creation of this dissertation would not have been possible without a lot of different people. My first thanks go to my supervisors who have always been interested and enthusiastic about my work: Guy Melançon, for introducing me to the colourful world of information visualisation and visual analysis a mere 5,000 Km away from home, and Bruno Pinaud, for giving me the opportunity to work on Porgy and getting used to me barging in his office with questions no matter when. I am really grateful to the both of them for offering me a complete "creative" freedom when doing researches even though the different ideas may not have always seemed essential, or even related, at the time. I sincerely hope they find the finished result was worthy of their time.

I wish to thank all the following persons for accepting to be part of my jury. Vladimir Batagelj, from the *University of Ljubljana* (IMFM), and Benoît Otjacques, from the *Luxembourg Institute of Science and Technology*, for their keen interest in my work and for accepting to review my dissertation. Hélène Kirchner, from the *Institut National de Recherche en Informatique et Automatique*, for presiding the jury during my defence as well as her eagerness and patience during our many meetings. Tatiana von Landesberger, from the *Technische Universität Darmstadt* (GRIS), for inviting me in her research team for a few weeks and offering me a different point-of-view on visual analysis. David Auber, from the *University of Bordeaux* (LaBRI), for sharing Tulip with the world and successfully getting me to run.

I would like to thank all my colleagues from the EVADoMe group and in the *Laboratoire Bordelais de Recherche en Informatique* who had to bear with me at one time or another in the last four years. Congratulations once more to the ones who have been quicker than me in completing their thesis, Noël Gillet, Patrick Kamnang-wanko, Alexandre Perrot, Joris Sansen, and Thanh Tung Tran; and I can only advise the ones whose turn is coming during the next two years, Aarón Ayllón Benitez, Rémi Delassus, Antoine Hinge, Antoine Laumond, Haolin Ren, and Gaëlle Richer, to take heart. Many thanks to David Auber, Romain Bourqui, Romain Giot, and Sofian Maabout for their feedback and advices when discussing or presenting my work to them throughout this thesis, and, of course, to our incredible group of engineers, Norbert Feron, Frédéric Lalanne, and Patrick Mary, for indulging either my awful sense of humour or technical illiteracy. Thanks also to Benjamin Renoust, from the *Japanese National Institute of Informatics*, for his support each time our paths have crossed in the last few years, and I want to commend all the administrative staff of the LaBRI who certainly make the life of all the researchers there much easier.

Many thanks to my collaborators Maribel Fernández, from *King's College London*, and Hélène Kirchner for sharing your ideas and listening to mine, this thesis would have been very different without their help. I am also grateful to the whole *opencare* consortium and in particular to Luce Chiodelli, from the *University of Bordeaux*, for her fortitude and poise as an administrator, to Alberto Cottica and Noemi Salantiu, from *Edgeryders*, for teaching me that everyone can help in making a difference, and to Amelia Hassoun, from the *University*

of Oxford, for coping with my technical jargon. I would like to extend my thanks to the whole *Graphisch-Interaktive Systeme* team from the *Technische Universität Darmstadt* for welcoming me and exchanging about our respective works.

I also need to thank all my other friends who have been a much appreciated distraction in my every day life and more occasionally for those whom are farther away. Whether they try to address computer problems, mathematical questions, or to repair humans, they are the bests.

Last but not least, I would particularly like to express my deepest gratitude to my parents Angéline and Jean-Michel, who never doubted me and always backed me up throughout my studies, although I am quite sure they have been left more than once wondering what I am doing exactly. Finally, thanks to Ludivine and Adrien, my "little" sister and brother-in-law, for being there for somebody who has not been around a lot lately.

This thesis is dedicated to all the persons searching for answers:
may they find what they are looking for but never stop asking new questions.

Contents

1 Introduction **1**
- 1.1 Context . . . 2
- 1.2 Contributions . . . 3
- 1.3 Outline of the thesis . . . 5

2 Definitions **7**
- 2.1 On graphs and their specifications . . . 8
- 2.2 Graph rewriting . . . 20
- 2.3 Strategic located graph rewriting . . . 30
- 2.4 Conclusion . . . 39

3 Modelling graph generation algorithms **41**
- 3.1 Related works . . . 43
- 3.2 Translating the small-world model . . . 47
- 3.3 Introducing a new social network generative model . . . 61
- 3.4 Conclusion . . . 73

4 Modelling information diffusion in social networks **75**
- 4.1 Propagation in social networks . . . 77
- 4.2 Modelling cascading and threshold behaviours . . . 81
- 4.3 Transforming a privacy-preserving dissemination model . . . 95
- 4.4 Conclusion . . . 108

5 Network visualisation using a compact overview **111**
- 5.1 Displaying graphs and networks . . . 113
- 5.2 JASPER: a pixel-oriented overview for large graphs . . . 123
- 5.3 User experiment: visualisation validation . . . 138
- 5.4 Conclusion . . . 147

6 Conclusion **149**
- 6.1 Summary . . . 149
- 6.2 Perspectives . . . 151
- 6.3 Discussion . . . 153
- 6.4 Conclusion . . . 154

Bibliography **157**

List of Figures **185**

A Author's publications **189**

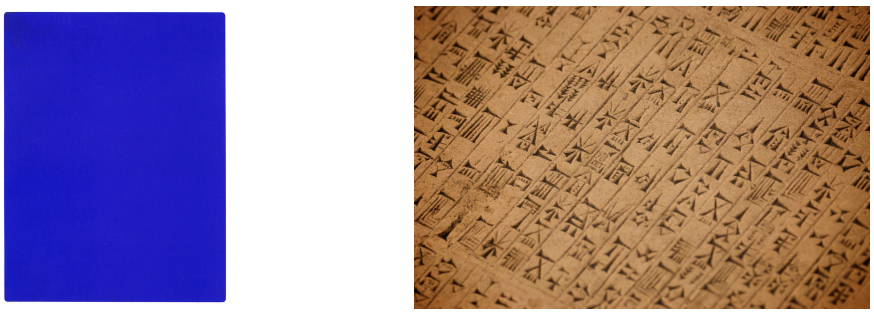
B Porgy computation times **193**

C Small-world model analysis **195**

Chapter 1

Introduction

Whereas, as the saying goes, a picture is really worth a thousand words is rather hard to prove or quantify. Though this affirmation may not be entirely true for every existing image (see Figure 1.1a[1] for instance), graphical depictions have nonetheless been used to indirectly transfer information for a very long time. Writing, the most important human invention, is undoubtedly the best evidence of such attainment (Daniels and Bright, 1996). From its first and most basic form using cuneiform script (as shown in Figure 1.1b), to more modern writing systems, such as the Latin alphabet this thesis is lettered with, writing has been used throughout the whole world during the better part of the last ten thousand years by diverse human communities and settlements (Woods et al., 2010). Obviously, the ability to represent knowledge in a graphical and comprehensive manner, which can be at the same time easily understood and replicated, offers tremendous advantages. This feature allows us to preserve traces of history, reflections, experimentations, results, achievements, and pretty much any other piece of information which can be described using a written language.



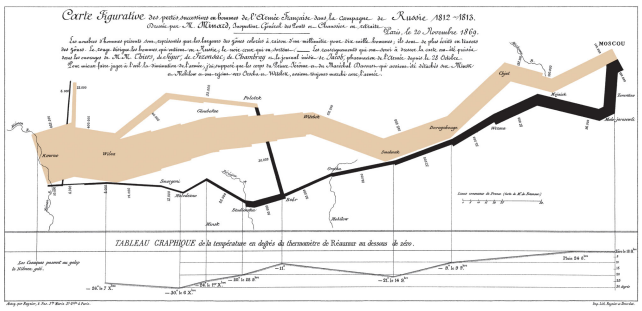
(a) Example of a minimalist monochromatic painting. "*IKB 191*" by Yves Klein (1962).[2]

(b) Photograph of a cuneiform script excerpt. "Mesopotamian writings" by Paul Hudson.[3]

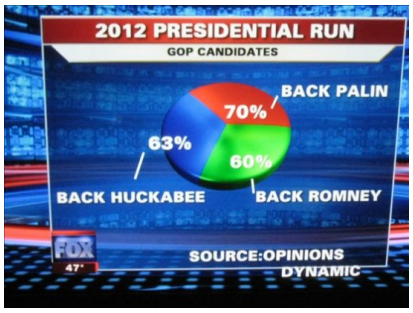
Figure 1.1: Example of visual representation in the context of art and writing. While pieces of art can sometimes be quite literal and display the scene almost exactly as the author captured it, like for paintings of the Impressionism movement (https://en.wikipedia.org/wiki/Impressionism), writings always convey an abstract and more complex idea even when the characters are pictographs.

[1] The best counter-example of this affirmation is incontestably this minimalist painting by Yves Klein which we believe could simply be described in one word: *blue*.

In many ways, the aforementioned possibilities are very close to the undertakings commonly asked of the visualisation specialists. Their main goal is to help others to understand and effectively communicate information through a visual medium. Evidently, different factors come into play which can either make a representation agreeable and particularly easy to understand (Fig. 1.2a), or at the contrary illogical and confusing (Fig. 1.2b). In particular, as each individual possesses different notions of aesthetics (Smith et al., 2004) and perceives the world differently (Ware, 2012), establishing universal solutions which can be effective on everybody with a similar efficiency can become rather complicated. Nonetheless, lessons have been learned along the years, several working solutions have been identified and sets of rules, guidelines, and dos and don'ts are available to everyone in any book introducing the field of information visualisation (e.g., Bertin (1983), Cairo (2012), Munzner (2014), Ward et al. (2010)).



(a) Minard's depiction of Napeleon's army during the 1812 Russian campaign



(b) Pie chart presented on Fox News displaying results of an opinion poll

Figure 1.2: Information visualisation at its best and worst. While Minard's 1869 graphic is often claimed to be the best statistical graphic ever drawn, more recent graphical depiction can hardly qualify as being rational (Source: http://www.datavis.ca/gallery/).

1.1 Context

The idea of information visualisation is very general and can be used to characterise any operation representing a piece of information through a visual mean. While a global understanding of a domain being studied is always useful, our interest in this dissertation lies toward a much narrower field, focussing on the depiction of *graphs*. These mathematical objects are extremely versatile and can be used to represent a wide variety of high-level systems. They are also particularly adapted to model sets of elements presenting different kinds of connections with one another. This allows the graph to be used to model systems as different as interpersonal relationships (groups of friends, genealogical trees), communication or power grid (Internet, electricity), and transportation infrastructures (roads, railroads). Based on these few examples, one can easily guess that, in term of visualisation, graphs are very popular objects. It thus comes as no surprise that several well established tools are available to represent them and perform different kinds of analysis and operations on them. Among the most popular solutions, we can mention Cytoscape[4] (Shannon et al., 2003), Gephi[5] (Bastian et al., 2009), Pajek[6] (Batagelj and Mrvar,

[2] https://en.wikipedia.org/wiki/File:IKB_191.jpg
[3] https://www.flickr.com/photos/pahudson/4288400913
[4] http://www.cytoscape.org/
[5] https://gephi.org/
[6] http://mrvar.fdv.uni-lj.si/pajek/

2004), and TULIP[7] (Auber et al., 2016) developed at the University of Bordeaux.

In the work we present in this thesis, we are interested in graphs but more specifically in ways to transform them. These modifications can take many forms, for instance, new elements can be created or existing elements are merged or altered. However, such transformations are not expected to occur chaotically. Indeed, in order to standardise these operations, we use a specific formalism called *graph rewriting*. A graph rewriting system operates on a single graph, which is to be transformed according to a few different *rewriting rules* belonging to the system. As these rules can be used to perform any kind of transformation on the graph without limitations, the possibilities offered by the graph rewriting formalism are virtually endless. This also means that it must be possible to *take any existing algorithm which can be applied on a graph and express the sequence of operations it performs as rewriting rules*. These rules could then be used in a graph rewriting system to reproduce the behaviour of the original algorithm using solely the graph rewriting formalism.

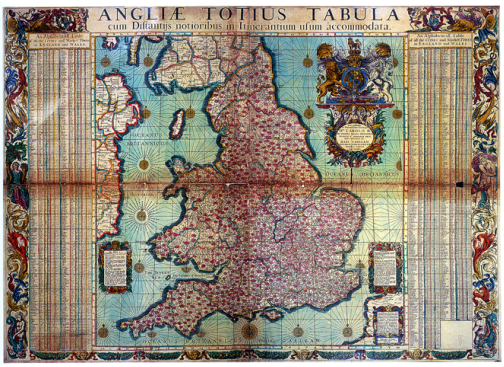
As we look to put this hypothesis to the test, we must first choose a graph rewriting system implementation. Among the different existing solutions, which we will present later in this document, we propose to use the visual rewriting platform called PORGY[8] (Pinaud et al., 2012), which is developed on top of the TULIP information visualisation framework. In addition to being open-source and developed in-house at the University of Bordeaux, the visual dimension proposed by PORGY makes the definition of the rewriting rules easier but it also allows the user to create representations of the graph being currently transformed to observe the rewriting operation taking place. While we use a specific rewriting platform, we think it is important to mention that the work developed in this dissertation is entirely reproducible on a different solution. However, it will be necessary for this new system to adapt its rewriting implementation to propose all the peculiar features defined in our work.

1.2 Contributions

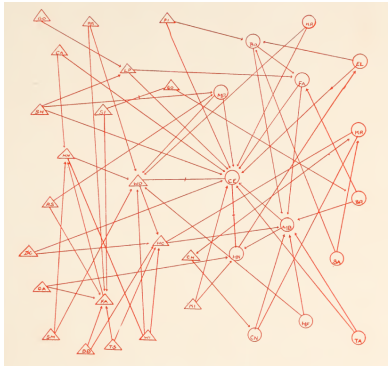
While graphs can be used in a lot of different contexts, their elements tend to be "decorated" with properties relevant to the situation at hand. Let us consider a transportation infrastructure for instance. Connections between cities, such as roads, can be described by their length, quality, speed limit, if there is traffic, if a toll must be paid, and so on. As they are more than simple mathematical objects defining a topology, such "decorated" graphs are commonly qualified as *networks*. Examining these more complex objects instead of plain graphs opens a lot of different possibilities to us. Multiple algorithms concerning transportation networks are widely used nowadays such as the Dijkstra algorithm[9] to compute the shortest road between any two cities (Fig. 1.3a). Trying to adapt an algorithm of this nature would incontestably be more interesting than studying the classical Depth-First Search and Breadth-First Search algorithms. However, we decide to give in to our own preference and to consider social networks instead.

These specific networks are used to represent interpersonal connections of any nature, such as family, friend, foe or colleague relations. While social networks have a long history (see Scott and Carrington (2011) and Fig. 1.3b), the rapid adoption of Internet and the success of online social networks has sparked a world-wide interest on the subject and offered a lot of material to work on. From a personal perspective, some key aspects of the social network inspire a very strong interest. The first one concerns how online social networks come to life and expand by inviting and integrating little by little new individuals. The manner in which the new arrivers decide

[7] http://tulip.labri.fr
[8] http://porgy.labri.fr
[9] https://en.wikipedia.org/wiki/Dijkstra's_algorithm



(a) Annotated map presenting the distance between English cities in 1679 (Adams, 1679)



(b) Representation of the social exchanges between 1st grade students (Moreno, 1934)

Figure 1.3: Visual representations of graphs (or networks) carrying a specific meaning.

with whom to connect or which groups to take part in or avoid reflects a lot about each person from our point of view. Our first proposition would be to consider a well-known generative model able to create artificial social networks and create its adaptation using the rewriting formalism.

The second aspect we consider is the diffusion of information throughout the network. As everybody is connected and able to communicate with a lot of persons almost instantly by using either electronic mailing lists or online networking services like Twitter[10] or Mastodon,[11] understanding how information comes to pass using word-of-mouth so quickly is essential. It would allow to either make the most of the technology (e.g., communication during emergency situations) or avoid any abusive case (e.g., false or sensitive information being spread). For our second proposition, we thus choose a diffusion algorithm and translate it to our formalism.

While a few other leads are available to us, we limit ourselves to the two aspects of social networks developed above in order to allow us to completely focus on them. Of course, we should not limit ourselves to a single adaptation for each aspect. While an adaptation of a generation algorithm is interesting, proposing a new generative model resulting in a graph presenting known characteristics of social network would be even better. In the same way, why not try to adapt different diffusion algorithms and create a new one. Furthermore, it would be interesting to see if and how the translation into a graph rewriting system differentiates or brings closer two given models. We thus propose to offer the following contributions in this dissertation with respect to our own graph rewriting system:

1. We formally define all the features needed to perform the different adaptations towards our graph rewriting system implementation.

2. We present through applied examples the translation process to follow in order to adapt network generation and information diffusion algorithms into our graph rewriting formalism.

3. We propose a new network generation algorithm based on the previous findings.

4. We propose a new diffusion algorithm also based on our graph rewriting formalism.

These four goals are our key contributions in term of graph rewriting.

[10] https://twitter.com/
[11] https://mastodon.social

In addition to the previous points, as we are using a visual platform to perform all of our transformations, it would be quite nonsensical to not propose at least a contribution to improve the visual representation of the results. While the graph being rewritten is going to evolve during the generation, the diffusion phase is going to leave our graph unchanged except for its elements' inner properties. This implies that we would need to design a new graph drawing method adapted to display elements subjected to information diffusion. Elaborating graph drawing algorithms and visualisations represent several specific challenges, especially if the solution needs to be validated through a user experimentation. We nonetheless propose to validate our solution to ensure that the final layout is usable and offers qualities lacking to some of the pre-existing representation methods. Consequently, we propose the following contributions generalised to avoid limiting its use to the graphs resulting from the rewriting transformations:

5. We produce a new general representation method to visualise graphs and evaluate it to ensure the solution offers better results than the existing literature.

While this last contribution is evidently different from the first ones more linked to the rewriting formalism, it nonetheless allows us to propose a rather complete workflow. We can thus start with a network generation, perform a model simulation, and conclude with a visualisation of the resulting network. With all of our contributions announced loud and clear, we are ready to develop our work.

1.3 Outline of the thesis

The thesis is organised as follows:

Chapter 2 establishes all the definitions we will use throughout this thesis. We first recall some basic notions about graph theory and follows with a detailed description of the graph rewriting formalism when using strategies and located rewriting rules applications.

Chapter 3 proposes to employ the graph rewriting formalism previously introduced to generate graphs. Starting with a quick overview of the related works, we then present two generative models, respectively composed of a few rewrite rules and a strategy, both able to produce networks displaying small-world characteristics.

Chapter 4 is used to explore a second problematic, turning this time our focus towards information diffusion models. Two propagation models are thus studied and adapted using our graph rewriting formalism to simulate propagation phenomena. We then turn toward a dissemination model which we use as a basis for inspiration and improve upon using the knowledge gained during the previous model adaptations.

Chapter 5 aims at tackling the visualisation of networks representing diffusion phenomena. We specifically present and evaluate a pixel-oriented visualisation adapted for displaying large graphs in a sensible amount of time.

We then end this document with some final perspectives, discussions and conclusion.

Chapter 2

Definitions

Contents

2.1 On graphs and their specifications . . . 8
2.1.1 Generalities . . . 9
2.1.2 Labelling and port graphs . . . 13
2.2 Graph rewriting . . . 20
2.2.1 Port graph morphism . . . 21
2.2.2 Port graph rewrite rule . . . 22
2.2.3 Matching . . . 26
2.2.4 Rewriting step and derivation . . . 28
2.3 Strategic located graph rewriting . . . 30
2.3.1 Located rewriting . . . 30
2.3.2 Strategic graph programs . . . 32
2.4 Conclusion . . . 39

Although the yearning reader is eagerly awaiting for some modelling, analysis and visualisations, this chapter, as its name so eloquently hints at, mostly focuses on introducing and defining the concepts presented and used throughout the rest of this thesis. Some of them are common knowledge and can most certainly be skipped over by anyone with the faintest notion of graph theory. However, several of the remaining concepts also introduce or are based on singular notions which will only appear familiar to readers acquainted with works on graph rewriting techniques. To avoid any possible misunderstanding with the readers unfamiliar with such topics, the different definitions are brought forth step-by-step, each building on the previous ones until all of the notions needed to express the context of strategic graph rewriting are finally introduced.

Most of the definitions presented in this chapter have already been introduced under one form or another in one of the papers presenting results for PORGY. However, several of them have sensibly evolved along the years as the software, being a research tool, is continually expanding, its inner system is subject to re-factoring and new functionalities are regularly implemented. Some of these changes are minor and may only affect a small part of one definition but others

This chapter borrows content from:
Maribel Fernández, Hélène Kirchner, Bruno Pinaud, and Jason Vallet. *Labelled Graph Strategic Rewriting for Social Networks*. Currently under review. Submitted to the Journal of Logical and Algebraic Methods in Programming (JLAMP) in November 2016; major revision submitted in September 2017.

introduce much wider revisions, implying numerous adjustments throughout several definitions. This still makes the existing definitions entirely accurate in the context of their respective papers, which we will present later, while also allowing the readers to witness and follow the evolution taking place in the definitions with each newly introduced operation or constituent.

For this thesis, we have adapted the existing definitions in their most recent occurrence, as presented in Fernández et al. (2016a). We thus propose to use directed labelled (port) graphs as a basis instead of the usual non-directed (port) graphs with attributes stored in an external record. Although both objects are adapted for the task and can be considered as equivalent, the labelled graph is, from our point of view, a more familiar object to manipulate. Furthermore, its formal definition and the graph implementation we use are both structured very similarly, thus making the transition from one description to the other seamless and coherent. Overall, the definitions presented in this thesis have been reworked to propose formulations using labels and labelling functions instead of records and have been generalised to handle both directed and undirected graphs.

In a few words, graph rewriting can be described as a technique allowing to transform, or *rewrite*, a given part of a graph. Objects named *rules* are defined to indicate which part of the graph to target and the exact transformations to apply on it. Ultimately, such powerful technique allows unrestricted modifications but also implies precise characterisation in order to accurately specify and target the elements to transform. Once several such transformations, or rules, are defined, we propose to use procedures called *strategies* to manage the different transformations to carry out on the graph and to describe how, when and where to apply them. To achieve this, strategies are built using their own language. Effectively, with a strategy, a set of rules, and a graph to transform, we have a complete system, equivalent to a program and the set of functions it applies.

The definitions proposed in this chapter are grouped in three thematics according to their context. Firstly, we propose a simple reminder of the standard definitions related to graphs from a general point of view, presenting them in a few of their different forms and recalling their corresponding properties. We complete this section with a more detailed account on labelled graphs and port graphs which we use as the structural basis for all of our graph transformations. Secondly, we define the whole process surrounding the graph rewriting operations, from the construction and specification of rewrite rules, to the resulting transformed graph and the trace of modifications it underwent. Lastly, we introduce the strategies as well as the notion of located (port) graph rewriting to precisely target chosen elements and administer complex and advanced transformations operations.

2.1 On graphs and their specifications

This first section proposes the more general definitions and offers some recalls and reminders on graph theory. Although we consider the readers to have basic notions of graph theory, the definitions and visual examples given along the way have been designed to allow anybody with a comprehension of set theory and mathematical functions to understand them. Due to the growing interest towards graphs as mathematical and computational objects, a plethora of books, courses and online resources has been written on the topic, and, apart for some terminology variations depending of the authors' originating field of research, all of these resources present ostensibly similar content. We propose to the interested reader the following short list of quality references, either focussing on graph theory (Bollobas, 1998; Diestel, 2000; Bondy and Murty, 2011) or using this very topic as a basis and expanding upon it in order to perform network analysis (Brandes and Erlebach, 2005; Batagelj, 2009; Cohen and Havlin, 2010).

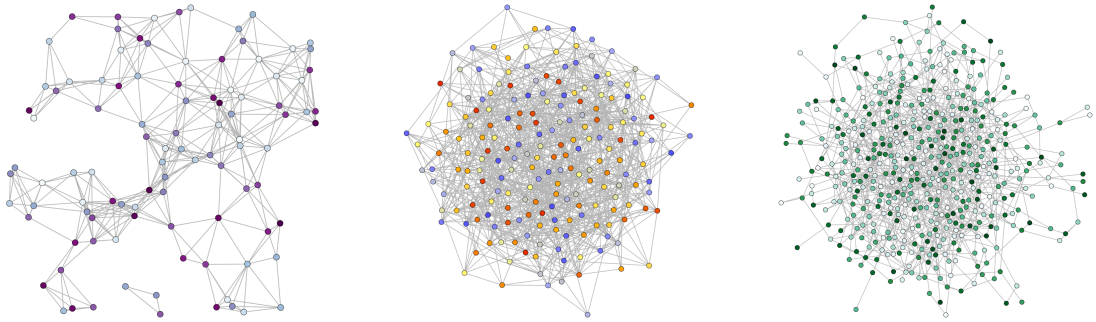
2.1.1 Generalities

Graphs are very common and well-known data structures. While Euler is often mentioned as the first mathematician to have used the concept of a graph to introduce the Königsberg bridges puzzle,[1] this mathematical objects is mostly used nowadays to represent relational data, that is any data where two entities are linked by a relation. We usually define a graph as follows:

Definition 1 (Graph). *A* ***graph*** *is denoted as* $G = (N, E, \mathscr{E})$ *where*

- N *represents a set of* nodes *(also called* vertices *or* entities*),*
- E *describes the set of* edges *(sometimes mentioned as* links*,* relations *or* connections*), and*
- $\mathscr{E}$ *is a function associating each* edge *to an* ***unordered*** pair of nodes *such as* $\mathscr{E} : E \mapsto N \times N$*.*

Alternative definitions exist to express graphs where, for instance, the function associating an edge to a pair of nodes is informally defined (e.g., Brandes and Erlebach (2005)). Definition 1, nonetheless, is very general and can be used to describe the arrangement of associations between nodes and edges of every possible graph. Figure 2.1 shows a few examples of graphs, presenting different number of nodes and edges, obtained using algorithms generating graph structures.



(a) Graph with a grid approximation structure inspired from Yao (1982) (100 nodes, 427 edges)

(b) Random graph following the $G(n, p)$ model of Gilbert (1959) (200 nodes, 982 edges, $p = 0.05$)

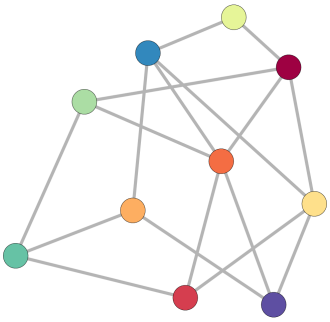
(c) Another random graph as described by Erdős and Rényi (1959) (490 nodes, 1000 edges)

Figure 2.1: Some examples of graphs with different numbers of nodes and edges.

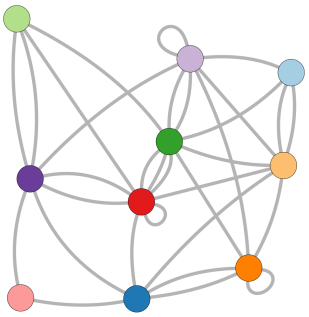
We call a given arrangement of associations between nodes and edges as shown above a **topology**. Among the different possible topologies a graph can present, a few cases with peculiar structural characteristics have specific names; we show a few examples of such cases in Figure 2.2. For instance, a graph is said to be **simple** (Fig. 2.2a) if it does not contain loops –$\forall e \in E$ and for $u \in N$, $\mathscr{E}(e) \neq \{u, u\}$– and only one edge can exist between any ordered pair of nodes –there is no such case, for $e_1, e_2 \in E$ and $u, v \in N$, where $\mathscr{E}(e_1) = \{u, v\}$ and $\mathscr{E}(e_2) = \{u, v\}$ when $e_1 \neq e_2$. The opposite of a simple graph is called a **multiple** graph (Fig. 2.2b), or multigraph, and allows loops as well as multiple edges between any ordered pair of nodes. Another specific type of graph topology of interest to us is the **tree** (Fig. 2.2c). A graph is a tree when there is no cycle inside it, that is there is only one possible path to follow in order to connect one node to another for any pair of nodes within the graph. The topologies can also induce some properties

[1] https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

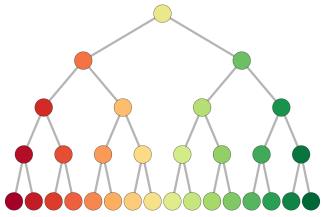
concerning the **degree** of each node, that is the number of edges associated to a node. If all the nodes in a graph have the same degree, a graph is said to be **regular** (Fig. 2.2d), moreover, if each possible pair of nodes is connected by an edge, then the graph is **complete** (Fig. 2.2e). Obviously, some of these definitions may sometimes overlap, for instance, a complete graph is also regular if it is simple, and a tree is always a simple graph but can never be regular nor complete.



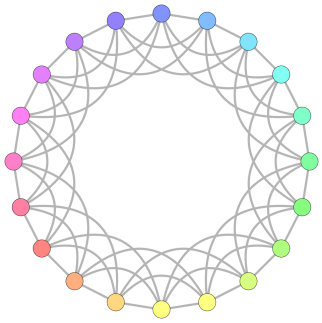
(a) Simple graph, a graph without any loop or multiple edge (10 nodes, 17 edges)



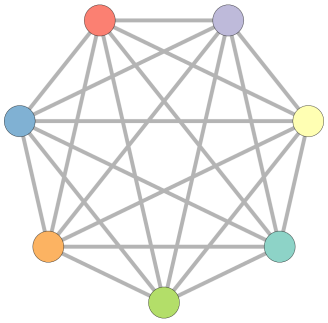
(b) Multigraph, a graph with loops and multiple edges (10 nodes, 32 edges)



(c) Tree, a simple graph without cycles (31 nodes, 30 edges)



(d) Regular graph, a simple or multiple graph where every node has the same degree (20 nodes, 60 edges).



(e) Complete graph, a simple regular graph where all nodes are connected together (7 nodes, 21 edges).

Figure 2.2: A few examples of different graphs with particular topologies.

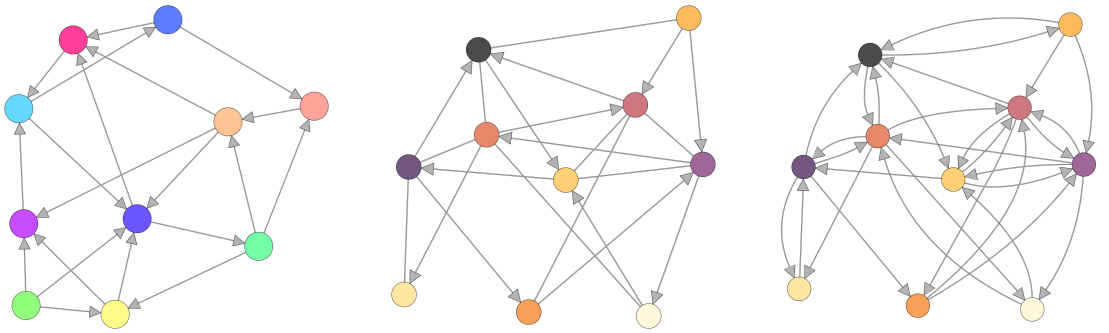
While all those topologies can be used to represent an infinity of different structures, graphs can also describe more down to earth situations. For instance, a graph can represent the equivalent topological structure of a road map, in the very same way it has been used by Euler to capture the available bridges to cross over the rivers in Königsberg. Such objects, similar to those existing in navigating systems, can then be used to identify the paths to follow to go from point A to point B. If multiple possibilities exist, one can be interested to either follow the **shortest path** between A and B, that is the one achieved with the least hops from neighbour to neighbour, or to identify the **longest path** between two nodes. The longest path existing in a graph is commonly called the graph's **diameter** and the average length of the shortest paths between all pairs of nodes is defined as the **characteristic path length** of a graph., But how, as we are interested in a graph representing a road map, could we express one-way traffic? The

graph definition needs to be extended to consider this situation; we consequently establish for our graph a new type of edge allowing for connections to be defined according to a given direction.

Definition 2 (Directed graph)**.** *A **directed graph** is a graph $G = (N, E, \mathscr{E})$ where*

- *N designates a set of nodes,*
- *E defines a set of **directed** edges (or arrows), and*
- *$\mathscr{E} : E \mapsto N \times N$ is a function which associates to each edge an **ordered** pair of nodes such that, for $e \in E$ and $u, v \in N$, if $\mathscr{E}(e) = (u, v)$, then $\mathscr{E}(e) \neq (v, u)$.*

Several examples of directed graph are proposed in Figure 2.3. The notion of the **source** of an edge (sometimes called origin) and its **destination** (or target) are common to describe respectively the node from which the edge starts and the one to which its arrives. While the arrows add a new meaning to the graph, the base topology can always be considered as undirected if need be. Some specific objects, called **mixed** graphs (Fig. 2.3b), even propose to only consider a part of the edges as oriented arrows and to keep the rest as undirected connections.



(a) Simple directed graph (10 nodes, 20 edges)

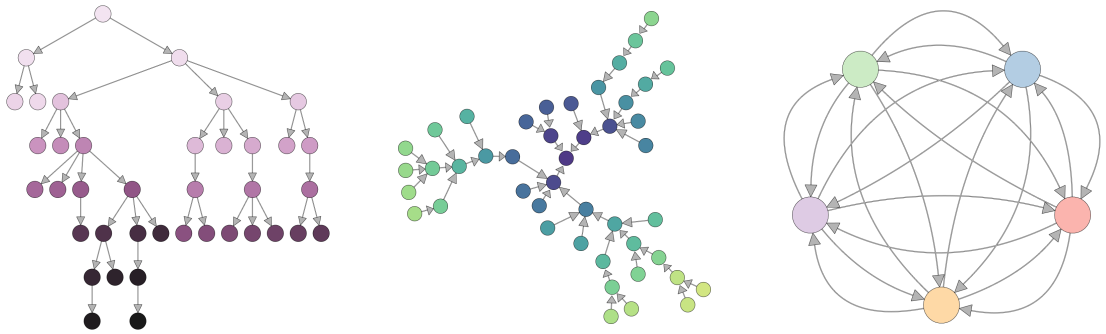
(b) Mixed graph with directed and undirected relations (10 nodes, 25 edges)

(c) ... which can also be represented using reciprocal arrows (10 nodes, 34 edges)

Figure 2.3: Examples of directed graphs using directed, mixed and reciprocal edges (arrows).

Quite naturally, all the specific topologies mentioned above can be adapted to fit this new definition. For instance, a simple graph forbids multiple edges and loops but not cycles, thus, for two nodes $u, v \in N$, we can now have two directed edges $e, f \in E$ between them but only when $\mathscr{E}(e) = (u, v)$ and $\mathscr{E}(f) = (v, u)$ (Fig. 2.3a and 2.3c). The definition of a tree with directed edges undergoes some changes as well. A **directed tree** can contain a peculiar node called a **root**. When all the edges point away from the root, the graph is called an **out-tree** (Fig. 2.4a); and, inversely, if all the edges point toward the root, then the graph is an **in-tree** (Fig. 2.4b). This change is very similar to the variation affecting the degree. Adding direction to the edges makes the notion of degree for a node insufficient, thus with the orientation taken into account, we define the **in-degree** and **out-degree** measures, respectively counting the number of incoming and out-going edges. Finally, the definitions for regular or complete directed graphs do not differ except that a simple complete directed graph will now need twice as much edges compared to its undirected variation (Fig. 2.4c).

Using directed graphs effectively allows us to add more information on the edges by specifying both source and destination. However, when working with graphs, sometimes the whole object is neither necessary nor simply relevant. If we reconsider the example of the road map



(a) Out-tree, a directed tree with edges pointing away from the root (39 nodes, 38 edges)

(b) In-tree, a directed tree with edges pointing toward the root (45 nodes, 44 edges)

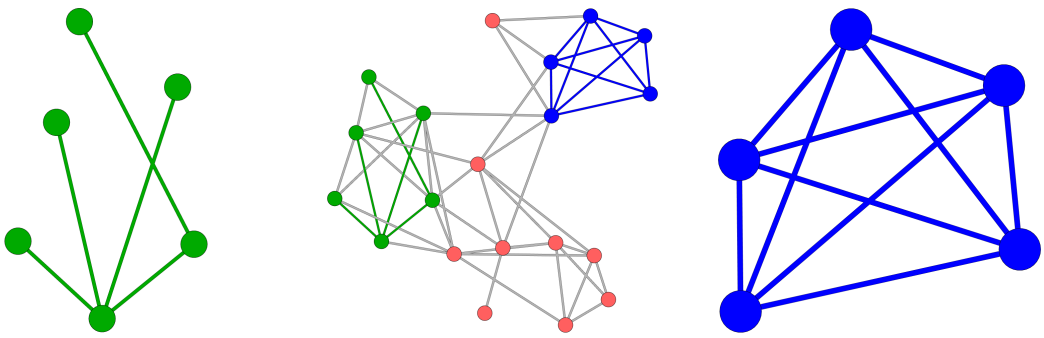
(c) Complete directed graph with twice as much edges than an undirected complete graph (5 nodes, 20 edges)

Figure 2.4: Additional examples of directed graphs with a structural peculiarity.

mentioned previously, an analogy would be to use a detailed map of the whole world when a partial map solely representing a much smaller section, for instance the Liechtenstein, would have been sufficient. In a similar fashion, partial graphs can be extracted from entire graphs if needed. These objects are defined as follows.

Definition 3 (Subgraph)**.** *A* ***subgraph*** *s extracted from a (directed) graph $G = (N, E, \mathscr{E})$ is a (directed) graph $s = (N_s, E_s, \mathscr{E}_s)$ where*

- *N_s is a subset of the nodes form G such that $N_s \subseteq N$,*
- *E_s represents a subset of the (directed) edges from G such that $E_s \subseteq E$, and*
- *$\mathscr{E}_s$ is a function interfacing $\mathscr{E}$ for the elements of subgraph s, which associates to each edge in E_s an (ordered) pair of nodes in N_s.*



(a) A subgraph consisting of the green elements extracted from b (6 nodes, 7 edges); the resulting graph is a tree.

(b) A simple undirected graph with two sets of highlighted elements in green and blue (20 nodes, 50 edges)

(c) A subgraph consisting of the blue elements extracted from b (6 nodes, 8 edges); the resulting graph is complete.

Figure 2.5: Examples of subgraphs extracted from a simple undirected graph.

Two examples of subgraphs are shown in Figure 2.5. As each subgraph is a graph in its own right, the different specifications defined earlier can be applied to it. All the characteristics of a graph however are not hereditary or automatically transferred to its subgraphs; for instance, a subgraph from a multigraph can be a simple graph, or a subgraph from a complete graph can be a tree. On the other hand, the orientation of the edges and their association with a pair of nodes remain constant throughout the graph, subgraphs, subsubgraphs, etc.

While subgraphs can be used to extract some subsets of a given graphs, they can also regroup the nodes in **clusters**. These assemblages are often defined based on the graph topology using clustering algorithms designed to gather nodes into *natural groups* (Gaertler, 2005). This method is especially pertinent when facing graphs representing social networks as groups of nodes are commonly encountered in the form of communities which designate users strongly connected with each other (i.e., with multiple common neighbours). The tightness of such group of nodes can be computed by counting the average number of common neighbours each node have with its neighbours. Furthermore, the value can then be averaged on the whole graph for all nodes to obtain the measure known as the **clustering coefficient** (see Watts and Strogatz (1998)).

2.1.2 Labelling and port graphs

The abstract definition of a graph given so far allows us to conveniently use the object in a few different contexts to represent particular situations. For instance, in addition to the road map example, graphs can easily be used to depict geographical information, like representing the map of shared borders for all the countries or the hierarchy of bifurcations and branching of the different rivers from the oceans to their sources (Strahler, 1957). While these graphs can preserve the structure of the information we store in them, that is the topology in itself, the information is not really kept as it should. Indeed, what good is a representation of all the borders for every country if the names of the countries can not be retrieve instantly? Other data could also be preserved in the graph like the total length of the border,[2] a historic tracing if and when the frontier has been disputed in the past, or even if the nodes (countries) in the resulting graph can be coloured such that two connected nodes never share the same colour (Wilson, 2002).

These additional information and computation can not be achieved with the graph definitions as given above. Indeed, some elements of the graph may contain different kinds of data (for instance, a name, a date, a colour), but the definition does not allow for such information to be preserved. This feature is however supported by a specific class of graphs called **labelled graphs**. As pointed out by its name, a labelled graph is a graph and thus respects Definition 1 (or Definition 2 if the graph is directed) which can preserve information through special functions attached to it, assigning labels to its nodes and edges.

Definition 4 (Labelling function)**.** *Let λ be a set of labels on nodes or edges, $\mathscr{L}$ is the* ***labelling function*** *for nodes and edges such as $\mathscr{L} : N \cup E \mapsto \lambda$.*

Labels are commonly used to store information such as weights for the edges or tags for the nodes, but we can generalise their content type to anything (e.g., a numeric value, a sequence of characters, a colour, a tuple). In our case, we affect to the graph several labelling functions in order to allow the storage of multiple different data at the same time for each element.

Definition 5 (Labelled (directed) graph)**.** *A* ***labelled (directed) graph*** *is designed as $G = (N, E, \mathscr{E}, \Lambda)$ where*

- *N, E and $\mathscr{E}$ are the sets and function as described in Definition 1 (respectively, Definition 2 for a directed labelled graph),*

[2] http://www.nationmaster.com/country-info/stats/Geography/Land-boundaries/Border-countries

- Λ *is a set of labelling functions* $\mathscr{L}$ *on* N *and* E*, such as* $\Lambda = \{\mathscr{L}_1, \mathscr{L}_2, \ldots, \mathscr{L}_k\}$ *where* k *is the total number of labelling functions.*

Any labelling function $\mathscr{L}_i$, where $0 \leq i \leq k$, can then be used to store an *attribute* for each element of the graph. From here on, these labels allow us to distinguish the different kind of entities or relations existing within the graph. For instance, nodes could be labelled using colours (with either their full names –e.g., *red*, *black*– or a RGB tuple) to discern them, while

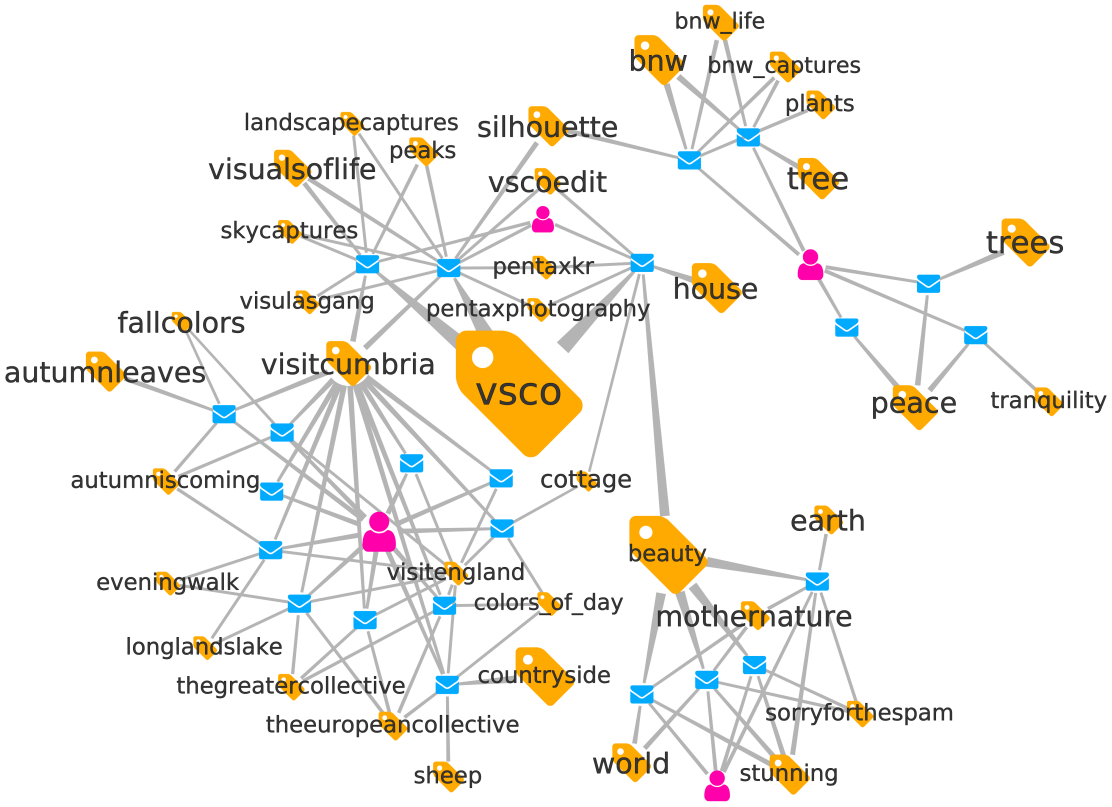


Figure 2.6: Example of labelled subgraph representing tagged content on Instagram (65 nodes, 122 edges). The dataset, originally described in Munk et al. (2016), is built from geo-tagged Instagram posts harvested over several weeks. Each post contains a media (picture or video), the identity of its author and a piece of text with a comment and some tags added by the user (e.g., `#london`, `#picoftheday`, `#sogood`). We build from this dataset a visualisation representing users, posts and tags. Each element contains several different type of information such as the name, photo, and biography of the user, where and when a post was submitted, the frequency with which a tag was used, etc. Each of these values is stored in a proper labelling function, e.g., for a labelled graph G, a node $n \in N$, and with the labelling functions $tag, type, colour \in \Lambda$, if $tag(n) =$ `autumnleaves` then $type(n) =$ `tag` and $colour(n) =$ `orange`. Because we can not display all the available information at once, we use some of the visual properties of the representation to express a few of them: the colour and shape map the nodes' type (*magenta* for users, *blue* for posts and *orange* for tags), the glyphs' size indicates with which frequency each element is referred to (its degree) and the label of the orange nodes is used to display the name of the tag.

the edges could be labelled with values (indicating the geographic distance between two nodes for instance) or unique numbers to enable their identification. Labelled graphs, also sometimes called attribute graphs, thus allow us to potentially store an infinite amount of information on the nodes and edges, and the graph can now be used to model a lot of different problems using the elements' attributes stored as labels. A popular example of application, further developed later on, is the representation of social networks in which the users are represented by the nodes, their relations by the edges, and all the persons and relations characteristics –such as names, age, exchanged messages, qualifier of the relation, etc.– can be stored as labels.

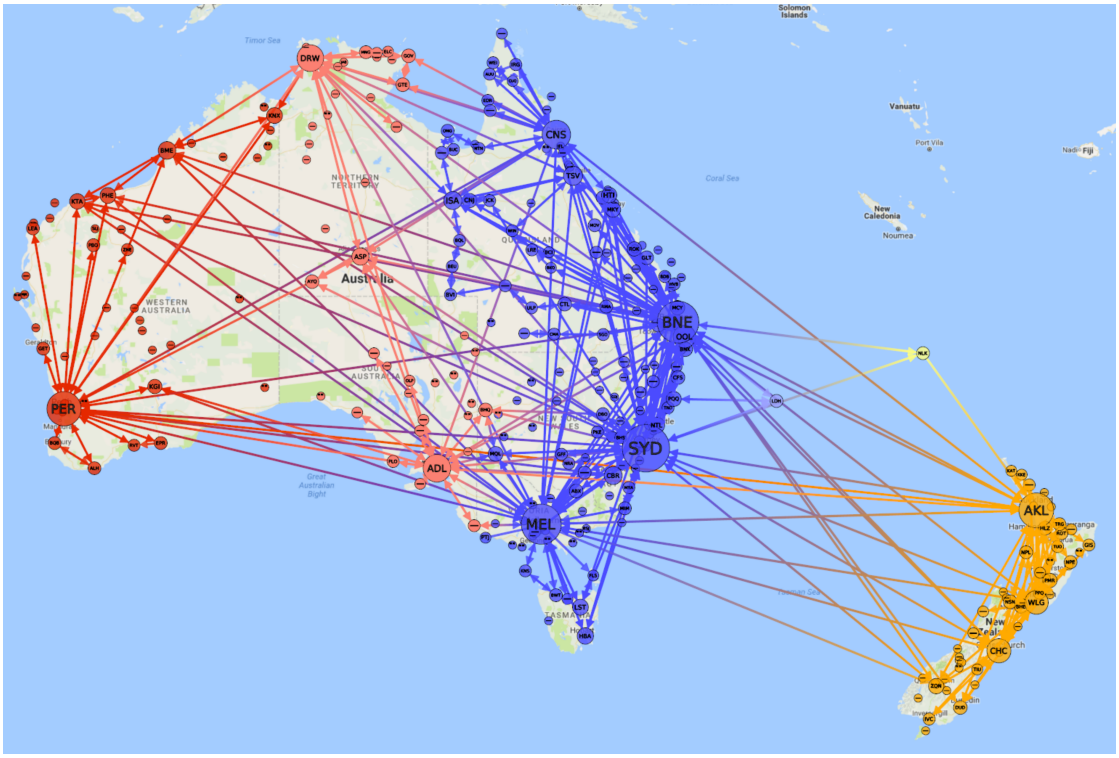


Figure 2.7: Example of labelled subgraph representing flying routes between the airports of Australia and New-Zealand (304 nodes, 1030 edges). The data originates from OpenFlights (https://openflights.org/data.html) and proposes a list of airports and flying routes. We know a fair share of information concerning the airports such as their geographic position (latitude, longitude, altitude), the city/country where it is located, the airport's name, the IATA/-FAA/ICAO acronyms (international identification for the airport, e.g., SYD for Sydney or MEL for Melbourne), the timezone, etc. as well as, concerning the flying routes, the name of the operating airline, the number of stops, the equipment, etc. All of these values must be stored in the corresponding labelling functions, e.g., for a labelled graph G, a node $n \in N$, an edge $e \in E$, and with the labelling functions $latitude, iata, equipment, destination \in \Lambda$, we can have $latitude(n) =$ `-33.9461`, $iata(n) =$ `SYD`, $equipment(e) =$ `717` and $destination(e) =$ `Sydney`. The visualisation represents some information using the visual properties of the graph: the latitude, longitude and altitude indicate where the airport (represented by a node) should be placed, the nodes' size gives the number of connections passing through the airport (its degree), the colour hints of the timezone, and the edges are directed to indicate the flying routes' direction.

Figures 2.6 and 2.7 present different examples of labelled graphs. This new type of graph can be adapted to all the different variations encountered earlier, whether it is directed and simple, regular and complete, or the subgraph of another larger graph. Trivially, any graph containing information in addition to its topological structure can be described as a labelled graph.

Nonetheless, although labelled graphs can be used in a lot of different situations, there is one concept in particular which can not be simply depicted using them. Let us, for instance, consider the modelling of biochemical networks, like those presenting protein-protein interactions as in Blinov et al. (2006) and Deeds et al. (2012), using a labelled graph as introduced above. We define two nodes to represent the molecular entities, set to be chemically bound together using an edge. Molecules can react together and create strong bonds when presenting ionised atoms (lacking or possessing more electrons than protons). When the two complex components undergo a chemical reaction compelling them to become connected, multiple binding sites, indicating points where ionised atoms are present and the two molecules can become attached, may be available at the same time. Instead of being connected directly like nodes with edges, the two molecules are attached through these binding points. A labelled graph structure, while quite flexible, is unable to simply represent those binding sites and manage whether they are free, already occupied, or in a transitional state. Note that using labels to store in each node the available binding points and in each edge the one occupied by the current connection is still possible but require a lot of different manipulations to finally obtain the needed information.

To address this issue in our modelisation tool Porgy, we have adopted a slight variation of the labelled graphs as we propose to use **port graphs** in their stead. Simply put, a port graph is a graph where edges are solely attached to nodes using specific connection points called *ports*. This model is not without similarities with computer networks, where a machine can be connected to the rest of the network through an Ethernet port, in turn connected to a hub or a switch, linking to other computers, servers, or even different networks. Each port is thus attached to a single node and every connection between two nodes must pass through at least one of their respective ports. We define them as follows:

Definition 6 (Labelled (directed) port graph)**.** *A **labelled (directed) port graph** G is defined as $G = (N, P, E, \mathscr{E}, \mathscr{P}, \mathscr{N}, \Lambda)$ where*

- *N represents a set of nodes;*
- *P designates a set of ports;*
- *E corresponds to the set of (directed) edges;*
- *$\mathscr{E}$ is a function which associates to each edge an (ordered) pair of ports such that $\mathscr{E} : E \mapsto \{P, P\}$ (respectively, $\mathscr{E} : E \mapsto P \times P$ when directed);*
- *$\mathscr{P} : P \mapsto N$ is a function which associates to each port the node to which it is attached to;*
- *$\mathscr{N} : N \mapsto \mathbb{P}$ is a function which associates to each node the set of ports $\mathbb{P}$ attached to it such that, for $n \in N$ and $\forall p \in \mathscr{N}(n)$, $\mathscr{P}(p) = n$;*
- *Λ is a set of labelling functions $\mathscr{L}$, such as $\Lambda = \{\mathscr{L}_1, \mathscr{L}_2, \ldots, \mathscr{L}_k\}$, with $\mathscr{L}_i$, for $0 \leq i \leq k$, being a labelling function for nodes, ports and edges such as $\mathscr{L}_i : N \cup P \cup E \mapsto \lambda_i$, where λ_i is a set of labels on nodes, ports or edges.*

The term **portnode** is commonly used to designate the object consisting of the central node (in N) and its attached ports (from P). We present in Figure 2.8 an example of port graph visualised using Porgy. As each element is labelled, nodes, ports and edges can store different

types of information. This generalisation can be used to represent any common labelled graph. Portnodes can then be adapted in different configurations, from providing a single port, through which each neighbour will be connected, to offering as many ports as wanted. An example of configuration with as much ports on a node as it has neighbours is presented in Deeds et al. (2012); we have also proposed another variation in Vallet et al. (2015a) where each portnode possesses two ports to distinguish incoming from outgoing edges. However, if a straightforward translation from a (directed) labelled graph to a (directed) labelled port graph is necessary, then proposing a single port for each port node is the simplest solution as it avoids any additional complexity (be it structural or visual) when defining the concerned port graph. We have notably

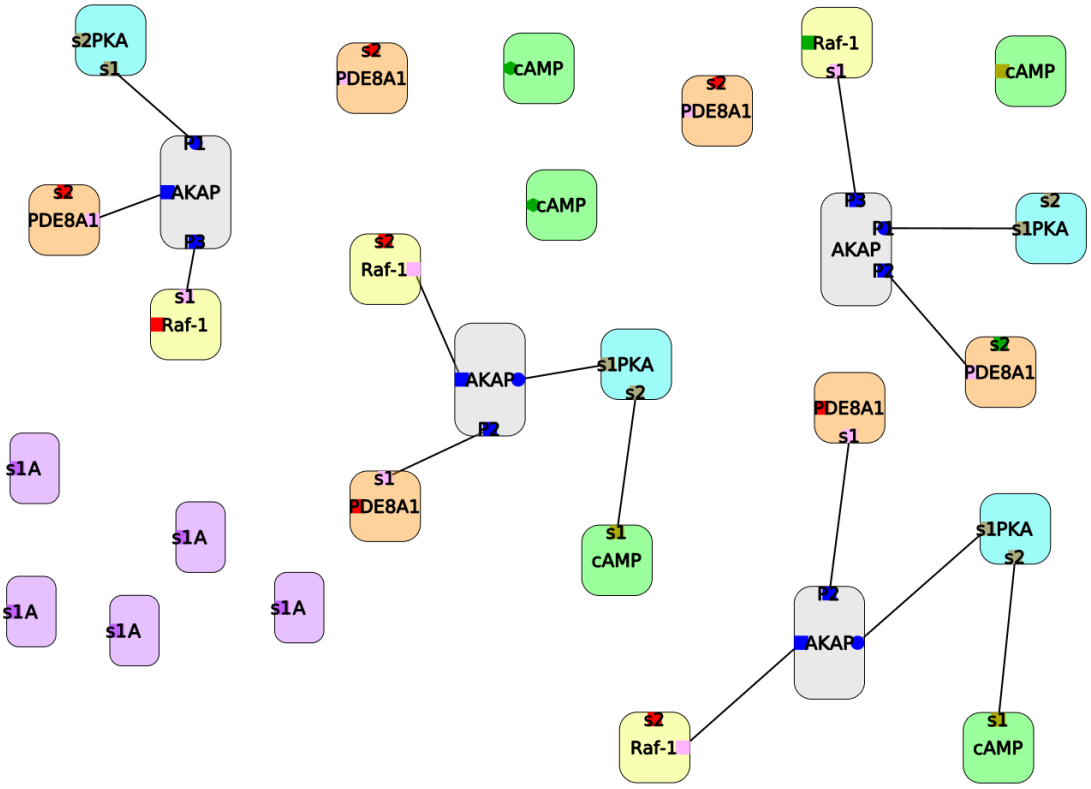
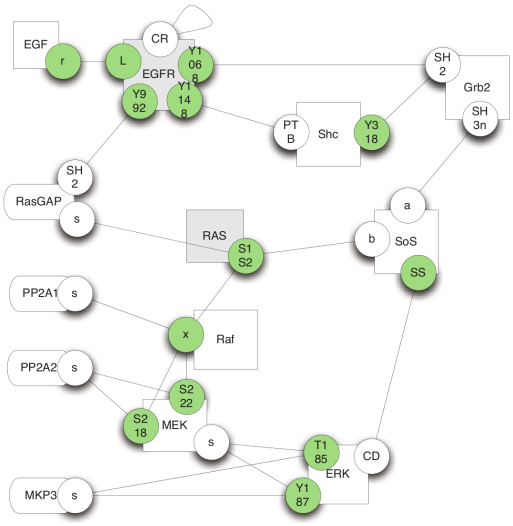


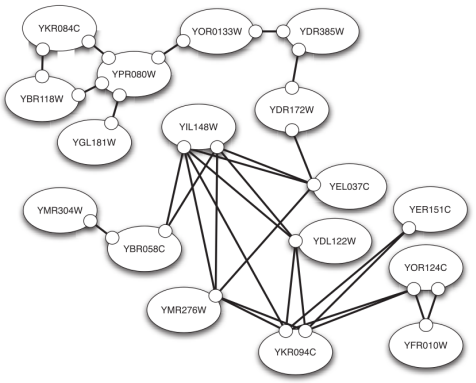
Figure 2.8: Example of labelled port graph from Andrei et al. (2011a) representing protein-protein interactions (28 port nodes, 51 ports, 14 edges). Each of the binding sites is characterised by a specific status which is stored in the element using a labelling function. To ease the acknowledgement of this status value on any of the ports, we use the visual properties of the elements and, more precisely, the shape and the colour of the ports. For instance, some of the cAMP, Raf-1, and PDEBA1 elements have a green port while other proteins of the same type present red or brown ports, marking them as unavailable. To help distinguishing them, each type of protein is also coloured distinctively. For a labelled port graph G, a node $n \in N$ and with the labelling functions $name, colour \in \Lambda$, we have for instance: $colour(n) = \texttt{green}$ if $name(n) = \texttt{cAMP}$, $colour(n) = \texttt{yellow}$ if $name(n) = \texttt{Raf-1}$ or $colour(n) = \texttt{orange}$ if $name(n) = \texttt{PDE8A1}$.

used this method in Fernández et al. (2016b) to represent nodes in a network as they are created by a generative model. We also propose another example in Figure 2.10, this time representing a map of flight connections. Despite not being explicitly defined as a port graph, the directed flight connections (i.e., the edges) connect to the nodes using specific points labelled by the arrival and departure time.

Overall, the decision of using port graphs in PORGY was made at the very beginning of the graph rewriting tool elaboration (Pinaud et al., 2011, 2012). Because its creation was motivated by the modelling of biological processes (Andrei et al., 2011b), and it is still used to this end these days, as in Andrei et al. (2016), the port graph model, as explained in the example detailed above, was essential to define the binding sites of the molecules. This need is very easily verified as several authors have opted for similar modelling solutions in the last ten years (e.g., Blinov et al. (2006); Danos et al. (2007); Andrei (2008); Yang et al. (2008); Faeder et al. (2009); Harmer et al. (2010); Chylek et al. (2011); Deeds et al. (2012); Chylek et al. (2015)). We present in Figure 2.9 two such examples of port graphs, modelling protein to protein interactions. Due to this initial need and the fact that normal graphs can be transformed seamlessly into port graphs, and conversely if the ports do not carry any specific meaning, all of our subsequent works thus have "adapted" to the PORGY platform, employing the port graph and portnode structures as a basis for describing our models. While the adoption of the port graphs can seem like an unneeded hindrance when compared to the more classical graph constructs, using one rather than the other does not change the global process followed, and although port graphs can seem alien at first sight, the use of ports to distinguish the connection points quickly ends up feeling simply natural and logical.



(a) Contact map of a protein to protein network, from Danos et al. (2007).



(b) Subgraph of proteins (nodes), binding sites (ports) and interactions (edges) originating from Deeds et al. (2012).

Figure 2.9: Examples of labelled port graphs obtained from the literature.

The definitions given thus far describe the data structure at the centre of our work. Although they may appear unusual at a first glance, (directed) port graphs can be used to represent any

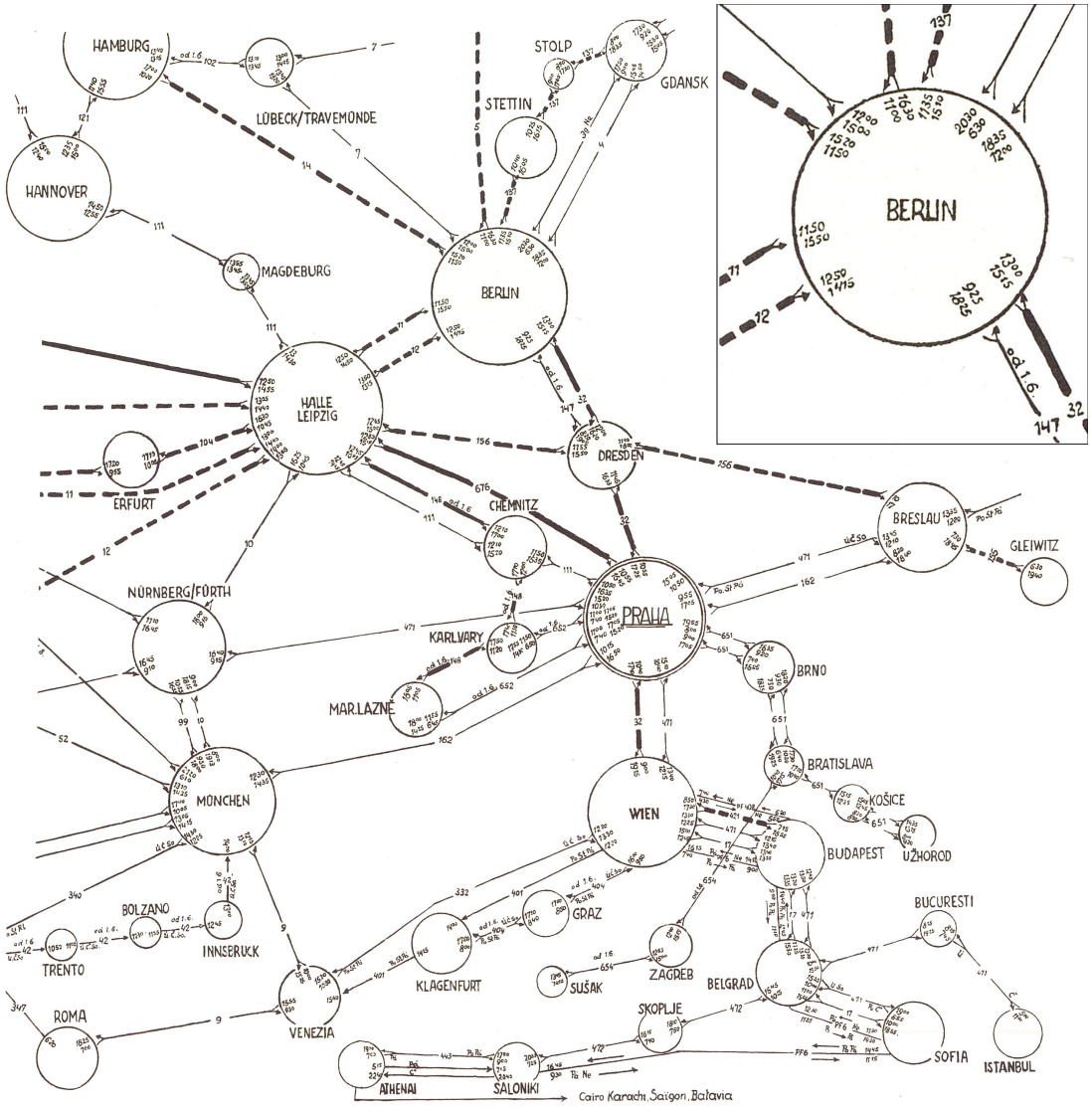


Figure 2.10: Example of a directed labelled port graph representing a partial flying map/schedule from 1933 between different European cities (e.g., Berlin, Prague, Vienna). This partial representation comes from Tufte (1990) and its caption is as follows: "*A comprehensive narrative description of a transport system requires a record of both time and spatial experiences. Here a complex network of routes is brought together with flight times and identification numbers in a brilliant map/schedule for the Czechoslovakia Air Transport Company in 1933. [...]*". Whereas the figure in Tufte's book is also accompanied with the brochure cover, we have instead added an enlarged detail of Berlin in the top-right corner. As one can see, nodes represent cities, edges indicate flying connections and, although no graphical distinction is used to represent ports, the attachment area for any edge to a node gives the specific time scheduled for arrival and departure with respect to the edge direction for this connection only. In addition to the written information (i.e., city names on nodes, scheduled connections on ports, flight identification numbers on edges), some visual cues are used on nodes (their size) and the edges (their line style: thin, dashed or bold line) to represent even more information.

conventional graph and, after a short time of adaptation, their handling becomes as much a habit as one can perceive when manipulating normal graphs. We next focus our interest towards (port) graph transformations and the definition of rewriting operations.

2.2 Graph rewriting

Several different formalisms exist to describe the evolution of an entity into something else. Chemical reactions, for instance, as mentioned earlier, are good examples of such occurrence. An initial element or groups of elements are specified as input materials and the resulting combination, with any existing byproduct, is given as the reaction outcome. With the transformation happening, the atoms in the molecules can be considered as simply being reordered to form new assemblages as expressed by the chemical reaction formula. Another example, using set of letters as atomic elements this time, can present a very similar mechanism. Starting from an ordered set of letters, like a *sentence* for instance, a formula could propose to change a subset of elements, e.g., a *word*, into another. We thus propose to search for a *pattern* and replace it with another one in a precise manner much like what is achieved when using regular expressions (Lawson, 2003) to find specific sequences of characters in texts, but also to replace them.

While the appellation *rewriting*, used to describe our formalism, makes sense in the context of finding sequences of characters and rewriting the sections of a text corresponding to specified patterns, the designation initially originates from the fields of term algebra and term rewriting (Baader and Nipkow, 1998; Courcelle, 1990). According to Baresi and Heckel (2002), the first proposal introducing similar transformations applied to graphs appeared in the late sixties/early seventies with web grammars (Pfaltz and Rosenfeld, 1969; Montanari, 1970), grammars for partial orders (Schneider, 1970) and λ-graph reduction (Wadsworth, 1971). Baresi *et al.* effectively present an introduction to the graph transformations from a software engineering point-of-view, however, the three handbooks compiled by Rozenberg (1997) and Ehrig et al. (1997a,c) truly present the subject in great detail; we can only refer the interested reader to them for more information about the foundations, applications and problematics related to graph transformations. Although we propose hereafter our view of graph rewriting and introduce the principle accordingly, several diverse definitions of graph rewriting exist in the literature, each using different kinds of graph structures or rewrite rules and formalisms. Fernández et al. (2016a) give an extensive list of publications which propose their own perspective on the problem, such as Barendregt et al. (1987); Lafont (1990); Barthelmann (1996); Corradini et al. (1997); Plump (1998); Habel et al. (2001); Andrei and Kirchner (2008, 2009). Furthermore, while the following definitions draw inspiration from previous works ranging from the earliest introduction of Porgy (Andrei et al., 2011a; Fernández et al., 2012) to the most recent publications surrounding the rewriting platform (Fernández et al., 2014; Vallet et al., 2015a; Fernández et al., 2016b,a), each have been adapted to describe rewriting operations on directed labelled port graphs.

Despite our entire focus being set onto the graph rewriting platform Porgy, several other graph transformation tools exist. The ones we most commonly mentioned in the past are PROGRES (PROgrammed Graph REwriting System) (Schürr et al., 1997), AGG (Algebraic Graph Grammar system) (Ermel et al., 1997), Fujaba (From Uml to JAva and BAck) (Nickel et al., 2000), GROOVE (GRaph for Object Oriented VErification) (Rensink, 2004), GrGen (Geiß et al., 2006) and GP/GP2 (Graph Programming) (Plump, 2009, 2011). However, a few others, more focussed toward modelling software systems, are also noticeable. We can mention solutions such as ViaTra (Visual Automated model TRAnsformation[3]) (Varró and Balogh, 2007), Henshin (based

[3] https://eclipse.org/viatra/

on the Eclipse Modelling Framework[4]) (Arendt et al., 2010), GREaT (Graph Rewriting and Transformation Language) (Balasubramanian et al., 2006), AToM3 (A Tool for Multi-formalism and Meta-Modelling) (de Lara and Vangheluwe, 2002), VMTS (Visual Modeling and Transformation System) (Mezei et al., 2007), Augur 2 (König and Kozioura, 2008), MOFLON[5] (Anjorin et al., 2014)... While a sound comparison of all theses solutions would be very interesting, it would also be out of scope in the current context. Being the solution we used to express the models introduced in this thesis, we thus restrict ourselves to solely discussing and defining the graph rewriting approach from our own point of view using the platform PORGY.

We focus in the following on the three processes needed to perform transformations as graph rewriting operations. We first need a technique to express the correspondence from a given graph to another one. Secondly, we express the definition for a port graph rewrite rule, allowing us to precise a pattern to identify and the conform transformation to perform. By bounding these two operations together, we are able to propose a matching procedure to select the appropriate elements to rewrite. The graph rewriting operation obtained once all processes have been successful, which we define as a rewriting step, is concluded by finally transforming the targeted elements according to the specifications given in the rewrite rule.

2.2.1 Port graph morphism

Our very first move is to define a way to express an existing correspondence between two distinct (port) graphs. Thus, we are interested in the notion of homomorphism (from the Greek *homos* meaning "same" and *morphe* signifying "form") between the two entities. As the port graphs consist of several elements of different nature –i.e., nodes, ports, edges–, we will need to express different functions to detail the possible mapping of one element from a given port graph to another. We thus propose the following definition.

Definition 7 (Port graph morphism)**.** *Given two port graphs* $G = (N_G, P_G, E_G, \mathscr{E}_G, \mathscr{P}_G, \mathscr{N}_G, \Lambda_G)$ *and* $H = (N_H, P_H, E_H, \mathscr{E}_H, \mathscr{P}_H, \mathscr{N}_H, \Lambda_H)$*, a (partial)* ***port graph morphism*** $\mathscr{M}$ *transforming* G *to* H *is defined by a set of functions* $\mathscr{M} = \langle \mathscr{M}_N, \mathscr{M}_P, \mathscr{M}_E \rangle$ *specifying the mapping of the sets of elements (respectively, nodes, ports and edges) of* G *to* H *such that:*

- $\mathscr{M}_P : P_G \mapsto P_H$ *is a mapping from the set of ports of* G *to the set of ports of* H*.*
- $\mathscr{M}_E : E_G \mapsto E_H$ *is a mapping from the set of edges of* G *to the set of edges of* H *such that, for two ports* $p_G, q_G \in G$*, two other ports* $p_H, q_H \in H$*, and two edges* $e_G \in E_G$ *and* $e_H \in E_H$ *(with* $\mathscr{E}_G(e_G) = (p_G, q_G)$ *and* $\mathscr{E}_H(e_H) = (p_H, q_H)$*), there is a morphism from* e_G *to* e_H *(*$\mathscr{M}_E(e_G) = e_H$*) when the edge extremities also present the respective morphisms* $\mathscr{M}_P(p_G) = p_H$ *and* $\mathscr{M}_P(q_G) = q_H$*;*
- $\mathscr{M}_N : N_G \mapsto N_H$ *is a mapping from the set of nodes of* G *to the set of nodes of* H *such that, for two nodes* $n_G \in N_G$ *and* $n_H \in N_H$*, we define their every attachment ports as* p_G *and* p_H*, respectively* $\forall p_G \in \mathscr{N}_G(n_G) \subseteq P_G$ *and* $\forall p_H \in \mathscr{N}_H(n_H) \subseteq P_H$*, if there is a morphism from* n_G *to* n_H *(*$\mathscr{M}_N(n_G) = n_H$*), then each of the ports* p_G *attached to* n_G *also possesses a distinct morphism* p_H *attached to* n_H *(*$\mathscr{M}_P(p_G) = p_H$*).*

As one can notice, the ports are truly at the centre of the morphism operation and we use those anchoring points as references at all time. By defining them as a basis during the mapping, we can compare both the nodes –to which they are attached, and by extension the whole portnode–, the edges –which connect two ports together–, and see if differences exist or

[4] https://www.eclipse.org/modeling/emf/
[5] http://www.moflon.org

find the appropriate correspondence. The ports are thus a great help to establish a basis for the port graphs topology and verify whether their respective structure are actually identical and each element has a correspondence. Furthermore, while N, P and E are used to express the morphism, the different functions defining the port graph (i.e., $\mathscr{E}$, $\mathscr{P}$, $\mathscr{N}$ and Λ) are not used to define the morphism. The ports are considered at all time as the basis and thus the results returned by the port graph functions are redefined accordingly. For instance, should two ports $p, q \in P_G$ be part of the same portnode in G but end up being attached to different nodes in H, the functions $\mathscr{P}$ and $\mathscr{N}$ will automatically adapt to produce the appropriate results such that even if $\mathscr{P}_G(p) = \mathscr{P}_G(q)$, we obtain $\mathscr{P}_H(p) \neq \mathscr{P}_H(q)$, and of course $p \notin \mathscr{N}_H(\mathscr{P}_H(q))$ and conversely $q \notin \mathscr{N}_H(\mathscr{P}_H(p))$.

This definition indicates the exact correspondence between two port graphs, as each element from G finds a match in H. This scenario is obviously not very common as most of the time we will only look for a smaller pattern in a larger port graph. However, as we are looking in cases like these for a partial morphism, we can reuse Definition 3 describing the principle of subgraphs. We start identifying potential candidates based on the pattern and then extract the eligible elements into a sub(port)graph. In the same fashion that a subgraph is not just a partial selection of the elements of a larger graph but a true graph in itself, a sub-port graph also respects the definition of a real port graph. This allows us to define the partial port graph morphism as an exact port graph morphism, only applied on a subset of the original port graph.

Identifying the correspondence between two port graphs is a stepping stone for the graph rewriting process. As we are able to map an element in a port graph to a different one in another port graph, though the type of the element (i.e., node, port, edge) must remain consistent, we can use this technique to declare patterns to search for. Hereafter, we present how such patterns can be described and how we can submit the selected elements to transformation to achieve our rewriting objective.

2.2.2 Port graph rewrite rule

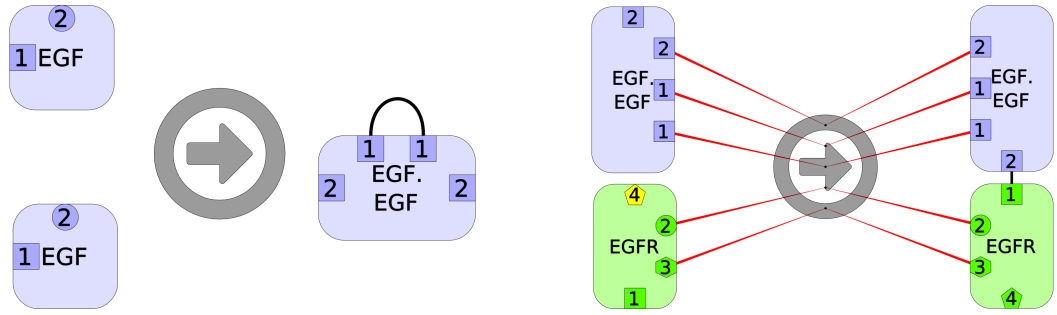
A *port graph rewrite rule*, most commonly noted $L \Rightarrow R$, indicates a transformation we refer to as a *rewriting operation* or a *rewriting step*. Several definitions exist however, in our case, the rewrite rule is itself a port graph, consisting of two sub-port graphs L and R and an *arrow* node, noted $\Rightarrow$, encoding the correspondence between some of the ports in L and R. We give the definition as follows.

Definition 8 (Labelled (directed) port graph rewrite rule)**.** *A* ***labelled (directed) port graph rewrite rule*** $L \Rightarrow_{\mathscr{W}} R$ *is a labelled (directed) port graph consisting of:*

- *a first labelled (directed) port graph L, called* left-hand side *(LHS), specifying the model of subgraph targeted for rewriting,*
- *a second labelled (directed) port graph R, called* right-hand side *(RHS), defining the subgraph set to replace L, where all the labelling functions in Λ_R in R must be expressed using algebraic combinations of the labelling functions in $\Lambda_{\mathscr{M}(L)}$ found in $\mathscr{M}(L)$, and*
- *an arrow node $\Rightarrow_{\mathscr{W}}$ with a set of edges, each connecting a port in L to a port in R through a port in the arrow node.*

The edges provided with the arrow node indicate the mapping of the ports from L to R and the edge rewiring to perform on the graph when the rule is applied. We affix to the arrow node the function $\mathscr{W} : \Lambda_L \wedge W \mapsto \mathbb{B}$ taking for parameter Λ_L –the set of labelling functions for the nodes, ports and edges of L–, W –a Boolean expression used to express the absence of elements–, and returning a Boolean value.

We will present in details the whole process followed when a graph rewriting operation is performed later, for now, we focus on the rewrite rule definition. As the left- and right-hand side are port graphs, the number of portnodes allowed to exist in either is not limited, we could thus describe any port graph in the LHS and rewrite it into any other port graph in the RHS, allowing thus an infinite number of transformations to obtain all the possible port graphs. Because graph rewriting is such a powerful and versatile tool, we need to take extra care to avoid unwanted transformation and thus we will always seek to limit the rules into performing just the modifications we need, exactly where we want them. An exhaustive definition of the LHS is the very first step. As an exact description of the pattern we seek is properly given, we can restrain the modifications to the eligible elements and rewrite them according to the rule.



(a) Two signalling EGF proteins form a dimer (EGF.EGF) represented as a single node.

(b) An EGF dimer (EGF.EGF) and a receptor (EGFR) bind on free sites.

Figure 2.11: Examples of port graph rewrite rules from Andrei (2008) obtained with PORGY. The transformations are a peculiar case of reaction pattern in the Epidermal Growth Factor Receptor (EGFR) signalling pathway fragment. The red edges linking LHS elements to RHS ones specify how to rewire elements once the rule application is complete to avoid unintentionally ending up with dangling edges.

We propose in Figure 2.11 a visual example of two (undirected) port graph rewrite rules. Although we have just given the definition for directed rewrite rules, undirected rewrite rules are very similar and allow us to introduce the concepts we will need through simpler examples; the eager reader can however peek into the following chapters to find examples of directed rewrite rule applications. The rules presented in Figure 2.11 originate from Andrei (2008) and represent reaction patterns in the Epidermal Growth Factor Receptor (EGFR[6]) signalling pathway. This visual rule representation shows a lot of information at the first glance. Indeed, in addition to simply depicting the topology of the LHS and RHS (arrangement and number of ports, nodes and edges), the rule also provides visual clues which may be used as characteristics in the rewriting operation. Overall, each value for the visual properties, such as the size, shape, colour or name, defining the ports, nodes and edges is stored in one of the numerous label functions defined in Λ. The first rule (Fig. 2.11a) symbolises two EGF proteins, in the LHS, bonding together to form an EGF.EGF element, in the RHS, with a central node shaped as an arrow to depict the arrow node $\Rightarrow_{\mathscr{W}}$. Several changes, occurring during this transformation, are shown at the same time here and we can use diverse visual cues to indicate existing differences. The two portnodes, labelled EGF, present two ports called 1 and 2. Beyond their different names, the shapes used to represent these ports are also distinct. The rewrite rule application will thus replace two of

[6] https://en.wikipedia.org/wiki/Epidermal_growth_factor_receptor

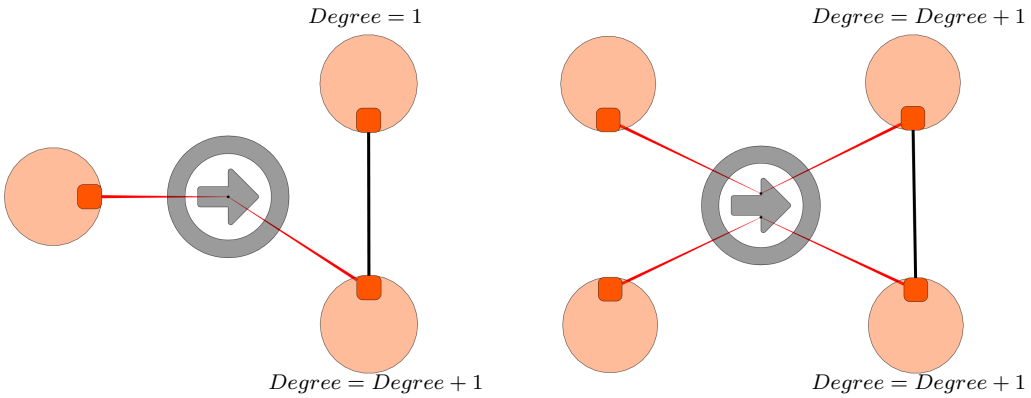
the EGF elements, each with two ports 1 and 2 (1 being a square and 2 being a circle), with a new element EGF.EGF with four squared ports (two ports named 1, and two ports named 2). A new edge is also added between the two ports labelled 1 attached to EGF.EGF in the RHS. This first example is quite straightforward and can be simply resumed as two elements merging together to create a new one. However, some information related to the transformation seem to be missing here: imagining that the two EGF elements are connected to other portnodes before the rewrite rule application takes place. What would happen to the potential edges connecting the rewritten elements to the rest of the graph?

Although this situation can not happen in this previous example, a second rule, shown in Figure 2.11b, presents a way to indicate the continuity of connections for elements being rewritten. We use special edges, depicted as red edges on the visual representation, linking ports from the LHS to ports in the RHS through the arrow node $\Rightarrow_{\mathscr{W}}$ and indicating the port to target when performing the edge rewiring once the rewriting operation is completed. In this second rule, an edge connecting with port 2 of the EGFR element for instance will be simply rewired to connect with port 2 on the rewritten EGFR element. Conversely, if an edge were to exist on the lone port 1 on EGFR, the rewriting operation would discard such connection as the rule does not expressly precise this rewiring operation must be performed. This method is quite efficient to avoid any dangling edge and has been used as a visual aid for a number of years by a few different researchers (e.g., Mu et al. (2007)). However, the rewrite rules must be defined with the utmost care to avoid any unintentional loss of connections after the rule application. Additionally, different configurations based on the arrow node connections between L and R can be used to express specific behaviours. The default operation occurs when there is no connection between an element of the LHS to the RHS. This case is described as a *black hole* as the edges previously connected to the rewritten element(s) have no specific place to re-attach and are consequently removed from the port graph to satisfy Definition 6 forbidding dangling edges. The two remaining cases are the *fusion* and *division*. Respectively, if two (or more) red edges connected to distinct ports in the LHS reach to the RHS and are connected to the same port there, then all the edges are rewired to that very port. Inversely, if several red edges originate from the same port in the LHS but are attached to distinct ports in the RHS, then the rewiring operation will duplicate these edges and reattach each of them to the targeted ports. Although those behaviour add generality, we will not use them in the work presented here. For the interested reader, more details concerning the edges connecting elements of the LHS and RHS through the arrow node are available in Fernández et al. (2016a).

As you can see in the rule, the shape of the ports but also the colours of both nodes and ports can be changed. Because we use labelled port graphs, we consider that all the attributes and information of the graph are stored using labelling functions, and thus, this is also true for the visual attributes. By following Definition 8, the different labels existing for each elements of the RHS can be defined from usual values, or they can be based on values of elements' labels existing in the LHS. For instance, using the rewrite rule $\mathscr{R}$ shown in Figure 2.11b, we define two labelling functions $name_{\mathscr{R}}, colour_{\mathscr{R}} \in \Lambda_{\mathscr{R}}$. For the ports $p_L \in P_L$ in the LHS and $p_R \in P_R$ in the RHS, we may have $name_{\mathscr{R}}(p_R) = name_{\mathscr{R}}(p_L) = \texttt{4}$ but with $colour_{\mathscr{R}}(p_L) = \texttt{yellow}$ which is different from $colour_{\mathscr{R}}(p_R) = \texttt{green}$. We will use this type of functionality very frequently in the rest of this thesis, however, defining the operation to be performed this way is quite long and inconvenient. In the same manner that we use PORGY, a visual rewriting platform to model our problems, we will use the visual representations of the rules to express the values stored on the different elements. This information could still be quite cluttered on rules like those in Figure 2.11 were we subjected to use rules with a lot of different ports, each with significant amount of data being stored and updated with each rewriting operation. This is however not the case for us in the context of this thesis. Due to our interest being focussed on social network

related modelling, we are able to greatly simplify our rules and only need portnodes with a single port used as the sole connection point to the other elements. As a result, the ports and nodes are only distinct elements due to PORGY's inner structure definition but otherwise will behave as one, as if they were a single element. We thus propose to explicitly mark on each rule visual representation which labelling value is modified and how it is updated. We show such an example of rule in Figure 2.12. The two rules can be used to generate a new random multigraph with a single component, respectively, Rule 2.12a creates a new node and connects it to an existing one, and Rule 2.12b connects two nodes together. For each rewrite rule $\mathscr{R}$, we define a labelling function $Degree \in \Lambda_{\mathscr{R}}$ to store for each portnode its current degree. Obviously, when we connect an edge to a node, we expect the value stored in its $Degree$ to be updated; this is taken care of during the rewrite rule application. Either the node is new (i.e., Rule 2.12a) and its degree value is initialised at 1 ($Degree = 1$), or the node was pre-existing and its degree value is incremented (with $Degree = Degree + 1$). There may be several ways to perform an operation, but in any case, the modifications always occur in the RHS and use values originating or derived from computations on labels existing in the LHS. Although visually pleasant and appropriate for simple operations, this method can however introduce some confusion in some case, especially when designing values previously existing on several elements at the same time. For instance, in Rule 2.12b, the operation $Degree = Degree + 1$ applied in the RHS on either the top or bottom portnodes does not precise which element from the LHS to use to obtain the degree value, indeed, it could signify either the bottom node or the top node. Thus, albeit this is not an existing restriction in the PORGY rewriting system, we consider for the visual representations that the label values defined for an element in the RHS are extracted from its mapped candidate from the LHS (as indicated by the red edge) unless explicitly stated otherwise.

A final point concerning the function $\mathscr{W}$ affixed to the arrow node $\Rightarrow_{\mathscr{W}}$ is needed to complete the explanations surrounding our port graph rewrite rule definition. However, despite being expressed in the port rewrite rule definition, $\mathscr{W}$ is also inherently connected to the next process needed to perform graph transformation. We thus delay our explanations for the moment but will clarify its role right after this next definition.



(a) Rule adding a new node and edge

(b) Rule connecting together two nodes

Figure 2.12: Example of rewrite rules to use for random multiple graph generation. The rules are virtually similar to the one given in Figure 3.1 on page 44. As we create new elements, we modify the label values of the pre-existing ones to keep them up to date. Here, we use this technique to process the degree value and preserve it in a labelling function $Degree \in \Lambda_{\mathscr{R}}$ in each rewrite rule $\mathscr{R}$.

2.2.3 Matching

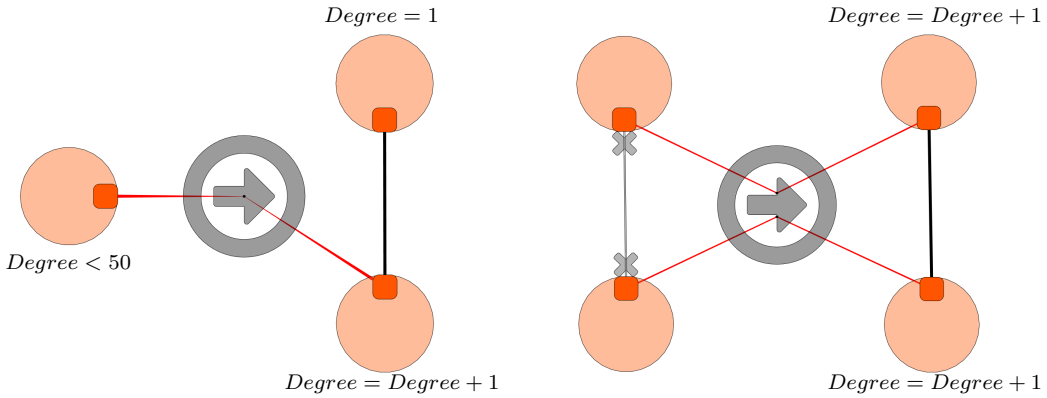
With the port graph rewrite rule globally defined, we are now able to express which changes to perform during the rewriting operation. Once combined with the morphism operation allowing the (partial) mapping between two port graphs, these two processes establish the basis for the matching operation which we define as follows.

Definition 9 (Match)**.** *Let $L \Rightarrow_{\mathscr{W}} R$ be a port graph rewrite rule and G a port graph, we say that a* ***match*** *$\mathscr{T}_G(L)$ of the left-hand side L is found in G if:*

- *there is a port graph morphism $\mathscr{M}$ from L to G (hence $\mathscr{M}(L)$ is a subgraph of G), and*
- *the value returned by the function $\mathscr{W}$ of the arrow node is true.*

The identification of an appropriate morphism $\mathscr{M}$ is obviously necessary to pin-point candidates to consider for the rewriting operation to take place. Several techniques allowing such matching exists with the most well known algorithm being attributed to Ullmann (1976), although it was hardly the first (Corneil and Gotlieb, 1970). This method has since been updated (Ullmann, 2011), some improvements have been proposed (Čibej and Mihelič, 2015) and a few techniques have been found to identify matches more easily (Čibej and Mihelič, 2014). The search of subgraph isomorphism is a whole research area with a full spectrum of solutions ranging from parallel algorithms (McCreesh and Prosser, 2015) and genetic algorithms (Kim et al., 2016) to some specific solutions adapted for graph databases (Lee et al., 2012). Although the efficient search for subgraph isomorphisms is a very interesting problem, albeit quite complicated as it belongs to the NP complexity class (Cook, 1971), we will not look into it in this thesis. We can note however that our current context of operation consists in identifying exact matches, opposed to approximate matching (Lu et al., 2016), by using a non-parallel and non-distributed single computer architecture. While this certainly simplifies the algorithm implementation, it also limits the speed of our solution when performing numerous and complex graph rewriting operations. We discuss in greater details the performances of PORGY in Appendix B on page 193.

Once some potentially matching elements have been identified and the morphism is set, effectively allowing us to know the correspondence between each element of the LHS, we need to be certain that they all have been appropriately selected. In addition to the topology and because we use labelled port graph, we are able to propose a conditional matching based on the possible label values with the function $\mathscr{W}$. As indicated in Definition 8, $\mathscr{W}$ considers all the values returned by the labelling functions of the LHS port graph, as well as a Boolean expression W to declare the necessity for certain elements to be absent, and return a Boolean value signalling if the elements selected to map those in the LHS are a fitting choice. We can thus decide to allow the application of a rule to a matching subgraph if only certain labels exhibit the desired values. For instance, for a rewrite rule $\mathscr{R}$ with a labelling function $Name_{\mathscr{R}} \in \Lambda_{\mathscr{R}}$ providing names for nodes, ports or edges, we could configure $\mathscr{W}$ to only accept as candidates nodes whose name is defined as A and edges whose names are not defined as B. This very process can simply be generalised to any of the labelling functions in $\Lambda_{\mathscr{R}}$ and adapted to offer the usual equality or inequality tests, with the ability to test as many label values as needed for all of the elements existing in the LHS. We propose, just like we did earlier for the redefinition of the RHS label values, to add annotations on the rewrite rules representations to visually express the conditions to meet for the elements to successfully match with the ones in the LHS. We present in Figure 2.13a an example based on the previous Rule 2.12a. We still track the degree value, as previously explained, but we also have added a new condition on the sole portnode of the LHS. Rule 2.13a basically indicates that, to successfully match with the portnode in the LHS, the candidate portnode must have a degree value inferior to fifty. As mentioned earlier, this principle can be generalised to nodes, ports, and edges, with several conditional tests also being allowed on a single element value.



(a) Rule adding a new node and edge if the initial node's degree is inferior to 50.

(b) Rule connecting together two nodes if the two nodes are not already connected.

Figure 2.13: Example of conditional rewrite rules to use for simple random graph generation. Each rule is subjected to a condition and can only be successfully applied when it is met. Because Rule b checks whether the two nodes are already connected before inserting a new edge between them, the resulting graph will be simple, with each node having at most 50 neighbours (Rule a).

With the first part of $\mathscr{W}$ specified for the conditional matching, we are only missing the Boolean expression W. According to Definition 8, W is used to express the absence of elements. This concept has been introduced to enable the modelisation of network algorithms and, more specifically, to facilitate the elaboration of tests checking the existence of edges between ports. This functionality is inspired from the *GP* programming system (Plump, 2009), and from a more general definition given in *Elan* (Borovanský et al., 1998), in which a rule may have a condition introduced by the keyword `where`. Applied to our case, a condition `where not Edge(p,q)` indicates that no edge exists between the ports p and q. This condition, being a part of the conditional matching, is also checked during the matching. Just like before, we propose a visual cue to indicate on the rewrite rules visual representation that such condition exist. We show in Figure 2.13b an example of edge nonexistence. The rule is similar to the one presented in Figure 2.12b with the addition of a grey edge between the two ports of the LHS. To help identify those "anti-edges" more easily, we also mark their extremities using glyphs shaped like crosses. In undirected rules like the one showed in the figure, we usually put such glyphs on each of the edge ends, however, in undirected rules, we only put a cross at the destination's end of the arrow to preserve the visual information signalling the edge direction. Despite its special meaning, an "anti-edge" is also considered as an element like the ports, nodes, edges, and as such, it is subjected to conditional matching too. For instance, let us define a port graph G, with two ports $p, q \in P$ respectively attached to the nodes $m, n \in N$ ($\mathscr{P}(p) = m$ and $\mathscr{P}(q) = n$), and an undirected edge $e \in E$, connecting p and q ($\mathscr{E}(e) = \{p, q\}$), and labelled by a function $weight \in \Lambda$ such that $weight(e) = X$. If we declare a rewrite rule searching for two portnodes, each with a single port, and connected by an "anti-edge" whose labelling function $weight$ must return a value such that $weight(e) \neq X$, then a matching is completely possible as there is no edge between m and n (through p and q) whose value is different from X.

Now that the function $\mathscr{W}$ is properly defined, we can clearly see that, for a match to be successful, three requirements must be satisfied. First of all, the rewrite rule LHS must have an appropriate topological morphism on which to map its elements. Secondly, the considered elements must also comply with the conditional matching with respect to their label values.

Lastly, if an "anti-edge" is defined in the LHS, then it must find no match with an existing edge. Should any of these three requirements failed to be enforced, then the graph rewriting operation can not be applied. We propose now to look at the complete rewriting step and detail how each of the processes defined above fall into place.

2.2.4 Rewriting step and derivation

Most of the time, a rewriting step is defined in a very intuitive way: we find a matching subgraph for the left-hand side pattern of the rule and replace it by the right-hand side pattern. Such formulation hides the intrinsic complexity of the matching procedure as well as the transformation itself. When it comes down to implementing such a rewriting system, even so when one needs to visualise the rules and the transformations, things need to be meticulously specified. We propose the following definition for it.

Definition 10 (Rewriting step)**.** *A **rewriting step** $\mathscr{S}$ on G using a rule $L \Rightarrow_{\mathscr{W}} R$ and a match $\mathscr{T}_G(L)$, written $\mathscr{S} : G \mapsto^{\mathscr{T}}_{L \Rightarrow_{\mathscr{W}} R} G'$, transforms G into a new graph G' in three steps.*

- *The* build *step: if a match $\mathscr{T}_G(L)$ for L is found in G, an instantiated copy of the port graph R named R_c is added to G (thus $\mathscr{M}_G(R) = R_c$), and the values of the labelling functions in Λ_{R_c} are computed according to the values found in $\Lambda_{\mathscr{T}_G(L)}$.*

- *The* rewiring *step: all edges connecting elements of the match $\mathscr{T}_G(L)$ for L to the rest of G are redirected to R_c. Let two ports $p_L \in P_L$ and $p_R \in P_R$ respectively in L and R, for each port p_A in the arrow node $\Rightarrow_{\mathscr{W}}$ connecting p_L to p_R ($\mathscr{E}_{L \Rightarrow_{\mathscr{W}} R}(e_{LA}) = (p_L, p_A)$ and $\mathscr{E}_{L \Rightarrow_{\mathscr{W}} R}(e_{AR}) = (p_A, p_R)$ with $e_{LA}, e_{AR} \in E_{L \Rightarrow_{\mathscr{W}} R}$), we must find all the edges e linking the ports $p_G \in P_G \setminus P_{\mathscr{T}_G(L)}$ –existing in G but not consider as matching elements for L– with ports in $\mathscr{T}_G(L)$ –the matching subgraph of L in G. Let $\mathscr{M}^L$ and $\mathscr{M}^R$ be the general port graph morphisms mapping respectively the transformations from L to G and from R to G, each edge e is then rewired according to its direction: if $\mathscr{E}_G(e) = (p_G, \mathscr{M}^L_P(p_L))$ then e is rewired such that $\mathscr{E}_G(e) = (p_G, \mathscr{M}^R_P(p_R))$ or, if $\mathscr{E}_G(e) = (\mathscr{M}^L_P(p_L), p_G)$ then e is rewired as $\mathscr{E}_G(e) = (\mathscr{M}^R_P(p_R), p_G)$. The port obtained by the morphism $\mathscr{M}^R_P(p_R)$ is found by the match $\mathscr{T}_G(R)$ for R in G, that is R_c.*

- *The* deletion *step: the subgraph $\mathscr{T}_G(L)$ is deleted from the transformed graph G, thus creating the final graph G', where the elements matching with L have been replaced by those matching with R and the edges have been rewired accordingly.*

Thus, these three steps describe a complete rewriting operation. Once the transformation to perform is expressed in a rewrite rule and an appropriate match has been identified, the rule can be applied and the corresponding changes made in the graph. The graph G is consequently transformed into G' and is susceptible to undergo further transformations as long as the rewrite rules LHS can find purchase in the graph.

Generally, for a given rule $L \Rightarrow_{\mathscr{W}} R$ and a port graph G, several rewriting steps may be possible depending of the matching elements. Consequently, the application of a rule can potentially produce various graphs when different elements are rewritten in G. As a result, if a single rewriting application can for instance produce γ port graphs, a succession of rewriting operations could produce an exponentially increasing number of possible graphs. We name such sequence of rewriting steps a **derivation**. Although some rules or succession of rules may increase the number of elements which could potentially be rewritten (e.g., Rules 2.12a and 2.12b), others can also limit this number after a certain number of applications (e.g., Rules 2.13a) or with each application (e.g., Rules 2.11a and 2.13b) like converging or diverging functions. In the latter case,

the available derivations are finite and the system is said to be **terminating**. Very naturally, a derivation can also be represented as a tree, with each new potential alternative result to a rewriting step being added to a new branch: these objects are simply called **derivation trees**. Each branch thus follows a rewriting scenario with each node in the tree proposing alternative routes whenever several rule applications can occur in different places in the graph. We show an example of such representation in Figure 2.14. The derivation has been obtained after successive applications of Rule 2.12a (creating a new port node and connecting it to one of the pre-existing port nodes) on a graph with a single initial port node. As one can see, each node in the tree shows a small preview of the port graph, and because each rule application creates a new port node (and thus a new candidate for the next rule application), every additional level in the tree at depth δ creates $\delta!$ new leaves and forbid the derivation to ever terminate. Obviously, rules which terminates, like the one proposed in Figure 2.13b, cannot produce such derivation trees as the resulting graph obtained after several rewriting steps will converge toward a single or several distinct solutions. Overall, the derivation tree is a wonderful tool to follow the different rewriting steps and their respective results.

The definition we give here for a rewriting step is our own but several different variations exist in the literature. Basically, the techniques proposed to tackle the graph rewriting problem either follow the Double Push-Out from Ehrig et al. (1973) or Single Push-Out (Raoult, 1984; Kennaway, 1987) approach (respectively shortened as DPO and SPO). We will avoid detailing those algebraic approaches here as their very formal definitions would significantly expand this section, however, we forward the interested reader to Ehrig et al. (1997b) for a specific comparison between the two approaches. We can note nonetheless that the graph transformation process we propose above is more closely related to the SPO approach as described in Löwe et al. (1993) and Löwe (1993); more in-depth explanations are available in Fernández et al. (2016a).

The definition proposed to achieve the rewriting step gives us all the essential ingredients we need to perform graph rewriting operations: a graph and a collection of rules with the appropriate techniques to match them on the graph. This set of components and tools is what we commonly refer to as a **graph rewriting system** (GRS). Although quite powerful, the graph rewriting formalism as expressed up to this point still lack some flexibility when we wish to perform minute operations and rewriting transformations on very precise sections of the graph. Indeed, the matching will indifferently choose candidates anywhere in the graph as long as their topology and their labels agree with the conditions expressed in the rule. Furthermore, despite

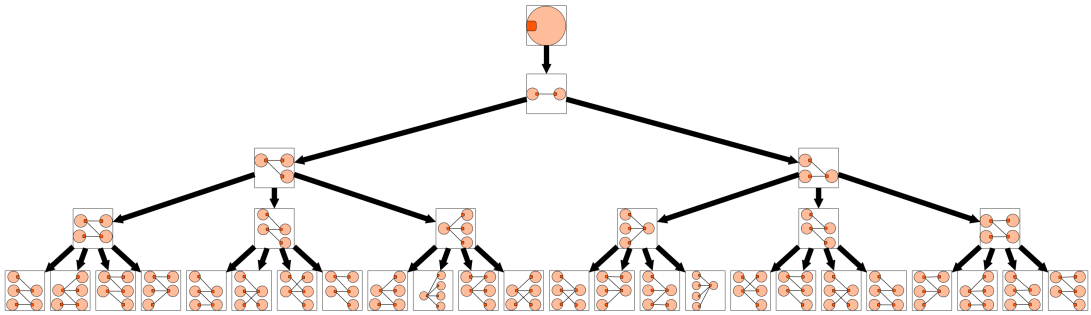


Figure 2.14: Example of derivation tree of depth 4 obtained after successive applications of Rule 2.12a on a graph with a single initial portnode.

the potential of graph rewriting operations, we have not seen so far how the graph rewrite rules could be organised or ordered to perform more convoluted transformations.

2.3 Strategic located graph rewriting

In this section, we present an extension to the basic port graph rewriting system introduced beforehand. Despite the versatility of the rewrite rules, as they can be used to express any possible modification on any possible graph, we are slightly limited by the port graph rewriting system in its current form. Imagine, for instance, a simple rule $\mathscr{R}$ setting the colour of a node to red. In a labelled port graph G where the elements colour is defined by the labelling function $Colour \in \Lambda_G$, our rule will just consider a node in G, match on it, and change its *Colour* value. The first observation we can formulate is that any node can become red: every time the rule is applied, a node (whether it is already red or not) is picked in G at random and is subjected to the transformation. While the rule is in its simplest form, repeating the operation on nodes already coloured red is a waste. We thus need to find a way to express which elements are available for rewriting, or even to directly force only some elements into being targeted by the transformation. To this end, we introduce hereinafter the concept of **located graph rewriting** as a mean to decide which elements are available for rewriting and which are excluded.

2.3.1 Located rewriting

The rewrite rules as expressed previously can be adjusted in numerous way to only transform the needed characteristics with a surgical precision. This effectiveness is however misspend by the roughness of the matching operation, indifferently selecting the candidates in the whole graph. To facilitate the specification of graph transformations, we propose to use the concept of **located graph** as defined in the following.

Definition 11 (Located (port) graph)**.** *A* ***located (port) graph*** $G^{\mathcal{Q}}_{\mathcal{P}}$ *consists of a (port) graph G and two distinguished subgraphs $\mathcal{P}$ and $\mathcal{Q}$ of G, called respectively the* position subgraph *(or simply* position*) and the* banned subgraph *(or simply* ban*).*

As their respective names can lead to believe, the position subgraph $\mathcal{P}$ is the subgraph of G under study, that is the elements most likely to be used during the transformation, whereas the ban subgraph $\mathcal{Q}$ can be seen as a protected subgraph, where no transformation concerning the elements inside can take place. Those two different sets can have different use only limited by the imagination of the expert designing the graphs and rewrite models. Most commonly, the position subgraph contains the nodes awaiting to be rewritten, whereas the ban subgraph is formed of "forbidden" elements or elements which have already been rewritten.

Obviously, this modification also touches the rewrite rule definition. Indeed, to inject some specific meaning into the two subgraphs $\mathcal{P}$ and $\mathcal{Q}$, their role needs to be integrated to all the aspects of the graph rewriting system. Let us first improved the definition of the rewrite rule.

Definition 12 (Located rewrite rule)**.** *A* ***located rewrite rule****, noted $L_{\mathcal{W}} \Rightarrow_{\mathscr{W}} R^{\mathcal{N}}_{\mathcal{M}}$, is given by a port graph rewrite rule $L \Rightarrow_{\mathscr{W}} R$, with an optional subgraph $\mathcal{W}$ of L, specifying which elements will be mandatorily chosen from $\mathcal{P}$, and two disjoint subgraphs $\mathcal{M}$ and $\mathcal{N}$ of R, defining respectively which elements will be reintroduced in $\mathcal{P}$ and $\mathcal{Q}$.*

Before applying the rule rewrite operation, the matching can elect several potential candidates and thus several possible morphisms. For the LHS subgraph morphism in G noted L_G, and because G is a located (port) graph being rewritten using a located rewrite rule, we propose

default behaviours for the elements. First, a LHS noted L will successfully match on the subgraph L_G if at least one of the elements in L_G exists in $\mathcal{P}$. This option can be pushed even further by using the subgraph $\mathcal{W}$ in the rule LHS to declare exactly which elements of L_G should be matched to elements existing in $\mathcal{P}$. Secondly, the existence of $\mathcal{Q}$ counteract this effect by making impossible any matching with nodes in $\mathcal{Q}$. By carefully managing the addition and subtraction of elements in $\mathcal{P}$ and $\mathcal{Q}$, the initial matching can thus be strongly oriented, to not say completely orchestrated, toward our rewriting goal.

Once the rule application is completed, the rewritten elements can be directly added to $\mathcal{P}$ or $\mathcal{Q}$ to steer the next rewriting operations. Elements from the RHS are thus either part of $\mathcal{M}$, $\mathcal{N}$ or neither subgraphs and are respectively added to $\mathcal{P}$, $\mathcal{Q}$ or neither, accordingly. If we reconsider the example of the rewriting rule colouring nodes in red, by inserting the freshly rewritten node in $\mathcal{N}$ in the RHS, that is in $\mathcal{Q}$ in the transformed graph, then the following rewrite rule application will no longer consider the red node, now in ban, as an acceptable candidate. By using $\mathcal{W}$, $\mathcal{M}$ and $\mathcal{N}$, we can properly specify which elements should absolutely be matched from $\mathcal{P}$ with $\mathcal{W}$ and which elements can never be matching candidates with $\mathcal{Q}$, which transformed elements are inserted into $\mathcal{P}$ with $\mathcal{M}$, and finally which elements are inserted into $\mathcal{Q}$ with $\mathcal{N}$.

We propose some default behaviours when applying a located rewrite rule. As mentioned above, the subgraph $\mathcal{W}$ indicates elements belonging to $\mathcal{P}$ but the definition does not forbid us to leave $\mathcal{W}$ empty in a rule. Consequently, when performing a matching, every node, port and edge outside of $\mathcal{Q}$ should be a potential candidate. To avoid cases where the amount of matching candidates would become excessively numerous, we have decided instead to impose the presence of at least one node from $\mathcal{P}$ when matching elements with the LHS. This small limitation keeps the transformations local and tend to prevent rewriting operation from going out of control, and, should the need for such operation arise, it does not hinder the possibility to declare all the elements of the graph as elements of $\mathcal{P}$.

With the located port graph and the located rewrite rule defined we can describe the operation occurring during the rule application. Quite naturally, this definition comes on top of Definition 10, and although we do not re-stipulate the three steps taking place during a rewriting step, the development is similar.

Definition 13 (Located rewriting step)**.** *A **located rewriting step** $\mathscr{S}_{\mathcal{L}}$ on G, using a located rewrite rule $L_{\mathcal{W}} \Rightarrow_{\mathscr{W}} R^{\mathcal{N}}_{\mathcal{M}}$ and a match $\mathscr{T}_G(L)$, is written $\mathscr{S}_{\mathcal{L}} : G^{\mathcal{Q}}_{\mathcal{P}} \to^{\mathscr{T}}_{L_{\mathcal{W}} \Rightarrow_{\mathscr{W}} R^{\mathcal{N}}_{\mathcal{M}}} G'^{\mathcal{Q}'}_{\mathcal{P}'}$.*

This means that the located graph $G^{\mathcal{Q}}_{\mathcal{P}}$ rewrites to $G'^{\mathcal{Q}'}_{\mathcal{P}'}$ using $L_{\mathcal{W}} \Rightarrow_{\mathscr{W}} R^{\mathcal{N}}_{\mathcal{M}}$ at position $\mathcal{P}$ avoiding $\mathcal{Q}$ according to the rewriting step $G \to_{L \Rightarrow_{\mathscr{W}} R} G'$ and with a morphism $\mathscr{M}_G$, defined in the match $\mathscr{T}_G(L)$, such that $\mathscr{M}_G(L) \cap \mathcal{Q} = \emptyset$, and $\mathscr{M}_G(L) \cap \mathcal{P} = \mathscr{M}_G(\mathcal{W})$, or simply $\mathscr{M}_G(L) \cap \mathcal{P} \neq \emptyset$ if $\mathcal{W}$ is not provided. The new position subgraph $\mathcal{P}'$ and banned subgraph $\mathcal{Q}'$ are defined as $\mathcal{P}' = (\mathcal{P} \setminus \mathscr{M}_G(L)) \cup \mathscr{M}_{G'}(\mathcal{M})$ and $\mathcal{Q}' = \mathcal{Q} \cup \mathscr{M}_{G'}(\mathcal{N})$; if $\mathcal{M}$ (resp. $\mathcal{N}$) is not specified then we assume that $\mathcal{M} = R$ (resp. $\mathcal{N} = \emptyset$).

In a nutshell, the management of the subgraphs $\mathcal{P}$ and $\mathcal{Q}$ allows us to target or ignore only specific elements, $\mathcal{W}$ offers additional precision, and $\mathcal{M}$ and $\mathcal{N}$ give us the possibility to adapt $\mathcal{P}$ and $\mathcal{Q}$ with each rule application; additional details concerning the use of the position and ban subgraphs are available in Fernández et al. (2016a). Obviously, all the different tools available for standard graphs, port graphs, rewrite rules and rewriting steps can also be adapted to their located variation. In particular, the derivation tree turned out to be a very useful structure to easily visualise the resulting transformations. However, some limitations still exist in our system. The precise handling of a single located rewrite rule proved to be possible using the position and ban subgraphs but how does such system hold when two or more rewrite rules are to be considered when performing transformations on a graph?

2.3.2 Strategic graph programs

Let us reconsider the example given at the beginning of this section with the rule $\mathscr{R}$ setting the colour of a node to red. Now, imagine a second rule $\mathscr{R}'$ setting the colour of a node to green instead. How should we handle the application of both $\mathscr{R}$ and $\mathscr{R}'$ on the same port graph? There is simple solutions available, like proposing to use one rule or the other according to a given probability or using a list to express a sequence giving the order in which each rule must be applied. Those propositions however lack the potential for dynamic and conditional rule application: for instance, we may wish to apply $\mathscr{R}$ as many times as possible and only then apply $\mathscr{R}'$ three times. To achieve such operation, we would need to transform our graph rewriting system into a programmable system handling loops and conditional applications. We thus propose to use **strategic graph programs**.

A strategic graph program is a program expressed using a **strategy**, itself consisting of a set of instructions allowing the management of the located subgraphs and the conditional application of rewrite rules. Consequently, a strategic graph program consists of a located graph rewriting system and a strategy expression that permit the applications of located rules and the handling of $\mathcal{P}$ and $\mathcal{Q}$. Each strategy is composed of a set of instructions following a complete syntax and formal grammar. Overall, the strategic graph programs offer a full programming language allowing us to handle a set of operating instructions to manage the rewrite rule applications. We present in Table 2.1 the complete grammar of the strategic graph rewriting language. A few different solutions also use solutions affiliated to strategic rewriting, a detailed survey is available in Kirchner (2015). As any of the rewriting applications we will present in this document will use this syntax, we propose to take a closer look to it in order to help the reader familiarise her/himself with the terms and available instructions.

Simply put, a strategy contains operations, and each operation in a strategy can either succeed or fail; these two outcomes can be directly indicated by the instructions `id` and `fail`. An operation is basically expected to either perform a rule application or change the located subgraphs. As long as the operations succeed, the strategy will continue its execution, however, once an operation has failed, the strategy usually comes to an end. To allow more complex treatments, an operation can also be considered as a test: a failed or successful operation can be caught in one of the loops or conditional constructs, leading the strategy to behave differently in accordance. As one can see in Table 2.1, the syntax is divided in four different construct groups: the **Compositions** –handling the loops, conditional applications and the general located subgraphs tests–, the **Positions** –which manage what we call the *focussing* operations, that is the instructions concerning either the position or ban subgraph–, the **Rules** –defining explicitly the rule applications–, and finally the **Properties** –handling the tests on the elements' properties or labels. Let us take a closer look at each of these groups.

According to our syntax (in the **Compositions** group), a **Strategy** is the entry point of our grammar and it can be recursively defined to enable several sequential operations. They can either return `id` or `fail` as atomic values, or they can initiate a rule application (A), a located subgraph update (U), a located subgraph comparison (C), a successive strategy application ($S;S$), or trigger diverse conditional or looping operations which we describe hereafter:

- `if`(S_1)`then`(S_2)`else`(S_3), a standard conditional operation; if the application of S_1 on (a copy of) $G^{\mathcal{Q}}_{\mathcal{P}}$ returns `id` (i.e., succeeds), S_2 is applied to (the original) $G^{\mathcal{Q}}_{\mathcal{P}}$; otherwise S_3 is applied to $G^{\mathcal{Q}}_{\mathcal{P}}$. As in most of the existing language, the second half on the instruction, specifying the `else`, is optional. The instruction fails if S_1 succeeds and S_2 fails or if S_1 fails and S_3 fails.

- (S_1)`orelse`(S_2), a more unconventional conditional operation; this instruction applies S_1 on $G^{\mathcal{Q}}_{\mathcal{P}}$ if possible, and otherwise applies S_2; it fails if both S_1 and S_2 fail. Despite what

Let L, R be port graphs, $\mathcal{M}, \mathcal{N}$ subgraphs of R and $\mathcal{W}$ a subgraph of L.
We define $k \in \mathbb{N}$ and $\pi_{i=1\ldots k} \in [0,1]$ such that $\sum_{i=1}^{k} \pi_i = 1$.
Let *label* be the name used to identify the labelling functions $\mathscr{L}$ from Λ,
$v \in \lambda$ is a valid value comparable to those returned by a labelling function.

Rules	**(Transformations)**	T	::=	$L_{\mathcal{W}} \Rightarrow R^{\mathcal{N}}_{\mathcal{M}} \mid (T \parallel T)$
			|	`ppick`$(T_1, \pi_1, \ldots, T_k, \pi_k)$
	(Applications)	A	::=	`all`(T) | `one`(T)
Positions	**(Focusing)**	F	::=	`crtGraph` | `crtPos` | `crtBan`
			|	$F \cup F \mid F \cap F \mid F \setminus F \mid (F) \mid \emptyset$
			|	`ppick`$(F_1, \pi_1, \ldots, F_k, \pi_k)$
			|	`property`$(F, Elem[, Expr])$
			|	`ngb`$(F, Elem[, Expr])$
			|	`ngbIn`$(F, Elem[, Expr])$
			|	`ngbOut`$(F, Elem[, Expr])$
	(Determine)	D	::=	`all`(F) | `one`(F)
	(Update)	U	::=	`setPos`(D) | `setBan`(D)
			|	`update`$(function\{parameters\})$
Properties	**(Properties)**	$Elem$	:=	`node` | `edge` | `port`
		$Expr$	:=	$label\ Relop\ v \mid Expr\ \&\&\ Expr$
		$Relop$	:=	== | != | > | <
			|	>= | <=
Compositions	**(Comparison)**	C	::=	$F = F \mid F$!= $F \mid F \subset F \mid$ `isEmpty`(F)
	(Strategy)	S	::=	id | fail | $A \mid U \mid C \mid S; S$
			|	`if`(S)`then`(S)`else`(S) | (S)`orelse`(S)
			|	`repeat`$(S)[(k)]$ | `while`$(S)[(k)]$`do`(S)
			|	`try`(S) | `not`(S) | `ppick`$(S_1, \pi_1, \ldots, S_k, \pi_k)$

Table 2.1: Syntax of the Strategy Language.

one may think, (S_1)`orelse`(S_2) and `if`(S_1)`then`(S_1)`else`(S_2) have indeed the same overall behaviour but they are not equivalent. The latter instruction applies the strategy S_1 two times instead of once for the former: at first, to try whether S_1 can be successfully applied, and once more to effectively apply it. Because we are choosing elements at random in sets, we can not guarantee that the two applications yield the same results whereas the `orelse` instruction tries applying S_1 and keep the very same result if the operation succeed. Additionally, when applying complex strategies with numerous instructions, such small improvement avoiding redundant operations can enhance the overall efficiency.

- `repeat`$(S)[(k)]$, a simple loop; this instruction repeats the application of S until it fails. An optional iteration counter is available such that, if k is specified, then the number of repetitions cannot exceed its value. The `repeat` construct is very useful to apply a strategy for as many times as possible. Because of its peculiar case of application, this instruction

can never fail, even if the application of its inner strategy does so.

- `while`(S_1)[(k)]`do`(S_2), a more complex loop differentiating the application strategy from the one used as a condition; this instruction is very similar to the `repeat` construct. Strategy S_1 is first applied on a copy of $G_{\mathcal{P}}^{\mathcal{Q}}$, if the application succeed, S_2 is applied, transforming $G_{\mathcal{P}}^{\mathcal{Q}}$ in $G_1{}_{\mathcal{P}}^{\mathcal{Q}}$. If S_1 successfully apply on $G_1{}_{\mathcal{P}}^{\mathcal{Q}}$, S_2 is applied and the graph is rewritten in $G_2{}_{\mathcal{P}}^{\mathcal{Q}}$, etc. If the optional iteration counter k is provided, the application stops with $G_k{}_{\mathcal{P}}^{\mathcal{Q}}$ otherwise it continues. For the same reasons previously stated for the `orelse` and `if-then-else` constructs, the operation `while`$(S)(k)$`do`(S) may seem inclined to return results similar to what `repeat`$(S)(k)$ would produce but it does not behave in the same way. Additionally, like the `repeat` construct, this instruction can never fail.

- `try`(S), a construct to try to apply any strategy; if S successfully applies on $G_{\mathcal{P}}^{\mathcal{Q}}$, the resulting $G'{}_{\mathcal{P}'}^{\mathcal{Q}'}$ is kept, however, if it fails, then $G_{\mathcal{P}}^{\mathcal{Q}}$ remains valid and the strategy keep going without failing. This construct can never fail and returns a result similar to (S)`orelse`(id).

- `not`(S), a negative operator; this construct inverses the resulting outcome of the concerned strategy: if S is successfully applied, then `not`(S) fails, and conversely, if S fails, then `not`(S) succeeds. This instruction can also be replaced by the construct `if`(S)`then`(fail)`else`(id).

- `ppick`$(S_1, \pi_1, \ldots, S_k, \pi_k)$, for probabilistic applications; a set of strategies is proposed, each with a probability of application $\pi_1, \ldots, \pi_k \in [0, 1]$ that $\sum_{i=1}^{k} \pi_i = 1$. Upon execution, the construct picks one of the strategies for application, according to the given probabilities, returning `id` if the application is successful and `fail` otherwise. This construct is very useful as a lot of models use probabilistic behaviours and transforms non deterministically.

These different constructs grant strategic graph programs the management of complex strategies and operations. The handling of loops and conditional or probabilistic instructions establishes a strong basis for the programming language and opens possibilities for the handling of complex situations and advanced transformations. Let us now take a closer look at the second group of constructs managing the position and ban subgraphs: the **Positions**. Three different standard subgraphs exist for our context: `crtGraph` –the whole current graph, potentially transformed by previous rewrite rules applications–, `crtPos` –the position graph, containing, among others, all the elements returned in $\mathcal{M}$ during the last rewrite operation and which will be targeted by $\mathcal{W}$ in the next rule application–, and `crtBan` –the ban subgraph, consisting of all the banned elements such as the ones returned by $\mathcal{N}$ in the last rule; these elements will be ignored for the matching of the next rule application. These three subgraphs can be targeted at any time to undergo set operations (union $\cup$, intersection $\cap$, minus $\setminus$) and are also subjected to be used as the basis for additional filtering operations we detail hereafter:

- `property`$(F, Elem[, Expr])$, the Swiss army knife for filtering elements based on their labels values; the operation is used to select elements (of type $Elem$) of a given graph (passed as a parameter in F) that satisfy a certain expression $Expr$ we present afterwards. It is truly a filtering construct, building a subgraph only from the elements for which the expression $Expr$ is respected.

- `ngb`$(F, Elem[, Expr])$, the selection of neighbours and connected elements; this operation returns the subset of adjacent elements of the located graph F providing they respect the conditions imposed by the optional expression $Expr$. Because we plan to use directed graphs, the edges direction needs to be taken into account in certain models, we thus propose the two variations `ngbIn` and `ngbOut` to consider the incoming and outgoing elements.

- `ppick`$(F_1, \pi_1, \ldots, F_k, \pi_k)$, for probabilistic choice of located graphs; this probabilistic pick follows the same principles as the `ppick` operation existing in the **Compositions** except that this time, we propose to randomly select a located subgraph to use based of the given probabilities.
- `update`$(function\{parameters\})$, a gate for custom subgraph extraction; this operation offers to any user the possibility to design their own subgraph extraction method and apply it on the graph at hand. The method simply needs to be implemented as a PORGY plugin, and the strategy will automatically launch it with the given parameters and return the resulting subgraph. This operation is very useful to obtain subgraphs respecting conditions too complex to properly define using our current syntax.

Once any of these operation is achieved, the corresponding subgraph F can be used as is or be affected to position or ban and used in the next rules' applications. Beforehand however, we pass through one last filter which allows us to either consider the subgraph F as a whole with `all`(F), or to only select a single element with `one`(F), with the result being affected to D. We can then use the operation `setPos`(D) (respectively, `setBan`(D)) to set the position subgraph $\mathcal{P}$ (respectively the ban subgraph $\mathcal{Q}$) to D. Overall, the subgraph operations and their affectation always succeed and return `id`.

It may be useful to remind the reader that, because $\mathcal{P}$ and $\mathcal{Q}$ are subgraphs, lone nodes, ports or edges can not be allowed within. A sub(port)graph is a proper (port) graph and should thus only contain portnodes or edges connecting two portnodes. As a result, for a port graph $G = (N, P, E, \mathscr{E}, \mathscr{P}, \mathscr{N}, \Lambda)$, setting a node $n \in N$ in one of the located subgraphs mean that all of its attached ports $\mathscr{N}(n)$ must also be part of the subgraph. The sole addition of a port $p \in P$ implies the insertion of its attached node $\mathscr{P}(p) = n \in N$ and all of the attached node's ports $\mathscr{N}(n)$ at the same time. Similarly, adding an edge $e \in E$ to a subgraph implies that we add to the same subgraph: both of the extremities $(p, q) = \mathscr{E}(e)$ (with $p, q \in P$), the nodes attached to the extremities $m = \mathscr{P}(p)$ and $n = \mathscr{P}(q)$ (for $m, n \in N$), and all of the ports of the two attached nodes $\mathscr{N}(m)$ and $\mathscr{N}(n)$. Formally, let a port graph $G = (N, P, E, \mathscr{E}, \mathscr{P}, \mathscr{N}, \Lambda)$, a subgraph F of G, $n \in N$, $p, q \in P$, $e \in E$ and with $(p, q) = \mathscr{E}(e)$, the addition of an element of G to F occurs in accordance to the following implications:

$$F \cup n \implies F \cup n \cup \mathscr{N}(n)$$

$$F \cup p \implies F \cup p \cup \mathscr{P}(p) \cup \mathscr{N}(\mathscr{P}(p))$$

$$F \cup e \implies F \cup e \cup \mathscr{P}(p) \cup \mathscr{N}(\mathscr{P}(p)) \cup \mathscr{P}(q) \cup \mathscr{N}(\mathscr{P}(q))$$

Among the operations in the group **Positions**, the constructs *Elem* and *Expr* are repeatedly used to express the type of the elements concerned by the filtering operation and the inequality proposed to actually select the appropriate elements; we define them into the group **Properties**. An element type, expressed using *Elem*, can either be a node, an edge or a port. This granularity allows us to address each of the port graph element indifferently. The construct *Expr* is used jointly to describe an inequation, where the value returned by a precise labelling function is compared, as specified by the *Relop* operator, to a given value v for all of the elements of type *Elem*. For instance, the strategic operation `setPos(property(crtGraph, edge,` $value \neq 2$`))` will find all the edges for which the labelling function *value* returns a value different from 2, and affect them all to $\mathcal{P}$. *Expr* can also be composed of multiple inequations to handle as many conditional valuation as desired, e.g., `setPos(property(crtGraph, edge,` $value > 2$ `&&` $value < 2$`))` which performs the exact same computation as above.

All these different constructs lead us to the final group handling the applications and graph transformations: the **Rules**. In a simple scenario, a strategy should be allowed to apply directly

a rule $L_{\mathcal{W}} \Rightarrow R^{\mathcal{N}}_{\mathcal{M}}$ (after the initialisation of the position and ban subgraphs). However, because we propose a random selection among the matching elements to pick the candidate(s) to rewrite, we have decided to introduce different operations allowing us to decide whether we wish to obtain a single rewritten instance of the graph or if we wish to get all the possible instances.

- `one`(T) computes one of the possible applications of the transformation T. Among all possible matching solutions, the operation selects one through an equiprobabilistic choice and apply it.
- `all`(T) results in the application of the rewrite rule for all the possible matching solutions. As for `one`(T), the transformation T is performed on the located graph at the current position, but the results create a new located graph for each application.

Of course, this unconventional output can only make sense if we can present the results coherently, and this is where the derivation tree shows its full potential. Naturally, a continuous application of single transformations using `one`(T) will produce a simple list of intermediary states, with each new operation generating a new element at the end of the list. However, using `all`(T) will create as many branches as there are possible matching solutions, with each child state presenting a copy of the initial graph transformed as specified by T. As one can see, although these two constructs are also available in the **Positions** group, the outcome of those operations is not the same. Conversely, the `ppick`$(T_1, \pi_1, \ldots, T_k, \pi_k)$ construct follows exactly the same behaviour as previously explained, with each transformation T_i having a probability π_i to be applied. Because this construct is classified as a transformation, a single application of the rules R or R' with equal probabilities will need to be expressed as: `one`(`ppick`$(R, 0.5, R', 0.5)$). Alternatively, the `ppick` construct from the strategy, which we defined earlier, can also be used to obtain the same result but will instead be expressed as: `ppick`(`one`$(R), 0.5,$ `one`$(R'), 0.5)$. Finally, the last operation available in the **Rules** group of constructs is the simultaneous application of transformations noted: $(T \parallel T)$. This operation applies two transformations on the same located graph and at the same time, thus only creating a single additional step in the derivation tree. An application of $(R \parallel R')$ can only succeed by applying R and R' concurrently, that is on different elements of the same graph without hindering each other application such that the intersection of elements being rewritten by both R and R' is empty. If such distinct morphisms are not available for the LHS of R and R', the construct application will fail and neither rule will be applied.

All the graph generation and information propagation models presented in this thesis use the strategy language to specify the transformations to perform and at which moment they must occur. Each of the strategy developed are detailed when introduced so that the reader does not have to constantly refer to the whole syntax given in Table 2.1. To achieve this section, we present a simple example of strategic graph rewriting in action. The transformations we propose here aim at giving a foretaste of Chapter 3 with the generation of a port graph. To achieve this, we re-use the two rules presented earlier in Figure 2.12a (starting with a single portnode, a second portnode is created and linked to the first) and Figure 2.12b (from any two distinct portnodes, an edge is created to connect them) and rename them respectively $R1$ and $R2$. Due to these generative rules properties, the graphs we can create will present some specific characteristics: they are undirected as the rules do not produce directed edges, they are connected as no portnode is ever created disconnected from an existing portnode and no edge is deleted, and, because we do not check for edge preexistence before connecting two portnodes (like in Rule 2.13b), the graphs can have multiple edges between the same two portnodes. When generating a graph, it is not uncommon to decide beforehand of the number of nodes and edges which will be created in the process. Consequently, we propose Strategy 1 to generate a port graph with $|N|$ portnodes

and $|E|$ edges. The graph G used as a starting point for this example only contains a sole port node, formed of a single port attached to a node. The strategy begins with the initialisation of $\mathcal{P}$ to the whole graph (with $\mathcal{Q} = \emptyset$). The next instruction is a bounded repeat loop, applying $|N|-1$ times the rewrite rule $R1$, thus resulting in a graph with $|N|$ portnodes and $|N|-1$ edges. The following instruction is once more a bounded repeat loop, but applying $|E|-(|N|-1)$ times the rewrite rule $R2$, thus resulting in a graph with $|N|$ portnodes and $|E|$ edges as desired. An example of port graph generated using this strategy is presented in Figure 2.15a.

Strategy 1: Ordered application of two rules for generating a multi random graph.

```
1 setPos(crtGraph); // Starts from a graph with a single portnode
2 repeat(
3   one(R1)
4 )(|N| − 1); // Results in a graph with |N| portnodes and |N| − 1 edges
5 repeat(
6   one(R2)
7 )(|E| − |N| + 1) // Results in a graph with |E| edges (|N| − 1 + |E| − |N| + 1) = |E|)
```

A second way to perform a graph generation using our rule is to follow a more random approach. We propose in Strategy 2 such method generating a graph by applying equiprobabilistically rules $R1$ and $R2$. Just like the previous strategy, we start with a graph containing a portnode with a single port. After the initialisation of the position subgraph $\mathcal{P}$ to contain the whole current graph, a first application of $R1$ is completed to create a second port node and thus allow any potential up-coming applications of $R2$ to find appropriate candidates for their LHS matching. We then instruct, using a bounded repeat loop, to either apply $R1$ or $R2$ with equal chances using a `ppick` construct; this equiprobable application is repeated $K-1$ times. Because we do not know which transformation is going to be applied at any time, we can not prognosticate the exact number of elements the resulting graph will contain. However, as the first rule creates a node and an edge while the second only creates an edge, we know that the final number of edges is equal to the number of times either rules have been applied, and thus, the graph will always have K edges. On the other hand, we are unable to properly know the number of nodes in the resulting graphs but we know it depends of the number of applications of $R1$. We can express the two extreme cases as follows: if $R1$ has never been picked during the repeat loop, the number of nodes stays at 2; conversely, if $R1$ has always been picked, then the graph will contain $K+1$ nodes. Because the probabilistic pick is equitable, the number of nodes in the graph is expected to be just in between these two extreme values; the graph is expected to have around $(K+3)/2$ nodes. An example of port graph generated using Strategy 2 is presented in Figure 2.15b for $K = 49$.

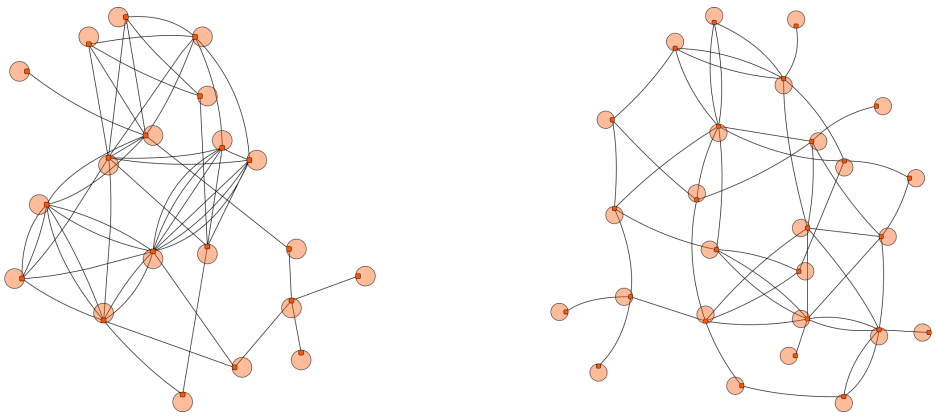
Strategy 2: Equi-probabilistic application of two rules for generating a random multi graph.

```
1 setPos(crtGraph); // Starts from a graph with a single portnode
2 one(R1); // Apply R1 one time to obtain a graph with two portnodes (and an edge)
3 repeat(
4   ppick(one(R1), 0.5, one(R2), 0.5)
5 )(K − 1) // Results in a graph with K edges
```

The examples shown in Figure 2.15 are only two instances of all the possible random graphs

which can be generated using Strategies 1 and 2. As one can notice, no loop (an edge connecting a node with itself) is actually created in the two examples unlike the model presented in Figure 2.2b. This is simply because neither rules can produce such an element. Particularly, the portnodes in $R2$ (Fig. 2.12b) are specified as distinct elements and can not match on the same candidate, thus forbidding any possible loop creation. Nonetheless, both generated graphs present the characteristics we were expecting as they are connected, undirected, and have multiple edges. Although we have mentioned it several times earlier as a useful tool to follow the rewriting steps performed on the port graph, the derivation trees for the two strategies presented above offer next to no interesting visual information. As the strategies only apply the rules one after another, the respective derivation trees are only composed of a single and very long branch following the transformations step by step along the fifty or so states tracking the rule and the located graphs just used. Considering the lack of interesting branching structure and the poor legibility of such long traces, we do not think the visualisation of the derivation tree to be necessary at this point; more appealing and developed examples are nonetheless detailed later in this thesis, and we can only redirect the eager reader to Chapter 4, Section 4.2.3 on page 92 for a quick glance at them.

When compared to the classical graph rewriting system, strategic located graph rewriting uses several additional complex concepts which allow to: firstly, precisely define subgraphs of elements to either target or ignore during the rewriting step, and secondly, offer a detailed language to express how, where and when to perform the rewriting operations. Furthermore, the joint usage of the located graphs and the strategies offers possibilities to manipulate the located graphs using set operators and potentially very specific filtering operations. All these points make the strategy language very flexible, so much in fact that it is proven as effectively able to simulate any Turing machine (Fernández et al., 2016a), and thus to achieve Turing completeness.



(a) Example of port graph obtained after the application of Strategy 1 (20 nodes, 50 edges).

(b) Example of port graph obtained after the application of Strategy 2 for $K = 49$ (26 nodes, 50 edges).

Figure 2.15: Examples of connected, undirected, multiple port graphs generated using Rules $R1$ (Fig. 2.12a) and $R2$ (Fig. 2.12b). Despite not being visible, the degree of each node has been computed during the rewriting steps and the values are stored in the labelling function $Degree \in \Lambda_G$.

Strategies are consequently small but fully functional programs, using functions, i.e., rules, to perform transformations on an object given as an input, i.e., a graph.

2.4 Conclusion

We have established in this chapter all the definitions concerning the inner workings of graph rewriting transformations. After a few short recalls on graph theory, we have explored the concept of (port) graph rewriting systems. Despite the possibilities offered by the technique, we quickly noticed that precise transformations or some more advanced operations were impossible to perform with the system as defined: although we were able to achieve any possible transformation, we lacked some measures to, first, allow us to specify which elements were to be rewritten and which should be left untouched, and second, manage the rule applications to allow complex operations. By introducing located (port) graphs, we have been able to address the former issue: the position and ban subgraphs can be used to limit the reach of our rewrite rules by either imposing or forbidding elements into being selected for the transformation. The latter problem, raising the question of rewrite rule management and complex operation, was resolved by the introduction of strategies. These procedures offer the possibility to express which rewrite rule to apply, under which conditions, how many times, and additionally propose methods to handle the located subgraphs. All of these tools are quite simply the basics we need to properly engage the models in the following chapters.

Chapter 3

Modelling graph generation algorithms using graph rewriting techniques

Contents

3.1 Related works 43
3.1.1 Probabilistic inductive class of graphs 43
3.1.2 Non-strategic graph rewriting approach 46
3.2 Translating the small-world model 47
3.2.1 Original algorithm 48
3.2.2 Translation to our rewriting formalisation 49
3.2.3 Formal validation of the translated model 59
3.3 Introducing a new social network generative model 61
3.3.1 Generation of a simple directed acyclic port graph 63
3.3.2 Creating complementary connections 64
3.3.3 Construction of communities using triads 67
3.3.4 Model validation and discussion 71
3.4 Conclusion 73

Among all the technological innovations which took place during the last twenty five years or so, the appearance and opening of Internet to the public is most likely the one which had the highest impact on the world. The popularisation of this world-wide network has allowed to connect people from different regions, countries and continents, shifting and transforming social interactions by creating a new "place" to discuss and exchange ideas and knowledge. In particular, the creation of online communities has created a fertile ground for social scientists (e.g., economists, sociologists, ethnographers). Although social studies have been commonly performed for quite some time now (Freeman, 2004), on-line communities have allowed exchanges and interactions on a scale never studied before (Leskovec et al., 2008; Newman, 2012). This

This chapter is based on:
Maribel Fernández, Hélène Kirchner, Bruno Pinaud, and Jason Vallet. *Labelled Graph Rewriting Meets Social Networks.* In Rewriting Logic and Its Applications, WRLA 2016, volume 9942 of LNCS, page 25, Eindhoven, Netherlands, April 2016. Springer International Publishing Switzerland.

increased magnitude created new ways to gather such information, analyse it, understand it, and, finally, communicate the results to other researchers to progress as a society. The study of such topics is directly at the heart of digital humanities.[1]

In response to this phenomenon, the research field of social network analysis, shortened as SNA, has boomed in recent years, as discussed in the first chapter of Jackson et al. (2006), in parallel to the development and popularisation of online social networks. The scientists of this area have successfully used a wide array of techniques originating from different scientific domains, such as statistics (O'Malley, 2013), graph theory (Brandes and Erlebach, 2005), or psychology (Westaby et al., 2014), to study, analyse and, ultimately, grasp and understand what is happening within these on-line social networks.

One of the existing roads available to us to shed some light on the inner mechanisms steering social networks would consist in recreating such network. Several approaches can help us attain such a goal:

- Mathematically or statistically-inclined researchers start by observing the existing social networks and quite naturally consider the analogy with their structure and graphs. They can then extract critical cues characterising the different elements composing them (number of elements, peculiarities concerning their distribution and topological features...), and define formulas and algorithms to generate graphs bearing such specificities.

- Social researchers, on the other hand, would much rather understand the action performed by the users, especially, how these persons decide with whom to communicate. By studying the way the individuals first interact with each other and how their relationship evolves from here, globally leading the network to its current state, the researchers can document and recreate the steps performed.

Whichever of these two solutions is preferred, in the end, both approaches describe a model able to build networks rather similar to those observed in real world situations. This ability to create new networks based on existing graphs does not only provide us with a simulacrum of the original, but allows us to study the generated network and, most importantly, to use it as a testing example for any of our hypothesis and new methods. Although real-world social networks come in many different forms, their number is quite limited overall when compared to the possible graph structures possessing identical characteristics one could generate. Using such generation models to create networks on which to work allows us to perform tests in somewhat similar conditions each time, and, essentially, to ensure the soundness and stability of the approaches we experiment with.

In this chapter, we propose to follow both of these approaches and implement the models respectively extracted using graph rewriting systems. Although graph generation algorithms can be quite complex processes, we present in the following how breaking-down the models to more manageable tasks can allow us to provide translations respecting the original algorithm specificities. We begin by presenting some related works proposing rule-based descriptions to express basic graph generation models. We then propose two different implementations of generation models translated to our strategic graph rewriting formalism. The first one is the well-known small-world model introduced by Watts and Strogatz (1998). This model description follows the mathematical approach with its resulting networks designed to display on average a high clustering coefficient and a short distance between vertices. The second implementation reflects a scenario –established by ourselves– aiming to mimic a social network creation with a design more consistent with the sociological approach.

[1] https://en.wikipedia.org/wiki/Digital_humanities

3.1 Related works

Graph generation has been a well studied and popular subject among researchers for quite some time (Erdős and Rényi, 1959, 1960; Bollobás, 2001). Naturally, when such a topic is applied to social networks, an item just as fashionable as of late (Carrington et al., 2005; Newman et al., 2006; Scott and Carrington, 2011), the resulting graphs attract a lot of attention (Watts and Strogatz, 1998; Barabási and Albert, 1999) with several diverse fields of application (Ioannides, 2005; Jackson et al., 2006). A lot of different papers introduce graph generation models, each with their own characteristics and properties respecting some of the specificities of social networks. Nonetheless, three models appear truly influential among the others, due mostly to their antecedence, and could be considered as seminal in the field of social network generation. The first paper, written by Erdős and Rényi (1959), has established the basis of random graph generation, leading to the creation of the standard and well-known ER graph generator. The two other papers, authored by Watts and Strogatz (1998) and Barabási and Albert (1999), introduce respectively the small-world and scale-free networks models, defining the corresponding properties at the same time. Basically, where small-world networks are characterised by their high clustering coefficient and short average paths, scale-free networks exhibit peculiar degree distributions, with edges being connected to nodes according to a preferential attachment scenario. Those notorious characteristics have also been encountered in several other different types of existing networks, such as the Internet (Lawrence and Giles, 1998) or co-authorship (Redner, 1998) and transportation networks (Banavar et al., 1999).

Usually, graph generation models are quite straightforwardly implemented as described in their respective original papers, with the authors, or sometimes third-parties (e.g., Batagelj and Brandes (2005); Nobari et al. (2011)), providing some form of algorithms detailing the different steps to follow for a successful and efficient implementation. The instructions consequently used to express the overall transformation, starting from next to nothing and resulting into a finished graph, simply break-down the models into atomic operations (indivisible), allowing us to understand them in depth and follow each and every applied modification. The rule-based formalism, used in graph rewriting systems and comparable applications, follows a very similar approach. We simply interpret the operations described by the model as atomic transformations to apply on the graph, such as: *"add a vertex here"*, or *"create an edge over there between those two vertices"*. Although we defined to some extent the graph rewriting system and the mechanisms we use as a basis in the previous chapter, other rule-based approaches exist (e.g., Kejžar et al. (2008)), some of which are also graph rewriting techniques only proposing a methodology different from our own (e.g., Taentzer et al. (2007)). Nonetheless, to the best of our knowledge, only two of those related works have proposed implementations of social network generation models using their own formalism.

Hereafter, we present in details those two works and their corresponding rule-based graph generation algorithm. We also discuss their respective advantages and drawbacks, as well as the underlying differences with our own graph rewriting system.

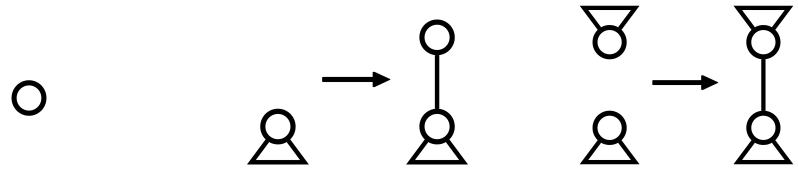
3.1.1 Probabilistic inductive class of graphs

The first rule-based formalism of interest we present exploit probabilistic inductive class of graphs (PICG). Initially introduced by Kejžar et al. (2006), the PICG have been further developed in Kejžar et al. (2008), which we use as a reference for the current section. PICG are based on the first inductive definition of a class of graphs, as given by Eberhard (1891), and further extended upon the notion of inductive class of graphs described by Curry (1963). The PICG formalism is defined as follows:

Definition 14 (Probabilistic Inductive Class of Graphs, from Kejžar et al. (2008))**.** *A probabilistic inductive class of graphs (PICG),* $\mathscr{I}$*, is given by:*

1. *a class* $\mathscr{B}$ *of initial graphs, the* basis *of PICG,*
2. *a class* $\mathscr{R}$ *of generating rules, each with distinguished* left element *to which the rule is applied to replace it with the* right element*,*
3. *a probability distribution specifying how the initial graph is chosen from class* $\mathscr{B}$*,*
4. *a probability distribution specifying how the rules from class* $\mathscr{R}$ *are applied, and*
5. *a set of probability distributions specifying how the left elements for every rule in class* $\mathscr{R}$ *are chosen.*

With the PICG extending upon the inductive class of graph by integrating a notion of probability, this definition draws a stunning similarity between the PICG formalism and the graph rewriting system we use. For instance, the standard set of graphs $\mathscr{B}$ on which one can possibly apply the rules and the actual set of rules $\mathscr{R}$ have the same respective uses. The two models however have distinct behaviours concerning the handling of the probability distributions deciding on which basis graph and particular elements to apply the rules (respectively point 3 and 5). The probabilistic decision pinpointing which rule to select exactly (point 4) is also handled quite differently. To further grasp those differences, we propose to study an example of PICG. To this end, we reuse and adapt the basic random graph generation model presented by the authors in Kejžar et al. (2006) to illustrate the principle of graph construction using the inductive class of graphs.



(a) Basis B (b) Rule $R1$ (c) Rule $R2$

Figure 3.1: Example of an inductive class of graphs $\mathscr{I} = (B, R1, R2)$ taken from Kejžar et al. (2006). The basis B is a single node; the rule $R1$ selects an existing vertex and connects a new node to it (a new vertex and a new edge are created); the rule $R2$ selects two existing vertices and connects them together (only an edge is created). The small trapezoids affixed to the vertices in the rules indicate the possible existence of edges linking the node to other elements non-exhaustively pictured in the rules.

The example used by Kejžar *et al.* is quite simple: starting from a single node declared in a basis $B \in \mathscr{B}$, two rules, $R1, R2 \in \mathscr{R}$, are successively applied; these different elements are shown in Figure 3.1. Because the example is expressed as a PICG, diverse probability distributions are defined: first, to indicate which basis from $\mathscr{B}$ to use; then, how the rules $R1$ and $R2$ are applied; and finally, how the elements subjected to rule application are picked out (respectively, point 3, 4 and 5 of Def. 14 on the previous page). The first one is straightforward as $\mathscr{B}$ only contains a single graph. The probability distribution managing the rule application and the element selection are a bit more complex to handle as the results may wildly diverge for different values. A simple solution would be to use plain equi-probabilistic distributions to choose both the rules

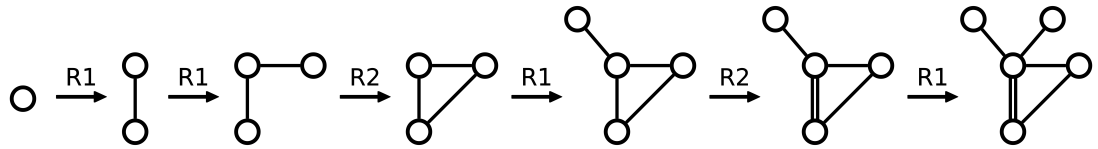


Figure 3.2: Example of random graph construction taken from Kejžar et al. (2006). Starting from the basis consisting of a single node, six successive rule applications are performed, as indicated by the right arrows, transforming the graph step by step.

and elements on which to apply them. $R1$ and $R2$ will thus each have a probability $p = 0.5$ to be applied, and every element can be the subject of a transformation with equal probability.

Based on these parameters, we show in Figure 3.2 an example of possible graph construction. Starting from the basis B, the figure shows six successive transformations obtained by various $R1$ and $R2$ applications as implied by the labels on the right arrows symbolising each rule operation. The resulting graph consists of 5 nodes and 6 edges, conforming with the four $R1$ and two $R2$ applications.

Overall, the generation example detailed above is quite simple and could be adapted to work with any rule-based formalism, including of course graph rewriting. The corresponding implementation of a random graph construction model using our strategic graph rewriting system would employ identical looking rules. However, the different probability distributions used in a PICG would be replaced by a strategy to steer both element selection and any rule application. While their goal are analogous, the two methods proceed differently to achieve them. A strategy is similar to a program where atomic instructions are rewrite rules (see Section 2.3.2 on page 32). Each rule can then be applied normally, several times using a loop, or only in some cases if a given condition is met. For instance, given a probability distribution for the rules from class $\mathscr{R}$ (Def. 14 on page 43, point 4) implies that, for each rule application, chances of applying rule R remain the same each time. Using a strategy to apply the rules fixes this issue by granting finer control over the transformations to perform. The rules can then be applied in a precise order if needed, and combined with additional instructions like `repeat`, `while-do`, `if-then-else` and `try`. The probability distribution (Def. 14, point 5), handling how the elements about to be transformed are selected, can also be replaced advantageously. By default, our graph rewriting system automatically performs an equi-probabilistic choice when selecting the element on which to apply the rule whereas PICGs propose to submit a specific probability distribution. Nonetheless, it is possible to use a strategy to achieve a filtering operation by manipulating the position or ban sets (respectively $\mathcal{P}$ and $\mathcal{Q}$, see Definition 11 on page 30) before applying the desired rules.

In the end, while PICGs propose a working solution to generate diverse types of networks,[2] our graph rewriting technique claims the advantage of the strategy-handling mechanism to conduct rule applications. With their adaptable probability distributions, PICGs can decide which rule to apply on which element. This feature enables a quite advanced rule management system by providing some fine control over the general transformation process. However, conditional applications, similar to what we may obtain when using instructions like `if-then-else` or `try` with our strategy, would be much more complex to reproduce.

[2]Another generation example is proposed in Kejžar et al. (2008). The authors present a preferential attachment model (from Barabási and Albert (1999)) using the vertices' degree to influence the probability of selection of a given node: the higher the degree, the higher the probability is to pick the vertex.

3.1.2 Non-strategic graph rewriting approach

The second related work which perform a graph generation using a rule-based approach is attributed to Barry et al. (2015). In their paper, the authors propose an evolutionary-oriented method to generate a graph with specific properties and features. To begin with, an initial set of rule-based transformations and their respective application probabilities is suggested to produce a graph. The resulting graph is then analysed and rated with respect to its expected features. Iteratively, the transformations' probabilities are updated and the set of rules reapplied to create a new refined graph which is re-analysed and rated once again. A genetic algorithm is used to find the appropriate probabilities and ultimately generate the graph with the desired properties.

Although the main contribution of the paper resides in generating graphs exhibiting specific features, our interest leans toward the creation model proposed in the paper as well as its underlying processes. Let us first mention some of the background concepts established by the authors. A graph G consists of a set of nodes V and a set of edges E such that each edge $e \in E$ corresponds to a unique pair of nodes (v_i, v_j) with $v_i, v_j \in V$. The formal definition of the graph generation model components is:

Definition 15 (Components of the graph generation model, from Barry et al. (2015))**.** *The graph generation model comprises the following components:*

1. *an initial state which comprises an initial graph* $G_0(V_0, E_0)$
2. *a set of rules* $\{R\}$ *of the form* $\alpha \to \beta$*, which transforms a graph* G *to a new graph* G'*; where* α *represents an existing construct in the graph* ($\alpha \subseteq G(V, E)$)*, and* β *represents a new construct which replace* α*.*
3. *a probability distribution* $\{P_i\}$ *over these rules,*
4. *the number of times* ($NumSteps$)*, to select and apply a rule.*

Just like encountered in the PICG (see Def. 14 on page 43, points 2 and 4), a set of rules is available to perform transformations and a probability distribution is used to overlook the rule application and decide which one should be used to rewrite the graph. For their graph generation model, Barry *et al.* use three distinct rules. The first one creates a new vertex and a new edge, connecting them to an existing vertex. The second rule creates an edge between two pre-existing vertices in the graph, checking beforehand that no edge already exists between the two vertices. The final rule replaces an existing vertex with a 'triangle'; two new vertices are thus created and connected to an existing vertex and another edge is added between the two new vertices. Tables 3.1a, 3.1b, and 3.1c on the next page show the corresponding pre- and post-conditions for each rule. As one can see, the two first rules (Tables 3.1a and 3.1b) are very similar to the ones proposed in the PICG's example (Fig. 3.1 on page 44). The second rule is a bit more precise as it adds a specific condition on the non-existence of an edge between the two vertices before creating one. This peculiarity forbids the generation of multigraphs, whereas such an operation can be performed with the PICG example (see fifth rule application in Fig. 3.2 on the previous page). The last rule (Table 3.1c on the facing page) is new however. An existing vertex is transformed into a triangle, creating two neighbours and all the appropriate edges in between.

Due to its formalism, the solution of Barry *et al.* is much closer from our own than from the PICG. When further compared to the PICG definition, the graph generation model of Barry *et al.* only propose to use a single possible starting graph instead of a set of possible ones. The authors nonetheless propose to directly specify the number of times each rule must be applied, thus adding eventually some control over the generation process, but the decision over which rule to apply is only handled by a unique probability distribution. Furthermore, while one can

Name	Add a vertex
Pre-condition	$v_i \in V; v_j \notin V$
Post-condition	$v_i, v_j \in V'; e(v_i, v_j) \in E'$

(a) Condition for Rule 1.

Name	Add an edge
Pre-condition	$v_i, v_j \in V; e(v_i, v_j) \notin E$
Post-condition	$v_i, v_j \in V'; e(v_i, v_j) \in E'$

(b) Condition for Rule 2.

Name	Add a triangle
Pre-condition	$v_i \in V; v_j, v_k \notin V$
Post-condition	$v_i, v_j, v_k \in V'$ $e(v_i, v_j), e(v_j, v_k), e(v_i, v_k) \in E'$

(c) Condition for Rule 3.

Table 3.1: Tables of conditions for the graph generation model's rules, from Barry et al. (2015).

switch the probability distributions specifying how the matching elements for the left-hand side of the rules are decided in PICG, no mention of such feature is made in Barry et al. (2015).

Overall, both works presented above are able to achieve a graph generation. Straightforwardly, the previous solutions, as well as our own graph rewriting system, proceed in the same manner: starting with a graph, a rule is applied on it, thus transforming some of its elements. The operation can then be repeated at convenience or a given number of times, with each of the above solutions having a probability distribution responsible for picking up the rule to apply. However, transformation after transformation, the rewriting operations are always repeated on the same set of rules with the same unchanged probability distribution. Thanks to its strategic language steering rule applications, our graph rewriting system proposes a more adaptive behaviour and proceed with a finer precision.

Leaving aside the two previous formalisms, we focus in the following on using our solution allowing graph rewriting operations managed by a strategy. To grasp the flexibility and fine control of our approach, we propose to follow first-hand the translation of an existing generation model using our specific formalism.

3.2 Translating the small-world model

Introduced in their 1998 paper, the generative model of Watts and Strogatz (1998) has been studied many times over the years. Several other researchers have proposed revised and extended versions of the original algorithm (Holme and Kim, 2002; Klemm and Eguíluz, 2002; Wang et al., 2006; Sallaberry et al., 2013), often adapting it to obtain degree distributions similar to those encountered in scale-free graphs as described by Barabási and Albert (1999). We focus in the current section on the standard model as initially proposed by Watts and Strotgatz.

Named after the small-world phenomenon popularised by Milgram (1967),[3] describing how any two persons in the world can relate to each other through at most five intermediary acquaintances, small-world networks topologies are neither regular nor completely random but lie in-between. From the point-of-view of the authors, such network is characterised using two measures. Firstly, the average distance between any pair of nodes (i.e., the average number of nodes in the shortest chain connecting any two nodes, described as the **characteristic path length**

[3] According to Barabasi and Frangos (2002), an earlier description of this phenomenon can be find in the short story "*Láncszemek*" (translatable as *Chains*) published in Frigyes Karinthy's 1929 book: "*Minden masképpen van*" (*Everything Is Different*) (Frigyes, 1929). A later theatre play called "*Six Degrees of Separation*" (Guare, 1990) can also be mentioned for spreading the results of Milgram's experimentation to a wider audience.

above Definition 2 on page 11) is considered quite small (with respect to the total number of nodes). Secondly, the average inter-connectivity of nodes with common neighbours (that is, the **clustering coefficient** as expressed after Definition 3 on page 12) is high (more so than in random graphs).

With several existing graphs showing similar properties (the examples given in Watts and Strogatz (1998) are neural networks, power-grid structures and collaboration graph of film actors), Watts and Strotgatz have decided to further study this peculiar "group" of graphs and propose a construction method allowing to create new ones. We establish in the following a translation of this creation method using our rule-based formalism. While not truly innovative, this adaptation shows with more details the modularity of our strategic graph rewriting system. Let us first recall the construction method proposed by the authors.

3.2.1 Original algorithm

We remind in the following the model algorithm, exactly as described in its original paper. The model is defined using three essential parameters to describe the network to generate: a number of vertices n, an initial degree k for each vertex in the ring lattice, and a global probability p indicating the chance for each edge to be rewired; the final graph will thus present $\frac{nk}{2}$ edges. The operations to follow have been divided in enumerated steps to allow direct references to them later.

SW.1 We start with a ring of n vertices, each connected to its k nearest neighbours by undirected edges.

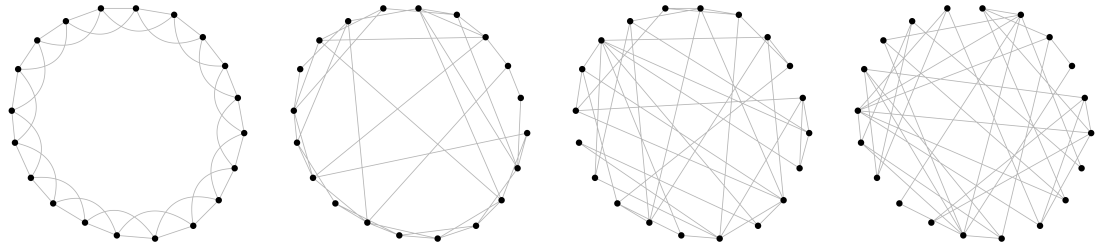
SW.2 We choose a vertex and the edge that connects it to its nearest neighbour in a clockwise sense. With probability p, we reconnect this edge to a vertex chosen uniformly at random over the entire ring, with duplicate edges forbidden; otherwise we leave the edge in place.

SW.3 We repeat this process by moving clockwise around the ring, considering each vertex in turn until one lap is completed.

SW.4 Next, we consider the edges that connect vertices to their second-nearest neighbours clockwise. As before, we randomly rewire each of these edges with probability p, and continue this process, circulating around the ring and proceeding outward to more distant neighbours after each lap, until each edge in the original lattice has been considered once.

Basically, the model is described by its authors as a "random rewiring procedure for interpolating between a regular ring lattice and a random network, without altering the number of vertices or edges in the graph." This statement is illustrated in Figure 3.3 showing a few different examples of networks obtained using the construction model described by Watts and Strotgatz. Depending of the value chosen for the parameter p, the rewiring operation can drastically change the resulting graph.

We complement this historic definition with a few additional details, aiming to alleviate some uncertainties. First, although some specifications are announced considering the values of k and n by the authors, such that $n \gg k \gg ln(n) \gg 1$, no limitation is proclaimed when considering their respective parity. To avoid possible complications appearing when both parameters are odd, we assume k to always be even. Secondly, in steps **SW.2**, **SW.3** and **SW.4** of the model, the authors mention the use of a clockwise iteration across the possible vertices to perform the sought-after transformations. To fulfil this requirement within the translated model, we propose to mark the starting element and use directed edges to allow us to always process the vertices in the same order by following the initial ring of vertices. The edges' orientation can be ignored



(a) $p = 0$ (b) $p = 0.2$ (c) $p = 0.6$ (d) $p = 1$

Figure 3.3: Examples of networks obtained using the construction model described in Watts and Strogatz (1998). Each graph has been generated with the following parameters: $n = 20$ (number of nodes), $k = 4$ (initial degree for each node of the starting lattice and average degree of the final network), and various values of p (specified for each network).

or removed at the end of the generation to obtain a non-directed graph. Lastly, although we use enumerated steps to detail the model, the translation proposed hereafter does not follow them exactly. For instance, as one can see, step **SW.4** is only the generalisation of step **SW.3**, itself being described as multiple ordered applications of step **SW.2**. We thus adapt our description to differentiate the regular ring lattice generation (**SW.1**) from the edge rewiring operations (**SW.2**, **SW.3** and **SW.4**).

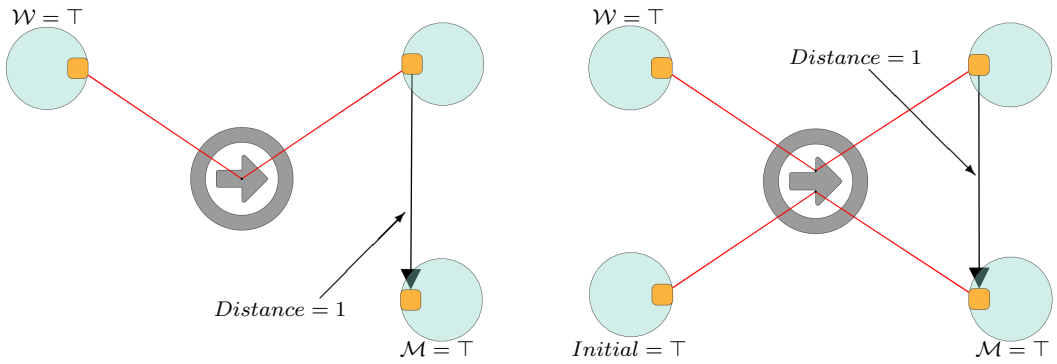
3.2.2 Translation to our rewriting formalisation

While strategic graph rewriting is a very powerful formalism, no implementation could be performed if the needed instructions to complete the program were not available. In such case, that is whenever a part of the algorithm is expressed using a higher form of operation (i.e., more abstract), we only have to break down the operation to low-level instructions, as would be done in any programming language (e.g., using subtractions to obtain a *modulo*, or multiplications to compute geometric series). On this point, the algorithm of the model is already expressed using mostly atomic graph operations, thus simplifying the translation process. The only part which necessitates a more abstract transformation would be the first step as it indicates a ring lattice is required as a basis for the small-world network. We could pass this first step altogether by considering such construction to already be at hand, and only perform transformations to rewire the edges. This circumvention, however, seems hardly fair; we thus need to begin by generating a ring lattice. While quite straightforward, this initial procedure allows us to recall and develop the (visual) syntax used to express rewrite rules as well as the grammar for the strategies.

Along the explanations introducing and detailing the rules and strategies hereafter, we provide an example and follow its evolution step-by-step. For legibility purposes, we limited the size of the network presented to $n = 20$ (number of nodes), and $k = 6$ (where $\frac{nk}{2}$ is the number of edges). We also choose a low value for p to preserve some of the lattice structure and avoid over-rewiring the edges with $p = 0.1$.

Generating a simple directed ring The first stage to construct a ring lattice is the creation of a simple ring. Reusing the parameters existing in the original algorithm to express our method, we need to generate a ring of n vertices (and n edges). This operation can be completed using the rules shown in Figure 3.4 on the next page and the corresponding Strategy 3 on page 51. By convention, the graph initially contains a single node at the beginning of the generation. This

first element is marked using the attribute *Initial* such that the attribute value will be set to $True$ for this starting element and set to $False$ for all the following ones. Initiating the operation using an empty graph would also be possible with the addition of an extra rule creating the first element before any other rule application.



(a) Rule *Add Node*. (b) Rule *Close Circle*.

Figure 3.4: Rewrite rules used for generating a simple directed ring network. Starting from a graph consisting of a single node with an attribute *Initial* set to *True*, Rule a (*Add node*) is applied to successively create new nodes in a chain. Applying Rule b (*Close Circle*) afterwards allows to close the chain by connecting the two extremities.

To achieve the ring construction, we propose to first create a chain of n nodes (and $n-1$ edges). To this end, we use the rule *Add Node* shown in Figure 3.4a which starts from a single node in the left-hand side and adjoins to it a new node when applied. Using *located rewrite rules* in our model, as described in Definition 12 on page 30, we can specify that the sole node in the left-hand side of the rule belongs to the subgraph $\mathcal{W}$. This means that this peculiar node must always be matched to one of the elements existing in the subgraph $\mathcal{P}$. We additionally use the subgraph $\mathcal{M}$, managing the injection of rewritten elements to $\mathcal{P}$, to restrict the element selection for the next rule application. The newly created node is added to $\mathcal{M}$ in this fashion, and will be considered as part of $\mathcal{P}$ in the following rule application. Once the initial node is added to $\mathcal{P}$, *Add Node* can be applied on it at the first rewriting application. Further rewriting operations are also able to succeed as the newly created node, now located at the chain's end, can be selected as a matching candidate in the next application.

Once the appropriate number of nodes has been generated with successive applications of *Add Node* (i.e., the chain is long enough), the rule *Close Circle* (Fig. 3.4b) is used to connect the chain's extremities. The first of the two nodes is simply identified by using the subgraph $\mathcal{P}$ which contains the newest node created by the rule *Add Node*. For the second extremity, the attribute *Initial* is used to identify the very first node which has been used to begin the chain construction. An edge is finally added to connect the two ends together and close the ring. Along with these transformations, an additional attribute named *Distance* is defined in both rules right-hand sides for latter use. For now, its value is initialised to 1 for each edge created using the rules *Add Node* and *Close Circle*.

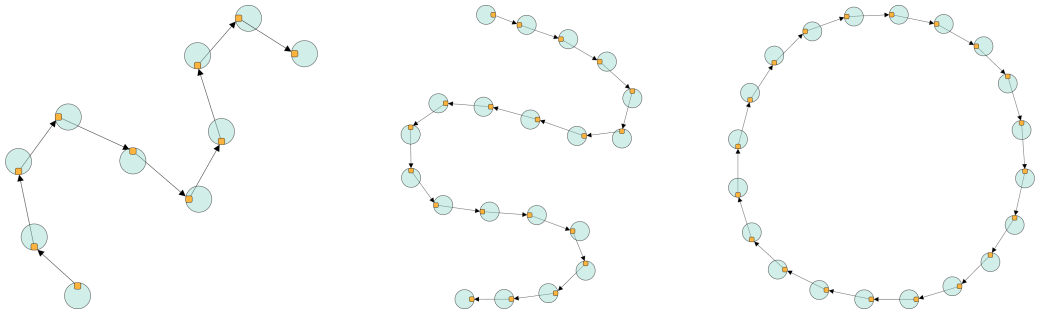
Overall, the keen observer will have noticed right away how all the rules used so far seem to look very much alike (see Figures 2.12 on page 25 and 3.1 on page 44). This comes as no surprise as these transformations allow to perform two basic yet essential operations to generate a new graph: create a new node and link it to an existing one (Fig. 3.4a), and connect two existing vertices (Fig. 3.4b). In this case however, some additional conditions are added to ensure that

all elements subjected to transformations are appropriately selected.

Strategy 3: *Ring creation*: Form a simple ring with n nodes.

```
1 setPos(one(crtGraph));
2 repeat(
3   one(Add Node)
4 )(n − 1);
5 one(Close Circle)
```

Strategy 3 shows how the transformations are ordered and managed. First, an initialisation of the position set is performed (line 1, instruction `setPos`) to select one of the elements existing within the initial graph as a matching candidate, thus adding it to subgraph $\mathcal{P}$. In our case, the sole node existing at the beginning of the generation is chosen. A `repeat` loop is then used to apply rule *Add Node* a total of $n-1$ times (lines 2-4). Each operation adds at the end of the chain a new node, thus creating $n-1$ nodes and $n-1$ edges (see Fig. 3.5a). The single final application of *Close Circle* (line 5) is performed once the expected number of nodes has been created (see Fig. 3.5b). The two ends are then bound together by creating an edge between them (Fig. 3.5c). The strategy application on a graph containing a single element thus successfully constructs a ring of n nodes and n edges.



(a) Example of chain with 10 nodes and 9 edges.

(b) Example of chain with 20 nodes and 19 edges.

(c) Example of closed circle of 20 nodes and 20 edges.

Figure 3.5: Different steps of completion for a simple directed ring network generation with $n = 20$, $k = 6$ and $p = 0.1$.

Transformation in a regular ring lattice Once the ring is formed, the remaining edges must be created to complete the lattice. This construct is achieved by linking each node to its k nearest neighbours (using the ring topology as a basis). To obtain this result, we must use rules connecting nodes at distance $2, 3, \ldots, \frac{k}{2}$ (we recall that k is assumed to be even). Although several rules could be used to achieve this goal, we propose in the following a peculiar solution. Due to the repetitiveness of the task, the different rules used for this step only present minor differences. Thus, instead of detailing each rule we use, we present a template version. The particularity of this rule is that it is expressed according to the value X in the same way that the value of a parameter is for a function.

Figure 3.6 on the next page shows the factorised/template version of such rule. We use the aforementioned attribute *Distance* attached to every edge to indicate the distance between two

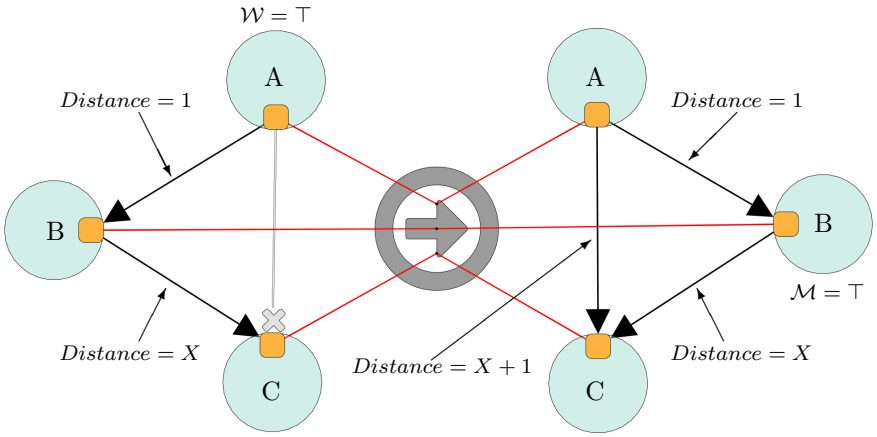


Figure 3.6: Rewrite rule *Create Jump X* to generate a regular lattice on a simple ring. Two elements A and C, located at a distance $X + 1$, previously indirectly connected through an intermediary node B, become attached to one another.

connected nodes with respect to the initial ring topology. As each node is already connected to two neighbours (due to the ring structure), we connect them to their $k - 2$ remaining closest neighbours. Once the rule is applied, an edge between two nodes located at a given distance $X+1$ is created. Within the rule, we express X as a variable which will take in turn the values $1, 2, \ldots,$ up to $\frac{k}{2} - 1$. Thus, for a node A being a direct neighbour of a node B (the edge between A and B is of distance 1), with B itself linking to a node C by an edge of distance X, we connect the nodes A and C with an edge of distance $X + 1$. Appropriately, before applying the transformation, we need to ensure that there is no edge already existing between the two targeted elements (A and C). This peculiar trait is expressed in Figure 3.6 by using an "anti-edge" (displayed in grey) specifying the absence of such edge (see Def. 9 on page 26).

In the same fashion than rule *Add Node*, *Create Jump X* is a located rewrite rule which uses the subgraph $\mathcal{P}$ to manage and steer the matching elements. The first node A is marked as belonging to $\mathcal{W}$, thus imposing the potential match for this element to belong to $\mathcal{P}$. The resulting transformations in the right-hand side show that B is added to $\mathcal{M}$ after the rewriting operation. The element B is thus added to the subgraph $\mathcal{P}$ for the following rule applications. This setup allows us to treat each node according to their creation order as we follow the ring topology (the edge of distance 1 between A and B). It is very easy to see how edges with a *Distance* of $X + 1$ are built based on edges of distance X. The ring structure can thus be used as a base to create edges with a *Distance* of 2. Successive applications of complete round of rules *Create Jump 1*, *Create Jump 2*,..., *Create Jump* $\frac{k}{2} - 1$, will correspondingly create edges of distance 2, 3,..., $\frac{k}{2}$.

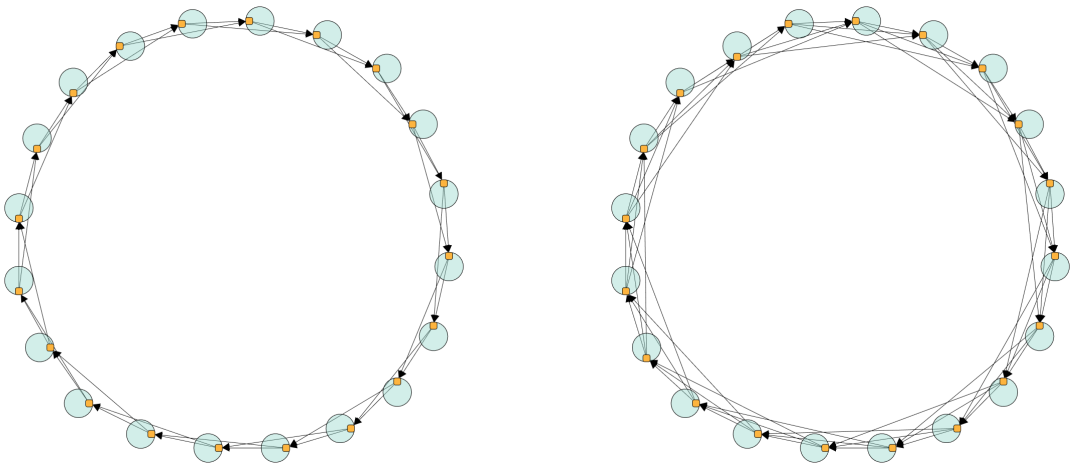
Strategy 4 on the next page exposes how the rules *Create Jump X* are applied for different values of X. The values taken by X need to follow an incremental order to allow the rules to build upon the previously created edges. After an initialisation of the position set[4] selecting the initial node (line 1), a first `repeat` loop, applying as many times rule *Create Jump 1* as possible, results in the creation of all possible edges with a *Distance* of 2 (lines 2-4, see Fig. 3.7a). Rule *Create Jump 2* can then use edges with a *Distance* of 2 to create every possible edge with a *Distance* of 3 (lines 5-7). This goes on until the application of *Create Jump* $\frac{k}{2} - 1$ has created all

[4]This initialisation is optional if the same $\mathcal{P}$ and $\mathcal{Q}$ subgraphs are reused from the previous strategy.

Strategy 4: *Weaving the mesh*: Insert "shortcut" edges between neighbours at distance up to $\frac{k}{2}$.

```
1  [setPos(one(property(crtGraph, node, Initial == "True", )));]
2  repeat(
3    one(Create Jump 1)
4  );
5  repeat(
6    one(Create Jump 2)
7  );
8  [...]
9  repeat(
10   one(Create Jump k/2 − 1)
11 )
```

edges with a *Distance* of $\frac{k}{2}$ (lines 9-11, see Fig. 3.7b). This particular sequence of steps allows us to ensure that the closest neighbours are the first to be connected together, and that any given node is eventually connected to k neighbours by the end of the strategy's application.



(a) Ring lattice in an intermediary state of completion for $X = 1$.

(b) Achieved ring lattice with n vertices and $\frac{kn}{2}$ edges ($X = 2$).

Figure 3.7: Different steps of completion for a regular ring lattice generation with $n = 20$, $k = 6$ and $p = 0.1$.

Rewiring the edges The two previous strategy applications have resulted in the creation of a regular ring lattice with n vertices, each with a degree of k. While maybe rather indirect and only achieving the first step (**SW.1**) according to the small-world network generation model, this introductory construction establishes some basis we can use from now on, such as the $\mathcal{W}$ and $\mathcal{M}$ subgraphs, matching using "anti-edges" or describing rules as templates. We can now focus on the second step (**SW.2**) of the algorithm described in section 3.2.1 on page 48 to perform an

edge rewiring.

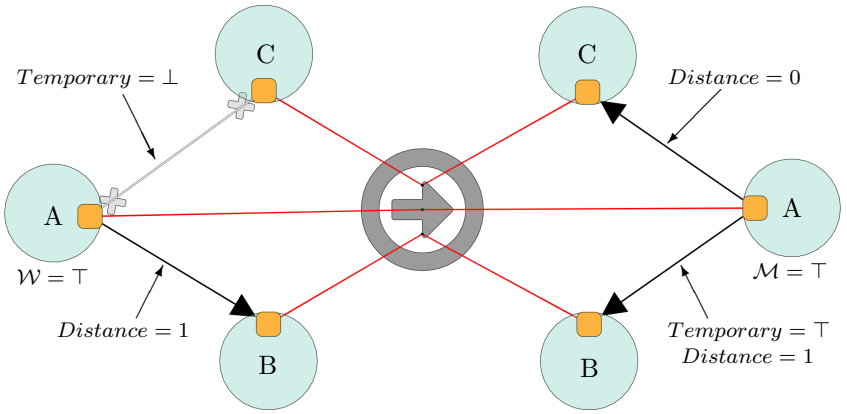
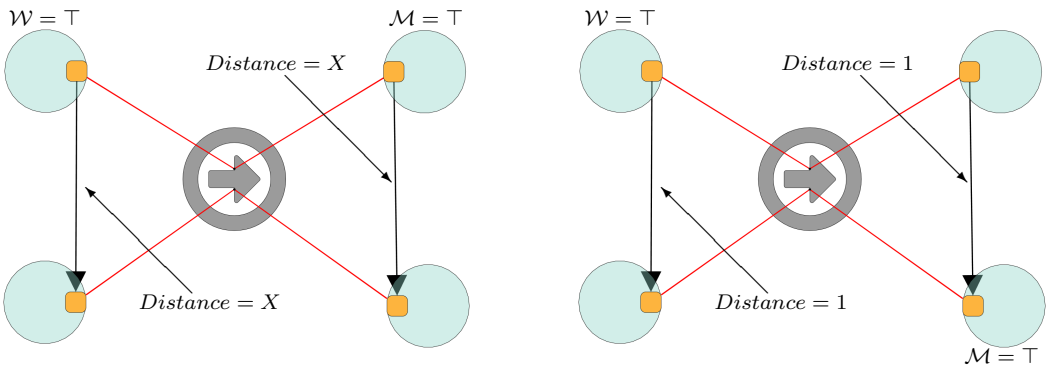


Figure 3.8: Rewrite rule *Perform Rewiring 1* to rewire edges between nodes at distance 1. The edge between two nodes A and B, previously indirectly connected, is rewired to connect A to C. A temporary edge between A and B is created to preserve the initial ring structure.

First, a vertex and the edge connecting it to "its nearest neighbour in a clockwise sense" (i.e., using the ring as a basis) are focussed on. According to the original model, two different scenarios can occur: either the end of the selected edge is reconnected to a vertex chosen at random while avoiding multiple edges, or it is left in place as it is, untouched. Both transformations are presented hereafter using either the rule *Perform Rewiring* or the rule *Leave Unchanged* respectively. Just like encountered previously in rule *Create Jump* (Fig. 3.6 on page 52), we introduce template versions of the rules using a variant parameter X which will take in turn for value $1, 2, \ldots$ up to $\frac{k}{2}$. However, for this step, the first iteration of *Perform Rewiring* (i.e., rule *Perform Rewiring 1* presented in Figure 3.8) is slightly different from its following applications (i.e., rule *Perform Rewiring X*, shown in Figure 3.10 on page 56). The reason for this irregularity is quite simple: as the model specifies in **SW.3**, the process must be repeated on each node by moving clockwise around the ring. The solution we selected for fulfilling this requirement is to use the initial ring structure as a basis to find the following node on which to apply the next rule (either *Perform Rewiring* or *Leave Unchanged*). We thus use the edges with an attribute *Distance* equal to 1 to change the node currently focussed on using rule *Next Node*, shown in Figure 3.9b on the facing page. The main issue with this peculiar solution is the behaviour to adopt when the edges on the initial ring structure are rewired and can thus no longer be used for this design. This concern is addressed by creating in their stead transitory edges marked using a new attribute named *Temporary*. This creation step can be seen in rule *Perform Rewiring 1* (Fig. 3.8). Once all the steps of the model are completed, these substitute edges are discarded using rule *Clean Edge* visible in Figure 3.11 on page 57. With this overall presentation finished, let us take a closer look at each of the rules used in this step of the model.

According to **SW.2**, we must start with a node and the edge connecting it to its closest neighbour (i.e., at a distance of 1). With a probability p, rule *Perform Rewiring 1* (Fig. 3.8) is applied to the selected node (identified as A), its neighbour (B) and a third node which will be used as the new end for the rewired edge (C). We can see that several specificities are given for the matching to be successful. First, the selected node (A) in the left-hand side of the rule is a part of the $\mathcal{W}$ subgraph which means that its matching candidates must belong to the $\mathcal{P}$ subgraph. Secondly, because *Perform Rewiring 1* is supposed to rewire the edges connecting

the closest nodes, the attribute *Distance* of the edge linking A to B is valued at 1. Thirdly, A and C must not be already connected to one another. We thus use two "anti-edges" (one for each possible edge orientation) to convey this predicate. Note that the existence of potential *Temporary* edges is plainly ignored as they will be removed in the final steps of the generation. The resulting transformation leaves us with the selected node (A) being now connected to both its original neighbour and the third node (respectively, B and C). The selected node A is still focussed on and is thus added to the $\mathcal{M}$ subgraph. The edge found in the left-hand side is rewired in the right-hand side to connect A to C. The attribute *Distance* for this edge is set to 0 as, with C being randomly chosen, we do not know what the distance between the two elements actually is. Finally, a new *Temporary* edge is created between A and B and its attribute *Distance* is set to 1 with respect to the original ring structure.



(a) Rule *Leave Unchanged X*.

(b) Rule *Next Node*.

Figure 3.9: Rewrite rules *Leave Unchanged X* to apply when the edge is left in place, and *Next Node* to hop focus from one node to another. Rule a (*Leave Unchanged X*) is a template rule applied whenever the edge of *Distance X* attached to the node in focus is not to be rewired. As one can see, no topological transformation or change in the attributes' value are performed and the same node remains in focus at the end of the rule application. Rule b (*Next Node*) may appear very similar to the first rule, however a few differences exist. With respect to the element candidate to the matching, the edge considered by this rule must belong to the initial ring structure (*Distance* $= 1$). No real transformations are performed *per se*, however the node in focus is different by the end of the rule application.

The second option presented in **SW.2** occurs with a probability $1 - p$; in such case, the selected node and edge are left intact and no rewiring occurs. Rule *Leave Unchanged X* shown in Figure 3.9a as the template version of the rule, is thus applied with $X = 1$ in the case of **SW.2**. There is little to say about this rule: the topology of the matching subgraph is unaffected, the value for the attribute *Distance* is passed from the left-hand side to the right-hand side, and the element focussed on (belonging to $\mathcal{W}$) remains so at the end of the application (by using $\mathcal{M}$). Despite its apparent purposelessness, this rule allows an accurate visual description of the operation demanded from the algorithm, furthermore, it is also necessary due to the strategy instruction `ppick` used to manage the probabilistic rule applications (more details in Section 2.3.2 on page 32).

Using either rule *Perform Rewiring 1* (with a probability p) or rule *Leave Unchanged 1* (with a probability $1 - p$) achieve the step **SW.2** of the generation model. The following operation, **SW.3**, aims at repeating step **SW.2** for all the nodes in a clockwise order until one complete lap around the initial ring structure is achieved. As the two previous rules perform their respectively

assigned transformations, some of the rewritten elements, and more particularly the node initially focussed on, are passed to the subgraph $\mathcal{P}$ by affecting the right-hand side element to the subgraph $\mathcal{M}$. We thus use rule *Next Node* presented in Figure 3.9b on the preceding page to select the node focussed on and its closest neighbour according to the initial ring structure (whether the edge connecting them together is *Temporary* or not), and change the focus to the second node. This rule application, which allows us to effectively jump from one node to its neighbour located at a distance of 1, is to be performed after a probabilistic application of the rules *Perform Rewiring 1* and *Leave Unchanged 1*. Repeating this set of operations n times leads us to achieve a complete lap of the initial ring structure, thus fulfilling step **SW.3**.

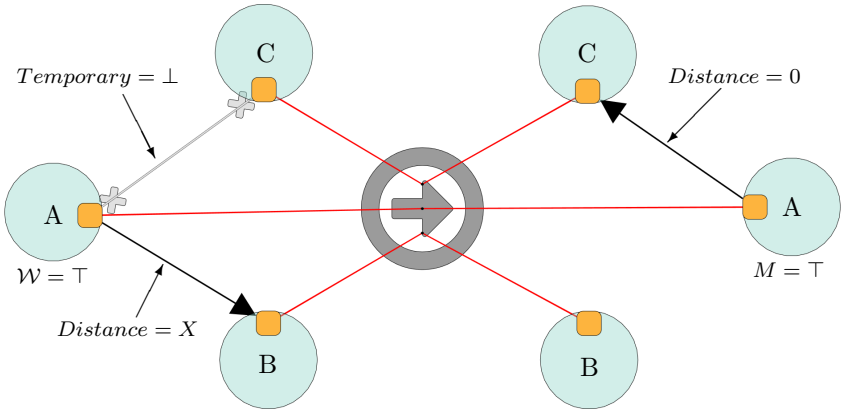


Figure 3.10: Rewrite rule *Perform Rewiring X* to rewire edges between nodes at distance X. In this template rule, an edge between two nodes A and B, located at a distance X and previously indirectly connected, is rewired to connect A to C. Unlike rule 3.8 (*Perform Rewire 1*), no temporary edge between A and B is created.

The final step described in **SW.4** intents to generalise the treatment proposed above using this time neighbours at distance X where X is valued 2, 3, ..., up to $\frac{k}{2}$. The operations to carry out are thus very similar to the ones presented previously: either a probabilistic rewiring is applied on the edge connecting a given node and its neighbour located at a distance X or no transformation occurs. Furthermore, the elements are still considered following a clockwise order using the initial ring structure as a basis. While the template version of the rule performing no modifications (rule *Leave Unchanged X*) has been already introduced, its counterpart, applying when the edge is to be rewired (with a probability p), has yet to be describe. We thus introduce rule *Perform Rewiring X*, presented in Figure 3.10. This template version of the rule is to be applied for any X different from 1. The rule itself is very similar to *Perform Rewiring 1* as it identifies three nodes (A, B, and C), two of which are mandatorily connected (A and B), with the third node (C) not being directly connected to the node in focus (A). As seen previously, the non-existence of the edge is verified using "anti-edges" (for which *Temporary* edges are still ignored). Once the rule is applied, the initial edge (between A and B) is rewired and used to connect the node focussed on (A) with the third identified node (C), however, no *Temporary* edge is created at this point. As before, we use the attribute *Distance* as an additional condition when searching for potential candidates to ensure that the selected edge is among the ones intended to be rewritten; the attribute's value of the rewired edge is also set to 0 to indicate the distance measured is no longer accurate. The second rule, applied on the selected elements when the rewiring is not performed, is still the same rule *Leave Unchanged X* (Fig. 3.9a on page 55) which

we introduced earlier. And just like before, we also use rule *Next Node* (Fig. 3.9b) to hop focus from one node to the next according to the initial ring structure.

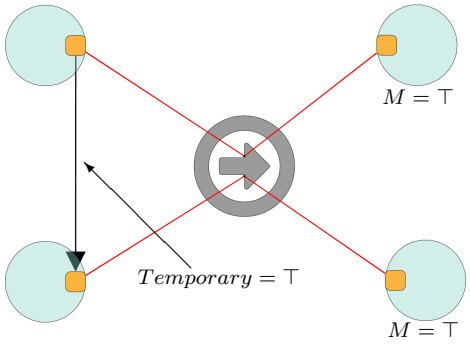


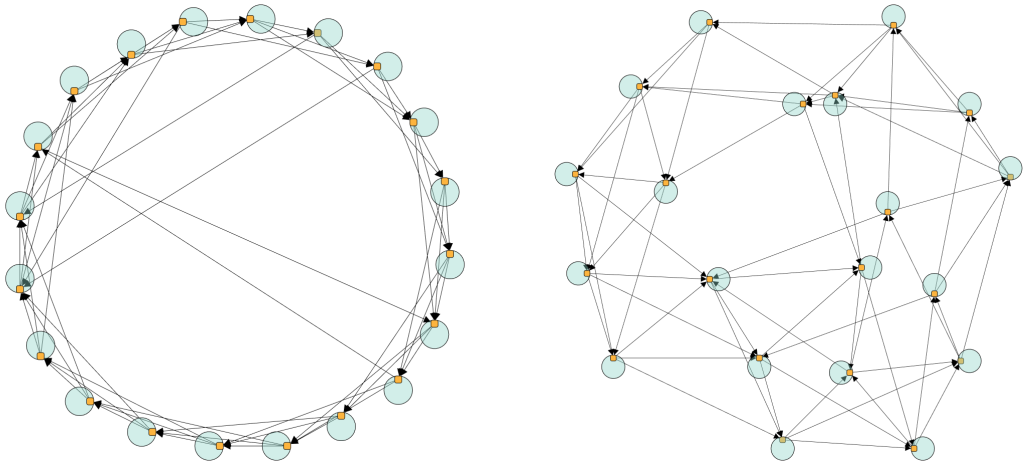
Figure 3.11: Rewrite rule *Clean Edge* to remove any *Temporary* edge from the final network. A temporary edge is selected and deleted from the graph. Subgraph $\mathcal{W}$ is not used in this rule as the operation can potentially concern every node. Each node is reinserted into the subgraph $\mathcal{P}$ through $\mathcal{M}$ to keep them available for further operations.

Finally, the last rule left to introduce to complete the model is shown in Figure 3.11. Rule *Clean Edge* is used to delete any *Temporary* edge existing in the graph. Its application must be repeated for as many time as possible to ensure that all of these edge are removed from the final network. As the rule can be applied on any suitable edge, indifferently of the order followed, the left-hand side nodes are not submitted to any peculiar constraint. The rewritten elements in the right-hand side however must be reinserted into the subgraph $\mathcal{P}$ for any potential following transformation.

Strategy 5: *Rewiring the edges*: Rewire the edges linking nodes at distance up to X on the regular ring lattice.

```
1  [setPos(one(property(crtGraph, node, Initial == "True", )));]
2  repeat(
3    ppick(one(Perform Rewiring 1), p, one(Leave Unchanged 1), 1 − p);
4    one(Next Node)
5  )(n);
6  repeat(
7    ppick(one(Perform Rewiring 2), p, one(Leave Unchanged 2), 1 − p);
8    one(Next Node)
9  )(n);
10 [...]
11 repeat(
12   ppick(one(Perform Rewiring k/2), p, one(Leave Unchanged k/2), 1 − p);
13   one(Next Node)
14 )(n);
15 setPos(all(crtGraph));
16 repeat(
17   one(Clean Edge)
18 )
```

The rules described above can now be ordered in a strategy managing the rewiring transformations. Presented in Strategy 5, this last set of instructions allows us to achieve the steps described in the points **SW.2**, **SW.3** and **SW.4** of the model. The first operation consists in setting up the subgraph $\mathcal{P}$[5] using a `setPos` instruction (line 1). We add to it the node initially present at the beginning of the graph generation, recognisable by its *Initial* attribute's value set to *True*. We then fulfil point **SW.2** by applying either *Perform Rewiring 1* or *Leave Unchanged 1*, respectively with a probability p and its complement $1 - p$ (line 3), pass to the next node by following the ring structure (line 4), and repeat the process n times, that is for all nodes, as expressed in **SW.3** (lines 2–5). This first `repeat` loop takes care of all the edges whose attribute *Distance* is equal to 1. The probabilistic rule application is taken care of by using a `ppick` instruction, which applies rules with respect to a given probabilistic distribution (see Section 2.3.2 on page 32 for further details). To satisfy **SW.4**, we duplicate the operation for the next possible value of the attribute *Distance* (e.g., see lines 6–9 for *Distance*$= 2$) and reiterate the process until the maximum value is reached: when *Distance*$= \frac{k}{2}$ (lines 10–14). We finally need to remove the supernumerary edges marked as *Temporary* to achieve the generation process. Considering the operation is to be applied on to the whole graph, we select all the elements and add them to subgraph $\mathcal{P}$ (line 15). We then repeatedly apply rule *Clean Edge* as many times as necessary to get ride of any unwanted *Temporary* elements.



(a) Small-world network obtained using the translated model.

(b) Same final network drawn with a force-directed layout.

Figure 3.12: Different layouts of an achieved small-world network generation with $n = 20$, $k = 6$ and $p = 0.1$.

The result of the generation process described using those three strategies can be seen in Figure 3.12. The two represented graphs are the same only using two different layouts. In the end, the product obtained with our example seems to respect the expected specifications of a small-world network. While generating several graphs with different parameters and comparing them to the original model could be achieve to show the statistical equivalence of the two methods, using the strategic graph rewriting formalism allows us to take another path.

[5]The subgraphs $\mathcal{P}$ resulting from the previous strategy application can also be reused.

3.2.3 Formal validation of the translated model

Although the strategies and rules describing the model seem to create an appropriate network, we need to ensure that the resulting generation is conform to the initial model and the final network is as specified. Thankfully, and due to the fine control over the rules and on how they are applied, the strategic graph rewriting formalism enables to formally verify that a strategy execution is producing the expected transformations.

Hereafter, we enounce three propositions, and provide their respective proofs, stating our proposed translation successively respects the specificities of the small-world generator model as described in Watts and Strogatz (1998). Each proposition corresponds to one of the different steps described in the sections above: **Generating a simple directed ring**, **Transformation in a regular ring lattice** and **Rewiring the edges** (respectively, Strategies 3, 4, and 5).

Proposition 16. *Given a positive integer parameter n, the strategic graph rewriting transformation as indicated by the rules in Figure 3.4 and Strategy 3 terminates and always successfully generates a directed ring structure with n nodes (and n edges).*

Proof. Of all the instructions in Strategy 3, only the `repeat` loop can be used to generate an infinite derivation trace. The termination property is however guaranteed due to a limit on the number of iterations set to $n-1$ (i.e., it is a bounded repeat).

The strategy is applied on a graph containing a single node with an attribute *Initial* set to *True*. This single element is always successfully selected as no parameters or conditions are specified to the `setPos` operation. Rule *Add Node* (Fig. 3.4a) is applied $n-1$ times. Each application considers the node focussed on, adds a new node and creates a directed edge going from the focussed node to the newly added node. This new node is then put in focus instead of the pre-existing node and the repeat instruction continue. This loop effectively forces the creation of a directed chain of n nodes and $n-1$ edges, starting with the *Initial* node. Rule *Close Circle* is applied a single time selecting the end of the chain and the *Initial* node, and creates an edge going from the former to the latter. The resulting graph is thus always a directed ring of n nodes and n edges □

Proposition 17. *Given an even positive integer parameter k, the strategic graph rewriting transformation as indicated by the rule in Figure 3.6 and Strategy 4 terminates and always successfully transform a directed ring structure, with n nodes and n edges, into a directed regular ring lattice, with n nodes of degree k and $\frac{nk}{2}$ edges.*

Proof. Strategy 4 uses $\frac{k}{2}-1$ unbounded `repeat` loops. While no explicit limits in the number of repetitions is given for each loop (although, it would be acceptable), the application of rule *Create Jump X* can only be performed a certain number of time. Indeed, as the rule is applied, new edges (whose attribute *Distance* is valued $X+1$) are created. Once a complete loop around the initial ring structure is achieved, the matching conditions given for the left-hand side can no longer be met due to the "anti-edge" definition between A and C. The same behaviour occurs with whichever value of X, thus guaranteeing the termination of the strategy.

The strategy is applied on the directed ring previously obtained composed of n nodes and n edges. Furthermore, one of the nodes is marked as being the *Initial* element, while the attribute *Distance* is set to 1 for all edges. The first instruction sets the *Initial* node into focus by adding it to subgraph $\mathcal{P}$; this operation always succeeds. Successive applications of rule *Create Jump X* (Fig. 3.6) are then performed using several `repeat` loop with variant X values. During the transformation, the node focussed on (A), its direct neighbour (B) and its neighbour's neighbour (C) are selected. For each case, nodes A and B are at distance 1 while nodes B and C are at distance X. Each rule application adds an edge between A and C of distance $X+1$ and change the

focus from A to B, hopping to A's next direct neighbour, thus effectively creating every possible edge of distance $X+1$ (with respect to the orientation). This leads each node to only possesses a single out-going edge for each given value of X. As explained above, once a complete circuit around the ring structure has been performed, the verified existence of an element matching with the "anti-edge" between A and C is the only reason which could lead a rule to fail. The resulting termination leads to the closure of the current `repeat` loop and the beginning of the next one (or the end of the strategy). Each loop thus applies rule *Create Jump X* n times and consequently creates n new edges of distance $X+1$. As X is successively valued at 1, 2, ..., up to $\frac{k}{2}-1$ with each passing `repeat` loop, $\frac{nk}{2}-n$ new edges are created overall, pushing the total number of edges to $\frac{nk}{2}$. The resulting graph obtained at the end of Strategy 4 contains n nodes, $\frac{nk}{2}$ edges, and is always a regular directed ring lattice as expected in **SW.1**. □

Proposition 18. *Given a positive rational parameter p (with $0 \leq p \leq 1$), the strategic graph rewriting transformation given by the rules in Figures 3.8, 3.9a, 3.9b, 3.10, and Strategy 5 terminates and always rewires a regular directed ring lattice's edges according to the specificities of the small-world model described in Watts and Strogatz (1998).*

Proof. Strategy 5 uses $\frac{k}{2}$ bounded `repeat` loops performing n reiterations and a single unbounded loop. The termination of each bounded loop is guaranteed as they will by definition come to an end once their allocated number of repetitions has been reached. The unbounded `repeat` loop is also unable to keep going on forever as the rule being call, *Clean Edge* (Fig. 3.11), can only be applied a limited number of times. Rule *Clean Edge* needs a *Temporary* edge to successfully be applied, which can only be created in rule *Perform Rewiring 1* (Fig. 3.8), itself being by extension only applied a limited number of time as the rule is solely called in the first bounded `repeat` loop of the strategy. As rule *Perform Rewiring 1* can only be applied a maximum of n times (when $p = 1$), rule *Clean Edge* will be applied a maximum of n times as well. The strategy is thus guaranteed to terminate.

Like previously, the first rule allows to set the *Initial* node into focus and always succeeds. Each of the `repeat` loop will first either apply rule *Perform Rewiring X* (Fig. 3.10), with a probability p, or rule *Leave Unchanged* (Fig. 3.9a), according to **SW.2**, followed by an application of rule *Next Node* (Fig. 3.9b).

Perform Rewiring In the first loop, for $X = 1$, applying rule *Perform Rewiring 1* results in a slightly different transformation than the one obtained from the following applications with rule *Perform Rewiring X* (where $X > 1$). Indeed, during that first iteration, a *Temporary* edge is created in place of the rewired edge to preserve the initial ring structure, thus still allowing us to visit all the nodes in the same order as previously achieved in the preceding strategies. In all *Perform Rewiring* variants, the node in focus (A), at first connected to its neighbour (B), becomes connected to another node (C) at the end of the rule application. As specified by the "anti-edge", A and C are not connected before the transformation, thus avoiding edge duplication. Although the matching candidate for A is selected only if belonging to subgraph $\mathcal{P}$ and B is also chosen under certain conditions (being A's neighbour), C on the other hand is chosen uniformly at random among all the remaining nodes. As the transformation occurs, the attribute *Distance* given to the rewired edge becomes invalid and is set to 0 while, the focus remains on A. In the case of rule *Perform Rewiring 1*, the *Temporary* edge's *Distance* is set to 1.

Leave Unchanged In each `repeat` loop, whenever the edge is not rewired, rule *Leave Unchanged X* is applied instead. The rule does not change anything and also keeps the focus on the same node.

Once either of these two rules is applied, rule *Next Node* is used to select the node in focus and its closest neighbour (using the ring structure as a basis). The focus is then shifted to the neighbour and the loop restart, thus effectively subjecting successively each node in turn to the process until one complete lap around the ring has been achieved, as specified in **SW.3**. Changing the attribute *Distance* used during edge matching allows us to start with the closest nodes and iteratively consider neighbours farther and farther apart. As each node only possesses a single out-going edge for each given value of X, and each node is always considered once within each `repeat` loop (imposing iteratively increasing values for X), each edge is guaranteed to be considered for rewiring once. With this final point fulfilling **SW.4**, our translation accurately respects the specificities of the small-world model generation. □

With each of the above proposition proved, our translation of the small-world generation model of Watts and Strogatz (1998) is finally complete. Adapting this first model to our formalism was not as straightforward as simply following the original algorithm. The dissimilarities between the generation steps initially proposed and the ones we finally used to construct the small-world network hint at some of the preliminary work needed to rearrange the transformations. More generally, this process is not entirely unusual as a very similar course would be performed by anyone looking to implement the same model into any programming language. Indeed, any given implementation is rarely a literal translation of the description of its respective model. For instance, some algorithmic improvements can be inserted to improve the time or spatial complexity, or sometimes, as in our case, adaptations due to the chosen language's graph-oriented formalism can be necessary.

3.3 Introducing a new social network generative model

In this section we address the second approach to graph generation as described at the beginning of the chapter. Whereas the model translated above is plainly designed to recreate the specificities encountered in small-world networks, we propose in the following to use a different approach by adopting the point-of-view of a bystander observing the construction of a network. Several different types of networks have been mentioned earlier such as a collaboration graph, or a neural network, but in this case, we propose to direct our attention toward online social networks. While being a complete research area on their own (Carrington et al., 2005; Newman et al., 2006; Scott and Carrington, 2011), online social networks have gathered much interest in the past fifteen years thanks to the wide public adoption of online social networking services such as Facebook[6], Twitter[7] or LinkedIn[8].

Our approach is designed around a simple scenario conceived to imitate the construction process encountered in real-world social networks. Basically, a social network is initiated by a single individual. This person can decide to reach to other people and include them into the social network or other persons can join on their own accord to connect with some of the people already present within. Whenever new users first join the social network, their number of acquaintances is thus very limited. We simplify this situation by connecting new arriver solely to the user who have lead them to join the network in the first place. After a time, individuals may connect to some of the people in the social network they already know. These preexisting relations are due to connections occurring outside the current network such as family ties, friendship, work-related connections, relations existing in other (online) social networks... This leads to

[6] https://newsroom.fb.com/company-info/
[7] https://about.twitter.com/company
[8] https://press.linkedin.com/about-linkedin

the creation of new connections within the network which may sometime seem random for any spectator only aware of the present social network, and therefore lacking the more global picture. Finally, individuals get to know the people with whom they are sharing friends in the network, potentially leading to the creation of new connections.

Our goal down the road is of course to design an algorithm to generate networks based on the above scenario with our strategic graph rewriting formalism. To be entirely thorough, some precisions are added to complete the description. First, as in the previous generation model, we work with a simple directed port graph. This time however we intend on using the edges' orientation to convey meaning instead of solely simplifying some of the rewriting processes. In most real-world social relations, two persons relate to each other with a simple mutual recognition. This is the case for instance in collaboration and communication networks (e.g., co-authorship relations from DBLP[9] in Yang and Leskovec (2012), or Enron[10] email communications in Leskovec et al. (2008)). However, some social networks present an asymmetric model of acknowledgement. While the most popular of them online is Twitter, classifying one of the users as a follower while the other is a followee, these connections can also be graded to express additional nuances if need be (e.g., votes in Wikipedia's request graph for adminship[11] in West et al. (2014) or trust ratings in diverse signed networks in Leskovec et al. (2010b)). Such relations can be very simply illustrated using directed edges, where a relation maintained by an individual n toward another individual n' is depicted by a single (labelled) edge going from the node representing n to the one representing n'. Secondly, the number of elements (nodes and edges) to create during the network generation is decided *a priori*, before the operations begin. We thus introduce $|N|$ and $|E|$ as, respectively, the total number of nodes and edges existing in the graph once the generation is over. The resulting number of communities to appear in the final graph is left to be decided during the generation process according to the enforced rewrite rule applications. Thirdly, in our scenario, individuals are mandatorily introduced into the social network by one individual already present within. Considering we start the network with a single user, each node will be connected to the same reachable subgraph leaving us with only one connected component.

To respect the three points above, we need to impose some limits concerning the inherent values $|N|$ and $|E|$. For one, the resulting network being a connected component, the minimum number of edges must be such that every node is linked to the principal component, thus $|E| \geq |N| - 1$. The maximum number of edges is also imposed due to the form of the final network we wish to obtain. In our case, a connection going from an individual n to another person n' should only exist once; we thus wish to obtain a simple directed (port) graph. As any ordered pair of nodes (n, n') can only be linked once, the maximum number of edges appears when the network is in a simple directed clique configuration, i.e., when $|E|_{max} = |N| \times (|N| - 1)$.

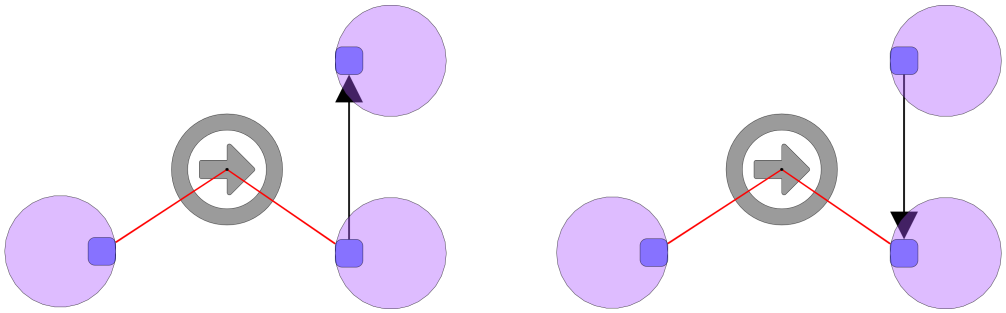
We present below the algorithm for social network generation following the aforementioned scenario. We use rewrite rules inspired from the above description to generate the graph in three phases: new users arrival (node generation, Sect. 3.3.1), introduction of connections established outside the social network (edge generation, Sect. 3.3.2), and connection between users sharing common acquaintances within the network (community generation, Sect. 3.3.3). This rough imitation of processes existing in real-world (online) social networks generates interesting networks with properties similar to those of small-world networks; for more details, the complete analysis of the model is available in Appendix C on page 195. Along the description of the algorithm, we provide an example of step-by-step graph generation coinciding with the rules and strategies introduced. The deployed example is set to generate a small graph of 20 nodes and 100 edges, the concerned figures provide additional information. This section is an extended version of the

[9] http://dblp.uni-trier.de/
[10] https://www.cs.cmu.edu/~enron/
[11] http://snap.stanford.edu/data/wiki-RfA.html

main example proposed in Fernández et al. (2016b).

3.3.1 Generation of a simple directed acyclic port graph

The first phase in the construction of the network uses the two rules shown in Figure 3.13. Contrary to the small-world model presented earlier in this chapter, the following rewrite rules do not use any sort of attribute for either identification or computation. The rewriting operations are solely based on the topological properties of the element to target, jointly with some management of the $\mathcal{P}$ and $\mathcal{Q}$ subgraphs to filter in or out the needed elements. The reintroduction of the elements into subgraph $\mathcal{M}$ in each rule, and consequently in $\mathcal{P}$ for the next rewriting application, is also no longer specified as the process is performed each time for every element. Both rules apply to a single node and generate two linked nodes, thus each application increases by one the total number of nodes and edges. The only difference between these two rules lies in the edge orientation as Rule 3.13a creates an outgoing edge on the initiating node, while Rule 3.13b creates an incoming edge.



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

Figure 3.13: Rules used for generating and attaching nodes to the social network. In both rules, the right-hand side creates an additional node and connect it 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 3.13a or oriented in the opposite direction in Rule 3.13b.

Proposing these two variations allows us to represent new individuals either invited to the network or joining it on their own to connect with some specific person already within. Let us demonstrate the difference with two examples; to this end, we reuse the usual terms employed in Twitter where "follower" describes a person who follows and "followee" designates one who is followed. In Rule 3.13a, someone wishes to follow an individual who is not part of the network; the person is invited, joins the social network and a relation is created going from the initial user –the follower– to the new user –the followee. Rule 3.13b is the opposite story. An outsider wishing to join the social network to follow a person of interest (friend, acquaintance, celebrity...) enters the network by himself and instantly follows the desired person thus creating a relation going from the new user –the follower– to the person previously present –the followee. While using only one of the two rules will produce a directed rooted tree (respectively, either an out- or in-tree), operating both with randomly ordered application generates a more diverse graph. The underlying skeleton of the graph remains still a simple acyclic tree.

The generation achieved during this first phase is managed using Strategy 6. It repeatedly applies the generative rules $|N| - 1$ times, enough for the graph to reach the targeted number of

Strategy 6: *Node generation*: Creating a directed acyclic graph of size N

```
1 setPos(one(crtGraph));
2 repeat(
3 //equiprobable application of the two rules used for generating nodes
4   ppick(one(GenerationNode1), 0.5,
5         one(GenerationNode2), 0.5)
6 )(|N| − 1) // Generation of N nodes
```

nodes specified at the beginning of the construction. As mentioned earlier, each rule application also generates a new edge; this means that once executed, Strategy 6 produces a graph with exactly $|N|$ nodes and $|N| - 1$ edges. While all nodes are similar, the orientation of each edge, as explained above, varies depending of the rule applied for its creation (either 3.13a or 3.13b). The `ppick` construct (lines 4–5) decides which rule to apply and ensures a probabilistic choice between the two rules. The probability distribution can be adapted to favour either one, but, in our case, we present a strategy managing an equiprobable solution.

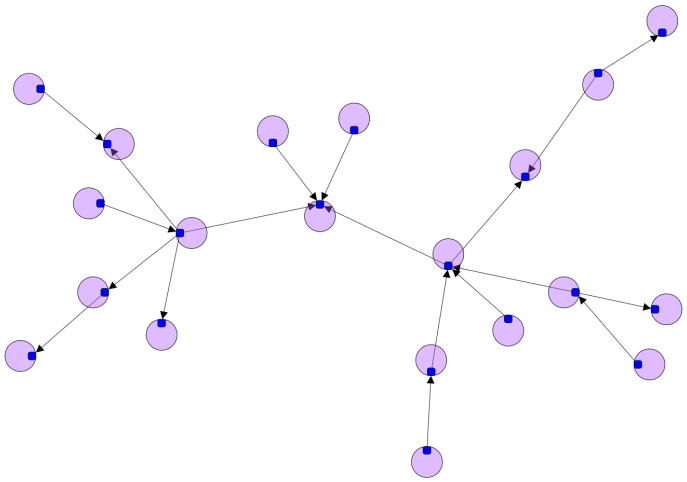


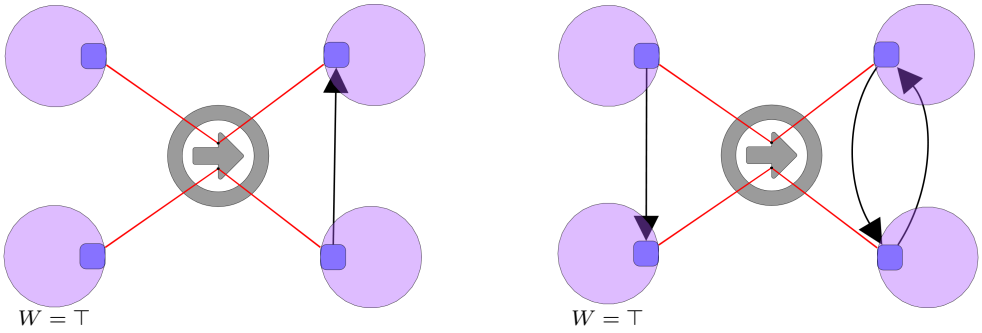
Figure 3.14: Example of resulting graph after applying Strategy 6. While the final graph is expected to have $|N| = 20$ nodes and $|E| = 100$ edges, this step produces a simple directed acyclic graph with $|N| = 20$ nodes and $|N| - 1 = 19$ edges.

Figure 3.14 shows an example of resulting graph obtained after application of Strategy 6 on a single node. As one can see, the edge directions are not consistent and chosen at random depending of the rule applied; as expected, the graph is also simple and acyclic. More generally, this first phase is very important as all the nodes used during the following rewriting operations are created once Strategy 6 is achieved. The graph obtained so far is then used as the backbone of the network we are composing. We focus next on generating additional edges.

3.3.2 Creating complementary connections

During this phase, we either create seemingly random connections between the network users or reciprocate already existing single-sided connections. As explained earlier, this process aims to

emulate the behaviour of individuals within the social network. Once they have joined, users start connecting to people with whom they are already acquainted through other social networks or circles. Any individual can thus initiate a connection with another person, or reciprocate if a connection linking them is already in place. Drawing a comparison with Twitter's actions, a user can, during the current phase, decide to become the follower of a person he has a previous knowledge of, as well as reciprocate a relation in which he is the followee by following the person in return.



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

Figure 3.15: Rules generating additional connections: between two previously unrelated nodes in a, by reciprocating a pre-existing connection in b.

We use two rules to achieve those possible actions, both of which link existing nodes, thus creating a new additional edge with each application. The first rule (Fig. 3.15a) 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. 3.15b) 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. As you can see in the representations, we use the $\mathcal{W}$ subgraph in each rewrite rule to specify the node to select in subgraph $\mathcal{P}$.

In both rules, the existence of edges between the nodes on which the rule applies should be taken into account. Since we aim at creating a simple graph rather than a multi-graph, the rules should not create an edge if a similar one already exists. This could be achieved by adding an "anti-edge" as in the small-world translation given earlier in this chapter. It would then allow us to ensure that the condition "`where not Edge(n,n')`" (see Definition 9 on page 26), forbidding the existence of an edge between the two matching candidates, is met. Here however, we propose a different approach by using position constructs (namely, both the position and ban subgraphs: $\mathcal{P}$ and $\mathcal{Q}$) to restrict the elements to be considered during matching operations. While both approaches achieve the same final goal of ensuring that no edge as specified exists, we believe the "anti-edge" and its visual cue is more legible and easy to understand than going through the whole strategy to follow how the position constructs are managed. The present case is thus only showed as an example of possible fine-tuning considering the rule application.

Strategy 7 handles the rewriting operations. Let us first present how the subgraphs $\mathcal{P}$ and $\mathcal{Q}$ are managed to avoid creating multiple edges. We initially need to filter the elements to consider during the matching. To this end, we randomly select a single candidate (line 3) among all the nodes whose outgoing arity (*OutArity*) is lower than the maximal possible value (i.e., $|N| - 1$). Once added to $\mathcal{P}$, this element, being the only available node, will be mandatorily chosen to replace the node declared as belonging to $\mathcal{W}$ in Rules 3.15a and 3.15b. We then proceed to

Strategy 7: *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'|)
```

ban (through $\mathcal{Q}$) all its outgoing neighbours to forbid them from being considered as potential matching elements (line 5). This way, we ensure that any future rule application will not use those nodes, thus effectively leading the rules, both creating out-going edges, to only apply on pairs of nodes not yet connected. This enables us to keep the graph simple with only one edge at most per ordered pair of nodes (n, n'). For each loop, $\mathcal{P}$ and $\mathcal{Q}$ are completely redefined, respectively with a new node and the set of all its out-going neighbours, to keep them updated.

With $\mathcal{P}$ and $\mathcal{Q}$ set, we once more use a `ppick` instruction to either apply Rule 3.15a or Rule 3.15b (lines 7–8). However, as we wish to guarantee the number of edges generated during the strategy execution, we need to ensure that an edge is created for each loop. In this phase, we desire to create $|E'|$ edges (line 9), where $|E'| \leq (|E| - |N| + 1)$ to remain within the number of edges $|E|$ defined at the beginning of the process. The use of the `orelse` construct (line 7 and 8) allows us to test all possible combinations of rule application, thus, if the first of the two rules can not be applied, the other one is applied in its stead. On the other hand, if neither rule can be applied, then the maximum number of edges in the graph has been reached, i.e., the graph is complete. In the end, if the value of $|E'|$ is not too high, we are left with $|E| - |E'| - |N| + 1$ remaining edges to create in the next phase. As it is the case in Strategy 6, the `ppick` instruction shown in Strategy 7 has an equiprobable outcome; this parameter can nonetheless be changed to transform the network accordingly. Note also that, because each node is randomly chosen among the possible matches when defining $\mathcal{P}$, we do not need to create alternative versions of these rules with reversed oriented edges.

From a graph theory perspective, the current phase simply adds random connections in the network, thus leading the average distance between nodes to diminish (see Section 2.1.1 on page 9). This very process is at the heart of the small-world model presented earlier. Although the model description we propose announces that this phase is intermediary, generating all the edges at this point, such that $|E'| = |E| - |N| + 1$, would result in creating a simple random directed network with a single component. Alternatively, this whole phase could also be ignored if $|E'|$ were set to 0. This unlikely situation could nonetheless happened if none of the individuals in the network were acquainted to someone besides the person which led them to join the network.

Figure 3.16 shows the continuation of our example presented in Figure 3.14. Applying Strategy 7 on the network creates $|E'|$ new edges (for $|E'| = 21$), linking either two previously unrelated individuals or reciprocating an existing connection between them. With the users in our network now connected to their acquaintances, we can forward to the next phase simulating the development of communities.

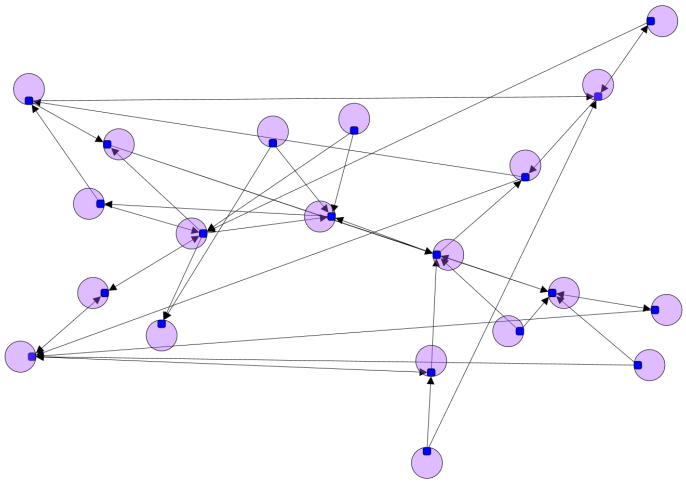


Figure 3.16: Example of resulting graph after applying Strategy 7. The final graph is expected to have $|N| = 20$ nodes and $|E| = 100$ edges. We insert during this phase $|E'| = 21$ new edges. The resulting graph remains a simple directed graph with $|N| = 20$ nodes but with a total number of $|N| - 1 + |E'| = 40$ edges.

3.3.3 Construction of communities using triads

This last phase aims at recreating the establishment of connections as individuals throughout the social network get to know the persons connected to their own acquaintances. This results in generating connections between individuals with a common neighbour. These smaller group of three users connected together are known as triads and can exist in different configurations. The study of triadic relations is the focus of several papers exploring in particular the notion of social balance initially developed by Heider (1946), and later generalised by Cartwright and Harary (1956). The notion is most commonly encountered in signed networks (Leskovec et al., 2010b). In recent years, social balance theory has been shown to be of assistance for signed edge prediction (Leskovec et al., 2010a), filtering edges to ease analysis and visualisation of networks (Nick et al., 2013), and help in person-to-person sentiment analysis (West et al., 2014).

While we are working with a directed network, our generation model does not take into account signed edges. Nonetheless, the configurations of triade relations can be adapted to our case. We present in Figure 3.17 the possible layouts of triads one can encounter in a directed network. When in one of the configurations presented in Figures 3.17a to 3.17d, an edge between two neighbours A and C, respectively coloured in blue and green, can either go from C to A or from A to C. In our case, the set of different possible configurations can be further reduced to four as the blue and green nodes do not need to be distinguished one from another. Indeed, the random selection method we use will identify a given node to either the blue (A), orange (B) or green (C) candidates indistinctively and with equal probabilities. As a result, configurations e and i, f and j, g and l, and h and k are mirrored versions of each other. We thus propose to take a closer look at the last four resulting triads (i, j, k and l) to study the connections established by individuals with a common neighbour.

To make sense of the directed edges, we once more use the terminology of the social networking service Twitter. We begin with Figure 3.17a, where an orange individual (B) follows two persons: A and C (coloured blue and green respectively). We can postulate that, as both individuals are

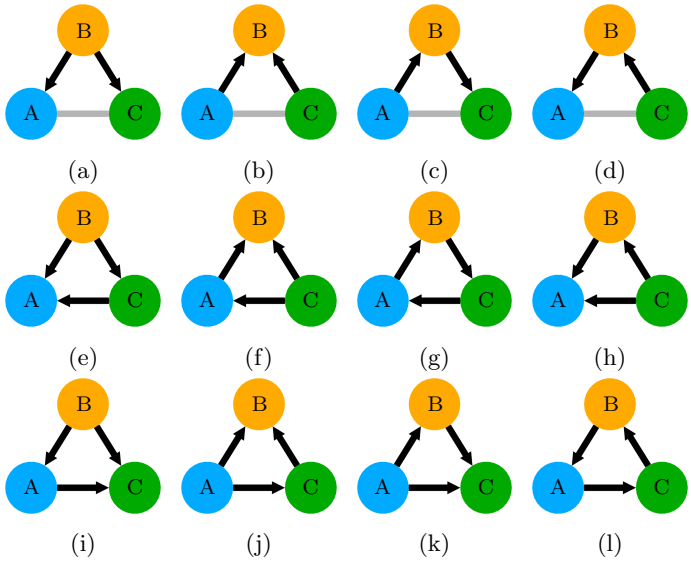
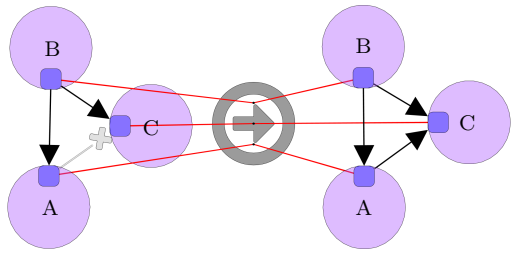


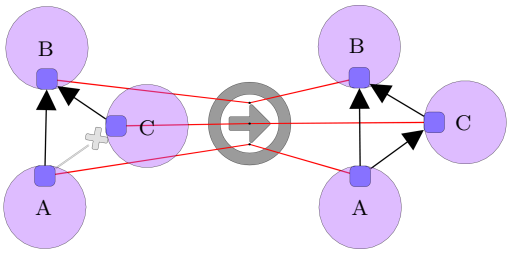
Figure 3.17: Layouts of all possible triadic configurations. The first triad of each column (a, b, c and d) shows the initial configuration with an undefined (grey) edge between nodes A and C. The remaining figures show the triads completed with a new (black) edge going either from A to C or from C to A.

followed by an individual with a set of interests, both A and C may be related through a common subject which is of importance to B. As a result, either A or C could be interested to follow the other, thus leading to Figure 3.17i where a link is created between the two users. Figure 3.17b shows a second possible triad where both A and C follows the same individual B. When in this position, A and C have a common ground and may start exchanging thus creating a link between the two of them as in Figure 3.17j. The next triad is shown in Figure 3.17k. In this configuration, based on Figure 3.17c, a first user A follows a second user B who follows in turn a third user C. This situation produces a transitivity as "the idol of my idol is my idol", meaning that A is much likely to follow C directly. Note that we use here the term "idol" instead of the more classical "friend" as we only consider single-sided relations. The last triad however, as shown in Figure 3.17l and based on Figure 3.17d, seems much less likely to occur to us. Indeed, while the previous triads all show some sort of common ground between the individuals A and C about to be connected, the circling relation between A, C, and B does not show such characteristic. This instinctive supposition has been verified in Leskovec et al. (2010b), where the authors compile and analyse signed relations through triads in social networks. Focussing solely on positive triads, this configuration is shown as the less likely to occur with only 35,093 occurrences out of 797,753 ($\simeq 4.4\%$). While still a possible triad, we decide to ignore this precise configuration to keep the strategy simple and only use the triads described in Figures 3.17i, 3.17j and 3.17k as a basis to establish new connections in the social network.

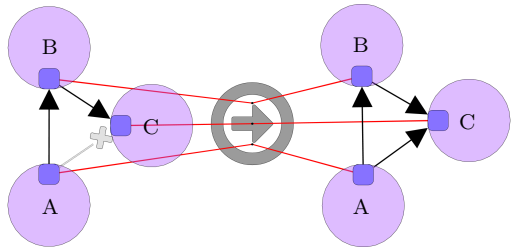
The current phase of the generation algorithm uses the rewrite rules presented in Figure 3.18, each respectively corresponding to the triadic configurations 3.17i, 3.17j, 3.17k, and 3.17l. While only the first three will be used in the upcoming strategy, all the possible rules are shown in the figure. The first triad rule (Fig. 3.18a) shows two individuals (A and C) being followed by a third person (B). The second rule (Fig. 3.18b) depicts an individual (B) being followed by two others (A and C). The third rule (Fig. 3.18c) considers a first person (A), following a second



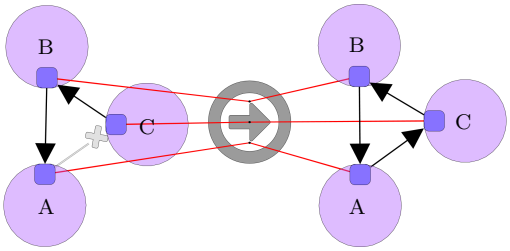
(a) Rule *CommunityDown*.



(b) Rule *CommunityUp*.



(c) Rule *CommunityLegacy*.



(d) Rule *CommunityRotation*.

Figure 3.18: Rules generating additional connections based on triad configurations. Each of the rules correspond to the configurations shown on the last row of Figure 3.17.

one (B), following in turn a third person (C). The fourth rule (Fig. 3.18d), which we will not apply, depicts a circling relation between the three individuals. These rules are very simple and solely use the topological characteristics of their left-hand side subgraph to find corresponding elements to rewrite in the network. We thus work with "anti-edges" (displayed in grey) to ensure that no directed edge already exists where a connection is about to be created. As one can see, no definition of $\mathcal{W}$ is given for any of these rules to specify which element to match in priority to one of the element of $\mathcal{P}$; this leaves each node equally eligible. While not explicitly stated in the figures, all the rewritten elements belong to subgraph $\mathcal{M}$ and are thus reintroduced into subgraph $\mathcal{P}$. This way, $\mathcal{P}$ remains updated and ready for the next rewriting operation without the need to repeat `setPos` instructions in the strategy. In each rule, we have annotated the nodes such that the new edge is always directed from A to C. The decision was an aesthetic choice rather than a structured one. Indeed, the corresponding matching elements are established arbitrarily upon application, thus avoiding the need to propose an alternative version of the rule where the new edge would be going from C to A.

From a graph theory perspective, the enforcement of triadic relations, while being a well-known and studied process as shown above, helps build communities by bringing together and linking individuals previously not connected. This is reflected by the utmost importance of triads when computing the average clustering coefficient of a network (see the description at the end of Section 2.1.1 on page 9 or in Definition 51 on page 195).

We use Strategy 8 to manage the three rules. Because we only use the topological information given by the rules, the interaction with the position subgraphs $\mathcal{P}$ and $\mathcal{Q}$ can be kept to a bare minimum. As a result, we begin the strategy by selecting all the elements in the current graph and add them to $\mathcal{P}$ (line 1). The position subgraph is then maintained by the rules themselves after each application without any involvement of the strategy. While this behaviour is convenient, the rules used in our case always reintroduce all the rewritten elements in $\mathcal{P}$ through $\mathcal{M}$. As all

Strategy 8: *Community generation*: creating edges to strengthen communities

```
1  setPos(one(crtGraph));
2  repeat(
3   ppick(
4     (one(CommunityDown))orelse(
5       ppick(
6        (one(CommunityUp))orelse(one(CommunityLegacy)), 0.5,
7        (one(CommunityLegacy))orelse(one(CommunityUp)), 0.5)
8     ), 1/3,
9     (one(CommunityUp))orelse(
10      ppick(
11       (one(CommunityLegacy))orelse(one(CommunityDown)), 0.5,
12       (one(CommunityDown))orelse(one(CommunityLegacy)), 0.5)
13    ), 1/3,
14    (one(CommunityLegacy))orelse(
15      ppick(
16       (one(CommunityDown))orelse(one(CommunityUp)), 0.5,
17       (one(CommunityUp))orelse(one(CommunityDown)), 0.5)
18    ), 1/3)
19 )(|E| − |E′| − |N| + 1)
```

our rules applications are performed within a `repeat` loop, this means the rewriting operations could go on for quite some time if left unmanaged. Consequently, and because we need to precisely handle the amount of edges created to reach the number agreed on at the beginning of the generation, the loop is bound (line 19) and a single, yet quite complex, `ppick` instruction is then used inside (line 3–18). We reuse here the feat developed in Strategy 7 by using `ppick` and `orelse` instructions to guarantee the number of created elements at the end of the process. To this end, we need to make sure that for every loop, a single edge is added to the network. We thus propose to either apply Rule 3.18a, 3.18b or 3.18c with equal probabilities (lines 4–8, 9–13 and 14–18). However, should the chosen rule be inapplicable (for instance *CommunityDown*, lines 4–8), we can not simply pass to the next iteration. In case of a failed application of its first member, `orelse` applies a second set of instructions, in this case: a `ppick` (lines 5–7). This nested instruction will either try to apply first one the remaining rules, then its counterpart in case of failure, or the other way around, reversing the consideration order of the two rules. The repeat loop is applied $|E| - |E'| - |N| + 1$ times (line 19), with each application creating a new edge, thus reaching the final amount of $|E|$ edges as decided at the beginning of the process (with $|N| - 1$ edges created during the first phase and $|E'|$ edges inserted during the second).

This final phase concludes the generation process. We present in Figure 3.19 the finished graph we used to illustrate the different steps during the generation process. As seen in Strategy 8, the loop is repeated $|E| - |E'| - |N| + 1$ times, with each operation creating a new edge. As, the end of the previous phase, the graph had $|N| = 20$ nodes and $|N| - 1 + |E'| = 40$ edges, this lift the total number of edges to $|N| - 1 + |E'| + |E| - |E'| - |N| + 1 = |E| = 100$ edges as expected. While the limited number of elements proposed in the example served us well to see up-close the possible transformations and potentially follow the evolution of some of the elements, a single generation on a graph with so few elements can hardly validate our rules and strategies. This point is of the utmost importance as we wish to formally guarantee several properties of the final network, especially the number of elements (nodes and edges) generated by the model. We

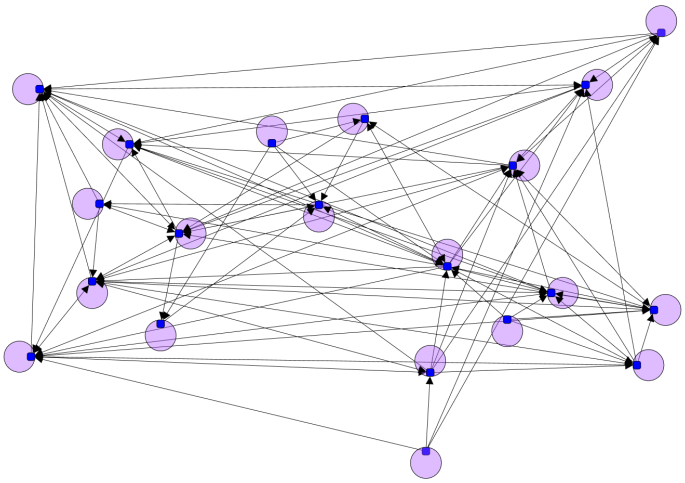


Figure 3.19: Example of resulting graph after applying Strategy 8. The final graph is simple, directed, with $|N| = 20$ nodes and $|E| = 100$ edges. This phase produces the $|E| - |E'| - |N| + 1$ final missing edges.

address this issue hereafter.

3.3.4 Model validation and discussion

As proceeded for the validation of our small-world model translation, we once more use a formal proof to verify whether the proposed rules and strategies respect the present model. A few different decisions have been made on our side when designing the model as we use equal probabilities when applying the rules and have decided to ignore altogether one of the triadic configuration. While these choices are subjective, the outcome of the respective `ppick` instructions should always be the successful creation of an element, thus the following proposition holds.

Proposition 19. *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 *(Strat. 6),* Edge generation *(Strat. 7) and* Community generation *(Strat. 8) 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).

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 7) and the forbidden edges in the rules for *Community generation* (Figure 3.18). 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 $\big(|N|-1\big)+\big(|E'|\big)+\big(|E|-|E'|-|N|+1\big)=|E|$. □

Although we have thus effectively proved that the final network obtained using the rules and strategies presented before is always a simple directed (port-)graph with $|N|$ nodes and $|E|$ edges, a few details need to be discussed. Firstly, the transformations presented above can easily be extended to create graphs with more than one (weakly) connected component. Several approaches are possible to achieve this feature; for instance, one could 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 applications of rules are developed in Fernández et al. (2016a)). Secondly, while we decided to opt for a directed graph when translating the model, adapting the rules and strategies to generate an undirected network instead is quite straightforward. Consequently, only one rule become necessary for each phase (attaching a new node to the connected component, creating a new edge between two disconnected nodes, closing a triad), with several simplifications removing the `ppick` and `orelse` instructions from the strategies. Thirdly, in the last phase, we decided to ignore one of the possible triadic configuration. While this decision was motivated by both our wish to keep the strategy simple enough and by the low probability of appearance of the said configuration, the principles necessary to integrate the corresponding rule (shown in Figure 3.18d) has been explain during the discussion detailing Strategy 8. The considerations about low probabilities of occurrence lead us to the next item. Whereas the strategies presented above make equiprobable choices when pondering upon which rule to apply, customised probabilities may be preferable to obtain a network with characteristics closer to those of real-world situations. For instance, in Strategy 6 handling the node generation, one may consider that new users tend to join a social network mainly because they wish to follow a certain person. In such case, opting for a 10%/90% distribution respectively for Rules 3.13a and 3.13b would make perfect sense. The interest would be similar in Strategy 8 where example of triadic configuration happenstance are given in the literature (Leskovec et al., 2010b). Finally, as signed networks have been mentioned at the beginning of the section, we could only suggest adapting the current model to work with signed edges. While the number of rules would need to be doubled and the application probabilities fine-tuned to match the characteristics of real-world networks, the resulting graph could be of use as a basis for trying out layout algorithms or visualisation solutions aiming to underline the sentiment of individuals toward each other in social networks.

Overall, this second model translation using the graph rewriting formalism ended up being simpler than the one proposed for the small-world model. While the model description does not truly hint of any difficulty in both case, Watts and Strotgatz's is much more precise, especially when considering the order of edges allowed to be rewired. The present model on the other hand is much more forgiving on such requirement, with most nodes susceptible to be chosen as candidates for rewriting at any time. Note that the only exception occurs in Strategy 7, due to our wish to showcase how an imposed absence of edge should be declared and handled without "anti-edges". Such differences in the implementations show us that the graph rewriting formalism has both advantages and drawbacks. For instance, creating a random edge anywhere in the graph is a rather simple feat. Specifying which node to use as the source and destination

of the edge however requires further efforts as the potential domain of eligible elements must be narrowed down to the two targeted nodes before the rule application.

3.4 Conclusion

We have presented here two models of graph generation adapted and expressed using a strategic graph rewriting formalism. Although all the necessary definitions concerning such system have been provided in the previous chapter, we believe these hands-on examples to be the best way to grasp the process followed during this work and have a better look at all the inner workings and how minute details are dealt with. These two generative models have been chosen specifically for several reasons. While one of them is the translation of a well-known and studied network generation method, the other has been designed using an original scenario. These differences allow to showcase the versatility of strategic graph rewriting techniques. The transformations to perform can be indifferently based on a strict and defined model or on looser ideas elaborated through empiric observations and behaviours described in the literature. More precisely, we have shown how the two position subgraphs, $\mathcal{P}$ and $\mathcal{Q}$, could be managed to steer transformations, by proposing, forcing or forbidding elements to be rewritten. Through different rules, we have illustrated how "anti-edges" could be used to condition rule application depending of the existence of edges. Finally, using the extensive instruction set available thanks to the strategy language, we have presented an instance of how such instructions could be used to perform finely-tuned operations to guarantee a specific outcome, namely the creation of a node or an edge.

Chapter 4

Modelling information diffusion in social networks using graph rewriting techniques

Contents

4.1 Propagation in social networks 77
4.1.1 Fast forward from the early days 77
4.1.2 Managing the influence: maximisation and minimisation 79
4.2 Modelling cascading and threshold behaviours 81
4.2.1 The independent cascade model (**IC**) 82
4.2.2 The linear threshold model (**LT**) 87
4.2.3 Comparison of the two models 92
4.3 Transforming a privacy-preserving dissemination model 95
4.3.1 Riposte (**RP**): a privacy preserving propagation model 96
4.3.2 Adapting the Riposte model with linear thresholds (**RP-LT**) 102
4.4 Conclusion 108

As we have seen in the different examples presented in the previous chapters, graph rewriting is a complex albeit powerful formalism. It allows us to perform literally any possible transformation, using one or several rewriting rules, to modify and rearrange any graph as we see fit. Although this aptitude is indeed quite impressive, it also comes with some particularities such as the meticulousness needed to express the rewrite rules and the strategies to avoid the transformations from going out of hand. While these different points have been mostly demonstrated using sets of rules and strategies aiming at generating different types of graphs, the proposed modelling has also brought to our attention a feature we deem most interesting. As one can recall, when detailing the few generative models in the previous chapter, we have noticed the recurring apparition of some of the rewrite rules, and in particular the ones allowing the creation of a new neighbour to an already existing node, or those simply connecting two nodes together.

Chapter based on:
Jason Vallet, Hélène Kirchner, Bruno Pinaud, and Guy Melançon. *A visual analytics approach to compare propagation models in social networks.* In Proceedings Graphs as Models, London, UK, 11-12 April 2015, volume 181 of Electronic Proceedings in Theoretical Computer Science, pages 65–79. Open Publishing Association, 2015.

In addition to be very likely to reappear in a lot of different models, these two simple rules are also providing us comparison points between the rewriting models. Indeed, for two graph rewriting systems using the exact same rules, analysing the strategy can indicate whether the two systems behave similarly or if and how they actually differ from one another. Conversely, a similar strategy operated on two different systems establish a common ground on which we can tentatively initiate all manner of comparisons. Although this approach could well be applied on the graph rewriting systems recreating the generative models we introduced earlier, we propose to implement this application while looking at a different feature of the social networks.

Most of the existing techniques allowing online social network analysis have been initially defined in the context of offline relations. However, with the ease of accessibility to Internet and the online social networks popularity, the users have been able to communicate far and wide with others, thus gradually expanding their own social networks. This obviously caused people to become more connected with each other, which ultimately allowed an ease in the dissemination of information, such as knowledge, interests, ideas, rumours, opinions or any other form of digital content. Those dissemination of information could occur so rapidly and reach so many persons at the same time that these behaviours were compared to those of viruses: infecting neighbours to proliferate and spreading efficiently. While the terminology can seem negative, the process can allow the diffusion of any information, relayed by the people within the social network to their friends, family members, and other relations they may have established inside the social network. The study of these phenomena has initiated a sustained interest in the research community, offering applications in various domains, ranging from sociology (Granovetter, 1978; Macy, 1991) to epidemiology (Hethcote, 2000; Dodds and Watts, 2005; Bertuzzo et al., 2010), and, of course, viral marketing and product placement (Domingos and Richardson, 2001; Chen et al., 2010). With such a wide array of different research fields, the advances in one domain were not automatically known of the others. Consequently, the literature contains multiple descriptions, attempts and models aiming at recreating these different propagation of opinion, disease, or promotional content.

As our rewriting formalism can be used as a common ground to express transformations occurring in a graph, we wish to see if, by describing different propagation models using a common formalism, we can evaluate their respective inner workings and ultimately compare them to one another. Considering the numerous descriptions available in the literature, we are mindful we will not be able to generalise the process to all existing models in this sole chapter, nonetheless, we believe it is important to describe the procedure to follow and study the result to evaluate the soundness of our approach. Additionally, identifying the elements of a solution which makes it unique is a strong advantage. Once the characteristics demarking a given model as being either more successful or specialised than its counterparts are isolated, then such knowledge could be introduced into another model to perfect it or generalise its use for a different type of situation.

In the present chapter, we are going to take a general look at the existing propagation models. By looking at some of the cases where they were used to perform different type of operations, we will come to discuss the elaboration, evaluation and the different use of such models. Because once described in a graph rewriting system using our techniques, all models are expressed through a common formalism, we then propose to adapt two of the most generic and recurrent models of information propagation in order to study them further and potentially draw a comparison between the two of them. Finally, we propose to set aside the propagation models to focus our interest on a dissemination model instead. Although using principles very similar to those observed in propagation phenomena, dissemination models are used to express means of diffusing information in social networks with a minimum input of the individuals taking part in it. After a first description, we then propose to enhance our dissemination model using a few improvements we have identified in the previous propagation models. Once translated into our graph rewriting

formalism, this whole process is achieved in order to ameliorate the dissemination model and improve the quality of the spread information.

4.1 Propagation in social networks

The idea behind a propagation phenomenon is very simple. Let us imagine we have two friends, Alice and Bill, who are discussing; suddenly Alice has a great idea and explain it to Bill. From here on Bill can adopt a number of different behaviours however we restrain this introductory example to the three most basic outcomes. Firstly, he can have a positive reaction toward the idea of Alice, find it to be grand, deserving to be spread far and wide, for the whole world to know. Conversely, Bill can express a negative opinion about the idea, totally rejecting it and maybe even finding it offensive. A last outcome would find Bill with a neutral opinion of Alice's idea, either deciding to ignore or not caring about it. Were Bill to find the idea pleasing, he would undoubtedly mention it to its own friends, Charles and Donald, who, in turn, would be confronted to the same choices: endorsing the idea, opposing it or remaining neutral. In the same way, if Bill were to find himself disapproving with the idea, he may also mention it to his friends in order to inquire their own reactions, a process they may feel the need to also achieve in turn. However, if Bill were not to care at all about it, chances are that the information would not transpire in any of his conversations, leaving Charles and Donald unaware of Alice's idea.

As we start considering more and more individuals, each with their own friends and connections, as expected in a social network, we can begin to imagine how the idea initiated by Alice may end up spreading throughout the whole graph, especially if a whole lot of persons were to start reacting to it (independently of that fact that the responses may be positive or negative). More generally, when an individual in a group performs a specific action (announcing an event, spreading a gossip, sharing a video clip, giving an opinion), she/he initiates a propagation phenomenon. Each neighbour responding to the content subsequently becomes a part of the propagation phenomenon, taking an active part in it. This person then informs her/his connections of her/his state toward the subject, giving them the possibility to react to the information and become active users themselves if they respond to the action. The process reiterates as the newly active neighbours start sharing the information with their own neighbours and so forth. The activation can thus end up propagating from peer to peer across the whole social network like wildfire, or, alternatively, it can die very quickly if no one effectively react to it.

While this general purpose example mention the positive, negative and neutral reactions of the individuals in the network, very few models actually propose such level of distinction (Chen et al., 2011; Stich et al., 2014; Wen et al., 2015), blurring the line between activation (sharing with their neighbours to show how brilliant or terrible the idea is) and endorsement (sharing the idea because we agree with it). To avoid confusion, we propose from now on to only consider a propagation step as resulting with a binary outcome: either the idea is refuted or it is accepted. In this section we will take a general look at the propagation phenomenon, with a quick overview of the literature and some of the most recurrent applications for them.

4.1.1 Fast forward from the early days

During the sixties, several sociologists have looked into the idea of collective behaviour (Smelser, 1962),[1] the social psychology behind it (Brown, 1965), how collective actions ended up taking place (Olson, 1965) and how a specific behaviour, mostly initiated by a limited group of individuals, was able to gain momentum and reach prominence (Wheeler, 1966). The keyword

[1] The theory detailed in this book is also resumed in Ormrod (2014)

contagion was at that point used to describe these situations where individuals were offered a choice of performing an action, thus becoming actors of the contagion process and making the global movement grow. The underlying idea is that, after some time, the decision of following the most popular opinion is more likely to occur. This phenomenon, which is more informally known as the *sheep effect* or sometimes *herd/flock behaviour*,[2] is not really impossible to conceive but seem to come in direct opposition to some other work originating from the field of game theory stating that an equilibrium should, at some point, exist (Nash, 1951). A solution unifying the two approach was finally brought forth by Granovetter (1978) with the introduction of a *threshold*, marking the tipping point where the decision made by the individual is balanced between the gains and losses of each alternative. While these authors speak of crowd behaviour triggering riots, strikes and migrations, mentions of applications also encompass innovation and rumour diffusion.

In more recent years, propagation phenomena have become of interest not only to sociologists but to a much wider array of researchers. With scientists abstracting the modelling of infectious diseases (Hethcote, 2000), epidemiological models have been used to simulate diverse possible outbreak scenarios where the contagion can take place in rural or urban areas (Eubank et al., 2004). Specific type of infections can also be represented by taking into account the vector easing the transmission, as for instance Bertuzzo et al. (2008) have achieved with cholera epidemics and the KwaZulu-Natal province river network. Nonetheless, such knowledge is not yet sufficient or precise enough to benefit in real world situations (Koenig, 2009; Bhattacharya et al., 2009; Ali et al., 2012) despite the researchers efforts when looking for solutions aiming at identifying ways to contain or slow down these infectious diseases (Kribs-Zaleta and Velasco-Hernández, 2000; Madar et al., 2004). It comes with no surprise however that a fair share of the existing works in the field propose to use networks as the underlying structure (Newman, 2002). In such cases, the edges represent the possible vectors of infection, like the river network in the case of cholera (Bertuzzo et al., 2008), while the nodes can either represent susceptible sites of infection (e.g., communities, group of families, villages) (Moore and Newman, 2000) or directly individuals, as considered in sexual networks for instance (Liljeros et al., 2003).

By considering the structure where the infection takes place as a network, we are able to further generalise the idea and apply the same knowledge gathered this far to other fields. The contagious element now longer needs to be an infectious disease, but can take the form of an innovation which can change the world (Young, 2000; Acemoglu et al., 2011), the photo of a pet (Cha et al., 2009), or even something as simple as a rumour (Lind et al., 2007). Additionally, using this special structure allows us to take into account the topology of the network being subjected to the infection to study particular traits or recurring characteristics (Moore and Newman, 2000; Kuperman and Abramson, 2001; Pastor-Satorras and Vespignani, 2001), a feature which has been shown as interesting when addressing viral infection problems in computer networks (Ganesh et al., 2005). The case which seems to gather the most interest however concerns the principle of information diffusion within social networks. A lot of different models exist to address the subject and their complexity can rank from the most simple cascade models (Watts, 2002; Kempe et al., 2003) to ones considering temporal evolutions (Goyal et al., 2010; Saito et al., 2010; Gomez-Rodriguez et al., 2014) or the neighbouring and community structures (Bao et al., 2013; Lin et al., 2015). While a complete list of all the existing models would be extremely long and overall quite monotonous, we can note that several tendencies seem to appear over the years. For instance, the specificities introduced by the spread of rumours (Lind et al., 2007),

[2] As expressed by Mark Twain: "*We are discreet sheep; we wait to see how the drove is going, and then go with the drove. We have two opinions: one private, which we are afraid to express; and another one –the one we use– which we force ourselves to wear to please Mrs. Grundy, until habit makes us comfortable in it, and the custom of defending it presently makes us love it, adore it, and forget how pitifully we came by it.*"

(mis)information (Acemoglu et al., 2010), or user-created content (Bakshy et al., 2009) have quickly been generalised to refer to the more generic information propagation/diffusion terms but mentions of *influence* also become more and more common.

4.1.2 Managing the influence: maximisation and minimisation

In a lot of the models describing information propagation, the notion of influence is used to describe the strength with which an individual in the social network can impact on the behaviour of its neighbours. While it can be seen as a concept similar to the reputation, the influence is different as its significance is not usually defined at a global level but more likely directly between two people. We also consider it asymmetrical such that the influence from an individual n on another person n' is different from the influence that n' exercises on n. Because the influence has such an important effect on the propagation rendition, it is essential to get the right values before performing any sort of simulation. This is however easier said than done as such information is usually private and each individual decides, consciously or not, the level of influence another person exercises on her/him. A few alternative solutions exist however to propose a solution to this problem such as learning, inferring or estimating the influence from previous operations (Goyal et al., 2010; Gomez-Rodriguez et al., 2012; Kimura et al., 2009b; Du et al., 2013). Alternatively, studying the user profile may allow us to extract attributes (On-at et al., 2017) which can then be used to infer interests (Saito et al., 2011) and the relevance of an information when spread to anyone. Additionally, classifying the different types of social relationships can also give us hints, as the closer two persons are, the more mutually influenced they are likely to be (Tang et al., 2011; Zhao et al., 2014).

By taking a step back, we can understand why knowing the influence of each person is sought-after. Let us imagine ourselves for a minute as product creator. While we have heard of the notion of viral marketing, an operation similar to the process commonly known as *word-of-mouth* (Goldenberg et al., 2001) but on a much larger scale, we have no idea on how to trigger such a craze around our own product. The solution is very simple: find the most influential person and convince her/him, usually through sponsorship, to promote our own product. Of course, as in traditional marketing, several factors have to be considered such as targeting consumers most likely to purchase our product (e.g., audio-speakers for a musically-inclined communities, trainers for a group centred around basketball, televisions for the film enthusiasts) or the timing of the operation. Additionally, targeting the most influential individual is not sufficient if the product adoption is only limited to its own connections; as we aim at promoting to as many people as possible, we need the neighbours to spread the word to their own neighbours and so on. Several works such as Domingos and Richardson (2001) and Richardson and Domingos (2002) present these aspects aiming at identifying the (commercial) value of people in the network. Once the right persons have been found, all that is left to do is sponsor them and the promotion through word-of-mouth should start and continue on its own.

The goal we aim at here is quite simply to maximise the diffusion by finding the right seed, the initial set of persons which will begin the diffusion. Several researchers have proposed different solutions going from the standard greedy algorithm (Kempe et al., 2003), to more advanced maximisation models such as SP1M (Kimura and Saito, 2006), PMIA (Chen et al., 2010), or InfluMax (Gomez-Rodriguez and Schölkopf, 2012), with the first two improving the greedy algorithms using heuristics (Chen et al., 2009), thus allowing it to scale up to graphs with millions of elements. Nonetheless, although the maximisation of influence is an attractive feat in the context of social networks (Sheldon et al., 2012; Wang et al., 2012; Yang et al., 2012), the same can not be said if we consider instead the spreading of negative content. Whereas the previous goal was to share and diffuse the content to as many people as possible, we can also be facing situations

where the propagation needs to be contained or stopped altogether: it requires to be minimised. Multiple cases can offer such scenario like leaked content, infected e-mails, spread of misinformation (Budak et al., 2011), and can even be generalised to the evolution of infectious diseases and contagions, computer viruses (Kimura et al., 2008), and even attacks on communication networks (Onnela et al., 2007) like the propagation of DNS cache poisoning[3] from authoritative DNS servers (Son and Shmatikov, 2010). In all these situations, the idea is not to cure but to restrain the diffusion process (Li and Tang, 2011), and the only solution is simply to avoid contact by removing connections (Kimura et al., 2009a; Khalil et al., 2013). Despite being more critical than its counterpart in the few examples we have given above, minimising the propagation is not easier than maximising it (Luo et al., 2014). Nonetheless, in both case, solutions exist to locate the source of the propagation and identify the individual or group of persons who have initiated the diffusion (Shah and Zaman, 2011; Pinto et al., 2012).

Finally, we propose to mention one last example in the context of maximisation or minimisation, as we leave aside the social networks for a moment to take a look instead on networks of enterprises, and more precisely, the financial relations between banks. After the financial crisis which occurred in 2007-2008, multiple efforts have been made to understand what happened at the time and more importantly, how to avoid it happening again in the future. Without going into too much unneeded details, the burst of the United States real estate bubble conjugated with the instability of a few key actors in the banking scene proved to be critical for the whole country economy.[4] While it is not uncommon for banks to have financial obligations toward one another or to own shares in the same businesses, it only makes them more vulnerable when one of the party fails. As some of the obligations came to be reclaimed, a few of the establishments proved to be unable to fulfil their responsibilities to other banks, thus creating a domino effect; the crisis ultimately culminated with the bankruptcy of the Lehman Brothers Holdings Inc.,[5] the fourth largest investment bank in the United States at the time. This phenomenon, called a *systemic risk* in finance, has since been studied through and through, whether by using a contagion metaphor (Moussa, 2011) akin to the ones we described above, by understanding how the network structure can help such relations to become more resilient (De Quadros et al., 2015), or even to study the aftermaths of such propagation (Acemoglu et al., 2015). Due to their large scale and all their ramifications, such contagions create aftershocks (Bardoscia et al., 2015) which may in turn "infect" smaller financial entities which had not defaulted during the main contagion process. In the end, these events have created an awareness of the existence of such weaknesses in the financial system and resulted in the creation of solutions allowing to test the robustness of such key institutional networks (Battiston et al., 2015) hoping to preemptively identify such flaws (Roukny et al., 2016).

Although the underlying principle of diffusion in networks is quite easy to understand, we can see that multiple approaches and applications exist, making some of the models quite specific to their own field whereas others have been designed to be very generalist. As we face all these existing models, it is sometimes hard to truly understand the minor differences existing between them and grasp if the different improvements they propose will effectively allow a more realistic modelling of a given diffusion phenomenon. Unfortunately, the latter can only be decided based on the characteristics of the network at hand; however, we believe the former can be addressed by comparing the different models on an equal footing. The first step is to define our context, thus while studying how innovations, infectious diseases, or financial failure spread is very interesting,

[3] https://en.wikipedia.org/wiki/DNS_spoofing

[4] For further details on the subject and a more complete depiction of the mechanisms involved in the crisis, we can only advise the interested reader to watch "*The Big Short*" (http://www.imdb.com/title/tt1596363/).

[5] https://en.wikipedia.org/wiki/Bankruptcy_of_Lehman_Brothers

we decide for the remaining of this chapter to focus on the spread of information in social networks. We then only have to find a way to express the models using a common formalism able to divide the models in atomic operations to express the modifications being performed by each one.

4.2 Modelling cascading and threshold behaviours

As we have seen, propagation in social networks occurs when one or several persons decide to perform a specific action such as relaying information, announcing an event, spreading a few gossips, or sharing a video clip. By carrying out this operation, the individuals become *active*, their neighbours are then informed of their state change and are offered the possibility to reproduce the action, thus becoming active themselves. The procedure then reiterates as the newly active neighbours share the information with their own neighbours, propagating the activation from peer to peer throughout the whole network. To replicate this phenomenon, some propagation models opt for entirely probabilistic activations (e.g., Kempe et al. (2003); Chen et al. (2011); Wonyeol et al. (2012)), where the presence of only one active neighbour is often enough to allow the propagation to occur, while others (e.g., Watts (2002); Kempe et al. (2005); Goyal et al. (2010)) consider the influence between individuals and use threshold values to determinate at which point the common influence of the neighbours building up against a specific person is enough to incite her/him to take part in the propagation. This threshold value, triggering the response to the phenomenon can be seen as the tolerance towards performing the action at the heart of the propagation: the more solicited a user is, the more inclined he becomes to activate. From a general point-of-view, several propagations can happen in one network during the same time window, but most propagation models focus only on one action at a time (e.g., relaying a specific information) as the other propagations are likely to be about entirely different subjects, thus creating few if any interferences between the two phenomena.

The graph rewriting formalism is, as mentioned in the previous chapter, a very flexible technique which can be used to translate a complex algorithm into a set of rewrite rules and a strategy. Whereas, for a single model, different sets of rules and strategies can be described to perform similar resulting transformations, proposing rewriting operations which can be compared for two different examples can be rather convoluted and a good understanding of all the models we aim to translate to oppose afterwards is necessary. In the following, we propose to detail two of the most basic propagation models which are commonly used as footing for a number of other models: the *independent cascade* model **IC**, as described by Kempe et al. (2003), and the *linear threshold* model **LT** (sometimes also called *general threshold* model), using the definition given in Goyal et al. (2010). Despite being both propagation models, **IC** and **LT** do not behave quite similarly. Consequently, to ease our work as much as possible in the upcoming task of comparing the models, we must define each transformation to perform as simply as possible, and thus only use atomic operations.

To facilitate the expression of both models, we define a few common ingredients which are used in the rewrite rules. While each model is expressed in a certain way in the paper's original definitions, we propose a few alterations such as renaming some of the attributes or using different colours to express the current state of a given node; such adjustments do not alter the models' mechanisms in the slightest but certainly helps us in presenting homogeneous models, easing the identification of potential similarities. Firstly, we assume that, at any given time, each node is in a precise *state*, which determines its involvement in the current spreading of information. The different states are represented using three values. Either an individual is *unaware* if she/he has not (yet) heard of the action at the heart of the propagation. Alternatively, a person is

considered to be *informed* if she/he has knowledge of the action, that is, if she/he has been influenced by their neighbours and thus been proposed to take part in the phenomenon without having accepted yet. Otherwise, if somebody is in neither of the previous states, then she/he is *active* as, having been successfully convinced, she/he has decided to propagate the action. We encode this information on each node using the 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 will describe effectively how the nodes' states evolve. An `unaware` node can become `informed` when at least one of its `active` neighbours tries to influence it, and an `informed` node will become `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. The second common ingredient we use in the two models is a naming convention as we call the first step *influence trial*, during which an `active` node n tries to influence an inactive neighbour n' (where n' is either `unaware` or just `informed`) and the following step is called an *activation*, where the node n' becomes `active` once it has been successfully influenced. Of course, the method used to decide whether an individual is supposed to become `active` is different depending of the models, however, the general workflow remains the same. We point it out in our last common ingredient, as we introduce for both models an attribute called *Tau* to store the influence level of the `informed` nodes. Its value is computed/updated during the *Influence trial* step and is used to decide whether an individual will become `active`, more details are given hereafter.

These common ingredients –the states, the two transformations and the attribute triggering them– already give us a few points of reference to compare the two models. For the remaining of this section, we start by recalling the definitions of these two models and show in which manner they can be described as instances of the same strategic graph program. Just like presented in the previous chapter, we introduce with each model visual representations of the rules applied to perform the rewriting operations. For each one, 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. Once both models are described, we take a closer look at an example of propagation phenomenon simulated using the graph rewriting platform Porgy. We then propose an analysis of the differences between the two models highlighted using the graph rewriting formalism as a common language.

4.2.1 The independent cascade model (IC)

The first model we describe is a basic form of cascading propagation. The **IC** model we choose to adapt originates from the definition introduced in Kempe et al. (2003). Being one of the simplest model available in the literature, it has several variations and refinements (Gomez-Rodriguez et al., 2010; Watts, 2002)) allowing, for instance, to simulate the propagation of diverging opinions in a social network Chen et al. (2011). The model is described as follows in Kempe et al. (2003):

> "*We again start with an initial set of active nodes A_0, and the process unfolds in discrete steps according to the following randomized rule. When node v first becomes active in step t, it is given a single chance to activate each currently inactive neighbor 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 neighbors, 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.*"

The details given by the authors provide us with all the precisions we need to express the **IC** model. Moreover, as we aim to propose a correct adaptation of the model using our formalism, we use this initial description to extract the key properties of the propagation our translation must reproduce to prove its accuracy. Indeed, where the exact algorithm to follow was clearly expressed for the small-world generation model (see Chapter 3, Section 3.2.1 on page 48), the same cannot entirely be said for the two propagation and the up-coming dissemination models presented in this chapter. Consequently, to prove that the translations implemented with our formalism are correct adaptions of the initial models, we start by identifying the models' key properties, extract them from the descriptions given in the original papers and finally, show how the proposed rules and strategy respect each and every one of these properties. Thus, for the **IC** model as described by Kempe et al. (2003) and once the notation of the elements has been adapted to our own preferences, we express the subsequent properties.

Proposition 20 (**IC** properties)**.** *These properties must be satisfied at each step k where an active node n is selected.*

IC.1 *n is given a single chance to activate each inactive neighbour n';*

IC.2 *n succeeds in activating n' with a probability $p_{n,n'}$;*

IC.3 *attempts of n to activate its inactive neighbours are performed in arbitrary order;*

IC.4 *if n succeeds in activating n' at step k, n' must be considered as an active node in step $k+1$;*

IC.5 *the process ends if no more activations are possible.*

Keeping in mind these few properties, we now present a more in-depth description of the **IC** model which we use as a basis for the translation using our formalism. First, we introduce the notations: let us assume that for each pair of adjacent nodes (n, n'), the influence probability from n on n' is given and that it is denoted $p_{n,n'}$ where $0 \leq p_{n,n'} \leq 1$. In this version of the propagation model, we note that $p_{n,n'}$ is history independent (its value is fixed regardless of the operations performed beforehand), and can be non symmetric, i.e., $p_{n,n'}$ does not have to be equal to $p_{n',n}$. We express as $N_0 \subset N$ the subset of nodes initially active, N_k the set of active nodes at step k, and let 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 active nodes evolves at each step and it is incrementally 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 a single 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. Additionally, for the model to fully comply with the properties stated above, the order used to choose the nodes n and their neighbours during the propagation is arbitrary.

As one can see, the model given above, as well as the ones following, are not directly expressed using labelled directed port graphs but common directed graphs. To adapt them to our formalism

is however rather simple and only consists in using port nodes with a single port instead of normal nodes. With this more precise description of the model, we can now begin to consider the attributes we are going to need, and elaborate our rewrite rules and strategy.

Attributes

In order 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. The value given to *Marked* is thus depending of ξ to help avoiding multiple influence tentatives, with the attribute being equal to 1 if $(n, n') \in \xi$, and 0 otherwise. The last attribute, *Tau*, used to measure how influenced a given node is, will have its value ranging in $[-1, 1]$, with 0 being the triggering value always used to indicate when a node can become `active`.

In our take on the propagation phenomenon, the *Influence* and the starting *State* are given parameters of the network being studied. Initially, the attributes' values of each node must be preset according to the node *State*. Thus, the `active` nodes have their attribute $Tau = 1$, while `unaware` ones see their attribute *Tau* set to -1 (note that all the `informed` nodes are `active` in the beginning). Afterwards, during the propagation, the value of the attribute *Tau* is updated in order to reflect the influence probability $p_{n,n'}$ stored in the *Influence* attribute:

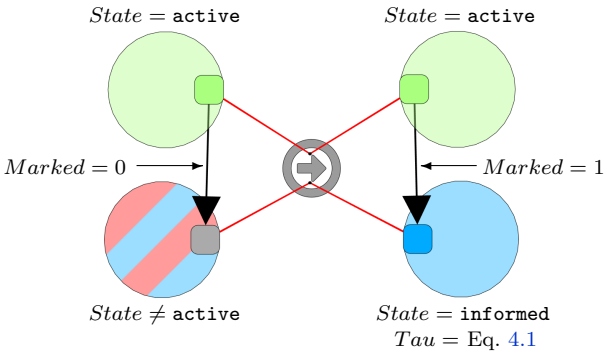
$$Tau = Influence - random(0, 1) \tag{4.1}$$

where $random(0, 1)$ is a random number in $[0, 1]$. We design the Equation 4.1 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 ($Tau \geq 0$). 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 $random(0, 1)$ 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 $random(0, 1)$ in $p_{n,n'}\%$ of cases, so $Tau = Influence - random(0, 1)$ is greater or equal to 0 in $p_{n,n'}\%$ of cases.

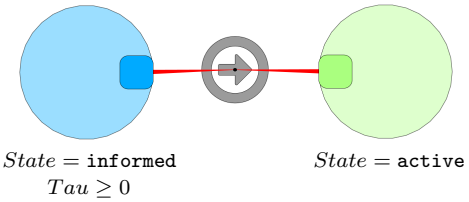
Rewrite rules

The rewrite rules used to represent the **IC** model are given in Figure 4.1. The first one, Rule *IC influence trial* (Fig. 4.1a), shows a pair of connected nodes in the left-hand side and their corresponding replacements in the right-hand side. The node n', initially `unaware` (in `red`), or already `informed` (in `blue`) by another neighbour, is influenced –successfully or not– in the left-hand side by an `active` node n (in `green`) connected 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 4.1. Furthermore, the directed edge linking the two port nodes is *marked*, by setting to 1 the attribute *Marked*. This operation effectively limit the number of influence attempt for each pair of active-inactive neighbours to one. Inactive nodes can thus still be influenced several times but only when visited by *different* active neighbours.

The second rule, named *IC activate* and depicted in Figure 4.1b, is much more simple as it only applies 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`). One can note that, for this rule, specifying the value of the attribute *State* in the left-hand side is not entirely necessary as *Tau* can only become greater or equal to 0 if the node n has already been



(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.

Figure 4.1: 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`).

`informed` by passing through Rule *IC influence trial*. Nonetheless, by indicating that the rule can only apply to `informed` node, we are also forbidding `active` nodes to be considered multiple times when applying this rule.

Strategy

The rewriting strategy used to represent the **IC** model is described in Strategy 9. The general principle followed to perform the propagation is quite simple. The first instruction (line 1) exclusively selects all the nodes whose *State* attribute is `active` and adds them to the position subgraph $\mathcal{P}$. Then a `repeat` loop is used to apply the two rewrite rules seen above. As we recall that a rule can only be applied if the matching subgraph contains at least one node belonging to the position $\mathcal{P}$ (and no element belonging to the banned set $\mathcal{Q}$), an `active` node is used as a mandatory element from $\mathcal{P}$ when calling the *IC influence trial* rule (line 3) to rewrite a pair of active/inactive neighbours. The right-hand side of the rule then follows the default behaviour given in Definition 13 on page 31, indicating that when no $\mathcal{M}$ subgraph is specified in the rule then the entire right-hand side is added to $\mathcal{P}$. Thus, at the end of the *IC influence trial* rule (Fig. 4.1a) application, both the newly `informed` node and the already `active` node are added to the position $\mathcal{P}$.

Strategy 9: IC propagation

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

The *IC activate* rule (Fig. 4.1b) is then immediately executed (line 4) to activate the potentially successfully influenced node. During the first loop, the position $\mathcal{P}$ contains all of the `active` nodes existing in the graph as well as the one `informed` node we have considered in the

previous rule. We thus can ensure that the node which have just been `informed` is tested and, if it qualifies (when $Tau \geq 0$), becomes `active` and is added to $\mathcal{P}$. The `try` instruction surrounding *IC activate* (line 4) is used to prevent any premature failure of the strategy if the `informed` node is not sufficiently influenced (no matching `informed` node with $Tau \geq 0$ can be found). In the following loop iterations, the position $\mathcal{P}$ still contains all of the `active` nodes existing in the graph but also all of the `informed` node which have been considered in *IC influence trial* but did not become `active`. As the strategy always try to apply *IC activate* after each influence, an `informed` node becomes `active` as soon as it is sufficiently influenced. The whole process is then repeated until no more propagation can be performed, that is when all possible edges are marked and all possible activations have been performed.

Validation and termination

As indicated earlier, we must now prove that our adaptation of the **IC** model is correct and corresponds to the initial model description given in Kempe et al. (2003). We thus consider in turn the different properties, extracted from the original depiction and listed in Proposition 20 on page 83, and show that our own implementation respects them.

Lemma 21 (**IC.1** of Proposition 20)**.** *Each active node n is given a single chance to activate each 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 (n, n') of active-inactive neighbours since no other rule resets the marked edge. □

Lemma 22 (**IC.2** of Proposition 20)**.** *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 4.1 on page 84) and the encompassing explanations. □

Lemma 23 (**IC.3** of Proposition 20)**.** *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 24 (**IC.4** of Proposition 20)**.** *If the active node n succeeds in activating its neighbour n' at step k, n' must be considered as an active node at step $k + 1$.*

Proof. All nodes in the right-hand side of the rules are put in $\mathcal{P}$ 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 $\mathcal{P}$. Thus, it is an eligible candidate on which to match the `active` node in rule *IC influence trial* during the next iteration of the loop. □

Lemma 25 (**IC.5** of Proposition 20)**.** *The process ends if no more activations are possible, i.e., there exists no pair of adjacent nodes (n, n') such that n is active, n' is inactive and n has not yet 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 26 (**IC** implementation correctness)**.** *The propagation process defined by the rules in Figure 4.1 and Strategy 9 respects the properties enounced in Proposition 20.*

Proof. Each of the property is proven in turn by Lemmata 21, 22, 23, 24 and 25. □

Proposition 26 thus demonstrates that our implementation respects the properties given in Proposition 20 and that our graph rewriting program is indeed a correct translation of the **IC** propagation model as described in Kempe et al. (2003). Furthermore, in addition to Lemma 25 which show that our model stops when in the same conditions than those described by the **IC** model, Proposition 27 and its proof also demonstrates that our graph rewriting program will always terminate as long as the treated graph is finite and no new elements are added to it.

Proposition 27 (**IC** termination)**.** *If the network is finite, the strategic rewrite program given by the rules in Figure 4.1 and Strategy 9 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 $\mathcal{P}$ containing all the active nodes (line 1 in Strategy 9), thus $\mathcal{P}$ 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 (each considered edge is *Marked*), 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. □

4.2.2 The linear threshold model (LT)

The second propagation model we propose to study is the **LT** model. Unlike the simple cascading behaviour offered in the **IC** 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 some examples of publications describing models using activation thresholds are proposed in Kempe et al. (2003), the model detailed below is based instead on a more generalised version described in Goyal et al. (2010). It is worthy to note that although the authors propose several alternative versions of their generalised **LT** model, we only consider their first depicted instance. More details on the particularities of this specific model are given below, however, the general threshold model is described in Goyal et al. (2010) as follows:

> "*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 neighbors become active. Time unfolds deterministically in discrete steps. As time unfolds, more and more of neighbors 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 neighbors* 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 neighbors of* u *that are active at time* t*.*"

The **LT** model is more difficult to understand than the **IC** model, and as one can see, the description proposed above is less precise than the one we excerpted from Kempe et al. (2003) for the **IC** model. Nevertheless, it is possible to follow an approach similar to the one adopted previously to implement this description using our formalism and notations. Here again, we can identify how 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. One can see that the focus is different in this propagation model: where **IC** was considering `active` nodes to influence and activate other nodes, **LT** instead seem to consider inactive nodes and influence them using `active` ones. These different points of operation however do not necessarily indicates that the two models are devoid of common ground. To ensure that our adaptation of the original model described in Goyal et al. (2010) is entirely sound, we extract the key properties of the model, adapt the elements names to remain consistent and express the properties as follows:

Proposition 28 (**LT** properties)**.** *These properties must be satisfied at each step* k *where a node* n *is selected:*

LT.1 *The node* n *has a monotone activation function* $f_n(S)$ *computing its* active *neighbours' joint influence value.*

LT.2 *An inactive node* n *becomes* active *at step* $k+1$ *if its neighbours' joint influence* ($f_n(S)$) *exceeds its threshold value* (θ_n)*.*

LT.3 *When* n *becomes active, its influence must be considered on its inactive neighbours.*

LT.4 *The process ends if no more activations are possible.*[6]

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, we express the monotone activation function $f_u : 2^{N(u)} \to [0,1]$ by the function $p_{n'}\big(S_{n'}(k)\big)$ as described in Goyal et al. (2010). As defined before, $N_0 \subset N$ is the subset of nodes initially active, N_k is 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 active 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' adjacent to n but whom n has not tried to influence yet.
- An inactive node $n' \notin N_{k-1}$ has its *active* neighbours' joint influence value computed using the formula:

$$p_{n'}\big(S_{n'}(k)\big) = 1 - \prod_{n \in S_{n'}(k)} (1 - p_{n,n'}) \tag{4.2}$$

[6] Although this last property is not explicitly mentioned in Goyal et al. (2010), we consider it an expected characteristic of the model.

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 $S_{n'}(k)$ to express the set of neighbouring nodes of n' being considered at the current step k. This process continues for as long as possible, that is until, for all the joint influences up-to-date, no more activation can be performed.

Similarly to the **IC** model, one can see that the **LT** propagation takes place in two phases: an influence computation followed by a potential activation. Before presenting the corresponding rules or the attributes we are going to use however, we need to specify more precisely the properties of the intended propagation model from Goyal et al. (2010) 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 $p_{n'}(S_{n'})$). Currently, we are looking to implement their static propagation model where $p_{n,n'}$ is expressed as a constant value, in opposition to their discrete and continuously evolving variants where the influence value changes as time pass-by. 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:

$$p_{n'}\big(S_{n'} \cup \{n\}\big) = p_{n'}(S_{n'}) + \big(1 - p_{n'}(S_{n'})\big) \times p_{n,n'} \tag{4.3}$$

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 the specificities of the **LT** model, two new attributes are needed in addition to the ones we introduced earlier in **IC** (i.e., we recall *State* and *Colour* –to define the nodes states–, *Influence* –to store the probability $p_{n,n'}$–, *Marked* –to mark edges connecting previously visited pairs of nodes–, and *Tau* –to store the influence trial outcome). 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 $JointInf = 0$ (as mentioned previously, there is initially no `informed` node in the graph). During the influence step, the value of the attribute *JointInf* on the node being `informed` is updated as specified by Equation 4.3, which we adapt into the following formula when using the appropriate attributes:

$$JointInf = JointInf_{\text{OLD}} + (1 - JointInf_{\text{OLD}}) \times Influence \tag{4.4}$$

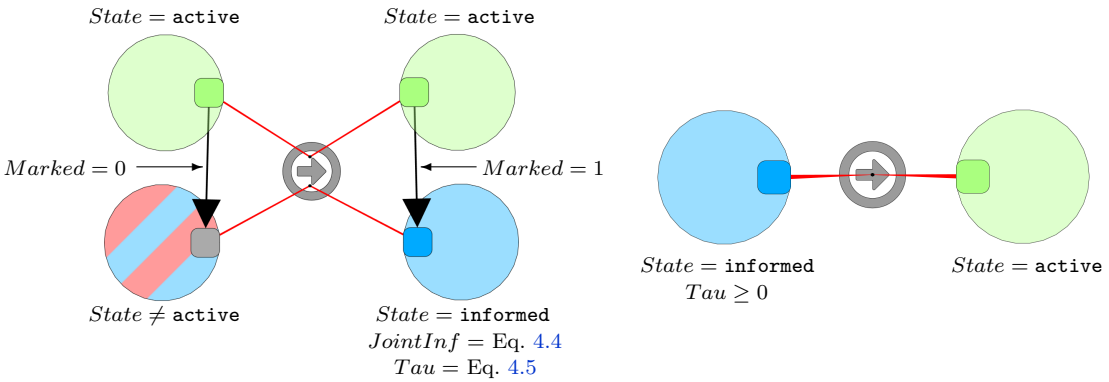
We can then compare this updated joint influence value for a newly `informed` node n' with its threshold value, stored in *Theta*, and assign the result to the attribute Tau:

$$Tau = JointInf - Theta \tag{4.5}$$

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 become `active`, thus ready to start influencing all of its neighbours.

Rewrite rules

As the propagation for the **LT** model takes place in two distinct phases, the two corresponding rewrite rules are quite similar to those introduced in **IC**. The first rule *LT influence trial* (Figure 4.2a) is applied on a connected pair of active-inactive nodes (respectively coloured in `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 during the rule *LT influence trial* in order to update the attributes of the inactive node n'. The new value for the attribute *JointInf* is computed by using the *Influence* value to update the previous joint influence probability measure; this is given by Equation 4.4. The attribute *Tau* then compares this value to the attribute *Theta* as described in Equation 4.5. The edge between the two nodes is then marked to avoid successive influence trials from n to n'. As a result, n' may be influenced by several `active` nodes before their joint influence is important enough to outweigh the threshold value of n', but each of its neighbours will only be able to influence n' a single time.



(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.

Figure 4.2: 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 indicates that the corresponding node for the match can be in either of the two states: `unaware` or `informed`.

The second rule, named *LT activate* (Figure 4.2b), is identical to the *IC activate* rule shown in Figure 4.1b. A successfully influenced node, identified by the positive value of its *Tau* attribute, simply sees its *State* attribute value set to `active`.

Strategy

We use the rewriting Strategy 10 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 their own rules.

As in the previous model, we start by defining a position $\mathcal{P}$ which gathers all the `active` nodes (line 1). We then use 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 on one of its inactive neighbours. At the end of the rewriting operation, these

Strategy 10: LT propagation

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

two nodes follow the default behaviour of the right-hand side elements and are added to $\mathcal{P}$.

We then *try* to apply the *LT activate* rule (line 4) on an informed node whose *Tau* attribute value exceeds or equals 0. Given that the rule is applied after each influence trial (which only influence one node at a time), the activation takes place as soon as the `informed` node considered before, and thus added to $\mathcal{P}$, is sufficiently influenced. 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, but the strategy application goes on and does not fail.

Validation and termination

With the graph rewriting program complete, we can now check whether our implementation corresponds to the **LT** model specifications described in Goyal et al. (2010). We look at each of the excerpted properties given in Proposition 28 on page 88 and demonstrate that our program respects them.

Lemma 29 (**LT.1** of Proposition 28)**.** *An inactive node n has a monotone activation function $f_n(S)$*[7] *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 4.4, according to which $JointInf \geq JointInf_{\text{OLD}}$. As $JointInf_{\text{OLD}}$ and *Influence* are both defined in $[0, 1]$, the activation function is monotone. □

Lemma 30 (**LT.2** of Proposition 28)**.** *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 4.5. 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. □

Lemma 31 (**LT.3** of Proposition 28)**.** *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 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. □

[7] We recall that in our case, $f_n(S)$ is defined by the function given in Equation 4.2.

Lemma 32 (**LT.4** of Proposition 28)**.** *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. The proof is similar to the one given in Lemma 25 using respectively rules *LT influence trial* and *LT activate* instead of *IC influence trial* and *IC activate*. □

Proposition 33 (**LT** implementation correctness)**.** *The propagation process defined by the rules in Figure 4.2 and Strategy 10 respects the properties enounced in Proposition 28.*

Proof. Each of the property is proven in turn by Lemmata 29, 30, 31 and 32. □

While we include Lemma 32 in the properties of the model, the original description does not explicitly specify when the propagation comes to an end. However, we (justly) 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. Of course, this limitation is also at the heart of Proposition 34 stating the termination of our graph rewriting program.

Proposition 34 (**LT** termination)**.** *If the network is finite, the strategic rewrite program given by the rules in Figure 4.2 and Strategy 10 terminates.*

Proof. Same proof as Proposition 27 using respectively rules *LT influence trial* and *LT activate* instead of *IC influence trial* and *IC activate*. □

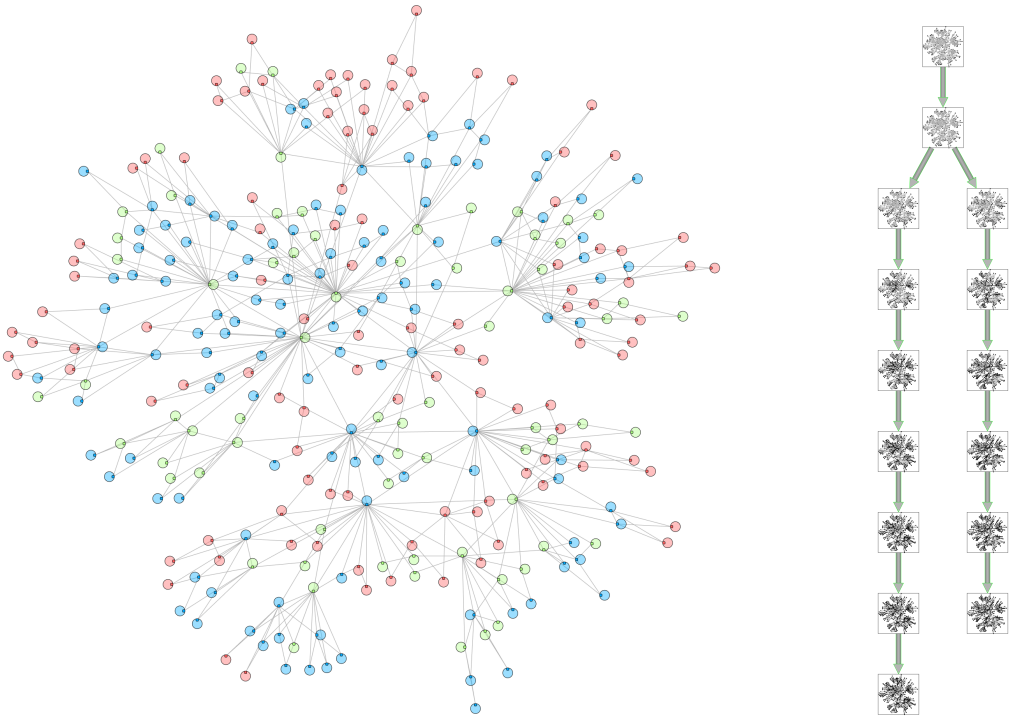
4.2.3 Comparison of the two models

With the two models finally implemented using our graph rewriting formalism, several observations can be devised. The most obvious ones are that the two models descriptions are quite similar and only differ on very specific points. Consequently, the rules *IC influence trial* and *LT influence trial* both present the same application conditions, while *IC activate* and *LT activate* are literally the same rules. The real difference here does not show in term of "physical" modifications on the graph, that is in changes in the topology, but is instead much more subtle and only takes place on the graph attributes. This type of operation is very different from what we have seen in Chapter 3, whether we consider Section 3.3 on page 61 where the transformations only take into account the topology of the graph, or Section 3.2 on page 47 as we go a step further, using more advanced operations.

Obviously, like every implementation, the models translations presented above using our graph rewriting formalism are our very own interpretations of the descriptions given in the original papers. They are thus only two possible solutions among the many alternative renditions one could come up with. This is also true for us as both adaptations of the propagation models have undergone several modifications since we first started exploring the translation of propagation models in Vallet et al. (2015b) and Vallet et al. (2015a). By refining and, at least from our point of view, simplifying the models translations over several iterations to make them more streamlined, we may have, unconsciously or not, brought them closer from one another. While this is not necessarily a bad thing, it still means that, should someone else propose their own implementations, their resulting interpretations of the two models may not be as similar as what we have observed with our own. Nonetheless, we have not been the only ones to witness the similarity between the two models, for instance, in Kempe et al. (2003), the authors have proposed a general model unifying the cascade and threshold models as well as a method to convert between them.

Overall, in our implementation, we have been able to minimise the differences between the two models to the sole computation of the attribute *Tau* which decides whether an informed

node should become active or not. In the case of the **IC** model, *Tau* is computed based on the *Influence* and a probabilistic draw as shown in Equation 4.1 on page 84 whereas the **LT** model takes into account the joint influence (*JointInf*), which is based on the nodes' *Influence* in Equation 4.4 on page 89, and the threshold value *Theta* in Equation 4.5. This simple alteration is more than enough to entirely change the behaviour of the models, which in turn also impact the propagation results.



(a) Network being subjected to a propagation phenomenon using the **LT** model (300 nodes, 1194 edges).

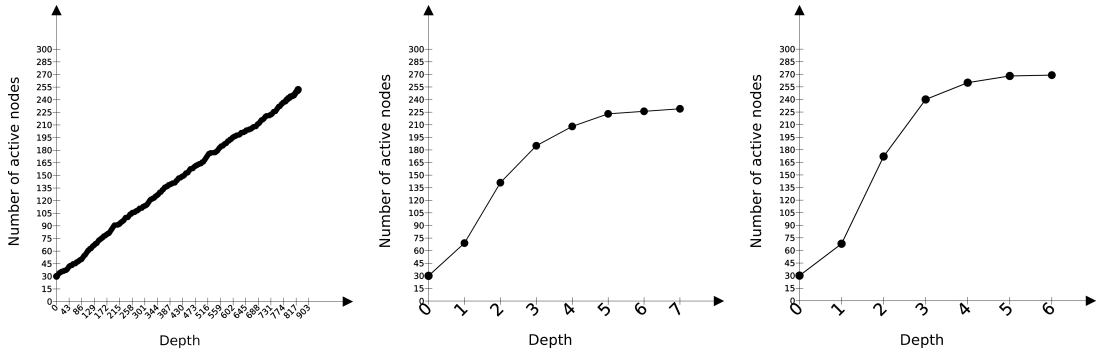
(b) Example of derivation tree (filtered)

Figure 4.3: Derivation tree and detailed representation of the graph being rewritten. The derivation tree depicted here is filtered to only keep intermediate representations; in its unfiltered form, the whole left and right branches both amount to more than 700 rewriting operations.

To observe how the models operate, we perform a few propagation simulation using Porgy and the models implementations described above. To this end, we first need a network with the appropriate attributes fulfilled and initialised as specified. As we do not possess such a network, we generate it[8] and set for each element the appropriate default values for the attributes *State* (= `unaware`), *Marked* (= 0), *Influence* (= $[0, 1]$), *Tau* (= -1), *JointInf* (= 0) and *Theta* (= $[0, 1]$). Our first operation consists in initiating a starting seed, that is a group of nodes which are initially `active` and whom commence the propagation phenomenon. After a few rewriting steps, the propagation is ongoing and some `unaware` elements have become `informed` (`blue`) or `active` (`green`). We show in Figure 4.3a the network we have considered to perform our

[8] We use one of Tulip's plugins to create a network structure similar to those described in Wang et al. (2006).

rewriting operations; it is depicted in an intermediate state as the propagation phenomenon is still growing. Additionally, by applying strategies and performing transformations, we are also building up the derivation tree (see Figure 4.3b), creating branches growing little by little as each rule application creates a new state for the graph which is then added at the end of the branch. With using Strategies 9 on page 85 or 10 on page 91, we perform a continuous loop of influence and activation thus resulting in a slowly but steadily evolving number of `active` nodes as shown in Figure 4.4a. However, this activation method, while corresponding perfectly to the model description, fails to represent the dynamicity existing during propagation phenomena.



(a) Evolution for Strategy 9 using the **IC** model

(b) Evolution for Strategy 11 using the **IC** model

(c) Evolution for Strategy 11 using the **LT** model

Figure 4.4: Analysis of a specific property evolution along a branch of the derivation tree. The three scatter plots give for different branches of a derivation tree the evolution of the number of `active` nodes with each passing graph transformation.

Consequently, we introduce Strategy 11 which tries to first influence as many nodes as possible, and then tries to activate them. The order of the rules is not changed and the modification does not invalidate any of the Propositions, thus the models are still correctly implemented.

Strategy 11: IC propagation using activation rounds

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

However, this process creates "waves" or *rounds* of propagation, which can be interpreted as if all the existing `active` nodes were trying to influence all of their neighbours at the same time, with the process repeating after each attempt until no more activations take place. Obviously, the dynamics of activation end up completely changing as shown in Figures 4.4b and 4.4c, obtained when applying Strategy 11 on the **IC** and **LT** models respectively.[9] The two scatter plots have been obtained after following the number of `active` nodes for each of the intermediate state of the rewritten graph in the derivation tree shown in Figure 4.3b. We can now see how each propagation starts slowly, accelerates and then gradually comes to an end; the same behaviour

[9]Strategy 11 must be modified beforehand to use *LT influence trial* and *LT activate* instead of their counterparts.

has been observed for each of our simulations. Of course, the propagation results are different once we change the initial seed or enable/limit the influence nodes exercise on one another (using *Influence* or *Theta*). Thus a smaller seed can link to a slower start of the propagation or even cause the propagation to die quickly as a seed too small can lack impact. Larger seeds, high *Influence* and low resistance (*Theta*) on the other hand help in enabling the propagation.

While we have run a few different propagation scenarios with each of the models, it is impossible to compare them based only on the success or the evolution of a small number of propagations. Indeed, the probabilistic aspect of **IC** in particular makes the model very complex to analyse as we would need multiple reiteration of the same propagation scenario (i.e., using the same *Influence* probabilities and the same starting seed of `active` nodes) to obtain a satisfying average instance. While this is still technically feasible, the sheer number of propagation to perform and the time needed to execute them make this specific analysis unrealistic. The **LT** model is more amenable in this case as the propagation results are always similar for each given scenario.

The two models we have detailed in this section give us the bases we need to model propagation phenomena. Once an information is announced out-loud by one of the nodes in the network, all its neighbours can decide to relay it to their own neighbours. However, this type of information transmission is only one among others. While information cannot be conjured out of thin air and thus we need individuals to first produce pieces of content to diffuse, the nodes in the social network may not always be the ones in charge of deciding what information should spread. We propose in the next section to first take a step back from the propagation models to consider the idea of information diffusion as a whole, before diving into a different type of information distribution using this time a dissemination algorithm.

4.3 Transforming a privacy-preserving dissemination model

Generally speaking, information diffusion can be performed using multiple approaches and different medium. For instance, information about current events, also known as *news*, can be transmitted using one-to-one channels (e.g., through interpersonal communications like word-of-mouth) or one-to-many mechanisms (e.g., television and radio broadcast or newspapers). Overall, when a propagation phenomenon occurs, the individuals in the network are usually at the heart of the process, with each person actively deciding whether the information is of interest before sharing it with their neighbours. In such cases, the users are considered to be responsible of the spread as the information can only be diffused to the rest of the network if people endorse it and agree to broadcast it. While this method of sharing information has some advantages, such as how the users need to actively decide to spread the information, it also comes with a few troublesome properties. In particular, you only need a few well placed nodes to agree to propagate an information for it to spread far and wide, without any hope of being able to contain it should the need arise. In the rest of this chapter, we propose to take a look at a dissemination algorithm which can avoid this type of problems. Although a dissemination is still a type of diffusion phenomenon, this method differs in our case from the propagation models studied earlier as it does not aim to replicate the behaviours of activation/infection/contagion in a network, and we use it instead to automatically spread information under certain conditions.

For the remaining of this section, we consider the dissemination model called *Riposte* (**RP**) described in Giakkoupis et al. (2015). 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 is not prone to engage others beyond the set of users who initially provided it. Based on these observations,

the dissemination algorithm will always try to promote and prioritise information which has been highly rated by numerous users, whereas the diffusion of poorly rated comment is hindered and deprecated. This method has for advantage that, even if some of the key individuals endorse an information, if the majority considers it uninteresting, the algorithm will weaken the possibility of this information to be diffused until it finally dies out. Moreover, when the dissemination algorithm is properly tuned, simply observing the information dissemination process is not sufficient to determine with acceptable confidence the opinion of any single user concerning the information that is disseminated. Indeed, for a node about to decide whether to endorse an opinion or not, the algorithm could propose to only enable the information to spread if it is effectively endorsed and always discard undesirable information. Or, on the other hand, the algorithm could also share with the same probability endorsed and ignored information for a complete opinion concealment. However, the optimum situation appears when the probability of *reposting* an interesting information does not utterly outbalances the probability of reposting a rejected information, thus still promoting on the long run the interesting information while also masking the opinion of the users about the specific pieces of information.

In the following, we first implement the **RP** diffusion algorithm as a strategic graph rewriting program reusing the diverse techniques learned and presented in the previous section. Once the model is completed, we iterate over it to propose a new dissemination algorithm considering the neighbour influence in addition to the interest manifested by the users towards the information being diffused. This modification of the graph rewriting program intends to show how easily our formalism can use an existing program and adapt it to incorporate features of other models.

4.3.1 Riposte (RP): a privacy preserving propagation 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, as mentioned above, independently of the user's opinions and consent concerning the information at hand, **RP** will from time to time disseminate information to the user's neighbours. The user's opinion still influences the probability of sharing a given piece of information 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. The authors considered that such a dissemination method could benefit to populations where information traffic is overruled by an authority able to censor content based on their controversial political or religious nature for instance. 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.

As we have done previously, we first consider the original model description given in Giakkoupis et al. (2015) to present the general definition of the algorithm in its original context:

> "*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 neighbors in G, or to not repost t, to anyone. The decision is randomized and depends on the user's (private) opinion on the information, and the number of the user's neighbors 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 neighbors 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.*"

As one can see, the **RP** model differs from **IC** or **LT** as it does not strictly use the same principle of influence followed by an activation. Instead, the nodes first learn the information, decide whether they find it interesting, then the **RP** algorithm decide whether the dissemination should take place based on the interest (or indifference) expressed by the user, with finally the diffusion potentially happening for all of the neighbours at the same time or none at all. Based on the initial description, we can isolate the following properties:

Proposition 35 (**RP** properties)**.** *These properties must be satisfied at each step k where a node n is selected:*

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

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

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 by using our formalism and notations, a few 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 user 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 $0 < \delta < 1 < \lambda$. In the original definition given by Giakkoupis et al. (2015), the value 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 instances 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. We thus define $\overline{S_n}$ as the set of nodes adjacent to n. After being informed by one of its neighbours, two different behaviours are possible for a node. If n wishes to diffuse the information (that is, n becomes active), then either all its neighbours or none will be informed of it with a probability $\lambda/|\overline{S_n}|$. Alternatively, if n does not wish to spread the information (thus, n remains "just" informed), then the information can still be passed to all its neighbours, but this time with a weaker probability $\delta/|\overline{S_n}|$.

Let $D_k \subseteq N$ be 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 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 ($\overline{S_n}$).

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. For this model, 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, as proposed in the previous models, the attribute *Tau* is used to store the result of the activation decision, computed as

$$Tau = Interest - random(0, 1) \tag{4.6}$$

where $random(0, 1)$ is a number uniformly and randomly chosen in $[0, 1[$. An `informed` node thus becomes `active` when $Tau \geq 0$, with its initial value being set to -1 for all nodes before the diffusion begins This time however, *Tau* is computed using the *Interest* attribute instead of the *Influence* attribute as in **IC** and **LT**. As we have seen before, **RP** has the particularity of separating the activation from the diffusion. To perform the dissemination according to the given parameters λ and δ, we thus need to create an additional attribute *Share* which we use to store the likeliness of n to share (i.e., spread) the information. Its value is computed for a given node as follows:

$$Share = \frac{isActive(\lambda - \delta) + \delta}{OutArity} \tag{4.7}$$

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 ($= |\overline{S_n}|$). As only simple operations can be performed on our elements, we use in this equation a handy manoeuvre to emulate a conditional application: by changing *isActive* value to 0 or 1, the fraction numerator switches from δ to λ thus allowing us to use either value based on another attribute. This small trick is not crucial for the model but it allows us to avoid the creation of two almost identical rules with only a small variation in the computation of *Share*. Furthermore, as *OutArity* is the number of outgoing edges from n, it is entirely possible that n does not have any (outgoing) neighbour to transmit the information to. When a node is in this situation, no diffusion can take place obviously, and thus *Share* does not need to be computed; in our implementation, we address this issue by having *OutArity* returning -1 in such cases instead of 0 to avoid errors.

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

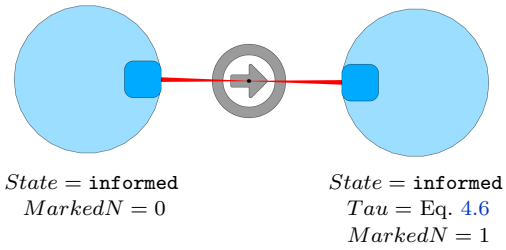
$$Sigma = Share - random(0, 1) \tag{4.8}$$

where $random(0, 1)$ is a random number chosen in $[0, 1[$. Initially, *Sigma* is set to -1, like *Tau*, on all nodes and the information diffusion from n to all its neighbours is only performed when the attribute *Sigma* of n is greater than or equal to 0. Where *Tau* is entirely responsible for the node activation, the attribute *Sigma* alone decides if a given node is supposed to share information.

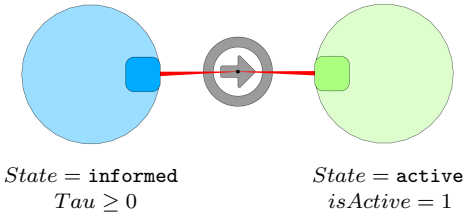
Although all these attributes are needed to emulate the dissemination process, it is important to note that, in real-world applications of the **RP** algorithm, the only visible information to an external observer is whether a node has heard of the information or not, i.e., if the node belongs to D_k 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

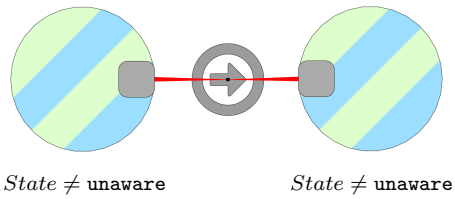
Following the definition of the **RP** model, we can now define the rules, presented in Figure 4.5, as the different steps to fulfil to enact the diffusion mechanisms existing in our dissemination model. The first rule, *RP initialisation* (Fig. 4.5a), is an opening step used to prepare the freshly `informed` nodes who did not yet tried to spread the information (i.e., unmarked nodes). A node



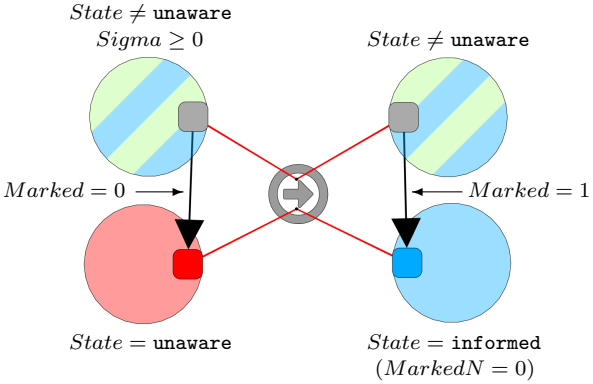
(a) *RP initialisation*: this rule is used to initialise an `informed` node already aware of the propagation subject.



(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.

Figure 4.5: Rules used to express the **RP** model. 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 either be `informed` or `active`.

is thus offered the possibility to be interested in the information, with *Tau* being computed accordingly (see Equation 4.6), and the attribute *MarkedN* being used to mark the nodes which have already been considered. 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. 4.5b) on it. Depending of the previously computed value for *Tau* on n, and more precisely if $Tau \geq 0$, the node is sufficiently interested to become `active`, thus also setting the attribute *isActive* to 1.

The *RP share trial* rule, shown in Figure 4.5c, then computes the attributes *Share* and *Sigma* of the node n being currently considered. As expressed in the model definition, n does not necessarily need to be `active`, i.e., willing to spread the information, but should still be aware of it (that is not `unaware`). The *Share* computation, performed following Equation 4.7, uses *isActive* to change the probability result depending on n's current *State*. *Sigma* then reuses the *Share* value (see Equation 4.8) 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 4.5d. 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'. While n could consider sharing the information to all of its neighbours n', regardless of their *State*, we limit the application here to the `unaware` nodes as the `informed` and `active` nodes are already aware of the information being spread and repeating the information process does not make n' more likely to `activate` or even share the content with others. Thus, only targeting `unaware` nodes, n' becomes `informed`, leading it to be considered as a new potential information spreading source in the next dissemination steps. The newly `informed` node has also its attribute *MarkedN* left untouched, thus still equal to its default value (0), and indicating that the node is ready to be subjected to the *RP initialisation* rule.

Strategy

The strategy used in this model is given below in Strategy 12. Much like for the previous models, we use a repeat loop (line 1) in the **RP** strategy to control the rewriting steps. We initiate the strategy by choosing the node which is to be at the centre of the rewriting operations in the initial step; we thus select an `informed` node which has never been considered to spread the information, that is, its attribute *MarkedN* is still equal to its default value (line 2).

Strategy 12: RP dissemination

```
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. 4.5a), 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 process comes to an end (line 3). However, if a matching node n exists in position P, the rule is applied on it and its attribute *Tau* is computed according to Equation 4.6. The rewritten node is then inserted in $\mathcal{P}$ and ready for the next rule application.

Once initiated, the candidate node at hand (which is the only node in $\mathcal{P}$) may endorse the subject being diffused and activates thanks to the *RP activate* rule (line 4). As shown in Figure 4.5b, 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, this is why, thanks to the `try` construct, that an absence of matching elements when applying this rule cannot cause the strategy to fail. We use the attribute *isActive* to store the result of the activation trial and add the rewritten node to $\mathcal{P}$.

We then apply *RP share trial* (Fig. 4.5c) 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 attributes *Share* (Eq. 4.7) and *Sigma* (Eq. 4.8) while keeping the rewritten node n in $\mathcal{P}$. 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. 4.5d) to all n's neighbours (line 6). If the (indifferently `informed` or `active`) node n has been selected to inform its `unaware` neighbours (n'), then n's attribute *Sigma* is greater or equal to 0. The rule application changes the attribute *State* of n' to `informed` and, through its attribute *Marked*, marks the edge connecting the two nodes to avoid multiple applications of the rule on the same pair of nodes again and again. While all elements of the right-hand side are added to the subgraph $\mathcal{P}$ 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 attribute *MarkedN* is still equal to its default value ($MarkedN = 0$).

Validation and termination

With our graph rewriting program complete, we can now check that the properties of the original model still hold in our implementation.

Lemma 36 (**RP.1** of Proposition 35)**.** *For each node n that learns an information item, the **RP** 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 all of them. Otherwise, when *Sigma*'s value is lower than 0, nothing happens, thus the information is not reposted to anyone. □

Lemma 37 (**RP.2** of Proposition 35)**.** *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 $Sigma \geq 0$ (see rule *RP inform*, Fig. 4.5d). However, *Sigma*'s value has a probability *Share* of being greater or equal to 0, with *Share*'s value itself ultimately depending of attribute *IsActive* (see Equations 4.7 and 4.8), where *IsActive* indicates if n likes the information. When n likes the information item, *IsActive* is equal to 1 and *Share* is equal to $\frac{\lambda}{OutArity}$. Conversely, if n does not like the information, then *IsActive* is equal to 0 and *Share* is equal to $\frac{\delta}{OutArity}$. As expressed before, *OutArity* (the number of edges outgoing from n) is used to represent $|\overline{S_n}|$, itself approximating s_n (the upper bound on the number of n's outgoing neighbours that have yet no knowledge of the information). □

Lemma 38 (**RP.3** of Proposition 35)**.** *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 12 corresponds to a dissemination step k. Strategy 12 can only stop when *RP initialisation* (line 3) or *RP share trial* (line 5) fails as the other instructions' unsuccessful executions do not halt their parent instruction when unexpectedly finishing. As *RP share trial* only fails if there is only no `informed` or `active` nodes in $\mathcal{P}$, an occurrence which can not happened if *RP initialisation* has succeeded, Strategy 12 only stops when *RP initialisation* fails. This scenario happens exclusively when there is no `informed` node left unmarked ($MarkedN = 0$), when $D_k = M_k$, which takes place when $\mathcal{P}$ comes out empty after instruction `SetPos` (line 2). This means that all the `informed` nodes have already been considered (and consequently marked) and thus no more diffusion is possible. We can also force the dissemination to come to an early end by using a repeat instruction with a bounded number of loops, and thus imposing a finite number of steps k. Additionally, although this is not the case for us, a new rewriting rule could be added to our graph rewriting program to add new nodes to the graph while the dissemination process takes place. If those incoming individuals are `unaware` of the information being diffused, the process can go on forever as long as new nodes are provided. □

Proposition 39 (**RP** implementation correctness)**.** *The dissemination process defined by the rules in Figure 4.5 and Strategy 12 respects the properties enounced in Proposition 35.*

Proof. Each of the property is proven in turn by Lemmata 36, 37 and 38. □

The validation of Proposition 39 ensures that our implementation of the **RP** model is consistent with the properties of the dissemination algorithm as expressed in Giakkoupis et al. (2015). While Lemma 38 mentions the possibility of a never-ending diffusion process, this behaviour can only appear in an ever-growing or infinite network. As the network we consider to apply this graph rewriting program is finite, we are able to ensure the termination of the strategy as shown in the following proposition.

Proposition 40 (**RP** termination)**.** *If the network is finite, the strategic rewrite program given by the rules in Figure 4.5 and Strategy 12 terminates.*

Proof. This proposition follows the property proved in Lemma 38, and the fact that the set M_k is always strictly growing with each new dissemination step k while being upper-bounded by the size of the network. □

4.3.2 Adapting the Riposte model with linear thresholds (RP-LT)

As we express the different diffusion models using a common language, i.e., our graph rewriting formalism, we are able to identify some of their key components which can then be employed to either differentiate or associate some of the models behaviours. Distinguishing the rules, in charge of local transformations, from the strategy, managing and steering their application, makes easier the comparison between the components of the different propagation models. This allows us to notice that the strategic programs implementing **RP** mainly differ in two aspects from the **IC** and **LT** models. Firstly, neighbour influence is replaced by personal interest toward the information being diffused, thus an information will be given the same consideration without indiscrimination based on its provenance. Secondly, the correlation between users activation and spread of information is mitigated through a sharing probability in **RP** whereas the diffusion is inferred by the activation in the two propagation models. While these specificities result

from distinctive characteristics of each model, the modularity of our approach offers us a great opportunity by breaking-down the models into smaller operations and giving us a more in-depth understanding of the operations being carried out.

The dissemination algorithm **RP**, unlike the propagation models presented earlier, completely ignores one of the typical features we find quite natural to consider when mentioning information diffusion, that is the neighbours' influence. In a *de facto* word-of-mouth scenario, a certain affinity must exist between two individuals before one of them accept to consider looking at any content send her/his way by the second person. This affinity can take different forms such as a personal or interpersonal appreciation of their relation or an overall reputation score for each person in the network. While **RP** rightly allows users to influence the dissemination by either endorsing or rejecting the piece of information at hand, we find this mechanism slightly too simple on its own and ill-suited to appropriately model the complex dynamics surrounding the diffusion of information. In order to address this issue, we take advantage of the adaptiveness of our approach and decide to design a new dissemination model by reusing some of the previously encountered components. The resulting model, which we name *Riposte with Linear Thresholds* (**RP-LT**), is designed to hide the users' reaction toward the information being diffused (endorsement or reject) while also taking into account the influence of the neighbour which shared the content in the activation process. This scenario is possible because, the users still know from which of their surrounding neighbours the information comes from.

We now recall the main elements needed to establish the new model, all the while 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' and its cardinality is denoted $|\overline{S_{n'}}|$.

Considering the core mechanism of the **LT** model, that is the possibility for a given node to be influenced multiple times to have a chance to activate, we cannot force the user to make a decision about the disseminated information after having heard about it only once. We thus add to our model counters tracking how many times the information has been transmitted to each node. Thus, let us define γ as the maximum number of times a node can be told an information before being asked to formulate his opinion. Of course, if a user is willing to endorse the information before his neighbours have informed her/him of it γ times, then the model validates the decision, however, a complete decline from the user is only noted once the maximum number of notification has been sent to give the chance to the user to change her/his mind. The limit imposed by γ could also be consider as way to emulate a rejection mechanism after being repeated the same information several times: if n has not been convinced the first γ times, continuingly trying to influence it is useless. In our current implementation, we consider γ as a parameter of the model and consequently offer the same number of chances to activate to all nodes, however, this limit could also be defined for each node independently.

Based on the established model description given above, we can determine the properties characterising **RP-LT**. Because the model merge elements and attributes of **LT** and **RP**, most of them are similar to the ones previously described in Propositions 28 and 35. We nonetheless recapitulate them hereafter:

Proposition 41 (**RP-LT** properties)**.** *Let $p_{n,n'}$, $p_n\big(S_n(k)\big)$, θ_n, λ, δ, γ and $\overline{S_{n'}}$ be as defined 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.*

Among all these properties, **RP-LT.3** is the only property exclusive to this dissemination model as the parameter γ is never mentioned in **RP** or **LT**. Once our implementation of **RP-LT** is complete, we will use these different properties to validate it and show our graph rewriting program behaves as expected.

Attributes

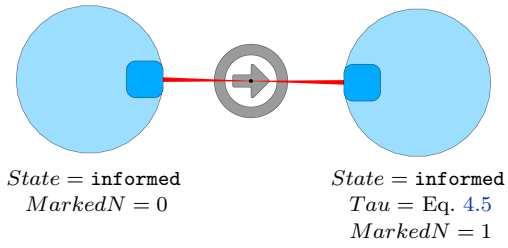
For the sake of completeness, we recall the different attributes already used in **RP** and **LT** which we are also going to employ. Obviously, we keep the general attributes: *State* and *Colour* to distinguish the nodes' states, *Marked* to mark the visited pairs of nodes, as well as *MarkedN* for 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* 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 4.4, 4.5, 4.7 and 4.8. The initialisation value for each of the attribute remains similar to ones used in the previous models.

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

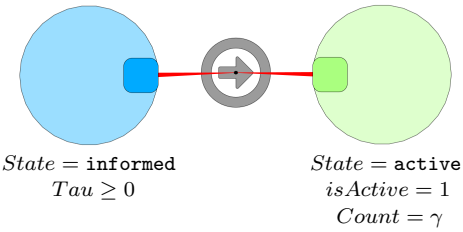
Rewrite rules

As one can expect, the rewrite rules, given in Figure 4.6, are quite similar to the **RP** rules. The first rule, named *RP-LT initialisation* (Fig. 4.6a), updates the attribute *Tau* (according to Eq. 4.5) of an `informed` node. When rewritten, the node stays `informed` and its attribute *MarkedN* is set to 1. The second rule, *RP-LT activate* (Fig. 4.6b), is in charge of the potential activations. When the attribute *Tau* 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. As node n activates, we also set the attribute *Count* to γ to indicate that the user took the decision to activate and thus that the node will no longer be responsive to the influence of its neighbours.

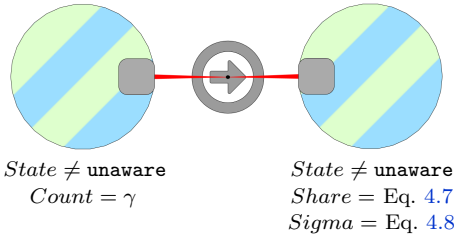
During the dissemination process, every node aware of the information being diffused who either decided to activate, or who has been fruitlessly influenced γ times to no avail, is considered to have made its mind. In this situation, we apply the *RP-LT share trial* rule (Fig. 4.6c). In order to simplify the strategy of the graph rewriting program, we filter the eligible nodes by looking at attribute *Count* to see if the node has taken a decision, in which case the value is



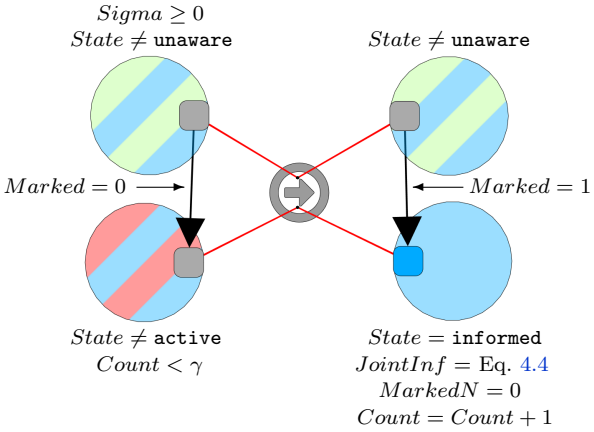
(a) *RP-LT initialisation*: the rule initialises an `informed` node, computes its attribute *Tau* and marks it using *MarkedN*.



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

Figure 4.6: Rules used to express the Riposte with Linear Threshold model (**RP-LT**). Colours keep their meaning from the previous diffusion models: `active` nodes are `green`, `informed` nodes are `blue` and `unaware` nodes are `red`. A bi-colour `blue/green` node can either be `informed` or `active` whereas a `red/blue` one is either `unaware` or `informed`.

equal to γ. The node is thus entitled to compute the values of its attribute *Share* and *Sigma* using respectively Equations 4.7 and 4.8.

The final rule we use in this model is *RP-LT inform* (Fig. 4.6d). 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 4.4 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 13. Just like the strategies used for **IC** and **LT** are comparable, the ones defining the **RP** and **RP-LT** are almost identical. We recall that, for each rule application considered hereafter, the newly rewritten elements are automatically reinserted in the position subgraph $\mathcal{P}$.

Strategy 13: RP-LT dissemination

```
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));
6   repeat(one(RP-LT inform));
7 )
```

As for the previous models, we use a repeat loop (line 1) to perform as many dissemination steps as possible. We start by selecting a single `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. 4.6a), we mark the selected `informed` node ($MarkedN = 1$), compare the *JointInf* and *Theta* attributes and store the result in *Tau* (Eq. 4.5). The value of *Tau* is used when the strategy tries to apply the second rule *RP-LT activate* (Fig. 4.6b) to activate the node (line 3). While it is only successfully applied if *Tau* is positive or null, the rule transforms the `informed` node into an `active` one, respectively modifying the values of attributes *isActive* and *Count* to reflect the node current *State* and indicate that its decision concerning the diffusion subject has been confirmed.

The application of rule *RP-LT share trial* (Fig. 4.6c) is the only difference between the instructions being used in this strategy and the ones followed by **RP** in Strategy 12. Here, the attribute *Count* restricts the rule application to nodes which have either been influenced γ times or for which the joint influence was sufficient to persuade them to endorse the propagation subject (and $Count = \gamma$ from the *RP-LT activate* rule). As a result, we cannot ensure the existence of matching elements every time; and must resort to `try` to perform the instruction; thus, only when the attribute *Count* of the node in $\mathcal{P}$ is set to γ will the rule successfully apply. In such a case, the attributes *Share* and *Sigma* are computed using respectively Equations 4.7 and 4.8.

Depending of value given by its attribute *Sigma*, the node n in $\mathcal{P}$ must share the information with its inactive neighbours n' which have been influenced less than γ times ($Count < \gamma$) and have not yet been contacted by n (the edge between them is not marked). The *RP-LT inform* rule (Fig. 4.6d) then marks the connection between n and n' and updates the `informed` node n' attributes such that: the joint influence *JointInf* is recomputed taking into account the new *Influence* of n on n' (Eq. 4.4), the influence counter *Count* is incremented to track the new influence tentative, and the marker *MarkedN* is reset to its default value, indicating that some changes have been applied to the attributes of n'. Of course, rule *RP-LT inform* is repeated as long as there is neighbours of n which have not yet been visited (provided that they are not `active` then nor that they have been already influenced γ times).

Afterwards, any of the nodes which have been influenced during this dissemination step can be considered during the next loop to test whether they are ready to activate of if they have been sufficiently influenced. If no dissemination takes place, then the set of `informed` nodes does not

grow. The process continues until all the available `informed` nodes have been considered and are thus marked (`MarkedN` = 0).

Validation and termination

With our Strategy and rules established, we are ready to validate our implementation and prove that the expected properties for **RP-LT** are respected by our strategic rewriting program.

Lemma 42 (**RP-LT.1** of Proposition 41)**.** *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 36. If n is supposed to diffuse 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 attribute *Marked*. Only neighbours which are active or have their attribute *Count* greater than γ are not concerned but these nodes are already aware of the information. □

Lemma 43 (**RP-LT.2** of Proposition 41)**.** *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 comparable to the one of Lemma 37 (using respectively rules of **RP-LT**). □

Lemma 44 (**RP-LT.3** of Proposition 41)**.** *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. However, once the node is marked in *RP-LT initialisation*, it will mandatorily try to activate, maybe disseminate information, and then will remain unchanged forever. □

Lemma 45 (**RP-LT.4** of Proposition 41)**.** *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 similar to the one given for Lemma 29 (but using *RP-LT inform* instead). □

Lemma 46 (**RP-LT.5** of Proposition 41)**.** *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 identical to the one validating Lemma 30 (with *RP-LT activate* instead). □

Lemma 47 (**RP-LT.6** of Proposition 41)**.** *The process terminates when no more diffusion is possible.*

Proof. Only the failed application of rule *RP-LT initialisation* can force Strategy 13 to come to an end. This condition occurs solely when no `informed` nodes remain or when all the *informed* nodes are marked (*MarkedN* = 1), meaning that all the nodes who could have performed a diffusion have already been considered. □

Proposition 48 (**RP-LT** implementation correctness)**.** *The dissemination algorithm defined by the rules in Figure 4.6 and Strategy 13 implements the RP-LT model as specified in Proposition 41.*

Proof. Each of the properties are proven in turn in Lemmata 42-47. □

With our adaptation complete and respecting the model properties, all that is left to do is to prove that our graph rewriting program terminates.

Proposition 49 (**RP-LT** termination)**.** *If the network is finite, the strategic rewrite program given by the rules in Figure 4.6 and Strategy 13 terminates.*

Proof. As mentioned in Lemma 47, rule *RP-LT initialisation* is the only rule application whose failure can cause Strategy 13 to stop. While the set of marked nodes M_k is growing with each successful application of *RP-LT initialisation*, rule *RP-LT inform* is also able to unmark nodes thus removing elements from M_k. In doing so however, the rule also increment attribute *Count* of the node being (re-)informed. Because Count's value is necessarily in the interval $[0, \gamma]$, each node n can only be removed from the set M_k at most γ times after which n will be finally added and never again withdrawn from M_k. The main repeat loop in Strategy 13 can thus only perform $|N| \times \gamma$ iterations at most and our program is consequently always able to come to an end. □

With this final proof, we conclude our implementation of the **RP-LT** model. Overall, the two dissemination models presented in this section show a more analytic and yet creative side of our graph rewriting formalism. While being more complex than the propagation models implemented above, **RP** also proposes a different approach by focussing on the users' interest toward the information being disseminated instead of just looking at the influence users have on one another. Of course, a user's opinion is important, but so is the confidence she/he has in the person who shared the information; this is why we have proposed to join the two methods in a single one with the **RP-LT** model. For an inactive node n', the threshold value $\theta_{n'}$, stored in *Theta*, can be seen as the resistance n' produces when facing an information presented by its neighbours. However, as somebody very curious about a piece of information is more likely to endorse it, an action only possible if the node's interest is high or if its resistance is low, and presenting either high resistance or low interest consistently means the information is quite likely to be rejected, we can consider the interest and the resistance as the two sides of the same coin, and even go as far as describe the resistance as the value measuring the opposite of interest. Consequently, our dissemination model does not simply trade of one activation mechanism for another but expend and build upon both of **LT** and **RP** techniques.

Obviously some drawbacks are also inherently bound to come with these modifications. Thus, because the diffusion process does not automatically follow an activation for **RP** and **RP-LT**, one cannot be certain that its neighbours are truly supportive of the information being disseminated. Indeed, the information actually received could also have been rejected by the neighbours while still being shared by the algorithm. This mean that uninteresting information can come from trusted sources, which could ultimately lead to users losing confidence in some of their neighbours for faulty reasons. Conversely, engaging information could also potentially comes from individuals usually considered as unreliable. However, this is in such cases that taking into account the interest (or resistance) expressed by each user is especially useful to avoid mistakenly continuing the diffusion of an unstimulating piece of information inadvertently spread by the algorithm through influential users. This property of plausible deniability brought by the model is at the same time unusual and thought-provoking and one cannot deny the significance of such feature in certain contexts where the freedom of expression is not granted as an inalienable right.

4.4 Conclusion

We have implemented in this chapter four diffusion models using our graph rewriting formalism. While the proposed translations are based on different definitions, the distinction brought by

the formalism –between the transformations to perform, which need to be precisely expressed, and the strategy managing their application– gives us the opportunity to break down the models at the lowest level and thus to identify and understand the operations being commonly used. In the previous sections, we have seized the occasion and took advantage of this mechanism in two different manners. Firstly, we have used our formalism as a basis to describe and compare two of the models in order to determine what distinguishes one from the other. The search of similarities has led to us to study in details the strategy and rules, but also the overall behaviour of the models. Secondly, after having analysed a dissemination model, we have established a new diffusion model by reusing some of the "components" proposed by other models. The resulting model presents the privacy-preserving characteristic of the initial dissemination model conjugated with the neighbourhood-based influence activation mechanism proposed in the second model.

While some visual analytic techniques have been used in this chapter to show and analyse a couple of propagation scenario, we can only deplore that diffusion processes end up looking all very much alike to one another after multiple simulations. We believe that the typical graph size used in our graph rewriting programs is not sufficiently large to present the dynamism of diffusion in all its glory. Nevertheless, now that we are familiar with different propagation and dissemination models, nothing forbids us to consider much larger graphs and to leave aside the graph rewriting formalism for a time to take a look at the diffusion behaviour on a larger scale. To appropriately visualise the resulting diffusion, we only need the right graph representation.

Chapter 5

Network visualisation using a compact overview

Contents

5.1 Displaying graphs and networks . . . 113
5.1.1 Standard solutions with respect to data types . . . 113
5.1.2 Visualisation design and task taxonomy . . . 116
5.1.3 Related works and overall visual encoding . . . 120
5.2 JASPER: a pixel-oriented overview for large graphs . . . 123
5.2.1 Phase I: layout using coarser graphs . . . 123
5.2.2 Phase II: node ordering and pixel-oriented representation . . . 126
5.2.3 Complexity analysis and execution times . . . 129
5.2.4 Resulting visualisation . . . 132
5.3 User experiment: visualisation validation . . . 138
5.3.1 Experimentation setup . . . 138
5.3.2 Experimentation results . . . 144
5.3.3 Discussion on the experimentation results . . . 146
5.4 Conclusion . . . 147

Information visualisation is defined in Ward et al. (2010) as "*the communication of information using graphical representations*". Despite its apparent vagueness, this concept is a very familiar one as a lot of different information representations are surrounding us daily and are used to visually communicate information. Such examples rank from traffic signs, imposing guidelines to respect for traffic safety, to any form of visual advertisement, promoting and suggesting the consumption of a product or a service. Even though marketing have mostly utilise such resources for commercial purposes, communication of information is an important objective to attain, especially in order to help individuals share knowledge and, ultimately, allow society to advance as a whole. Information visualisation can thus be used to represent and communicate, what we refer to as, meaningful information, that is a piece of information which present intelligent content (e.g., creative, inventive, rational, analytical).

This chapter is mostly based on:
Jason Vallet, Guy Melançon, and Bruno Pinaud. *JASPER: Just A new Space-filling and Pixel-oriented layout for large graph ovERview.* In Conference on Visualization and Data Analysis (VDA 2016), volume 2016 of Electronic Imaging, pages 1–10, San-Francisco, CA, United States, February 2016.

Because information visualisation in general is a very wide research field, we restrain our interest in this chapter to the problem at hand, that is the visualisation of graphs. More specifically, as we build upon the content of the previous chapters, we wish to find a pleasant and efficient way to represent the networks resulting from our prior contributions. Although different graph representations have been proposed during the definitions, generations and propagations to illustrate the concepts and transformations occurring at the time, it is obvious that the limited graphs' size considered then are unrealistic when compared to real networks. Additionally, the currently used representation method, showing nodes as disks and edges as lines connecting nodes together, i.e., the node-link diagram or sociogram introduced by Moreno (1934), as well as the force-directed drawing algorithms by Frick et al. (1995) and Hachul and Jünger (2005) used so far have known limitations when tackling larger graphs as shown in Ghoniem et al. (2004). We thus are in need of a solution allowing the representation of graphs composed, at the very least, of several tens of thousands elements.

Overall, visualisation has deep roots in the field of perception and cognition, as shown by Ware (2012). Vision is a great help in everyday life and a lot of information is communicated to us using visual representations, the most basic of which is quite simply the medium of *writing*, a finite collection of abstracted symbols ordered in peculiar configurations to express a meaning. This has naturally drawn the interest of a lot of researchers over the years to understand the processes followed from the visual perception to the processing of information, with studies going as far as evaluating the impact of colours, textures, motions and even memory. For a more in-depth look at all these developments, we refer the interested reader to the book of Ward et al. (2010) which discuss all of the previous aspects from an information visualisation point-of-view. However, representing and displaying information is not nearly enough. The content has to be legible and comprehensible by most, and the time needed to process it is widely expected as being much lower than the actual time needed to read a detailed description of the information we wish to communicate, otherwise, why should the observer even bother looking at it? While Fleischer and Hirsch (2001) enounce that "*people want to visualise information*", and thus, as far as we can infer, want to understand what they are visualising, a lot of obstacles are still in the way. Firstly, the visual perception ability of each human being is different. For instance, some people may not see details up close while some others may have colour vision deficiencies, thus making the task of creating a single visualisation suitable to all these persons very complex. Secondly, the unique and inherent sense of aesthetics existing for each of us can help a lot in certain contexts by engaging the observer to look at the image longer or with more focus (Smith et al., 2004), but what is appealing to some can end up being boring or even disagreeable to others. Because the field is so well-studied, several guidelines already exist and can assist when making decisions in the elaboration phases of the solution (Munzner, 2014). In the end however, the only solution to properly gauge the effect of a visualisation is simply to proceed with tests on large samples of users and perform experimentations and evaluations (Purchase, 2012).

We propose, in this chapter, to develop a novel method of representation to visualise (reasonably) large graphs. Due to our previous interest towards propagation phenomena, this method is developed with specific characteristics in mind. After an initial presentation of different existing solutions tackling problems akin to ours, and accompanied by discussions concerning their advantages and restrictions when compared to each other, we motivate and introduce our own method completed with all the design and implementation details. Despite a few accepted concessions formed due to the relatively large size of the graph we wish to visualise, we design our solution to allow the creation of a resulting visualisation in a relatively "short" amount of time. The extent of this swiftness is then proved through several benchmarks, computed to measure the time needed to execute our solution on characteristic graphs presenting different sizes, with several examples of resulting representation being also showcased at this point. Finally, and

because a quickly computed method does not guarantee its efficiency nor quality, we perform a user experiment to properly evaluate our method and check if the resulting representation is deemed "better" than the classical solutions.

5.1 Displaying graphs and networks

Information visualisation through graphs and, by extension, graph drawing, is still being the subject of extensive researches throughout the years, and this popularity has notably led to the development of a lot of different visualisation techniques. In reaction to this increasing number of solutions, articles detailing the state-of-the-art have been issued to propose a classification of these different techniques, with papers such as Herman et al. (2000) paving the way into establishing a lasting common ground for the existing layouts, navigation and interactions. For the current section, we focus our interest on a more recent report targeting large graphs in their general form (von Landesberger et al., 2011) and we consequently refrain ourselves from relating in details the other state-of-the-art papers which concentrate only on specific graphs or applications (e.g., Beck et al. (2014) about dynamic graphs, or Blascheck et al. (2017) on visualisation of eye tracking data).

5.1.1 Standard solutions with respect to data types

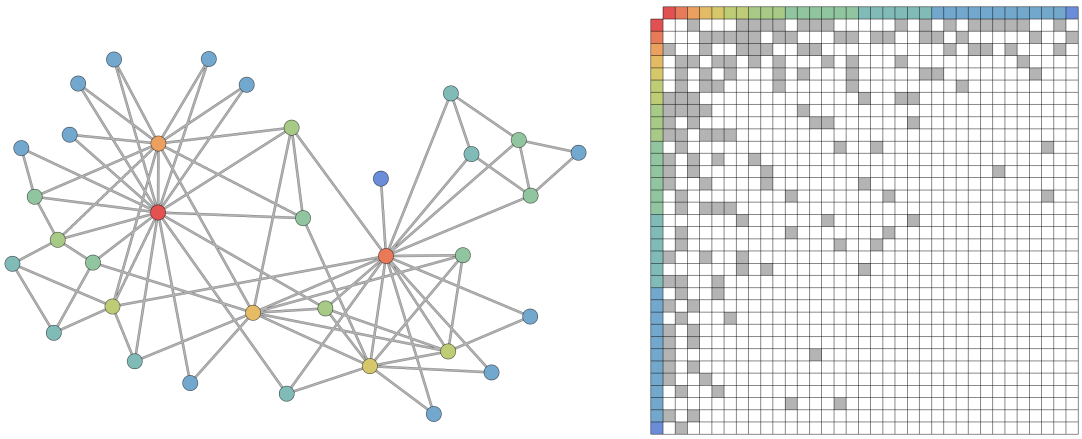
In order to find an appropriate representation for our information, the first step is to understand the type of data and dataset being used. Although we know we are planning to visualise graphs, these can come in various forms, as seen in Chapter 2. This identification step is essential as some sorts of representations are more appropriate for certain type of graphs, indeed, some traits or topological characteristics can sometimes be used to adapt the representation and make it more legible. In the state-of-the-art report of von Landesberger et al. (2011) used as a reference, graphs are classified according to their structure as either trees, graphs, and compound graphs. These last objects propose groupments of several elements in subgraphs and represent them in the graph as **meta-nodes**, with edges existing between elements inside and outside the meta-node being replaced by **meta-edges** (e.g., see Archambault et al. (2008) for additional details). This list of categories allows authors to sort and group the different visualisation solutions along the same classification. We synthesise their findings hereafter by listing, for each type of graph studied, the classification established and an example of reference proposing such solution:

Trees: node-link techniques: Herman et al. (2000);
space-filling techniques (enclosures: Van Wijk and van de Wetering (1999), adjacency: Stasko and Zhang (2000) or crossings: van Ham and van Wijk (2003));
hybrid approaches: Zhao et al. (2005);

Directed and undirected graphs: matrix representations: van Ham et al. (2009);
node-link representations (force-based layouts: Fruchterman and Reingold (1991), constraint-based layouts: Dwyer et al. (2009), multiscale approaches: Frishman and Tal (2007a), layered layouts: Bachmaier et al. (2009));
combination of matrix and node-link approach (multiple synchronised views: Henry and Fekete (2006), matrix with link overlay: Henry and Fekete (2007) or partial matrix and node-link representation: Henry et al. (2007));

Compound graphs: node-link visualisation: Archambault et al. (2008);
treemap-based visualisation: Fekete et al. (2003);
matrix view with links: Henry and Fekete (2007).

This method of categorisation is not uncommon and is also present in Chi (2000) where several visualisation techniques are sorted according to the data to visualise (including non graph related data like scientific visualisation with Card and Mackinlay (1997) and text- or spreadsheet-oriented visualisations in Chi et al. (1997)). Similarly, in Munzner (2014), the author, while discussing about the design of visualisation solutions aiming for analysis, also expresses her process as being initially defined by the type of data one wishes to visualise.



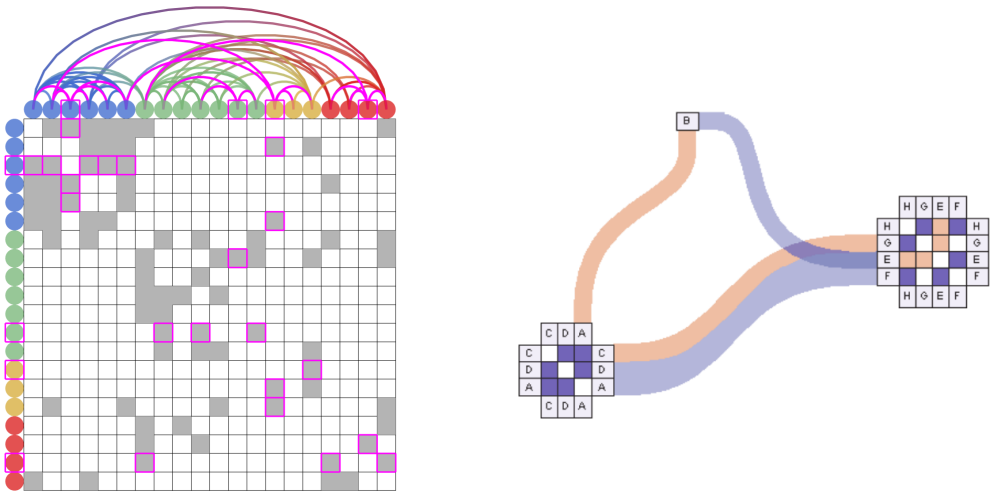
(a) Node-link diagram drawn using a force-based layout algorithm defined in Frick et al. (1995)

(b) Matrix diagram; the rows and columns are sorted accordingly to the nodes' degree.

Figure 5.1: Representations of the Karate Club network from Zachary (1977) (34 nodes, 78 edges). The colour of each node is rendered according to its degree (ascending from blue to red).

Although few variations are proposed, we can notice that the node-link and matrix representations, demonstrated in Figure 5.1, are two commonly used techniques regardless of the graph structure. This, however, comes without surprise as both techniques have been used for a long time and are quite familiar and natural representations for any graphs. We can however wonder as to why these visualisations are so widely accepted considering their respective limitations. Node-link diagrams, for instance, are well appreciated for their ability to show the global structure of a graph, i.e., the links between nodes. This peculiar trait, however, become severely diminished when dealing with larger graphs and, more particularly, when the edge density increases. User studies such as Ghoniem et al. (2005) and Henry et al. (2007) have shown that a high number of edges is an impairing factor for the graph overall readability. In the same way, matrices, which are simple to draw and display, and can be useful to appreciate the whole graph structure in one glance, are deemed nonetheless less natural to work with, and matrices of large graphs can be difficult to handle (Sansen et al., 2015). This trait is quite obviously shared by other popular approaches using any alternative sort of matrix representation such as the hybrid visualisations (e.g., Henry et al. (2007), Henry and Fekete (2007) and Rufiange et al. (2012), see examples in Figure 5.2) mentioned earlier, but the limitations appear to be to a lesser extent.

For our current use, that is the visualisation of social networks, we are interested to represent directed or undirected graphs in a specific state. This means we are, at least at first, only looking to display the state of the network at a given time, leading us to ignore the dynamicity issue for the moment. In Archambault and Purchase (2015), the authors have noticed how, despite its flaws, a node-link diagram is often preferred to represent social networks. Furthermore, while multiple layout algorithms exist for this representation (a general categorisation is given



(a) Matlink from Henry and Fekete (2007) (20 nodes, 80 edges); some of the elements are highlighted in magenta.

(b) Nodetrix, presented in Henry et al. (2007) (8 nodes, 8 edges); a few edges are highlighted in light salmon.

Figure 5.2: Examples of hybrid visualisations combining matrix and node-link representations.

in Nazemi et al. (2011)), networks are mostly displayed in small samples using force-directed layouts which seem to give rather good results on any kind of graph. Force-directed algorithms use spring embedded models (Eades, 1984) to generate a legible layout: typically, all nodes push unconnected elements away while pulling their neighbours closer. This step is repeated several times until an equilibrium is reached, which is, most of the time, decided when the layout has not changed too much between two application rounds. More details on those methods can be found in Brandes (2001). The advantage of such layouts resides in their efficiency as the spring-embedded behaviour allows isolated nodes to be pushed away easily while connected elements, which are surrounded by their neighbours, stay mostly put. However, very few articles propose extended user studies to compare the different existing graph layouts. Conclusions drawn from Pohl et al. (2009), in which the authors compare three graph layouts (orthogonal, force-directed, and hierarchical) and use eye-tracking technology to understand what makes a drawing successful, indicate that force-directed layouts give the best overall results. Furthermore, additional results from Hachul and Jünger (2007), where the authors evaluate and compare six different layout algorithms, place the force-directed FM^3 algorithm, described in Hachul and Jünger (2005), as an excellent candidate to create pleasing graph drawings, making it seemingly a perfectly appropriate solution to our visualisation problem.

Despite this optimistic foreshadowing, some issues are still present. More precisely, in recent years, the graph drawing community has focused more and more on increasingly larger graphs. This point is of particular interest to us, especially when considering the popularisation and multiplication of the ever expanding and evolving social networks. In general, graphs resulting from the mining of such large networks force us to face several hundreds of thousands (Leskovec et al., 2008), if not millions (Leskovec et al., 2010b), of nodes at once, thus pushing the existing representations, commonly used to display graphs of more moderate sizes, to their limits. Indeed, as shown in Hachul and Jünger (2007), usual force-directed algorithms can not be computed on large graphs in a small amount of time. Using the FM^3 algorithm (Hachul and Jünger, 2005) would force us to limit ourselves to work with smaller networks, which is of course unacceptable.

An appropriate solution to our problem should be able to handle large-scale datasets (with several hundreds of thousands or millions of elements) in a satisfactory amount of time.

The difficulty of displaying millions of elements in a comprehensible fashion is obvious and, consequently, existing layout algorithms are not always efficient in representing such graphs. We believe that the genericity of the considered methods is mostly to blame in this case as the proposed layouts, while adapted to our dataset type, do not truly comply to our expectations and are not completely adjusted to our need. In such case, and although an effective visual analysis can only be performed through appropriate visual representations, a good visualisation must also be designed based on relevant tasks for the targeted users. We thus propose in the following to properly establish our needs and the questions we aim to answer when using a visualisation.

5.1.2 Visualisation design and task taxonomy

When elaborating a solution, it is imperative to know what will be expected of the finished product: who will use it, in or under which conditions and to what end. While this sentence can seem to directly originate from a project manager's handbook, it is important to understand that visualisation solutions are developed to allow the representation of information. In the first place, information visualisations are used because "*they help us solve problems faster or better, or they let us learn something new*" (Ware, 2012), thus assisting us in comprehending the represented information. Of course, and although the exact definition of a good visualisation is subjected to personal preferences as everybody has a unique sense of aesthetics (Bennett et al., 2007; Smith et al., 2004) and sensibility of perception (Ware, 2012), we expect the representation to provide the information laid out in an understandable fashion. A few studies going in this direction have been performed to identify the key elements in generating a pleasing layout which maximise user understanding (Purchase, 1997; Huang et al., 2013). Even though those papers initially focus on node-link diagrams, some of the observed results can also be applied to more general visualisations. Nonetheless, no solution is truly perfect in the end and each advantage often comes as a trade-off, like how minimising the number of crossings improves the overall legibility of the graph layout but implies a much longer computation time (Vismara et al., 2000).

This notion of compromise is ever-present and need to be kept in mind but the representation must still allow an effective communication of the visualised information. To this end we establish a few needs our solution must meet from a potential user perspective:

P1 When representing propagation phenomena, the visualised (social) network must indicate the current state of the propagation: more precisely, the observers want to know which persons have already received information, which ones are currently susceptible to spread information and which ones are unaware that a propagation is taking place.

P2 Social network users are very often connected with friends, colleagues or family members, and consequently tend to be part of at least one community. Because an information spread by somebody is more likely to first register on people close to her/him, whom will latter repeat it, observers want to know if and how closely two persons are related.

P3 Following the previous points, we know that large graphs are complex to handle and represent but observers interested in propagation phenomena rarely care about complexity, only about results. Thus, *a)* the network must be available as a whole and shown as such, not partially, and *b)* the solution must return results quickly enough and not after several minutes of computation.

These guidelines are very important as they allow us to identify the observer's needs for this specific case of application. This clear announcement gives us the direction to follow in develop-

ing our visualisation solution as well as the goal to achieve in doing so. To pursue our solution elaboration, we base our design on the nested model for visualisation design and validation introduced in Munzner (2009) (later extended and completed in Meyer et al. (2012) and Munzner (2014)) and illustrated in Figure 5.3). The model proposes a nested structure where the encountered users problems define the operations to perform, which establish in turn the interactions and visual encoding to use, which finally characterise the solution implementation. To initiate the process, we establish the three guidelines (**P1**, **P2**, **P3**) given above as *characterising the problems of real-world users*. The second nested level leads us to *abstract these problems into operations on data types*. Quite obviously, our data type is a labelled directed graph where nodes represent users of the social network. We consider that all users know if they have received the information being propagated, if they are willing to spread this information further or if they have never heard of it, thus completing item **P1**. To address the real-world problem **P2**, we propose to use a community detection algorithm and communicate the results obtained to the observers. Finally, according to **P3**, we know that the graph cannot be altered and that the computation time should be acceptable (a measure we keep vague for the time being but will detail later on).

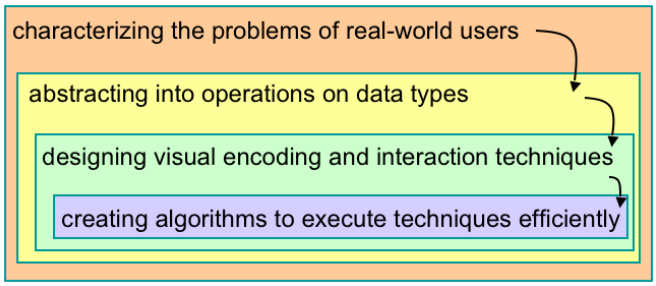


Figure 5.3: The nested four-level model for visualization design and evaluation proposed in Munzner (2009). The model indicates the step to follow when elaborating a visualisation, from the problem characterisation to the algorithm creation.

The third nested level necessitates that we design both *visual encodings* and *interaction techniques* to solve our problems. Using specific visual encodings for our nodes is a perfect way to differentiate those which have spread information from those who have not yet done so and from the others which are not aware of any information at all. While multiple options are available to us at this point, as discussed in Bertin (1983), Nazemi et al. (2011) and Munzner (2014), some solutions can be more efficient than others (Healey et al., 1993). In our case, using nodes' colours seem the most obvious and simple choice for solving both **P1** and **P2**. For instance, nodes from the same community could be of the same colour by default but their colour should change once they have received information, and once more as they start spreading it. Although the number of colours at our disposal is quite large from a technical point of view (i.e., $256^4 \approx 4.10^9$ variations for the RGB colour model with a greyscale/transparency/alpha channel), the real number of distinguishable colours at the same time is much smaller and appears to be between five and ten (Healey, 1996; Ware, 2012). A few researchers like Harrower and Brewer (2003) have produced special colour scales offering truly distinctive colours but the number of colours in such case only amounts to a dozen. We thus propose to use distinctive colours to identify communities (**P2**) and diverse transparency or greyscale levels, using the alpha channel for instance, to differentiate the nodes' state (**P1**). The more general visual encoding we need to use to represent the graph, that is the type of layout or in which fashion nodes and edges will

be displayed, is left unspecified for the moment but will be shortly addressed.

To establish the second item of the third nested level, i.e., a list of the interaction techniques we need in our solution, we can take a look at the existing task taxonomies. Several authors have established different task taxonomies but the most famous is likely the one proposed in Shneiderman (1996); it is often resumed by citing the *Visual Information Seeking Mantra* (or Schneidermann's) stating: "*Overview first, zoom and filter, then details-on-demand*". In his paper, the author describes seven high-level tasks allowing to explore the information visualised in accordance to his mantra (paraphrased from Shneiderman (1996)): *overview* of the entire data collection; *zoom* in on items of interest; *filter* out uninteresting items; select an item or group and *get details* when needed; *view relationships* among items; keep a *history* of actions to support undo replay and progressive refinement; and allow *extraction* of sub-collections and of the query parameters. Complementary references propose zoom and pan operations, focus+context techniques, visual clustering of elements (Herman et al., 2000), projection and distortion for multi-dimensional data (Keim, 2002) but also low-level (Amar et al., 2005) or topology and attribute-based tasks (Lee et al., 2006) (a more complete list with additional bibliographical references is available in von Landesberger et al. (2011)). Of course, all of the interactions above can be interesting but are they with respect to the tasks we have defined? In the case of **P1**, the Schneidermann's mantra with zoom and pan, and focus+context operations are essential to know the state of the different nodes as well as to get more details if asked for. Once joined with the visual clustering of elements or a distortion of the layout to bring together nodes belonging to the same community, these interactions would also allow to achieve **P2** easily. Finally, the creation of a representation proposing an overview of the graph and available at any time is sufficient to address **P3**, under the condition that all the computations are achieved in a reasonable amount of time. Although we do not go into to much details concerning them, it is important to note that several taxonomies dedicated to dynamic graphs are also available such as Ahn et al. (2013) or in Kerracher et al. (2015). While a propagation phenomenon is inherently dynamic, we propose to represent this evolution using small-multiple representations, where the progression is shown in successive steps (see Figure 5.4). As mentioned in Munzner (2014): "*small multiples are better than animation if equivalent information is shown (Tversky et al., 2002) and the segmentation is carefully chosen (Zacks and Tversky, 2003)*", and as showed in Robertson et al. (2008), although animations are fun to watch, small-multiple representations lead to fewer errors and appear to be more effective.

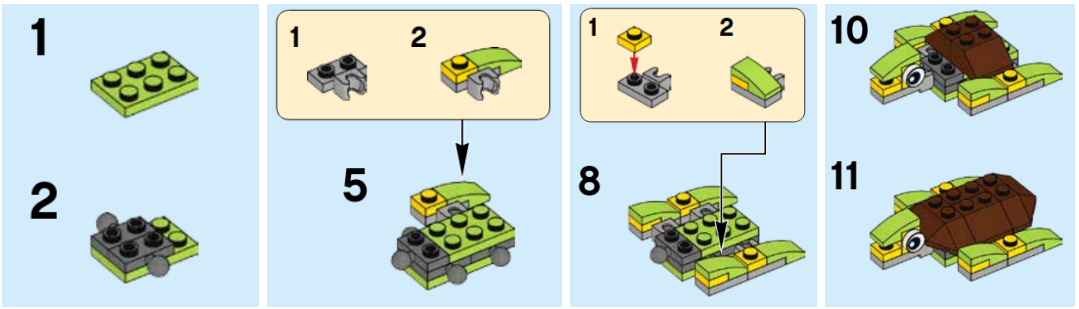


Figure 5.4: Example of small-multiple representation used for information visualisation. Small-multiples are very intuitive and are commonly used to depict instructions, as in Zacks and Tversky (2003), or constructions steps. Here, we show some partial building instructions for the set LEGO Happy Turtle (30476); instructions taken from https://www.lego.com/en-gb/service/buildinginstructions/search?search&text=30476#?text=30476.

The fourth nested level of the visualisation design is focussed on the effective implementation of the final solution. This last level is obviously linked with the second part of the problem described in **P3**: the obtaining of quick results. Once the algorithms are achieved, the different levels are finalised to form the whole solution solving the problems initially described. This distinction in defining the problem(s), the data types, the encodings/interactions and the implementation allows to properly separate the potential issues and fix them independently from one-another.

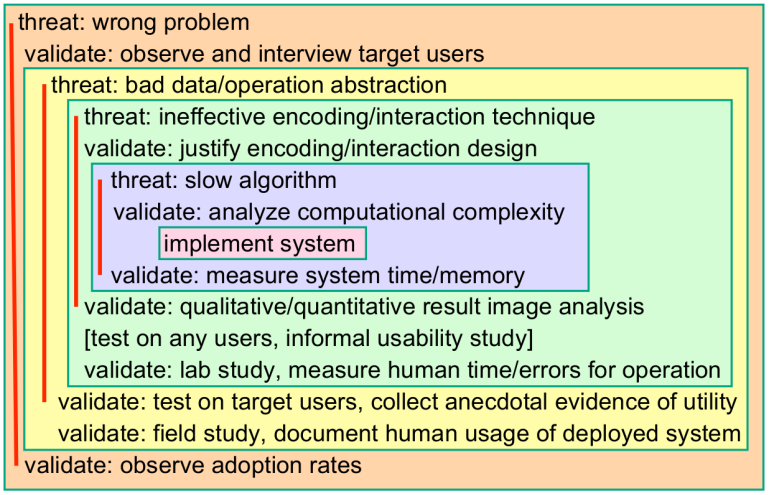


Figure 5.5: Threats and validation for each of the four levels of the nested model, as proposed in Munzner (2009). A threat is defined for each level with, sometimes, a preemptive validation step to limit errors early in the visualisation design. The threat can only be completely shed aside once the nested levels are cleared and the post-implementation validations are performed.

In complement to the nested model, Munzner propose a listing of the most common possible threats the visualisation solution may be subjected to, like a wrong problem definition or a bad data abstraction in the first or second nested level, and the validation operations to complete to ensure the quality of the final "product" (see Figure 5.5). While the low-level task validation, i.e., the implementation, requires some benchmarks and software testing, the middle-level ones ultimately need laboratory and field studies for a thorough validation of the solution (e.g., expert and user experiment (Purchase, 2012)). We keep this necessity in mind for the validation of our finalised solution, but for the moment, we are interested in the preemptive validation steps, and more precisely, the first two ones: observe and interview target users and justify encoding/interaction design. When defining our needs, we have not interviewed external users but have formulated the problems which seem relevant in our context of graph visualisation with some flavours added by the underlying idea of analysing a propagation taking place within. Consequently, the interaction and encoding designs of the elements are typically what would be expected of a standard visual analysis solution. However, what we are not sure of is the visual encoding design we need to use to represent the whole graph.

While the standard solutions such as the node-link and matrix diagrams have been used successfully in the past, we have seen that the size of the networks to visualise impede these representations, thus hindering the results legibility. In the following, we propose to take a closer look at a few existing visual encoding designs elaborated for larger graphs.

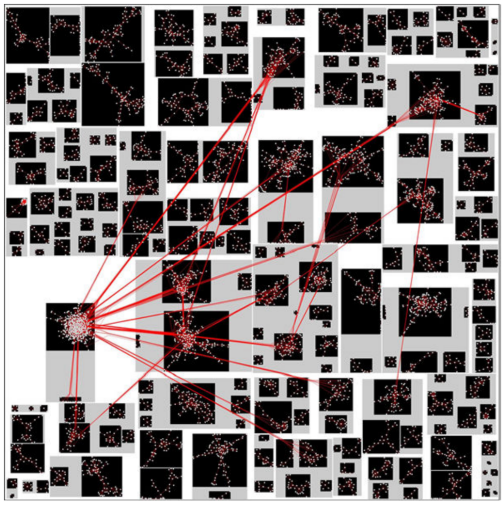
5.1.3 Related works and overall visual encoding

With our three problems **P1** (need for a visual indication of the state of each node), **P2** (need for a visual clustering of nodes closely connected) and **P3** (need for a complete, and fast, overview of the graph) properly defined, we can look through existing solutions while keeping in mind the few key characteristics our final solution must be capable of.

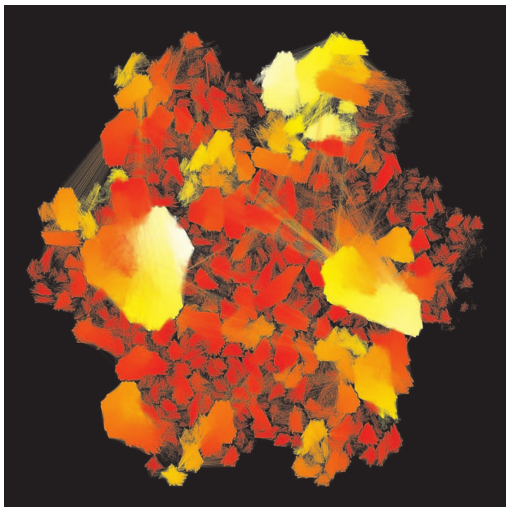
In the direct research of references proposing visualisation of propagation phenomenon using a graph representation, the results are quite limited as statistical approaches are often preferred (Wang et al., 2014) or the authors simply use standard node-link diagrams to represent the networks, even when the edge density becomes so large that it is impossible to say with certitude to which node a given edge is attached to (Lu et al., 2014). Considering the possible overwhelming number of nodes and edges in large social networks, as we may want to visualise hundred of thousands of elements, the combination of **P1** and **P3**, imposing that each node must be visible at the same time, may be hard to fulfil. Nonetheless, solutions able to conform to the problems **P1** and **P2** already exist. Three examples of such works can be found in Auber et al. (2003), Archambault et al. (2008) and Shi et al. (2009) where the authors propose multiscale visualisations of social networks. Basically, these representations decompose hierarchically the graphs into several smaller groups using nested subgraphs. This technique is particularly adapted to large graphs as one can limit the number of level to visualise, thus only displaying part of the hierarchical structure or a subset of the elements according to the hierarchical branch currently tracked. This restriction is, however, at the same time, an advantage and an inconvenient as the resulting representations can be used to visualise large graphs but still do not succeed in showing them in their entirety. These solutions are consequently unable to help us resolve **P3**.

With the multiscale solutions irrelevant for our own case, we could think ourselves back to square one, but the technique raises nonetheless an interesting point in decomposing the graph into smaller components. More precisely, if the hierarchical groups coincide with the communities existing in the network, the problem **P2** is solved. Such node grouping can also be used to simplify the layout computation in considering each cluster as a whole entity instead of trying to find an appropriate position for each node separately. This method has been used in Huang and Nguyen (2007) and Didimo and Montecchiani (2012), see Figure 5.6a, to visualise up to fifteen thousand nodes and forty thousand edges. For both resulting visualisations, all the nodes are individually visible even though the number of elements displayed is quite big. This feat is possible as the authors use space-filling layouts as a basis to order the clusters in the resulting layout, allowing them to create space-efficient representations. Such quality is obviously important when dealing with large graphs but more so in our case as we wish to isolate the colour of each displayed node. Other algorithms, for instance Muelder and Ma (2008), have proposed a similar method, by ordering the nodes along a space-filling curve as shown in Figure 5.6b. These curves –such as those described in Peano (1890), Hilbert (1891) or Morton (1966)– are specially designed to "fill" plane areas by placing the nodes at regular distances in a given order and pattern. The solution proposed in Muelder and Ma (2008) has good advantages which address **P3**: its computation is quick, the technique can scale up to display large graphs (tested by the authors using a graph with 1M nodes and almost 3M edges), and the produced layouts are, at least from our point of view, clearer than those obtained with force-directed algorithms when displaying larger graphs as a whole. Additionally, the fact that the ordering operation used to place the nodes along the space-filling curve is based on a clustering algorithm (Clauset et al., 2004) also provides us with a solution for **P2**. Nonetheless, the presence of numerous edges still impair the node visibility in the densest parts of the graph, a problem known in many visualisations (Ghoniem et al., 2005), thus offering no solutions for **P1**. A somewhat similar solution has also been proposed in Auber et al. (2013) only using hierarchical data, and consequently making it inappropriate for general

network visualisation. In this later case however, edges are no longer needed as the hierarchy is indicated using coloured nested regions and borders, dividing the layout like a nested Tree-Map (Johnson and Shneiderman, 1991).



(a) Hierarchically clustered drawing of a social network from Didimo and Montecchiani (2012) (5,835 nodes, 13,815 edges). The 286 clusters have been computed using Louvain's algorithm (Blondel et al., 2008).

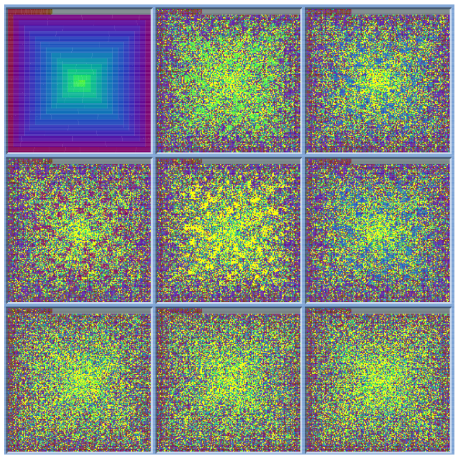


(b) Layout obtained using Muelder and Ma (2008) approach (28,854 nodes, 1,180,816 edges). The representation is based on a Peano-Gosper curve (Mandelbrot, 1977), sometimes called a flow-snake

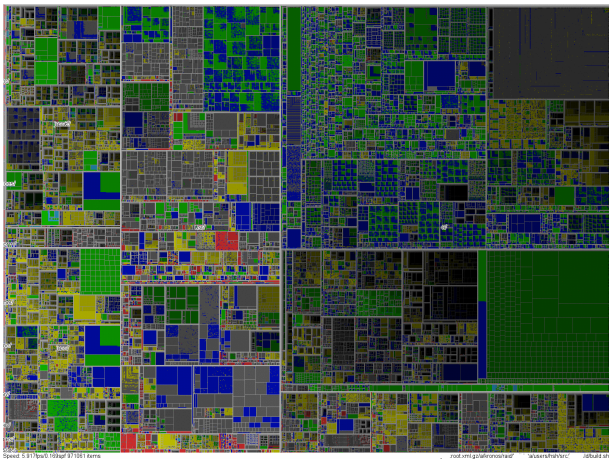
Figure 5.6: Examples of representations maximising the use of space to display information.

The idea of using space-filling curves is elegant and allows to efficiently use space. This can however be pushed even further as demonstrated with pixel-oriented visualisations, initially introduced in Keim (1996), see Figure 5.7a. This method is commonly used to analyse records and visualise the possible correlations between two measures, one being displayed using the position of the elements along a space-filling curve while the second is depicted using a simple colour mapping; whenever the two analysed measures are correlated, peculiar colour patterns, e.g., chunks of the same colour, tend to appear in the representation. Using this kind of visualisation would give us the capacity to effectively display all the nodes at the same time (achieving **P1** and **P3**), however, edges –carrying information on the connectedness of each node in the network– cannot be explicitly represented using a pixel-oriented method. Duarte et al. (2014) have encountered a similar problem and addressed it quite effectively. Their *Nmap* layout is introduced as being a neighbourhood preservation space-filling algorithm able to display connected nodes close to each other. The resulting visualisation is rather reminiscent of the existing Tree-Maps (Johnson and Shneiderman, 1991; Van Wijk and van de Wetering, 1999; Schreck et al., 2006) which have been successfully used to visualise very large datasets containing up to a million of elements (Fekete and Plaisant, 2002), see Figure 5.7b, and to perform social network analysis (Stein et al., 2010; Gove et al., 2011). By achieving this spatial proximity in the representation to indicate the connection of elements from a structural point of view, as expected for **P2**, the edges do not need to be drawn as their existence is hinted by the nodes adjacency in the visualisation.

As mentioned above, considering the number of elements we wish to visualise at the same time, some compromises are bound to be necessary. By representing the information in its simplest form, where a node is a pixel, we are able to represent a lot of elements at the same time,



(a) Pixel-oriented visualisation from Keim (1996) (24,000 elements with 8 variables). These representations are ideal to find correlations between variables: the elements are coloured according one variable and ordered according to another.



(b) Tree-Map (Johnson and Shneiderman, 1991) of a file system from Fekete and Plaisant (2002) (971,061 files). The area of each rectangle is specified according to the file's size it represents, while the colour indicate the file's type. A grey-scale is also used to make the deeply nested directories appear darker.

Figure 5.7: Examples of visualisations using pixel-oriented representations.

and by changing the pixel's colour, one can visualise the state of the element being represented, thus achieving **P1** and the overview mentioned in **P3**. Although our solution does not allow us to directly display the edges, we can use interactors to obtain that piece of information. A complementary way we introduce is to also use a visual metaphor to suggest the presence of existing connections directly in the visualisation. The way we propose to achieve this, and solve **P2** at the same time, is by using the layout of the nodes to cluster elements of the same communities, thus suggesting that two nodes next to each other in the representation are more likely to be connected than any other two nodes which are not neighbours. To this end, a layout algorithm offering to keep each node close to its neighbours, even if only approximately, seems to be a good concession, and a small price to pay for visualising the numerous nodes without the hindrance of overlapping elements. Though it can not be applied on a smaller scale to obtain an exact representation, the described solution seems to present sufficient qualities to be used to compute overviews on larger graphs. Furthermore, it is important to note that the representation method we have chosen here is not without drawbacks; in particular, there are some limitations for observers whom might wish to visualise propagation phenomenon in dynamic graphs with evolving topologies. This operation is beyond the scope of our current application at the moment as we are using snapshots of social networks in a fixed configuration to study the evolving transmission of information between users. It will nonetheless become relevant in future work as the structure of a social network is expected to develop and transform. In this thesis, we limit our visualisation effort on large static graphs and the state modifications happening to the different nodes.

While multiple solutions exist to visualise graphs, finding the one which is appropriate to address the problems at hand can be difficult. In addition to the representation and the design of the elements visual encoding, interactions can also be used to extract further information from the visualisation. It then becomes important to characterise which part of the solution will

help to resolve which question and to keep the solution as straightforward as possible to avoid unnecessary complex operations. We now propose to describe how we built our visualisation.

5.2 JASPER: a pixel-oriented overview for large graphs

After the definition of the three problems we need to address with our visualisation and an overview of the different algorithms, interactions and methods at our disposal to do so, only half the work is done. We present in the following the implementation details of our visualisation, and in order to be thorough in the design and verification of our solution, follow the recommendation of Munzner (2009) about the necessary algorithm validations shown in Figure 5.5. We thus accompany our solution with both a complexity analysis as well as some benchmarks on graphs of different sizes to better grasp the limit of our approach and more particularly to identify the potential bottlenecks existing in our solution and preventing its scalability.

We present in Figure 5.8 the workflow followed by our solution. As explained previously, we have drawn inspiration from several already existing visualisations to find characteristics and techniques to help us better represent the information. Our solution, which we have named JASPER (for *Just A new Space-filling Pixel-oriented layout for large graph ovERview*) is divided in two main phases. We first compute a layout based on a coarse representation of the initial graph. This step allows to improve the computation time as we limit the number of elements, thus simplifying the graph to produce. We then reorganise all the nodes along a space-filling curve, with the ordering of each node on the curve being based on its spatial position in the previous layout. The resulting layout is a space-filling representation with similitudes to pixel-oriented layout (only the nodes are displayed and existing edges are invisible by default, see Fig. 5.7a). We end up with a compact visualisation where nodes belonging to the same connected group are set to be displayed spatially close to each other.

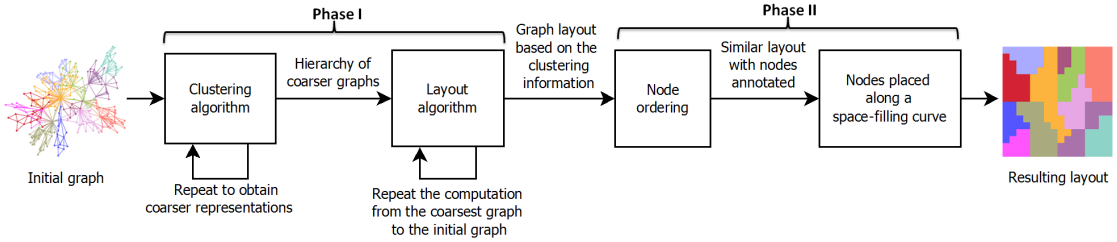


Figure 5.8: Workflow showing how operations are pipelined in JASPER, starting from a graph with an unspecified layout and returning the same graph with a node arrangement similar to a pixel-oriented layout.

In this following, we take a more in-depth look into the operations performed during both phases of the workflow. We also propose an analysis of the solution complexity, completed with a few benchmarks, as well as several example of resulting representations.

5.2.1 Phase I: layout using coarser graphs

As we are tackling larger graphs, we know that the usual force-directed layouts will take a long time to compute. However, if we were able to somehow use instead a less detailed version of the initial graph by keeping only the most important nodes and edges, we could compute what we believe to be a quality layout more quickly. This less detailed, or cruder, version of the initial

graph is sometimes mentioned as a coarser graph or a skeleton graph. This method allows to describe an initial graph by a much simpler representation, keeping only its backbone, with the different nodes being regrouped according to a specific parameter. Similar process can be found in different layout algorithms and analysis techniques such as Auber et al. (2003), Frishman and Tal (2007b), Itoh et al. (2009), Didimo and Montecchiani (2012), and Nick et al. (2013). We present in Figure 5.9 an illustration of the different steps undertaken during this first phase.

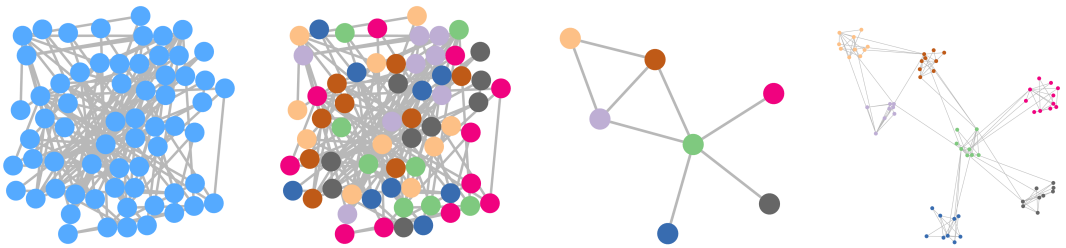


Figure 5.9: Intermediate resulting representations obtained on an example graph (64 nodes, 125 edges) during the different steps of Phase I.

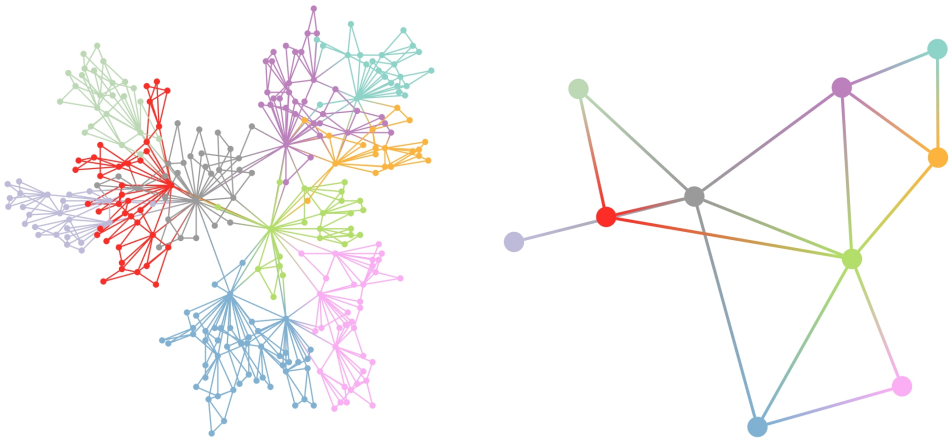
Grouping nodes

The starting point of our method is the application of a clustering algorithm on the initial graph. Typically, clustering algorithms aims at finding the "*decomposition of a set of entities into 'natural groups'*" (Brandes and Erlebach, 2005) and are consequently used to detect clusters or groups in a graph. In our current case, we are looking for clustering algorithms prompt at describing a subgroup of nodes particularly connected to each other; as we have seen before, in graphs representing interactions between persons such as social networks, those groups are often called communities. The subject is further developed by Fortunato (2010) where an overview of different existing clustering algorithms is presented and several of those techniques are compared. According to the author, the Louvain algorithm proposed by Blondel et al. (2008) and Infomap of Rosvall and Bergstrom (2008) provide the overall best results. However, the execution time on large graphs provided by Blondel et al. (2008), further completed with the additional support expressed in Didimo and Montecchiani (2012) toward this algorithm, showing the prominence of Louvain's clustering technique, have convinced us to opt for this solution in our method.

As shown in the example presented in Figure 5.9, the nodes are considered homogeneous in the initial graph (all nodes are of the same colour). Once we apply the clustering algorithm on a graph, each node will be put in the same subgroup as their closer neighbours (we colour nodes of the same group similarly to identify them). We naturally wish for each node to be regrouped only with its closest neighbours when displayed. To achieve this task, we entirely rely on the clusters quality –thus on the clustering algorithm efficiency– to appropriately regroup each node with its proper and closest relatives.

Building on clusters

Once each node is set in the adequate cluster with its closest neighbours, we have information on how to group users together. We start working on the initial graph to identify the different clusters and create a map of their relations. We thus obtain a first coarse graph. Repeating



(a) Initial graph G_0; each of the clusters in C_0 is coloured differently to facilitate their identification.

(b) Coarser representation G_1; each coarse representative is displayed using the same colour as its cluster.

Figure 5.10: Example of a coarsening process on a simple graph divided in 10 clusters.

those operations several times produces coarser and coarser representations. Figure 5.10 shows an example of such treatment on a small graph. Starting with an initial graph (Fig. 5.10a), a computation of a coarser representation (Fig. 5.10b) gives us the connections between the clusters and results in a simpler graph. More generally, upon each computation of a coarser graph, we first group close nodes together then communities linked to one another. We use the groupings herein created to decide how close two users should be displayed.

Using a more formal approach, coarser representations can be described as follows. For a graph $G_k = (N_k, E_k, \mathscr{E})$ identified by a set of nodes N_k and a set of edges E_k, where $k \geq 0$; we define G_0 as the initial graph we want to visualise and $G_1, G_2, \ldots$ as its gradually coarser representations. By applying a clustering algorithm on G_i, where $i \geq 0$, a set of clusters C_i is created such as each node $n \in N_i$ belongs to one and only one cluster $c \in C_i$ (described as $cluster(n) = c$). Those groupings are then translated to a coarser representation of the graph, *i.e.*, for each cluster c in G_i, a node m is created in a new graph G_{i+1} such as $m \in N_{i+1}$ and, if an edge $e \in E_i$ exists between n and $n' \in N_i$, we update G_{i+1} following those two cases:

a) n and n' are in distinct clusters ($cluster(n) \neq cluster(n')$), then, an edge $\epsilon \in E_{i+1}$ is set between the two nodes m and $m' \in N_{i+1}$ –respectively representing $cluster(n)$ and $cluster(n')$. When an edge linking m and m' already exists, a weight can be set and updated to indicate the multiple connections between the two clusters.

b) n and n' are in the same cluster c ($cluster(n) = cluster(n') = c$), then no transformation is performed in graph G_{i+1}.

Layout computation

Once we have achieved the computation of the wished number of coarse representations, a final layout for Phase I can obtained through successive steps. After a layout computation of the coarsest graph G_p, we gradually use the layout computed on the clusters in G_i (where $p \geq i > 0$) to place the subgroups in G_{i-1}. The operation is repeated until G_0 is reached and treated.

To perform the first layout computation, we have to choose an efficient layout algorithm offering legible drawings in a small amount of time. In its survey on spring embedders and force-directed layouts, Kobourov (2012) mentions some possible candidates for such application. Ultimately, as we had mentioned its qualities before, we have selected the Fast Multipole Multilevel Method (FM^3) introduced by Hachul and Jünger (2005). The choice has been motivated based on the results showed by Hachul and Jünger (2007) and Archambault et al. (2007) as this layout method proposes a good balance between execution time and drawing quality with minimal edge-crossings and overlapping edges. Those two points are essentials as we wish to keep a readable layout of the clusters to easily distinguish one from another. An additional perk of the layout resides in its weighted variant offering advanced control if needed.

Computing the layout of the coarsest graph G_p is straightforward as it simply requires to apply the algorithm layout on G_p. The layout of any of the following coarser graphs G_i, where $i < p$, needs a few additional steps. First, a local layout is computed in each existing cluster $c \in C_i$ by only considering the corresponding subgraphs. Then, freshly computed coordinates of each node $n \in N_i$ attached to the cluster c (with $cluster(n) = c$) are transformed through translation and scaling so that all nodes in c are displayed in a (small) area as defined by $m \in N_{i+1}$, representing c in the coarser graph G_{i+1}. Each node coordinate in a coarser graph G_{i+1} is thus used as an anchoring point to layout the nodes of G_i. Those two steps are repeated for decreasing values of i until we reach G_0 and have computed the coordinates of each initial node.

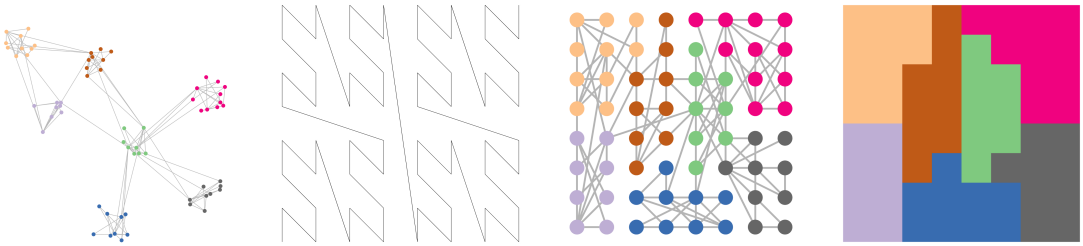
Where classical spring-embedded or force-directed layouts are time-consuming to compute when facing thousands of elements (Archambault et al., 2007), this nested method allows to compute several small layouts quickly. Furthermore, those operations and the geometrical transformations affecting node coordinates can be done independently from one another and thus be easily distributed on supporting systems. Furthermore, as an additional possible performance improvement we have observed on the testing graphs presented in the following, computing the inner layout of each cluster is not always the best choice as connections with nodes from outside the subgroup will not be accurately considered. In most cases where nodes within clusters are densely connected, a simple random layout gives acceptable results and allows to speed up the application by averting the nested layouts computations.

On a more general note concerning the whole Phase I, the rapidly decreasing number of nodes for each coarser representation makes the construction of G_{i+1} and each application of the clustering and layout algorithms on it quicker. This is not surprising considering the graphs obtained from the clusters are getting simpler, however, based on our observations on the testing graphs, repeated applications to obtain several levels of coarse graphs do not necessarily improve the resulting layouts. As a result, in our applications, one coarser representation (G_1) has proved to be sufficient more often than not, with a second level of coarse representation only appearing to be of interest when used on the larger graphs. We offer nonetheless the general method as such, considering structured graphs with established hierarchical communities will benefit from it. In any case, the choice of the number of coarse level is left to be decided upon application.

5.2.2 Phase II: node ordering and pixel-oriented representation

Pixel-oriented visualisations have been presented in Keim (1996) as a representation allowing to display important quantities of data in a minimal space. To do so, elements one wishes to display are ordered and placed along a space-filling curve. Our method uses an analogous process with nodes being ordered according to their spatial position computed by the end of Phase I. We divide the space using a technique similar to the one used to create k-d trees. Once each node is isolated from its neighbours, they are ordered according to a space-filling pattern and

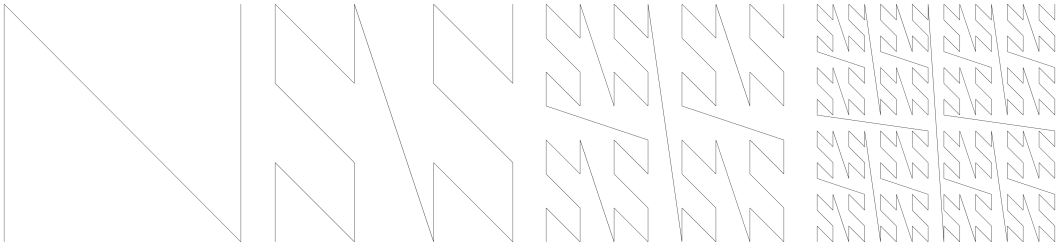
are laid on the corresponding curve at their given position. The size and shape of the nodes are maximised and the edges are finally hidden to obtain the maximum legibility and avoid overlapping elements. We show in Figure 5.11 the continuation of the example started in Phase I with the different steps performed during this second phase.



Intermediate layout used for space division | Space-filling curve used for ordering | Nodes positioned on the space-filling curve | Resulting layout

Figure 5.11: Intermediate resulting representations obtained on an example graph (64 nodes, 125 edges) during the different steps of Phase II.

During the elaboration of this Phase, we had to agree on the appropriate model of space-filling curve our solution could use. As mentioned earlier, there is several possibilities at hand like those described in Peano (1890) or Hilbert (1891). We propose to use the simple space-filling curve popularised by Morton (1966). Sometimes called Z-order curve, we will however prefer the original layout using an N shape in our application as shown in Figure 5.12. The choice of the space-filling curve is rather important as it indicates how the elements are to be divided and ultimately ordered.



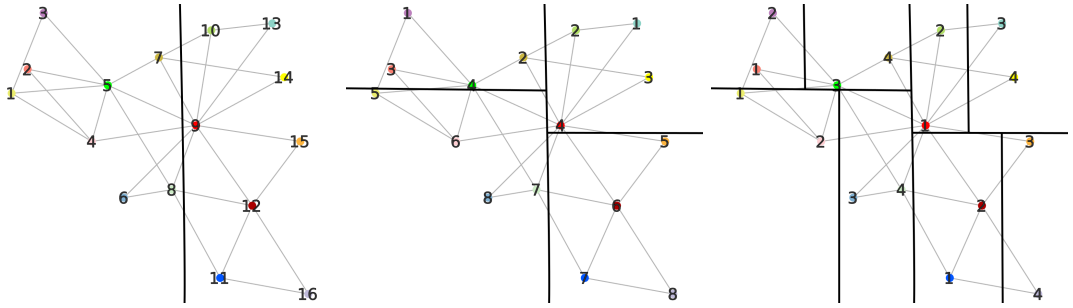
(a) Level 1 (4 nodes) | (b) Level 2 (16 nodes) | (c) Level 3 (64 nodes) | (d) Level 4 (256 nodes)

Figure 5.12: Successive developments of the N-order space-filling curve introduced by Morton (1966). The layout is similar to a Z-order curve but with a 90 degrees rotation. Each level δ of the curve contains 4^δ nodes and $4^\delta - 1$ edges.

Space division and node ordering

The method we come up with to achieve the node ordering can be computed fairly easily. We use a technique similar to the one employed for k-d tree construction (Bentley, 1975). A k-d tree is a data structure used for the storage of k-dimensional data, however, in our case, the only dimensions considered are those specified with the nodes layout position (x and y coordinates). We first divide the graph in two parts with a similar number of nodes in each halves using a

pivot value on the first dimension (e.g., x). The two resulting halves are then divided again in two equal parts but using a pivot value on the second dimension. This process is repeated while alternating dimensions until each node is isolated and thus is given an order; the first steps of the process are demonstrated in Figure 5.13. As we know the size of the graph, we can compute from the start how many divisions will be necessary to isolate all the nodes. However, if we wish to conserve a regular shape for the layout, as it is our case, the number of division have to be equal for each halves. This means regions with fewer nodes have to incorporate "holes", that is white-spaces which are given an order too. Moreover, because the number of nodes is equally distributed in each halves during the division, the "holes" are evenly spread throughout the whole space-filling representation.



(a) First division performed on x (b) Second division done on y (c) Third division on x once more

Figure 5.13: First iterations of space division on a graph with 16 nodes. Each time, the nodes are ordered (in their set) according to the dimension considered during the step, and are divided in two subsets of equal sizes –or approximately if the number of nodes is odd. The other dimension is then chosen and the division is performed again. These operations are repeated until each node is isolated in a subset.

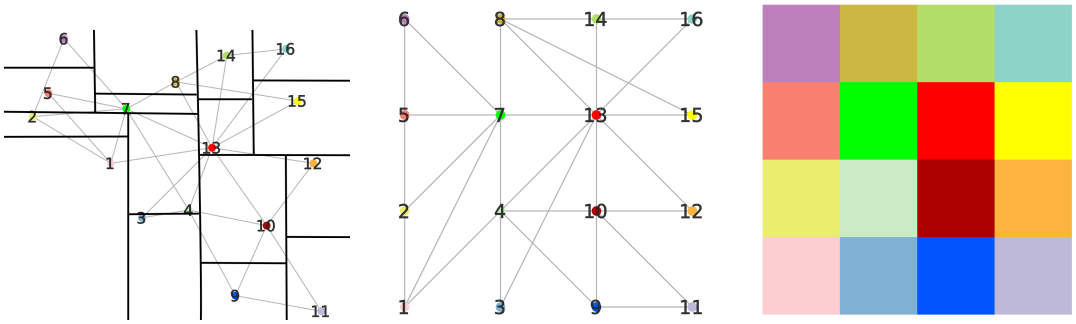
To avoid creating representations with too many white spaces, we suggest to use a variable shape for the representation such that the resulting shape of the layout will be either a square or a rectangle depending of the number of nodes to display. While our ideal visualisation ratio is set to be 1:1, a square is appropriate. However, the successive divisions and the rearrangement along a space-filling curve –explained hereafter– do not give us the possibility to freely rearrange nodes. The appearance of these white spaces is noticeable on representations with a limited number of nodes, as depicted on Figure 5.15c, but becomes less visible when facing graphs with more nodes, like the one in Figure 5.16b. A variable layout shape allows us to avoid incorporating too many "holes" and to use most of the space available by following a simple rule: let $n \in \mathbb{N}$ and $|N|$ the number of nodes in the graph to display, the resulting shape will be a square if $2 \times 4^{n-1} \leq |N| < 4^n$ and a rectangle otherwise (when $4^n < |N| \leq 2 \times 4^n$). We test the number of nodes during the first space division (e.g., on the x coordinate) to either spread the data on a square or on half that surface, i.e., a rectangle. This is implemented by either using the whole space-filling curve or only half of it if the number of nodes allows it.

Placement along a space-filling curve

Following the subsets computed during the space division operation, we can order the nodes. As shown in Figure 5.14a, one can see how the successive divisions have created a sort of tilled assemblage, somewhat reminiscent of some of the compositions by Piet Mondrian.[1] The space

[1] https://en.wikipedia.org/wiki/Piet_Mondrian

divisions along x and y give an appropriate order for each node to be displayed along our space-filling curve as we establish a correspondence between each tile and the points on the Morton curve. Once ordered, the nodes can be laid out according to the curve shape, as in Figure 5.14b, and finally be reshaped and resized to obtain a more evenly tilled depiction akin to a pixel oriented representation (see Fig. 5.14c).



(a) Once nodes are isolated, they can be ordered following the space-filling curve shape.

(b) Ordered nodes are repositioned according to the N-order space-filling curve layout.

(c) The size of each node is increased and its shape is changed to create a pixel-oriented layout.

Figure 5.14: Nodes' ordering and placement along the space-filling curve.

Similar results can be obtained when using different curve designs or orientations –such as the Hilbert or the Z-order curves– as long as the node ordering during the division respects the path followed by the curve. This last point is crucial to conserve adjacency in the final layout, otherwise, nodes which should be close end up being separated if the space division and reconstruction do not follow the same pattern.

As shown in Figures 5.11 and 5.14, the successive application of these two phases allows us to create a compact representation where nodes connected to each other end up in the same vicinity. While the solution seem to be working according to our expectations on the smaller graphs, let us now see how it copes on real world cases.

5.2.3 Complexity analysis and execution times

After having introduced the algorithm followed by our solution, we now propose to study the complexity of its implementation. Despite the good resulting representations obtained so far on the small examples, we are not entirely sure of the effectiveness of our solution once applied on much larger graphs. As we have seen previously, network visualisations with several hundred of thousands or even millions of elements are hard to compute and some of the existing force-directed layouts for instance can take from a few minutes to several hours to compute (Hachul and Jünger, 2007). Although, in the end, we believe the execution time to be a much more valuable information from the point of view of the person who will use the algorithm to compute representations, we first give in the following the complexity analysis of our solution to observe its theoretical scalability.

We stay in the context previously introduced and keep the algorithm separated in two Phases. Following the notation developed earlier, let $|N_0|$ and $|E_0|$ be respectively the number of nodes and edges of the initial graph G_0, $|N_1|$ the number of clusters computed from G_0 (equal to the number of nodes from G_1) and $|E_1|$ the number of edges connecting those clusters (number of

edges in G_1). As we find that using more than one level does not significantly improve the final layout and to avoid performing an overextensive analysis, we analyse the solution complexity when computing only one coarse representation (G_1) from the initial graph.

	Phase I: coarser graphs		
Step	Clustering	Coarse graph construction	Layout computation
Complexity	$O(\|E_0\|log\|E_0\|)$	$O(\|E_0\|)$	$O(\|N_1\|log\|N_1\| + \|E_1\|)$

	Phase II: pixel-oriented representation	
Step	Node ordering	Placement on the curve
Complexity	$O\big(\|N_0\|(log\|N_0\|)^2\big)$	$O(\|N_0\|log\|N_0\|)$

Table 5.1: Complexity of Phase I and II of JASPER applied on a graph G using one level of coarsening. $|N_0|$ and $|E_0|$ define respectively the number of nodes and edges of the initial graph G_0 while $|N_1|$ expresses the number of clusters computed from G_0 (equivalent to the number of nodes of G_1) and $|E_1|$ designates the edges connecting those clusters (similar to the number of edges of G_1).

The details for the results are given in Table 5.1 for each of the main steps existing in Phase I and II. The first measure concerns the Louvain clustering algorithm (Blondel et al., 2008). While generalised versions (Meo et al., 2011) or parallel implementations (Que et al., 2015) of this algorithm exist, we use the version updated by the authors[2] and implemented in TULIP. The authors do not give any exact information concerning the complexity of their algorithm, only hinting that the execution on large graphs suggest the method to be linear as the modularity is easy to compute and tends to converge really fast. As we are interested in the worst case complexity, we opt for a more conservative opinion in comparison to greedy optimisation methods and use instead a near-linear time complexity in the number of edges: $O(|E_0|log|E_0|)$. Once the clustering achieved, the coarse graph G_1 is built in linear time and its layout is computed using the force-directed layout FM^3, as implemented in OGDF,[3] with a worst-case running time of $O(|N_1|log|N_1| + |E_1|)$ (Hachul and Jünger, 2007). In the second Phase, the node ordering operation presents the same complexity as a k-d tree construction using a sort algorithm with a linearithmic complexity[4] on each space division. Lastly, the placement of the nodes on the curve can either be performed by looking for the nodes in the k-d tree but also be performed right after a node receives its order.

We define the overall worst-case time complexity to be $O\big(|N_0|(log|N_0|)^2 + |E_0|log|E_0|\big)$ with the first Phase being performed in $O(|N_1|log|N_1| + |E_1| + |E_0|log|E_0|)$ while the second phase is completed in $O\big(|N_0|(log|N_0|)^2\big)$. To simplify the expression, we have maximised $|N_1|$ and $|E_1|$ to $|N_0|$ and $|E_0|$ respectively. We learn from this worst-case scenario that, although both the number of edges and nodes impact the algorithm efficiency and the complexity is not linear, neither is it quadratic. However, because the number of edges is always bigger than the number of nodes in a network, sometimes up to an order of magnitude, we can predict that the number of edges, and more precisely, the part of the solution whose complexity is impacted by it, i.e., the clustering algorithm, will be our pitfall in terms of efficiency. The time complexity values given above concern non-parallel executions but due to the current operating systems abilities and processor architectures, we hardly recommend the use of an exclusively single-core implementation when

[2] https://sites.google.com/site/findcommunities/
[3] http://www.ogdf.net/
[4] As implemented in the standard C++ library: www.cplusplus.com/reference/algorithm/sort

the parallelisation of instructions can be achieved easily with compiler tools like OpenMP.[5] While we do not propose a complexity analysis of our solution on multithreaded architecture, several tasks in our implementation have been developed to allow OpenMP to take advantage of such available resources, thus ultimately improving the computation times in comparison to the performances one could have expected from the complexity analysis. These improvements, unfortunately, cannot be used to enhance either the clustering algorithm nor the force-directed layout computation.

	Graph size		Time (seconds)				
Dataset	$\|N\|$	$\|E\|$	Phase I*		Phase II	Total	σ
email-Enron†	36,692	367,662	.23	.35	.27	.85	.02
soc-Slashdot0922†	82,168	948,464	.77	1.16	.59	2.52	.04
com-DBLP†	317,080	1,049,866	3.11	1.32	2.37	6.80	.16
com-Youtube⋆	1,134,890	2,987,624	8.35	5.87	9.08	23.30	.64
wiki-Talk‡	2,394,385	5,021,410	9.84	4.18	20.24	34.26	1.64
com-LiveJournal⋆	3,997,962	34,681,189	190.59	20.78	33.79	245.16	5.97

Table 5.2: Presentation of the execution times for JASPER. The two first columns express the number of nodes $|N|$ and edges $|E|$ of each graph and the five last columns give the time measures. Each time value is an arithmetic mean measured on 10 runs and expressed in seconds. *Phase I** details both the time for the clustering algorithm execution and the rest of the Phase I operations. *Phase II* corresponds to the whole time needed for the second Phase (pixel-oriented layout computation). *Total* indicates the total execution time, sum of the previous values, and σ gives the standard deviation of the *Total* value.
†: Leskovec et al. (2008); ⋆: Yang and Leskovec (2012); ‡: Leskovec et al. (2010a,b)

To test our implementation, we have chosen a few real-world graphs of different sizes, ranging from hundreds of thousands of elements to several millions as initially targeted. We have completed the time measurement by the addition of a much larger graph, counting four millions nodes and almost thirty five millions edges, to test the limit of our solution. The datasets used are all freely available online on the *Stanford Network Analysis Project* website.[6] We show in Table 5.2 the execution times obtained when using our solution. The implementation and running time measurements have been done on a computer using an Intel i7-3840QM processor (2.8GHz, 4 hyper-threaded cores) with 32GB of RAM, a NVidia K2000M, running a Linux distribution (Ubuntu 14.04) and the TULIP visualisation software in its latest version at the time (4.8). The time measurements have been performed with the Boost library (`http://www.boost.org` – version 1.55) and consider the real process time execution (processor time used exclusively for the program execution[7]). All the graphs can easily fit in random-access memory during computation as less than 8GB are used on the testing machine at all time. The graphic rendering and display times are not included in the measures, thus only concerning the layout computation. As an informal measure, we note that, once computed, the *wiki-Talk* layout is typically displayed in less than 20s. However, it is important to note that displaying the larger graphs has a huge impact on the amount of memory used; this is due to the OpenGL elements used to draw the graph in TULIP. For instance, the full rendering of the *com-LiveJournal* dataset takes around 25GB of memory.

[5] An API for multi-platform parallel programming in C/C++: `http://www.openmp.org/`
[6] `http://snap.stanford.edu`
[7] Corresponds to the *process_real_cpu_clock* from the Boost library

The results presented in Table 5.2 are average values obtained after 10 runs for each dataset. As we use the existing implementations of Louvain and FM^3 available in Tulip, a (very) small overhead due to the inner mechanisms of the framework is added to our workflow. Nonetheless, we end up being able to compute the layout on relatively large graphs (2M nodes and 5M edges) in roughly 35 seconds. The table shows some details concerning the time needed for intermediate computations as well. A point of particular interest to us is to observe the time required by the clustering algorithm . While it is responsible on average for 30% of the total computation time on the first datasets, the algorithm becomes accountable for around 75% of the computation time needed for the last graph. Thus, as expected, the clustering algorithm impacts significantly the rapidity of the solution, especially when dealing with larger and denser graphs. On the other hand, the layout algorithm (FM^3) executed on the coarse graph presents a relatively small workload (15% in the worst case) in comparison; its execution time is measured as a part of the second column of Phase I. Finally, we can see that the execution time of Phase II scale as anticipated, i.e., it seems to almost linearly with the number of nodes.

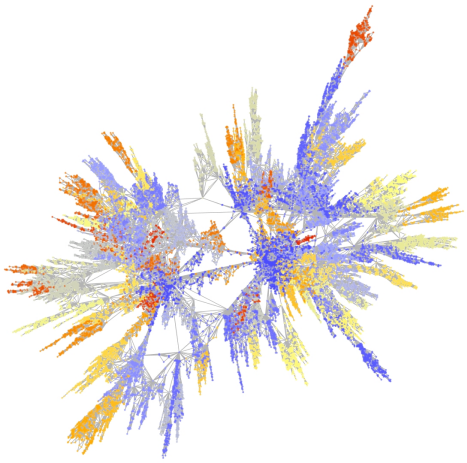
As planned, the results for some steps based on the previously announced complexities are different, the information given earlier being only valid for single threaded executions. Thus, both the space-filling and node ordering computations are quicker than initially expected thanks to their parallel implementations. Furthermore, and as expected, the last dataset (*com-LiveJournal*) shows the limits of our solution as the layout is computed in around four minutes. While the time needed to obtain a result in such cases is not unacceptable considering the size of the graphs at hand, we believe the computation of an overview layout should hardly take more than a minute as the representation is not an exact one. The resulting time measures proposed in the table underline how dependent our solution is of the clustering algorithm, and while a clustering of quality is needed when deciding the groupings, the operation must also be quick. However, our solution propose the advantage to be highly modular as the clustering algorithm detecting communities and the layout algorithm drawing the coarse graph representations can be changed at will should alternative algorithms need to be used.

With the complexity and time execution analysis out of the way, we can now take a look at some of the resulting representations obtained with our solution with a few different graphs.

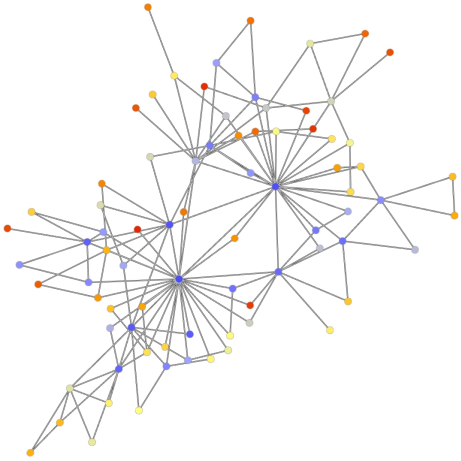
5.2.4 Resulting visualisation

We illustrate the general results obtained with our technique using four datasets of different sizes. The first one, presented in Figure 5.15, is a random network generated following the model of Wang et al. (2006). The graph is rather small, only consisting of 30,000 nodes and twice as much edges, but presents free-scale and small-world characteristics. Colours are mapped on the clusters to help differentiate one from another. The number of clusters, while being relatively low (between 70 and 80), is still significant enough that no existing colour-scale can provide enough discernible variations (Guastello, 2013). Consequently, some cluster colours appear to be quite similar while truly being different. We show the initial graph drawn using a force-directed algorithm (Fig. 5.15a), the coarse graph (representing the connections between communities, Fig. 5.15b) and the final layout obtained when using our algorithm (Fig. 5.15c).

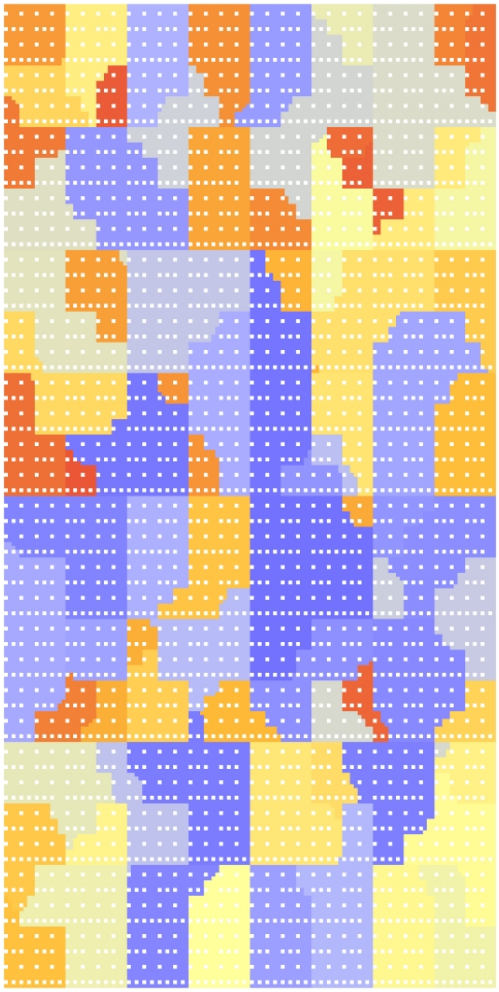
One can use the colours to identify the correspondences in the clusters between the two layouts, when looking at the coarse graph and resulting layout. Viewing those two representations side by side also allows to witness how our solution tends to preserve locality as connected clusters ends up next or not far one from another. Likewise, we can see how the nodes placement in Figure 5.15c respects the clusters position from Figure 5.15b (for instance: the two orange clusters on the bottom-left or the “reddish” ones in the top-right area). Similar cross-examination is unfortunately much harder between the force-directed layout of the graph and our solution’s



(a) Node-link representation of the initial graph with the FM^3 algorithm



(b) Layout of the coarse graph obtained after Phase I

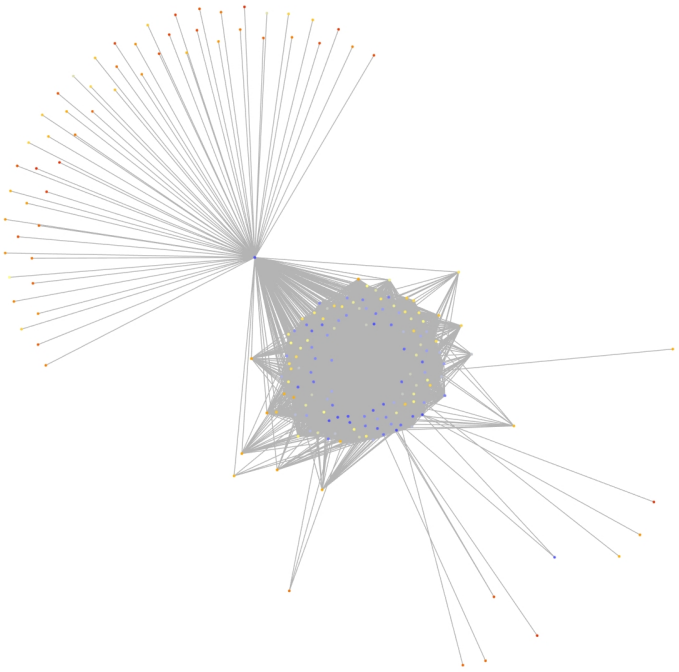


(c) Resulting pixel-oriented representation of the graph obtained after Phase II

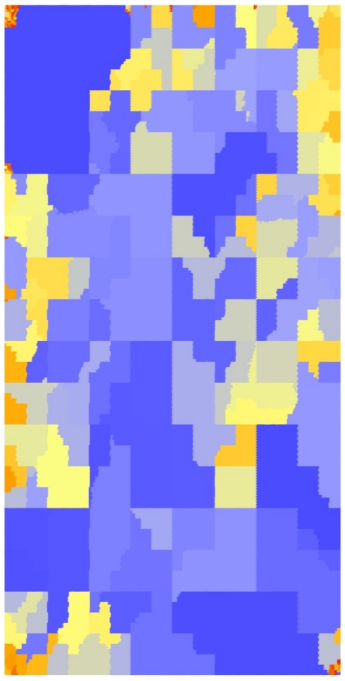
Figure 5.15: Application of JASPER on a random graph (30,000 nodes, 59,997 edges) generated with the model of Wang et al. (2006); the first figure (a) shows the complete graph displayed with a common force-directed layout (FM^3), the second figure (b) shows an intermediary layout of a coarser representation of the initial graph, while the third figure (c) shows the layout resulting from our solution.

resulting representation due to the random initialisation performed by the layout algorithm FM^3 used in both drawings. Additionally, this small graph overview is ideal to observe the white spaces mentioned earlier. One can see that the visualisation effectively forms a complete rectangle, even though its number of nodes should not allow it, and multiple "holes" are visible, regularly distributed throughout the representation. These white spaces help in fulfilling the surface and are evenly scattered to avoid us ending up with odd distortions in certain areas.

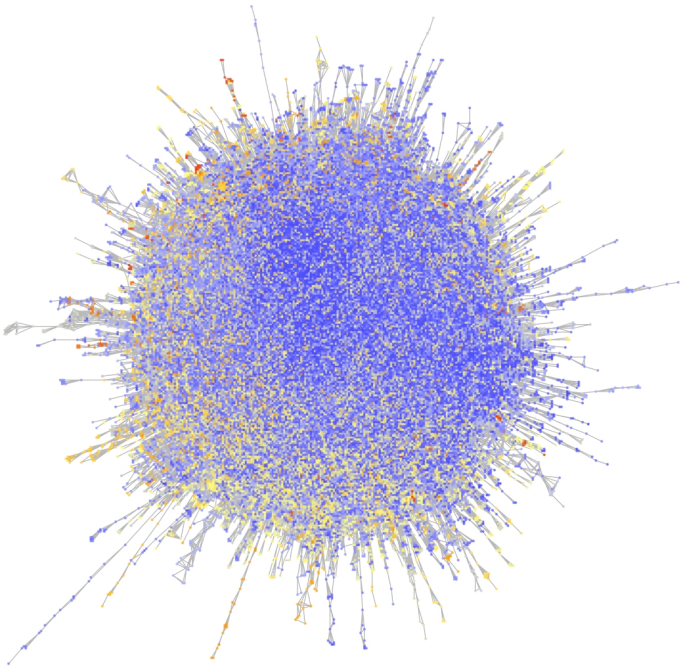
Altogether, the force-directed layout in Figure 5.15a provides a clear and understandable



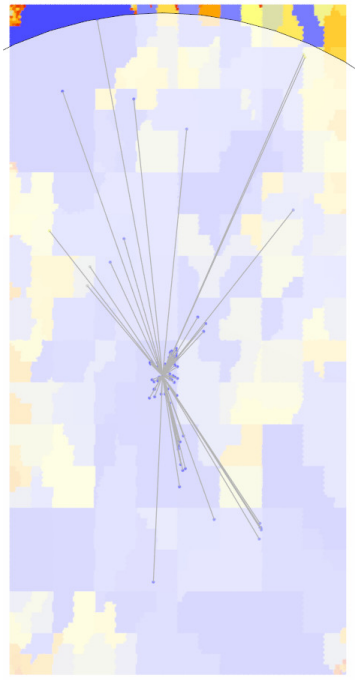
(a) Layout of the coarse graph obtained after Phase I



(b) Result using our solution



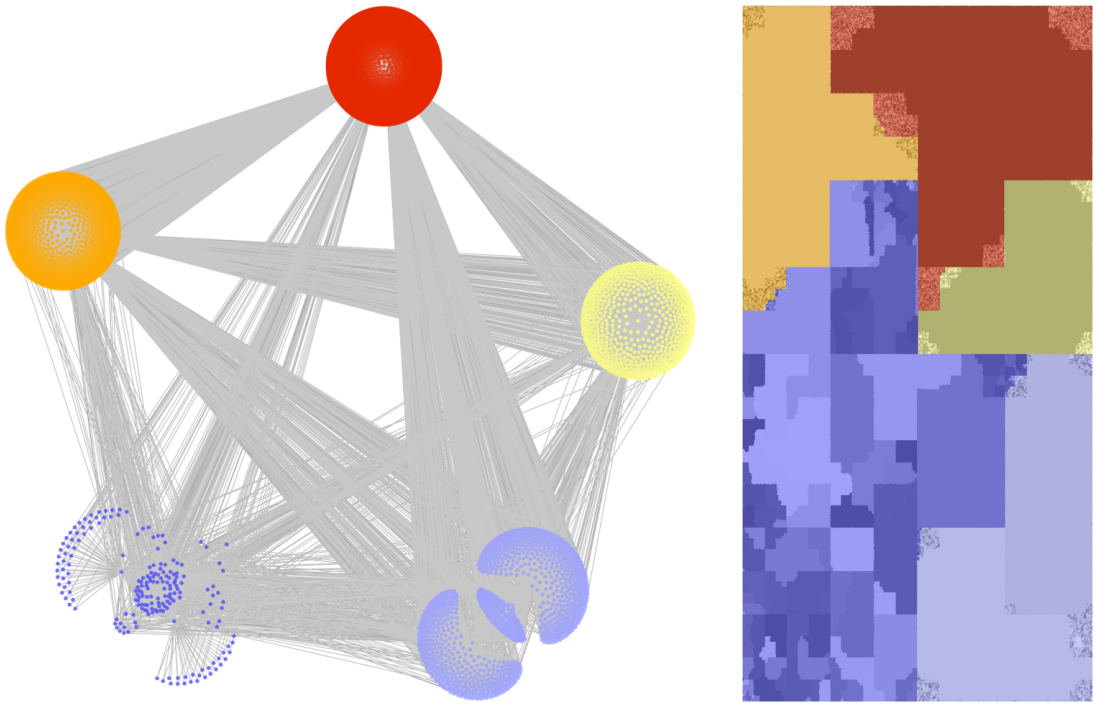
(c) Node-link representation of the network using the FM^3 algorithm; layout computed in 180 seconds using OGDF



(d) A neighbourhood highlighter interactor in action

Figure 5.16: Application of JASPER on the DBLP graph presented in Leskovec et al. (2008) (317,080 nodes, 1,049,866 edges and 430 clusters); the first figure (a) shows the coarser graph displayed with a force-directed layout while the second figure (b) uses a layout computed using JASPER. The third figure (c) shows a standard force-directed layout of the graph. For the final figure (d), we use the representation (b) previously obtained, select a node and visualise its direct neighbours using a visual filter.

view due to the small size of the graph. However, when facing densely connected or larger networks, force-directed representations are not clear enough and may take a long time to compute. To illustrate this point, we use the DBLP dataset introduced in Leskovec et al. (2008). Figure 5.16 shows the difference between the two rendered representations. The classic visualisation (Fig. 5.16c) using the force-directed algorithm alone is unreadable and the communities, somewhat still discernible when solely using the force-directed algorithm with the previous example, now melt into a single shapeless mass. On the other hand, our resulting layout (Fig. 5.16b) offers a much clearer overview of the graph. Each cluster appears more distinctively, even with limited colour variations, and, as connected communities are most likely laid out one next to another, additional information and insights concerning the group relations can be gathered by the person visualising the graph. This point is especially true if the visualisation framework proposes specific interactors for neighbour identification or exploration such as the neighbourhood highlighter presented in Moscovich et al. (2009) and available in TULIP. As shown in Figure 5.16d, using such tool allows users to examine and study the graph as well as the connections between nodes. The absence of edges improves the visualisation overall legibility but those elements are not deleted and can still be visualised. Moreover, organising and positioning the nodes in clusters gives some structure to the graph, helping users in seeing which groups of nodes are connected.

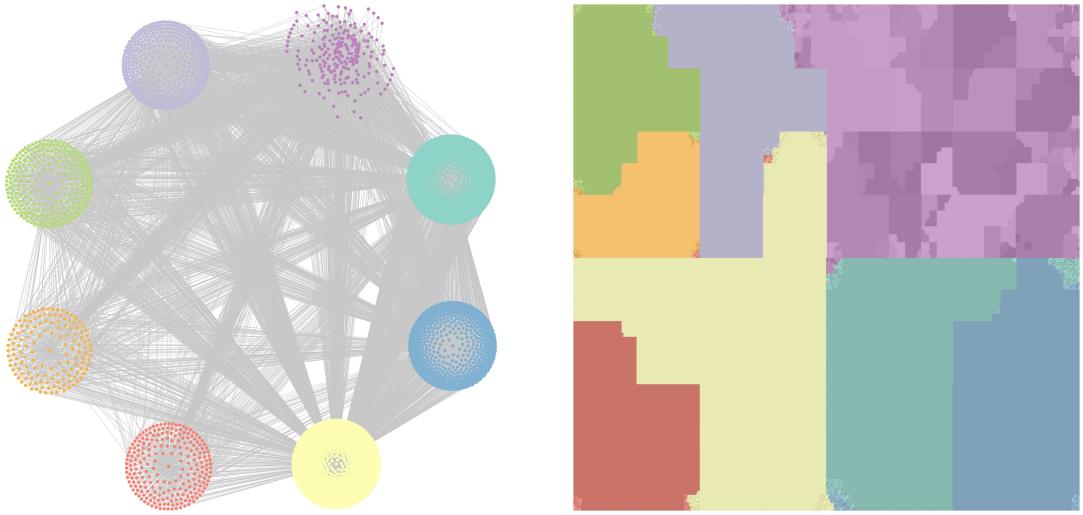


(a) Layout of the coarse graph

(b) Result using our solution

Figure 5.17: Application of JASPER on the Youtube graph presented in Yang and Leskovec (2012) (1,137,890 nodes, 2,987,624 edges and 9k clusters); the first figure (a) show the coarser version of the graph while the second figure (b) shows the final layout obtained with our algorithm.

The two next datasets, Youtube and LiveJournal from Yang and Leskovec (2012), are both quite large graphs making their representations impossible to generate using normal force-directed layout techniques. When the number of nodes exceeds half a million (and for at least



(a) Layout of the coarse graph

(b) Graph displayed with JASPER

Figure 5.18: Application of JASPER on the LiveJournal graph from Yang and Leskovec (2012) (3,997,962 nodes, 34,681,189 edges and 6k clusters); the first figure (a) show the coarser version of the graph while the second figure (b) shows the final layout of our algorithm.

twice as many edges), a classic layout algorithm such as FM^3 is not sufficiently efficient to compute a representation in a matter of minutes. On the other hand, JASPER can consider graphs as large as these and compute a layout in 25 seconds for Youtube and in around 4 minutes for LiveJournal. The resulting layouts for both datasets are shown in Figures 5.17 and 5.18. In each figure, a layout of the coarser graph is first shown, and followed by a representation of the complete graph using JASPER.

Finally, we present in Figure 5.19 an example of propagation visualisation achieved with our solution. As before, the nodes are distributed in clusters and each element is very likely to be spatially close to an element with which it is connected. Because the network topology does not evolve, we only need to compute the layout a single time. For this graph, a layout using the force-directed algorithm FM^3 is obtained in around thirty seconds whereas a representation with JASPER takes just one. The same layout is used consistently, allowing us to preserve the mental map, even if such necessity is still discussed and its benefit appear to be inconsequential (Purchase et al., 2007; Archambault et al., 2011; Archambault and Purchase, 2012) for any reason other than aesthetic preferences. In order to maximise the legibility of the representation from afar and to help observers differentiate the communities, we coloured the clusters using as few colour as possible. While six colours (light, medium and dark blue, and yellow, orange and red) are sufficient for the graph this time, we have obviously arranged them to avoid using the same colour on neighbouring areas. For a complete contrast with the colours in use, the nodes activated during the propagation are blackened, starting with the community initiating the first propagation step (visible on the right side of the graph). The resulting visualisation is arranged as a small multiple view presenting sixteen of the intermediary state of the propagation. Because our representation uses a space-filling curve, each node is laid out without any possibility of overlapping. This means that, as long as the visualisation is displayed with sufficient precision (i.e., on a screen with enough pixels), JASPER provides a pixel-oriented representation where each in-

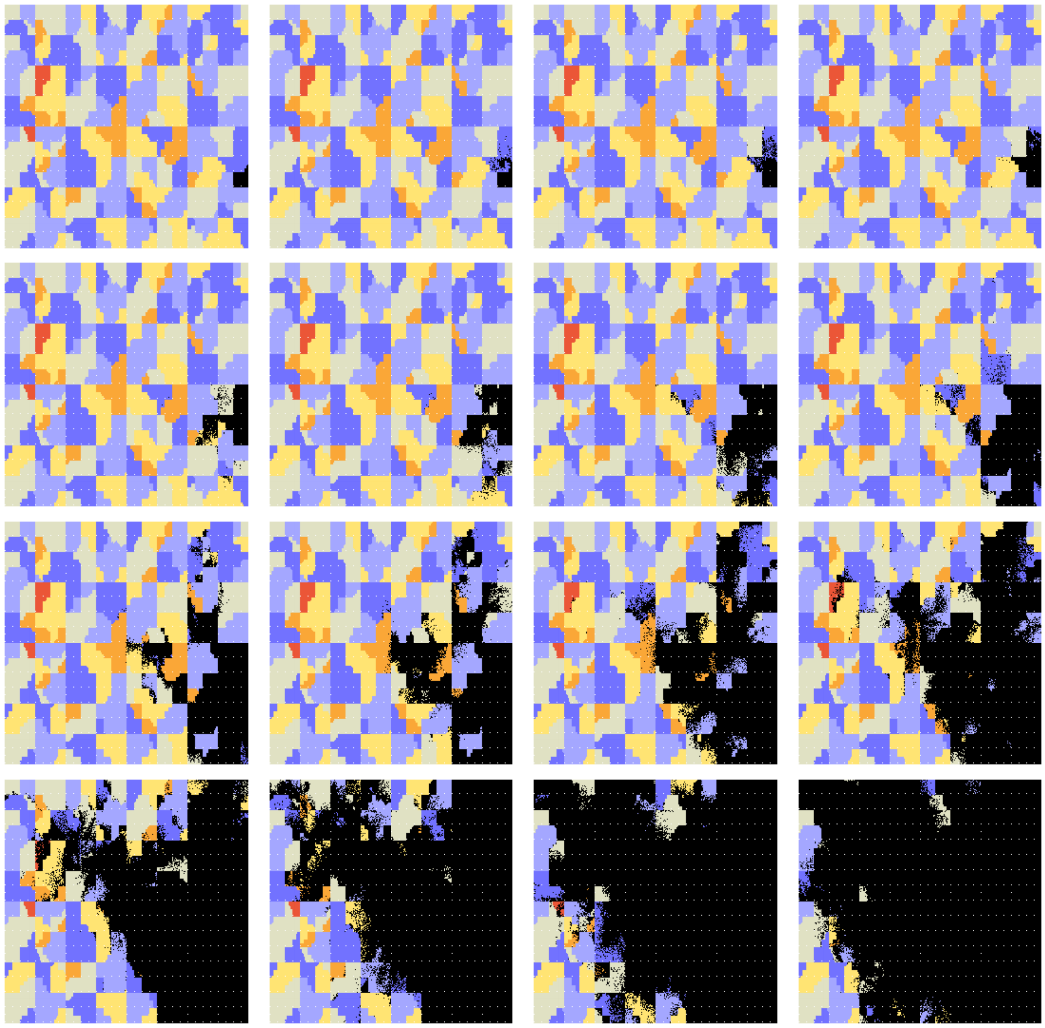


Figure 5.19: Small-multiple view of a propagation phenomenon on a medium graph generated by Wang et al. (2006) (65,000 nodes, 129,997 edges and 108 clusters). All the nodes within one of the cluster are considered active, and an independent cascade model (Kempe et al., 2003) is used to perform a propagation. The active nodes are coloured in black to quickly identify them.

dividual node is effectively displayed.[8] As one can see, the propagation hops from community to community, initially touching the most vulnerable/well connected nodes and spreading to their neighbours. The simulation showed in this example is of course only an illustration intended for the demonstration of JASPER. Consequently, the characteristics of the network have been adapted to allow a fast and relentless propagation to take place, thus explaining why all the

[8]Whether a pixel, i.e., a node, is actually visible in the final layout cannot be guaranteed however as it depends of both the display resolution (or pixel-per-inch) and the observer's visual acuity (Ware, 2012). This limitation is nonetheless common for any highly detailed representation. In the current example, as each propagation step of the graph is drawn in a square of 256 by 256 ($256^2 = 65,536$ which is large enough to draw the 65,000 nodes), the small-multiple view will effectively show all the elements at the same time if this page is printed in full-size on A4 paper (in 300 dpi) or if the whole representation is displayed on at least 1150 pixels (with the white spaces).

nodes successfully activate.

By presenting good execution times and legible layouts, our solution seems incredible on paper. However, while we know the inner mechanisms of Jasper and understand easily enough the information communicated by the representation, we need to see how it fares against other visualisation. Particularly, we need additional opinions concerning the resulting layouts and we want to see if our solution can produce more accurate or faster results than some other representation. The verification of all these points requires a user experiment.

5.3 User experiment: visualisation validation

In the same way that a new vaccine has to be thoroughly tested, in both *in vitro* studies and on real-cases, before being massively produced and widely distributed, our solution currently lacks an actual and proper usability study. The benchmark and the few examples of resulting visualisations are certainly encouraging and have allowed us to assert the computational complexity and measure the time needed to compute the representation. However, we still do not know if users working with our solution can find better or more complex information than when using other representations or if they can execute their work faster by using Jasper. We thus need to know if our solution is actually useful and if users are ready, and willing, to adopt it. This step is a normal part of the process of validating a solution and is mentioned in Figure 5.5 on page 119, as we are trying to validate our choices for visual encodings and interaction techniques by measuring the time and error rates obtained by users on specific tasks. While Munzner (2009) describes this validation step as a lab study, the experts of Human-Computer Interaction more generally label this process as an evaluation or an experimentation.

To achieve the validation of our solution, we have designed a user experimentation comparing Jasper to other popular representation methods. We hope, by using several different datasets and objectives to complete, to see the overall usability of our solution and whether or not it is actually a good solution. Although the visualisation of propagation phenomena have been our interest for developing Jasper, we have removed such focus from the experimentation design. We remain nonetheless interested in the performance of operations involved with the study and identification of communities in social networks. Thus, we propose to use our solution as a more general graph overview visualisation.

In the following, we start by describing the setup of our experimentation, detailing the datasets, presenting the representation methods and introducing the tasks given to the users. After a few words on the experimentation design and protocol, we finish by presenting in details the experiment results and discuss them.

5.3.1 Experimentation setup

The design of the experimentation has been greatly influenced by Purchase (2012), where the author details all the key points to help in achieving the experimentation. We thus reuse her terminology to express our own study.

By definition, an experimentation is motivated by a research question, in our case: is Jasper a better solution? The answer to this question mainly depends of two conditions: whether our solution enables user to respond faster or if their answers are more accurate, with the ideal result being a noticeable gain on both sides. To measure the advantage of our solution, we use different factors. First, we need a few different experimental objects, i.e., datasets, to test our solution in the more general case. Great results on a single experimental object can only indicates that our solution is adapted for visualising this particular dataset but consequently is not proved to be

better from a more general point of view. Second, we must define a set of tasks that the users will perform. The completeness and success of these tasks is measured and used to compare our solution to the others. This leads us to the last factor: the conditions, or in our case, the different representations. Because the sample of users taking part in the experimentation or evaluation cannot be strictly identical every time, the users must test both the solution being evaluated and the other ones to which it is compared to.

Datasets We use five different datasets throughout the whole experiment. A first one, a small random network of 10,000 nodes and twice as much edges, based on the generative model proposed by Wang et al. (2006), is used exclusively for training purposes. This dataset is thus not considered in the final experimentation results. During the experiment, it is only employed to introduce the different layouts and tasks, and help the users to familiarise themselves with the representations. Once these tutorials are finished, and in order to place users in similar situations to those encountered in social network analysis, we use real-world datasets of different sizes, ranging from tens of thousands to a few hundreds of thousands nodes. This aims to analyse the responses of users depending of both the layout used and the size and density of the networks visualised. All the following datasets are freely available as part of the Stanford Large Network Dataset Collection[9] (Leskovec and Krevl, 2014).

D1. email-Enron is a set of email messages, called the Enron corpus, representing exchanges between employees of the Enron corporation (Klimt and Yang, 2004). Additional information concerning the dataset, its origin, and a non-exhaustive list of research works using it is available online.[10] With its 33,696 nodes and 180,811 edges, this is the smallest dataset we use during the experimentation.

D2. soc-Slashdot0922 is a snapshot from February 2009 of the user community from the website Slashdot.[11] Although in the initial dataset, users have tagged each other positively or negatively, we are solely interested by the topological network created by those interactions. The Slashdot social network contains 82,168 nodes and 948,464 edges.

D3. soc-sign-epinions is a dataset extracted from the Epinions website.[12] More specifically, as a review website, Epinions proposes to users to select other users and "trust" them as well as their opinions on a wide range of products sorted by categories; more information on the dataset is available in Leskovec et al. (2010b). This network has 119,130 nodes and 833,695 edges.

D4. com-DBLP is a filtered version of the co-authorship network extracted from the computer science bibliography website[13] hosted by the University of Trier. Introduced in Yang and Leskovec (2012), this is the largest network used in our experiment with its 317,080 nodes and 1,049,866 edges.

To simplify the experiment and visualise networks with at least a common ground, the two datasets based on *email-Enron* and *soc-sign-epinions*, which are composed of several disconnected groups, have been filtered to only keep the largest weakly connected component.

Although we designed Jasper as a solution to represent the overview of rather large graphs, one cannot help but notice that the size of the graphs we have selected for our evaluation are more

[9] http://snap.stanford.edu/
[10] http://www.cs.cmu.edu/~enron/ and http://enrondata.org/content/research/
[11] https://slashdot.org/
[12] http://www.epinions.com/
[13] http://dblp.uni-trier.de/

modest than what we have presented before. This is simply due to the fact that this experiment aims at comparing our solution with others, and, while JASPER is able to handle larger graphs, we deem it unfair to evaluate the other representations on such extreme criterion.

Tasks Considering our strong interests toward social networks, the tasks we wish to perform should obviously answer to some existing need but also be oriented toward specific features encountered in such graphs. More precisely, an important point of interest in social networks analysis concerns the study of communities and the social ties existing between these groups. We thus propose solely community-oriented tasks, allowing us to measure how efficiently the different visualisations can represent this information:

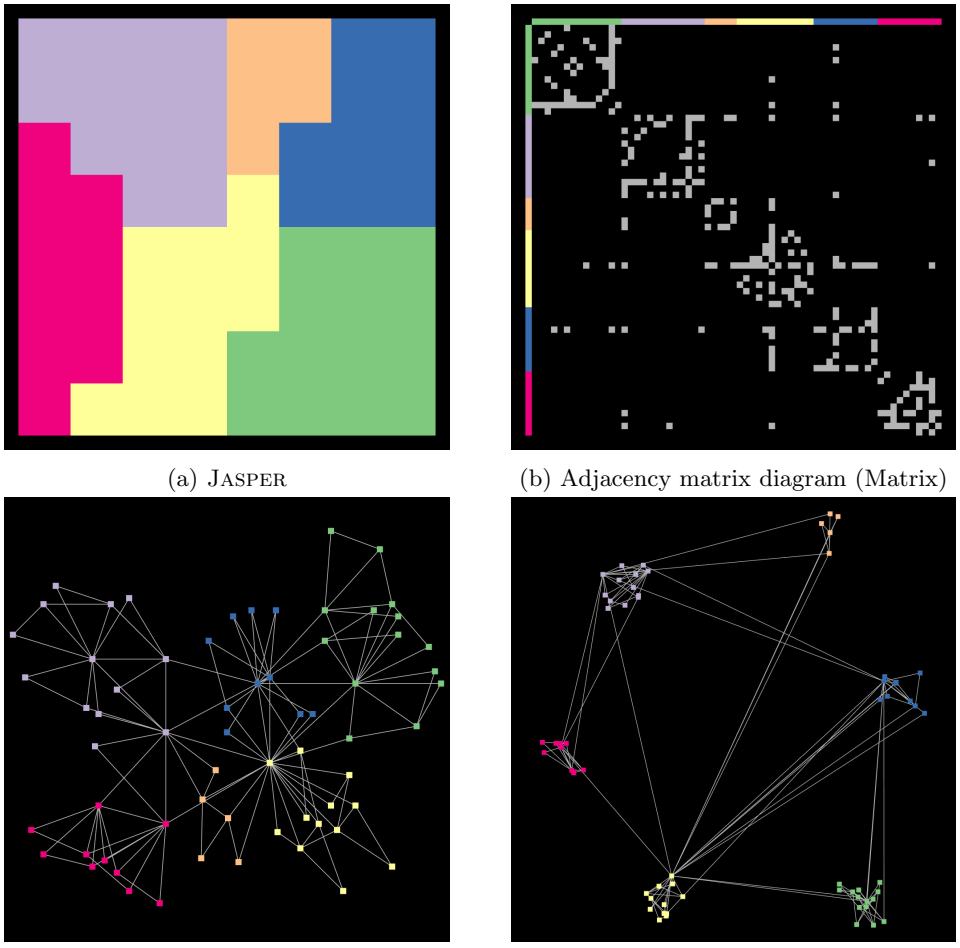
T1. Which highlighted community is the largest? As a community delimits a group of people with strong and numerous ties with one another, it will possess a very high edge/node ratio. Large communities are remarkable for this density is widened to many of individuals. When analysed, these sizeable groups often have to be identified, isolated, and studied aside to make sense. Being able to easily compare and pinpoint these large communities is thus an important criterion.

T2. How many highlighted communities is there? Although highlighting is an efficient way to bring information into focus, one still need to be able to discern this visual feedback. For this task, we extend this requirement by displaying several highlighted communities at the same time in a single graph. This allows us to measure how efficiently users can identify precise communities and differentiate them from the others.

T3. Are the highlighted communities connected? In the same way that most usual graph visualisations try to legibly indicate the existence of a link between two nodes, we wish to know if two communities are actually connected, i.e., if there is at least one of the nodes from the first community connected to one of the nodes from the second community. We propose this task to evaluate the adjacency-rendering abilities of the different visualisations.

Representation methods We choose four different visualisations to represent the aforementioned datasets. An example of the visual appearances of each method can be previewed in Figure 5.20. We use two well-known methods to compare to JASPER as a foundation: the adjacency matrix and the node-link diagram. In order to be fair throughout our experiment with respect to the nature of the tasks being community-oriented, we propose two different layout algorithms to draw the graphs with the node-link diagram. Although both use a force-directed layout, one additionally propose a more noticeable spatial separation of the communities. For each graph, we use the Louvain clustering algorithm (Blondel et al., 2008) to identify the communities and mark accordingly each node with an index specifying its cluster number. Based on this index, we then colour the nodes using a red-yellow-blue colour scale, so that each community colour is different. This colouring is given for each graph and is thus kept across all the representations. Furthermore, for all visualisations, nodes are drawn using square shapes, edges are coloured in grey and drawn as straight lines, and the background is coloured in black.

R1. Jasper As introduced above, JASPER is an adjacency-preserving, space-filling, and pixel-oriented layout. As a preliminary hypothesis, we believe our solution to be particularly adapted as an overview, and thus, should be the most efficient for finding communities and evaluating their size, as asked for task **T1**. It should also perform quite well with any community numbering related task, such as task **T2**, due to its construction. Its space-filling characteristic however is likely to impede to a certain extent the acknowledgement of existing connections between communities, thus likely giving poor results for task **T3**.



(a) Jasper

(b) Adjacency matrix diagram (Matrix)

(c) Node-link diagram using a force-directed layout (NLD)

(d) Node-link diagram using a force-directed layout with community separation (NLD-com)

Figure 5.20: Example of the four different kind of representations used during the experiment. The graph represented is the same in each visualisation. The nodes are shaped and coloured identically across all the representations.

R2. Adjacency matrix diagram (Matrix) This visualisation is a basic representation laying out a graph as a square Boolean matrix. As the main characteristic able to make a difference between two adjacency matrix diagrams is the selected node ordering, we arranged the nodes using the cluster indices. This regroups the nodes belonging to the same cluster, placing them next to one another. As a preliminary hypothesis, we believe the adjacency matrix diagram to be a rather dense and complex representation lacking the appeal offered by other visualisation techniques. Nonetheless, the matrix still shows all the information needed for each of the tasks, its only weakness being the difficulty for unfamiliar persons to gather this knowledge as shown in the literature (Sansen et al., 2015). We thus expect an overall fair rate of accurate answers but with probably longer response times than the other representations for **T1**, **T2** and **T3**.

R3. Node-link diagram (NLD) The first NLD uses the well-known force-directed layout FM^3 described in Hachul and Jünger (2005). While certainly the most common and preferred visualisation (Archambault and Purchase, 2015), node-link diagrams are known to provide mediocre to poor visual representations for dense graphs with as few as a hundred nodes when considering classical graph visualisation tasks (e.g., counting nodes, finding paths; Ghoniem et al. (2005)). As the tasks considered for our experiment are community-oriented, we may possibly end up with very different results in the end, however, with the representations being unsatisfactorily, especially for such large graphs, we state as a preliminary hypothesis that the NLD remains unlikely to excel at any of the given tasks.

R4. Node-link diagram with community regroupment (NLD-com) This second NLD uses the coarser graph, or quotient/skeleton graph, of the communities to display a much legible view of the graph. The coarser layout is computed using FM^3 and the communities are displayed more densely, allowing nodes belonging to the same community to be spatially closer to their neighbours than outsider nodes. This representation is similar to the visualisation obtained at the end of the Phase I of Jasper. The preliminary hypothesis formulated when using this layout is more positive than the previous one when considering the tasks **T2** and **T3**. Indeed, placing the nodes as a group improves the visibility and the identification the communities; when laid out this way, the connections between group of users are also quite discernible from connections between users of the same community. On the other hand, the task **T1** remains precarious as nodes from the same community can appear more and more indistinguishable from one another the larger the layout is.

Experiment design As we are looking to know whether there is any difference in performance when using different representation methods, we must also ensure the generalisation of our experiment. We thus propose to use two different series of questions: although the datasets, tasks, and visualisations used are similar in both cases, the communities subjected to the tasks are changed. This allows us to further validate the solution by using an alternative scenario. It also proves that the visualisations are not specialised in the study and identification of only a few communities but that the results would be almost similar should any other community be highlighted. In the end, all three tasks are applied to four different datasets (not including the smaller graph used for the introductory exercises) using four representation methods thus generating 3 tasks $\times 4$ datasets $\times 4$ representation methods $= 48$ results per user experimentation.

To perform the experimentation, we use a simple web application, proposing for each question two pictures of the same graph visualised using the same representation but with different highlighted communities. An example of the complete graphical user interface used during the experimentation is shown in Figure 5.21. As one can see, the task to complete is expressed at the top, with more detailed instructions given underneath and a time counter is placed in the top-right corner to indicate the time remaining to the user to complete the task. While the pictures are static representations of each of the datasets, an interaction is offered to the users: the possibility to hide the highlight on the concerned communities whenever the picture is hovered by the mouse cursor. Additionally, and in order to simplify the tasks and avoid unequal quality when dealing with non-deterministic layout at different iterations, the same graph layout is used for each pair of graph and visualisation in a given series. This restriction is useful to guarantee that the selected communities are not too small and can be noticed in any of the representations for instance.

Protocol Overall, the experiment is divided in four sets of questions, with each set using only a single representation method. To keep the users attentive, short recesses of up to two minutes

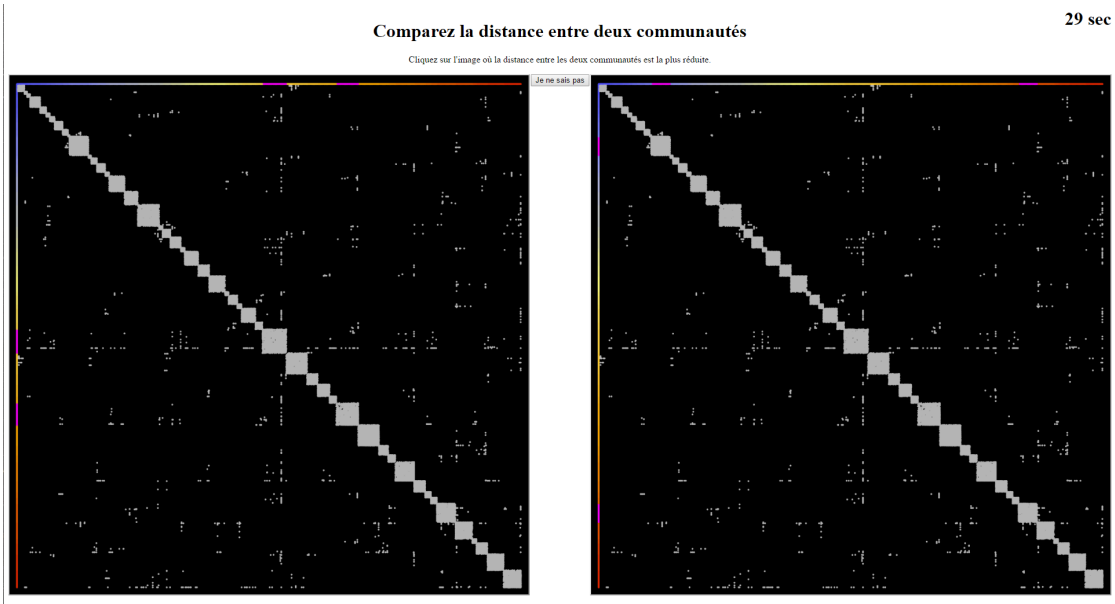


Figure 5.21: Evaluation graphical user interface. The task at hand is specified at the top of the page; the chronometer informing the user of the remaining time is in the top-right corner; the two pictures of the current graph are displayed in the center, on the left and right sides; finally, the "Surrender" button is located between the two visual representations.

are proposed between each set of questions.

When the experiment begin, the users are not familiar with the representations or the tasks at hand. For the users to perform at their utmost capability, they must learn the tasks and how to use the visualisations to achieve them according to Purchase (2012). This is ensured by using training questions at the beginning of each set. We submit a total of six training questions per set to the user. The first three are used to introduce the tasks with the current representation, and give users the opportunity to ask for help or additional explanations concerning the visualisation. The remaining three training questions are simply standard questions, selected at random in the alternative series and whose results will be ignored, but which allow the users to test the representation on a real dataset. Each user is thus asked to answer to 6 training questions $\times 4$ representation methods $+48$ normal questions $= 72$ questions in total.

For each question, users are shown two pictures of the same graph with different highlighted communities. The concerned nodes are coloured in pink to make them more apparent; the edges on the other hand remain of the same colour (grey). The user then reads the task proposed and clicks on the appropriate picture. If the user does not know the correct answer, a "Surrender" button can be clicked to pass to the next question. An informative time limit of 45 seconds is set for each question. Past that time, users can still look at the visualisation for as long as they need but any given answer will be marked as false. We hope that by leaving them the time to study a graph representation in depth, we give them the opportunity to forfeit one question for a better understanding of the visualisation, thus leading to better result with the following questions. The ordering of tasks, graphs, and representation used in each question is managed

using Latin squares[14] to avoid identical scenarios to take place for different experiments.

At the end of the experiment, each user must answer to a survey asking them to sort the representation methods according to their preference for a given task, and to pick her or his favourite representation. An inquiry for diverse remarks is also performed.

5.3.2 Experimentation results

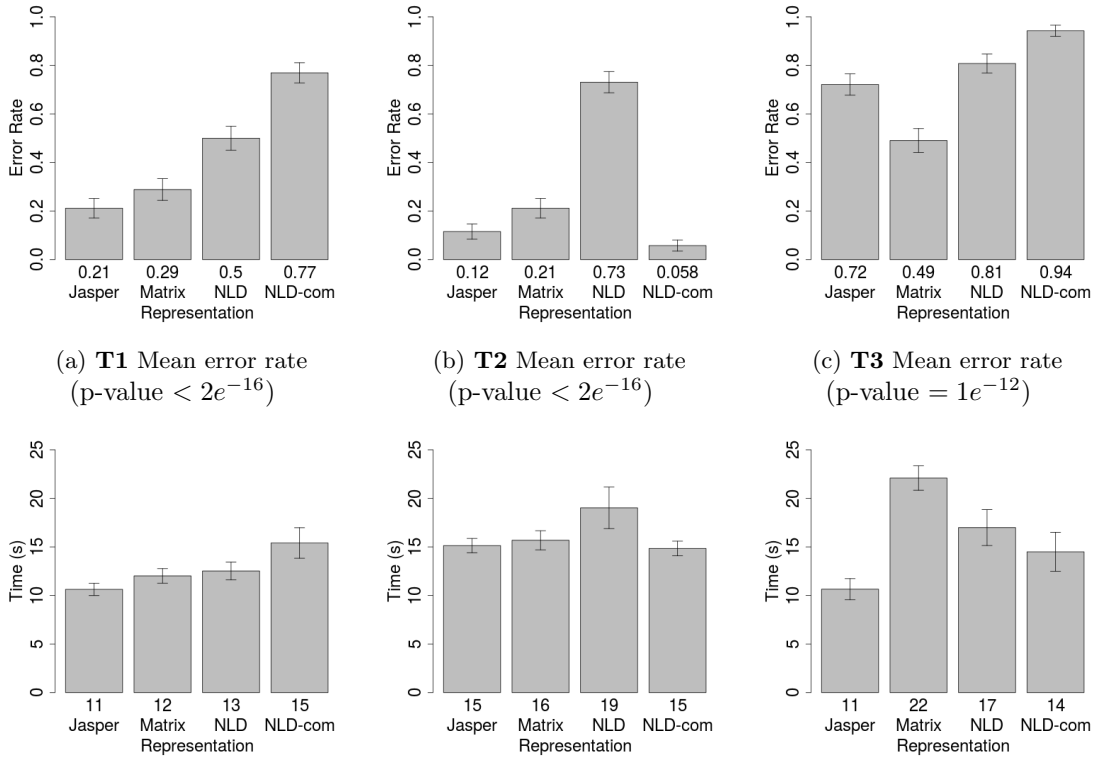
In the following, we discuss the results obtained during our experimentation. As mentioned above, we are mainly interested in two parameters concerning this experiment: the error rate and the response time. With these two measures being performed for each task on different graphs, we can compare the final results obtained with each representation method. During the whole experimentation, a total of twenty nine participants formed of associate professors, researchers, engineers, post-doctorates and students have volunteered as participants. All were asked to have at least a basic understanding of graph theory (at an undergraduate level) but no prior knowledge concerning information visualisation or network analysis was requested. Out of these twenty nine persons, three were present for the pilot tests, thus leaving us with a core of twenty six volunteers whose results are taken into account for the final user study.

We use the logfile returned by the web application stating the time and response accuracy of the users and analyse the results and their statistics; all statistical computations have been performed with the *R* project.[15] For both the time and the error rate, we use each time two different statistical tests. The first test offers a general view, allowing us to verify if there is a significant distinction existing between all the used representations. The second test is then applied on each pair of representations to compare one to the other. More precisely, to study the error-rate, we use Pearson's Chi-squared test[16] to compare the results using contingency tables and detect any possible global difference. A series of Wilcoxon's rank-sum test[17] to compare each pair of representation is then performed to precisely identify where the significant differences appear. For the response time variations, we first analyse them globally using Kruskal-Wallis' rank-sum test[18] and compare each pair of representation with Wilcoxon's rank-sum test whenever a significant difference is found out. As one can notice, we use non-parametric tests for both the error-rate and the response time, as both sets of results follow non-normal distributions. We nonetheless use a standard significance level $\alpha = 0.05$ to declare whether a significant difference exists in our results. With the time and error rate analysed, we then proceed with the second part of our analysis which considers the qualitative comments and marks gathered at the end of the experimentation as a survey. We use once more a Kruskal-Wallis' rank-sum test to compare for each task the users appreciation concerning the different representations. We can then rank the existing solutions according to this qualitative study and the users comments.

We present in Figure 5.22 the results obtained from our experimentation. Quite logically, we differentiate the results obtained according to the tasks the users were expected to complete.

T1. Which highlighted community is the largest? JASPER (**R1**) and the Adjacency Matrix Diagram (Matrix, **R2**) give here the best results with the lowest error rates. Indeed, user responses when using these representations were significantly more correct than with either of the Node-Link Diagrams visualisations (**R3** or **R4**). The response times for JASPER (**R1**) and Matrix (**R2**) are also better but the difference is only considered significant when either is compared to NLD-com (**R4**).

[14] https://en.wikipedia.org/wiki/Latin_square
[15] www.r-project.org
[16] https://en.wikipedia.org/wiki/Pearson%27s_chi-squared_test
[17] https://en.wikipedia.org/wiki/Mann%E2%80%93Whitney_U_test
[18] https://en.wikipedia.org/wiki/Kruskal%E2%80%93Wallis_one-way_analysis_of_variance



(a) **T1** Mean error rate (p-value $< 2e^{-16}$)

(b) **T2** Mean error rate (p-value $< 2e^{-16}$)

(c) **T3** Mean error rate (p-value $= 1e^{-12}$)

(d) **T1** Mean response time (p-value $= 0.01$)

(e) **T2** Mean response time (p-value $= 0.5$)

(f) **T3** Mean response time (p-value $= 3e^{-7}$)

Figure 5.22: Results of the user experimentation. We give for each task the mean error-rate and response time on the right answers measured during the experiments. The exact value obtained for each representation is specified under the bar on the charts, while the standard derivation for each group is given on top of them. The different p-values indicate the significance of the distinction between all the results.

T2. How many highlighted communities is there? For this task, a huge gap appears between NLD (**R3**) and the three other representation methods in terms of error-rate. NLD-com (**R4**) gives here a significantly lower error-rate than the Matrix (**R2**) while Jasper (**R1**) sits halfway in-between. The response time analysis shows only small variations, and, while the average NLD response time seems to be longer, no true significance is detected.

T3. Are the highlighted communities connected? This task proved to be much more complicated than initially anticipated in the pilot experimentation as shown by the high error-rates. Using the Matrix (**R2**) returns the lowest error rate by far, with Jasper (**R1**) and NLD (**R3**) coming next. However, the average response time when using Jasper (**R1**) is significantly better than all the other representations, it even cuts by half the average time needed to answer when using the Matrix (**R2**).

On a side note, while the datasets have been chosen to bring the experimentation as close as possible to real case, the selected networks are not identical and some statistically significant

variations exist for certain tasks. For instance, the successful completion of **T1** takes less time on DBLP (**D4**) and the completion of task **T3** has been significantly more successful on Epinion (**D3**) –with still 49% of errors– as well as sensibly faster –but not significantly– than on the other graphs. Task **T2** on the other hand does not show any significant differences in the response times and error rates across the different datasets.

Finally, we complete these results with the qualitative information obtained from the survey taken by all participants at the end of the experiment. Presented in Table 5.3, the marks given by the users are rather conform to the response times and error-rates measured. For task **T1**, JASPER (**R1**) and Matrix (**R2**) are both preferred to the node-link representations. Task **T2** however brings NLD-com (**R4**), JASPER (**R1**) and Matrix (**R2**) at the same level due to mixed opinions with the last method (**R3**) being majoritarily rated low. NLD-com (**R4**) and JASPER (**R1**) end up with the best marks for task **T3**, following the response time tendency but ignoring their high error rates (especially for **R4**).

Task	**R1.** JASPER	**R2.** Matrix	**R3.** NLD	**R4.** NLD-com
T1	1.64	1.96	3.16	3.24
T2	2.12	2.20	3.64	2.04
T3	1.80	3.20	3.28	1.72

Table 5.3: Average marks given by the users at the end of the experiment for each representation method depending of the task. The users were asked to rank the methods according to their efficiency with respect to the task at hand. This decision was made without any knowledge of the experiments results. The marks go from 1 (best) to 4 (worst) without any possible equality.

5.3.3 Discussion on the experimentation results

While no representation emerges as the clear winner from our experimentation, we have however certainly noticed a few points which we deem interesting. Let us start with the results obtain in task **T1**. The ranking based on the error-rate for each of the methods is very close to our initial expectations. While we are surprised by the relatively quick response time obtained from both the Matrix (**R2**) and the node-link diagram (**R4**), we find the error-rate of both JASPER (**R1**) and the Matrix (**R2**), although being quite low, to still be higher than awaited. Task **T2** sees the prevalence on all fronts of the node-link diagram rearranged for displaying communities (**R4**). This is because the nodes are arranged as tight clusters and can be easily identified, even in the largest networks. When compared to the normal node-link diagram (**R3**) on this task, one can clearly see the interest in regrouping the nodes as communities, nevertheless, this advantage turns to be a double-edged sword for the representation as **R4** falls back on the two other tasks whereas NLD (**R3**) present average results instead. Finally, task **T3** has certainly the most striking results with its the high error-rate. While we knew the task was somewhat complex after our pilot experiments, we were far from expecting such poor results across all representations. Indeed, had the users responded at random to the questions, the answers would had been closer to 50% of error-rate from a statistical point-of-view; thus, not only are the representation ill-adapted for this task, they also seem to mislead the users. As the identification of neighbouring elements is one of the atomic operations one would expect to perform on a node, the generalisation of this task to a community seemed like a necessary operation in a context where communities are so essentials. In the end, while the task may have been too intricate, we believe that the proposed representation methods are also not adapted for this type of task; alternatively, a multi-scale visualisation may be a much more efficient solution.

Overall, and despite its popularity, the node-link diagram (**R3**) does not excel anywhere and gives almost consistently worst results than the Matrix representation (**R2**) with respect to the tasks at hand. Its variation, NLD-com (**R4**), proposes extremely improved results on task **T2** in terms of error-rate but the representation does not bring improvements anywhere else. The Matrix representation (**R2**) showed the expected efficiency but the responses are faster than what we had initially predicted except for task **T3**. This proves that, given a quality ordering of the nodes, the adjacency matrix diagram is still a relevant representation method even for moderately large graphs. Lastly, JASPER (**R1**) did not reveal to be an incredible novel and ground-breaking representation method, but it was not designed with such goal in mind. For both tasks **T1** and **T2**, our solution comes as one of the best with no significant differences found in the experimentation results. Task **T3** shows more mixed results with a slightly better-than-average error-rate, in comparison to the other representations, but also with the best response time, coming at half the time needed for the Matrix (**R2**). Had the experimentation tasks been designed to only be completed once the answer given by the user is accurate, the error-rate distribution may have been quite different. So while JASPER (**R1**) is not the best at every task, its global performance seem to mark it as an all-encompassing acceptable overview solution, a fact sustained by the marks given by the users at the end of the experiment.

The propagation visualisation example presented earlier exposes perfectly our main intention when initially designing this layout algorithm: to propose a legible overview for large graphs. It is necessary to acknowledge the limitations of our method as we emphasise the *overview* characteristic of the resulting visualisation as performing analysis or in-depth studies on a graph can not be achieved using our method alone. Obviously, if one tries to discover all there is to know about a graph using a single approach, it will not be sufficient to understand every singular detail; attaining this outcome will indeed require different methods and points of view. Only then, may it ultimately result in a more complete picture. This is true for our method as, due to our focus on node representation, we have mostly hidden the edges, thus occulting some information to the observer. Even though we use node placement and colours to give hints of detected communities and existing connections, such metaphors are far from perfect and the visualisation produced is not entirely sound. Nevertheless, with the assistance of supplementary interactors and visual representations, additional insights on the structure of the graph and connections established through the edges can be easily accentuated; the neighbourhood highlighter available in TULIP or a matrix/node-link drawing focusing on specific sub-graphs are such tools. Furthermore, this lack of explicit information is not solely encountered when using our method as large graphs typically contain more information and details than one can possibly display simultaneously. Even other solutions similar to JASPER, using pixel-oriented layout to maximise the amount of information displayed in limited space, can only give an approximation (Shneiderman, 2008). For instance, considering the wiki-Talk dataset ($2,394,385$ nodes), we realise that simply using one pixel to display each node is impossible as the sheer number of elements to display is bigger than the number of pixels available on nowadays common display interfaces (a "full HD" resolution is only able to display $1920 \times 1080 = 2,073,600$ pixels). Multi-level solutions like Auber et al. (2003) and Shi et al. (2009) can overcome such restrictions but only up to a certain point as their resulting layouts solely provide an approximated or partial representation of the graph.

5.4 Conclusion

We have presented in this chapter a novel visualisation solution. Unable to find an existing layout which suited our needs, that is to visualise quickly large social networks without occluding any

node to see how propagation spreads, we were left with little choice but to create our own graph layout, obviously custom-fitted for our requirements. We thus have developed a new layout algorithm in adequation to our needs which we called Jasper (Vallet et al., 2016). We have used the lessons learned in studying different visualisation methods to design our solution to compute graph layouts with space-filling, pixel-oriented, and adjacency-preserving qualities. While initially created to be a quick method to compute overviews of large graphs, a user experimentation has proved our solution to be quite efficient on some community-related tasks, leading us to believe that Jasper can also be useful in some situations where a visualisation to assess the state of a large graph at a glance is required. As graphs will only grow larger and larger as more data is harvested to be exploited through data-analysis techniques, specific visualisation methods will have to be developed to cope with the ever increasing number of elements considered. Although in the end none of the representations considered during our experimentation has proved to be utterly better suited than the others for large graph visualisation and community-oriented tasks performance, we believe the issue worth of interest.

Chapter 6

Conclusion

The main goal of this last chapter is to bring all the work presented in this thesis to a conclusion. We also wish to offer some complementary perspectives, directions for future research, and discuss about some of the improvements brought to Porgy. But first, let us recall the outline of this thesis and summarise the work presented in this document.

6.1 Summary

From start to finish, we have tried to follow a logical workflow in the presentation of our work. We opted to begin our endeavour from next to nothing, starting with a graph, containing but a single node, which we then proceed to build up and expand. To this end, we first proposed to use generative models to create entire networks around our initial node. Secondly, we instilled some activity and dynamism in the graph through diffusion algorithms to perform simulations of information cascades. Finally, and while some visual analytics can also be performed during or after diffusion simulations, we employed visualisation techniques to study the graph and represent the operations which took place on it. Of course, performing these operations would hardly be worthy to mention had they been completed with standard and common methods; in our case however, we have instead defined and used a specific formalism employing only ordered graph transformations to achieve our goal and perform the aforementioned operations.

While, overall, this formalism makes our work slightly less straightforward, marginally more time-consuming and is not as efficient as some of the commonly used approaches, graph rewriting offers nonetheless a very accurate and unmatched expressiveness, allowing us to consider virtually any possible graph and transform it into anything else with just a single rule application. As one can guess, this kind of potentiality does not come easily nor without atonement and thus the formalism needs a precise situation and operational context to limit the transformations and avoid any unwanted or unruly byproduct. When we have presented all the details surrounding our approach in **Chapter 2**, we have described how the whole process of graph rewriting unfolds. The principle is quite easy to grasp: several elements existing in a graph are selected, extracted from it, transformed, rearranged, and finally replaced in the appropriate position. While this summary only grazes the surface of the different processes achieved during a rewriting step, the procedure truly followed is of course more intricate and every minute detail existing in the graph, down to the individual value of the elements' properties, becomes very important. However, the real strength of our formalism, as well as the originality setting our solution apart from other graph rewriting systems, plainly resides in the strategy and in our use of located graphs. Whereas the former component is essential to manage the transformations, offering for instance loops and

conditional or probabilistic rules applications, the latter items are indispensable to specify or dismiss at will the elements which can or cannot be transformed.

Once a proper definition of the formalism is given, nothing can stop us from establishing rules and strategies and applying them to obtain different results. Our rewriting platform PORGY, proposing a graphical interface to build the rules and visualise the results of their application on graphs, is able to work on different types of data; however, based on our own interest toward social networks analysis and visualisation, we have decided to investigate the possible applications for graph rewriting transformations on graphs of this nature. While several uses come to mind, the initial one we chose to explore was responding to a basic need: to apply rules, we first need a graph. Despite the large number of network generators available in the literature, the decision was easy to make as we naturally decided to explore the seminal model which initiated the craze for small-world networks. The complete presentation of the process we followed to adapt this model into rewrite rules and a strategy is detailed in **Chapter 3**. Additionally, after having proven our translation is correct and follows the specificities of the original model, we also use this experience to define and implement another generative model we created using behaviour observations we made when studying social network evolution. After analysis, we can guarantee that executing our model also results in generating a network presenting small-world characteristics. While the two models use different sets of rules and their strategies describe distinct rule application, seeing that two different models could lead to the creation of graphs presenting the same characteristics has left us wondering if the expressiveness of our formalism could be used to investigate, compare and identify which rule or section in the strategies leads to this result.

This question has been studied using what we believe to be a much simpler example to begin with. One of the recurring interest existing when studying social networks concerns the diffusion of rumours, content, or more generally any type of information. Because social networks represent the structure of interactions between individuals, understanding the relations between the persons gives us the possibility to better comprehend what, when, and in which manner pieces of information are going to spread throughout the network. Although there is a lot of different diffusion models with their own specificities, we have restrain ourselves in **Chapter 4** to only study four of them in-depth. The first two are standard influence propagation models where individuals diffusing a piece of information try to court their neighbours into relaying it to their own neighbours. On one hand, we have a probabilistic transmission, where one person tries to share the piece of information with somebody else and succeed with a probability correlated to the influence. On the other, we can use the joint influence method, where all the individuals sharing the information combine their personal influence to work together and convince another person to help in diffusing the information. The mechanisms presented by either models are at the core of several other diffusion models and have been extended by taking into account additional outcomes, influence resistance, time-dependent influences, etc. As their initial descriptions are quite different, both models present contrasting approaches accordingly. For instance, where one consider influence to be a peer-to-peer operation, the second prefers a many-to-one peer pressure effect; alternatively, the first model sees the influential individual as the person of interest whereas, the second model focus on the influenced user instead. However, the implementation using our formalism does not show such flagrant variations: the strategic programs are similar, and the only difference appears in the computation which lead to a person agreeing to relay an information. By completely expressing the models using a common formalism, we have found a legitimate way to describe the models on a neutral ground where they can be properly compared. This concept has also lead us to try another proposition. As we are able to identify small details differing between two existing models, why not try to start from an actual model and insert such divergence. To this end, we have used the remaining two models, describing a dissemination

algorithm instead of a propagation phenomenon. Whereas individuals (nodes in the graph) are key elements allowing a propagation phenomenon to go on, a dissemination process mostly ignores the users' opinions and decides, essentially on its own, which piece of information gets to be shown to one's neighbours and when it is retained. The initial dissemination model behaves as expressed previously. Every person however gets to vote if they consider that the information being diffused is relevant, in which case, the algorithm will further enforce the dissemination, and inversely. We propose to reuse this model and rework it to add to it some of the features observed in the earlier models. The graph rewriting program applying this reworked model ends up, as expected, being very close to the one using the initial model; it also presents whichever of the properties inherited through the small modifications being inserted.

The application of any of the above models on an existing network, once the different properties are properly initialised, allows us to simulate scenarios of information diffusion. We showed a visual representation of such network when in an intermediate state. Although, the graph used at that time is an example aiming to illustrate how it evolves as the graph rewriting program execution proceed, it is only natural to propose such feature for any who wishes to observe the current state of the network and to know if the diffusion is still on-going, slowing or accelerating for instance. This knowledge can of course be brought forth using different visual analytic techniques; nonetheless, to reach a more in-depth understanding of the transformation which occurred, one needs to be able to envision where the changes take place to wholly comprehend how the process unfolds. This point has been addressed in **Chapter 5** where we introduce a new space- and time-efficient method to draw large graphs. The main idea of the layout is to visually group nodes in clusters and use the graph's coarser representation, as well as a space-filling curve, to lay out the nodes in a compact fashion while preserving their spatial proximity with the other nodes of the same community. This solution has been successfully used to compute layouts for large graphs with a few millions of elements in half a minute, and with quicker results still (a matter of seconds) on graphs counting hundreds of thousands of elements. To ensure the quality and usefulness of the resulting visualisation, we have also performed a user experimentation, comparing our solution to the mostly commonly used visual representations. While the layouts produced with our method do not drastically outperform in terms of quality or completion time the other contestants, our solution managed to be consistently on par in the lead for each of the task being evaluated, as well as being the best rated according to the average users' opinion.

6.2 Perspectives

In this document we have independently created networks, produced scenarios of information propagation or dissemination, and finally proposed a solution to visualise the final results. Once combined, these three steps can be seen as a complete global workflow to simulate and observe the results of information diffusion. By switching the type of graph, diffusion model, and representation method being used, one can effectively create a custom solution to perform simulations according to precise specifications. To obtain this result however, some guidelines need to be followed to avoid problems. The most important one is that the different components must be developed and work completely independently from one another to avert dependencies which would otherwise impair the modularity. This also mean that the different initialisations, which are for instance needed by some of the models we have presented so far, have to be taken care of using rewrite rules and strategies.

In its current form, the visualisation step is some sort of outlier in the workflow as it relies on the visual advantages offered by Porgy (and thus Tulip) but does not employ the graph rewriting formalism to compute the layouts. Should we decide to, the workflow could be expressed

using only graph rewriting operations and work like Zhang et al. (2002), proposing a grammar-based graph drawing method, could be used as a basis to create layout techniques. Additionally, the generation-diffusion-visualisation workflow we describe above can naturally be extended to incorporate further modules. We have attempted it superficially earlier for instance as we were proposing to use visual analysis in Section 4.2.3 on page 92, however, a more common occurrence would likely take the form of a generation-diffusion-analysis-visualisation workflow. While several operations need to be performed to achieve a true analysis, some experiments should be carried on on simple examples first. Some calculation like the clustering algorithm presented by Raghavan et al. (2007) using label propagation should not stray far from our existing model and could be developed as a proof of concept before trying on more complex methods.

For the different models we have proposed in this thesis, considering the developed rewrite rules and strategies in the context of such a highly modular workflow gives us a different point of view which also underlines the limitations of our formalism. The first one we identify considers how the strategic graph rewriting syntax leaves us currently without a mean to transfer information handled at the strategy-level into the rules and vice versa. This shortcoming is most striking in Section 3.2 on page 47 where using a parameter to indicate the value to use for the attribute *Distance* would have made the implementation much simpler and the strategies shorter. Additionally, this also implies the necessity of updating the strategy syntax to add iterating loops and variables to store the counters. The second limitation we have found concerns the conditional matching. At the moment, the attribute's value of any element in the left-hand side is defined using a constant given as a specification of the rule (e.g., attribute *Colour* of node n_1 must be equal to `red` where n_1 is in the left-hand side and `red` is a constant). This could be changed to define the matching characteristic of elements based on other elements in the left-hand side (e.g., attribute *Colour* of node n_1 must be equal to attribute *Colour* of node n_2 where n_1 and n_2 are both in the left-hand side). Proposing this feature would make conditional matches on elements possible but it also creates some issues which need to be thought through and addressed first, notably how to handle deadlock situations (e.g., attribute *Colour* of n_1 must be equal to attribute *Colour* of n_2 and attribute *Size* of n_2 must be larger than attribute *Size* of n_1).

Other improvements to further generalise our formalism are still currently lacking in our formalism. Nonetheless, the following features can not be truly considered as existing limitations as we have not (yet) encountered cases where they were absolutely necessary to successfully implement the transformation. The first proposed feature is as follows. At the moment, only two located subgraphs, $\mathcal{P}$ and $\mathcal{Q}$, are available in the syntax. By proposing additional located subgraphs, it becomes possible to perform more elaborate filtering operations while leaving the position and ban subgraphs to be solely used to respectively either store the nodes and edges to focus on or the forbidden elements. The second feature is the introduction of "anti-nodes", which are elements to use in the rules; they are akin to the anti-edges presented in 2.2.3 on page 26 but using nodes instead. These anti-elements are used to precise in a rule's left-hand side where and when no elements should exist. For instance, let imagine the left-hand side of a rule contains a node n_1 and an anti-node n_2 and neither bear any distinctive sign (no attribute). For the rule to apply, a candidate must be found for n_1 but the anti-node must be left without a match: this rule can thus only apply on graphs with a single node. Once more, this refinement is easy to understand but it needs advanced considerations to cover properly all the possible scenarios (e.g., one or several anti-nodes, connected with or without edge and anti-edge).

On the visualisation side, multiple enrichments are possible concerning Jasper. Most notably, the topic of dynamic data has been briefly mentioned earlier but not developed further afterwards. Obviously, the quick execution of our method allows us to recompute a whole layout whenever an element is added or deleted to the graph. However, the non-deterministic characteristic of the drawings, imputed mostly to the underlying force-directed layout, implies a complete

transformation of the graph, thus disrupting the user mental map. While we believe it is better to keep a certain continuity in the position of the elements, entirely recalculating a graph layout is a mandatory step in some cases. Indeed, targeting key elements, like deleting the node at the centre of a community or inserting a new edge between two close communities, can lead to the division or merger of the said communities by the clustering algorithm, an event that our representation must absolutely display.

Of course, we cannot finish this section considering the possible perspective of our work without mentioning the limited performances of PORGY when working on large graphs. As shown in Appendix B on page 193, sufficiently large graphs cannot yet be computed in a fair amount of time using graph rewriting programs. Compared to the shortcomings and lacking features detailed previously, this quite down to earth but still rather important. The need for better performance has truly begin quite recently has the different models were finally becoming more stable and further experimentations were performed. We have already accomplished several performance analysis on our solution and we have been able to identify some of the code sections creating bottlenecks during the execution. However, it is likely that resolving these issues will not be sufficient and that only through further improvements, like those proposed in the publications mentioned in Section 2.2.3 on page 26, will our solution become truly able to handle larger graphs.

6.3 Discussion

Overall, it is interesting to see from where the graph rewriting platform PORGY started, and how it evolved during the last few years to allow the implementation of the different graph transformations and strategic rewriting programs which are presented in this document. In the beginning, the matching condition was mostly based on the topology with only two attributes, *State* and *Label*, to store values on ports and portnodes. Step by step, we have thus generalised the concept to handle as many attributes of different types as wanted. Parameters have been consequently added to each rule to manage which attribute(s) to use during the matching operation as well as the possibility of matching on edges. While these modifications were changing PORGY and making it more modular than before, thus allowing us to create more advanced models, we quickly realised that it was still far from sufficient to tackle the most intricate models. To continue the advancement and propose even more complex operations, we thus developed the computation component to perform calculations on the left-hand side attributes' value and affect them on the right-hand side elements being rewritten. The complete control over the attribute's value, and the diversity of computations available for them led us to deploy a mean to visualise the evolution of specific elements as the graph rewrite program keep applying new rules and transforming the graph. We have thus proposed to reuse the standard histogram and scatter plot visualisations, already available in TULIP, to respectively: count the number of nodes grouped by attribute's value, and display a parameter for each graph corresponding to a rewritten step for a given rewriting branch in the derivation (see example in Figure 4.4 on page 94). The latest improvements on the platform have added the management of directed edges and, shortly after, the inclusion of anti-edges to specify where elements should not exist. In total, these developments, while few in numbers, have completely changed the way PORGY behaves and how it can now be used; quite simply, none of the rewriting models presented in this document would have been possible with the earlier version of the software.

It is easy to understand how, at first glance, all the specifications surrounding our formalism can seem needlessly and overly convoluted. Particularly, using it to perform the simplest transformations can appear like using a bulldozer to build the most humble of sandcastle. However, the visual interface provided by the graph rewriting platform PORGY is to commend for mak-

ing the construction of rules and elaboration of strategies simple and fast. Although there is a learning curve to apprehend the tool, the task is not more complex nor does it take more time than what people face when discovering a new visualisation tool or programming language. For instance, creating a rule and strategy which change the colour of all nodes to blue, and computing it, should hardly take longer than writing and executing an equivalent program in any of the modern languages (e.g., C, Java, Python, R). Any new formalism will always seem alien to individuals who are not used to it, however, thinking in terms of graph transformations comes rather easily for the ones among us who work with these mathematical objects all day. Furthermore, while some purists would consider it superfluous, the visual definition of rewrite rules makes their construction, understanding and verification much simpler especially when working with complex rules counting a lot of elements. In the end, the different improvements applied on the graph rewriting definition and on the strategic language syntax bring us to a point where graph rewriting programs defined using our formalism truly present all the characteristics of a computer program as the graph rewriting system achieve Turing completeness.

6.4 Conclusion

This dissertation has presented a collection of network generation and information diffusion models expressed using a located strategic graph rewriting formalism, as well as a novel representation method to visualise large social networks. Using the graph rewriting formalism, we have used located transformations, jointly with strategic graph programs, to adapt existing algorithms and develop new ones for our graph rewriting system. While the translation to the formalism was not always evident at first for some operations, the different algorithms have finally all been successfully adapted. We have thus proved that the graph rewriting formalism was able to handle graph generation and information diffusion tasks.

Overall, the work achieved during this thesis has allowed us to study several generation models, diffusion algorithms and visualisation solutions. Because the graph rewriting operations are so different from the graph manipulations performed usually, we have been forced to precisely analyse and completely break down each of the studied models in order to identify all of their specificities. This allowed us to adapt the models truthfully for the graph rewriting formalism. It is important to note that, while all the definitions given at the beginning of this document are already complete, this was not the case for us at the time. Each new model thus brought at least some amount of frustration as the syntax was not always sufficient to perform all of the transformations immediately. Obviously, the first models we considered at the beginning of the thesis (the propagation models) showed us we were lacking a lot of features. This was not only true for the strategic syntax, but also for some parts of the graph rewriting definitions. This has lead to us to continuously improve and generalise the formalism with each new model being translated. The advantage is that with each new model, less and less new features are becoming necessary to implement; even to the point where the last model being adapted for the graph rewriting system did not necessitate a single modification of our definition, which is encouraging.

In the end, this thesis has effectively offered me the possibility to look in great details into both the processes of social network creation and information diffusion. While my curiosity concerning these topics has been satisfied, the process also raised additional questions and interests on related subjects. In particular, multiplex networks are fascinating objects which present a lot of potential and appeal in terms of analysis. Of course, the visual analytics dimension should not be forgotten. Our visualisation solution showed some promises in term of usability on graph counting a few millions of elements at a time. However, the ever-growing amounts of data produced by web

companies[1] call for more efficient solutions. All these data is obviously not meant to be analysed by humans but it is without a doubt a good indication of what the future holds for us. At least, we will not run out of data to crunch anytime soon; this certainly feels like a good time to be a data scientist.

[1] http://www.planetrisk.com/Big-Data-Infographic/

Bibliography

Daron Acemoglu, Asuman Ozdaglar, and Ali ParandehGheibi. Spread of (mis)information in social networks. *Games and Economic Behavior*, 70(2):194–227, November 2010. URL http://ideas.repec.org/a/eee/gamebe/v70y2010i2p194-227.html.

Daron Acemoglu, Asuman Ozdaglar, and Ercan Yildiz. Diffusion of innovations in social networks. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 2329–2334, Dec 2011. doi:10.1109/CDC.2011.6160999.

Daron Acemoglu, Asuman Ozdaglar, and Alireza Tahbaz-Salehi. Systemic risk and stability in financial networks. *American Economic Review*, 105(2):564–608, February 2015. doi:10.1257/aer.20130456. URL http://www.aeaweb.org/articles?id=10.1257/aer.20130456.

John Adams. Angliae totius tabula cum distancii notoribus in itinerantium usum accommodata, 1679.

J. Ahn, C. Plaisant, and B. Shneiderman. A task taxonomy for network evolution analysis. volume PP, pages 1–1, 2013. doi:10.1109/TVCG.2013.238.

Mohammad Ali, Anna Lena Lopez, Young Ae You, Young Eun Kim, Binod Sah, Brian Maskery, and John Clemens. The global burden of cholera. *Bulletin of the World Health Organization*, 90(3):209–218A, 2012. doi:10.2471/BLT.11.093427.

Robert Amar, James Eagan, and John Stasko. Low-level components of analytic activity in information visualization. In *Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, INFOVIS '05, pages 15–, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7803-9464-x. doi:10.1109/INFOVIS.2005.24. URL http://dx.doi.org/10.1109/INFOVIS.2005.24.

Oana Andrei. *A Rewriting Calculus for Graphs: Applications to Biology and Autonomous Systems*. Theses, Institut National Polytechnique de Lorraine - INPL, November 2008. URL https://tel.archives-ouvertes.fr/tel-00337558.

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.

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.

Oana Andrei, Maribel Fernández, Hélène Kirchner, Guy Melançon, Olivier Namet, and Bruno 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, 2011a. doi:10.4204/EPTCS.48.7. URL http://hal.inria.fr/inria-00563249/en.

Oana Andrei, Maribel Fernandez, Hélène Kirchner, Guy Melançon, Olivier Namet, and Bruno Pinaud. Porgy: Strategy-driven interactive transformation of graphs. In Rachid Echahed, editor, *6th International Workshop on Computing with Terms and Graphs (TERMGRAPH 2011)*, volume 48, pages 54–68, Saarbrücken, Germany, 2011b. doi:10.4204/EPTCS.48.7. URL http://hal.inria.fr/inria-00563249.

Oana Andrei, Maribel Fernández, Hélène Kirchner, and Bruno Pinaud. Strategy-Driven Exploration for Rule-Based Models of Biochemical Systems with Porgy. Research report, Université de bordeaux ; Inria ; King's College London ; University of Glasgow, 2016. URL https://hal.archives-ouvertes.fr/hal-01429890.

Anthony Anjorin, Erhan Leblebici, Andy Schürr, and Gabriele Taentzer. *A Static Analysis of Non-confluent Triple Graph Grammars for Efficient Model Transformation*, pages 130–145. Springer International Publishing, Cham, 2014. ISBN 978-3-319-09108-2. doi:10.1007/978-3-319-09108-2_9. URL http://dx.doi.org/10.1007/978-3-319-09108-2_9.

D. Archambault, T. Munzner, and D. Auber. Topolayout: Multilevel graph layout by topological features. *Visualization and Computer Graphics, IEEE Transactions on*, 13(2):305–317, March 2007. ISSN 1077-2626. doi:10.1109/TVCG.2007.46.

D. Archambault, T. Munzner, and D. Auber. Grouseflocks: Steerable exploration of graph hierarchy space. *Visualization and Computer Graphics, IEEE Transactions on*, 14(4):900–913, July 2008. ISSN 1077-2626. doi:10.1109/TVCG.2008.34.

Daniel Archambault and Helen C. Purchase. The mental map and memorability in dynamic graphs. In *Pacific Visualization Symposium (PacificVis), 2012 IEEE*, pages 89–96, 2012. doi:10.1109/PacificVis.2012.6183578.

Daniel Archambault and Helen C. Purchase. On the effective visualisation of dynamic attribute cascades. *Information Visualization*, 2015. doi:10.1177/1473871615576758. URL http://ivi.sagepub.com/content/early/2015/04/02/1473871615576758.abstract.

Daniel Archambault, Helen C. Purchase, and Bruno Pinaud. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Transactions on Visualization and Computer Graphics*, 17(4):539–552, 2011. ISSN 1077-2626. doi:http://doi.ieeecomputersociety.org/10.1109/TVCG.2010.78.

Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. *Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations*, pages 121–135. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-16145-2. doi:10.1007/978-3-642-16145-2_9. URL http://dx.doi.org/10.1007/978-3-642-16145-2_9.

D. Auber, C. Huet, A. Lambert, B. Renoust, A. Sallaberry, and A. Saulnier. Gospermap: Using a gosper curve for laying out hierarchical data. *Visualization and Computer Graphics, IEEE Transactions on*, 19(11):1820–1832, Nov 2013. ISSN 1077-2626. doi:10.1109/TVCG.2013.91.

David Auber, Yves Chiricota, Fabien Jourdan, and Guy Melançon. Multiscale visualization of small world networks. In *Proceedings of the Ninth Annual IEEE Conference on Information Visualization*, INFOVIS'03, pages 75–81, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7803-8154-8. URL http://dl.acm.org/citation.cfm?id=1947368.1947385.

David Auber, Romain Bourqui, Maylis Delest, Antoine Lambert, Patrick Mary, Guy Melançon, Bruno Pinaud, Benjamin Renoust, and Jason Vallet. TULIP 4. Research report, LaBRI - Laboratoire Bordelais de Recherche en Informatique, September 2016. URL https://hal.archives-ouvertes.fr/hal-01359308.

Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, Great Britain, 1998.

Christian Bachmaier, Franz J. Brandenburg, Wolfgang Brunner, and Gergö Lovász. *Cyclic Leveling of Directed Graphs*, pages 348–359. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-00219-9. doi:10.1007/978-3-642-00219-9_34. URL http://dx.doi.org/10.1007/978-3-642-00219-9_34.

Eytan Bakshy, Brian Karrer, and Lada A. Adamic. Social influence and the diffusion of user-created content. In *Proceedings of the 10th ACM Conference on Electronic Commerce*, EC '09, pages 325–334, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-458-4. doi:10.1145/1566374.1566421. URL http://doi.acm.org/10.1145/1566374.1566421.

Daniel Balasubramanian, Anantha Narayanan, Christopher P. van Buskirk, and Gabor Karsai. The graph rewriting and transformation language: Great. *ECEASST*, 1, 2006. URL http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/89.

Jayanth R. Banavar, Amos Maritan, and Andrea Rinaldo. Size and form in efficient transportation networks. *Nature*, 399:130–132, 1999. ISSN 0028-0836. doi:10.1038/20144. URL http://dx.doi.org/10.1038/20144.

Qing Bao, William K. Cheung, and Yu Zhang. Incorporating structural diversity of neighbors in a diffusion model for social networks. In *Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2013 IEEE/WIC/ACM International Joint Conferences on*, volume 1, pages 431–438, 2013. doi:10.1109/WI-IAT.2013.61.

A. Barabasi and J. Frangos. *Linked: The New Science Of Networks Science Of Networks*. Basic Books, 2002. ISBN 9780738206677. URL https://books.google.fr/books?id=8OwOlwEACAAJ.

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. ISSN 0036-8075. doi:10.1126/science.286.5439.509. URL http://science.sciencemag.org/content/286/5439/509.

M. Bardoscia, S. Battiston, F. Caccioli, and G. Caldarelli. DebtRank: A Microscopic Foundation for Shock Propagation. *PLoS ONE*, 10:e0134888, July 2015. doi:10.1371/journal.pone.0134888.

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.

Luciano Baresi and Reiko Heckel. *Tutorial Introduction to Graph Transformation: A Software Engineering Perspective*, pages 402–429. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. ISBN 978-3-540-45832-6. doi:10.1007/3-540-45832-8_30. URL http://dx.doi.org/10.1007/3-540-45832-8_30.

A. Barry, J. Griffith, and C. O'Riordan. An evolutionary and graph-rewriting based approach to graph generation. In *2015 7th International Joint Conference on Computational Intelligence (IJCCI)*, volume 1, pages 237–243, Nov 2015.

Klaus Barthelmann. How to construct a hyperedge replacement system for a context-free set of hypergraphs. Technical report, Universität Mainz, Institut für Informatik, 1996.

Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks, 2009. URL https://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154/1009.

Vladimir Batagelj. *Large-Scale Social Network Analysis*, pages 8245–8265. Springer New York, New York, NY, 2009. ISBN 978-0-387-30440-3. doi:10.1007/978-0-387-30440-3_489. URL https://doi.org/10.1007/978-0-387-30440-3_489.

Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Phys. Rev. E*, 71:036113, Mar 2005. doi:10.1103/PhysRevE.71.036113. URL http://link.aps.org/doi/10.1103/PhysRevE.71.036113.

Vladimir Batagelj and Andrej Mrvar. *Pajek — Analysis and Visualization of Large Networks*, pages 77–103. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-642-18638-7. doi:10.1007/978-3-642-18638-7_4. URL https://doi.org/10.1007/978-3-642-18638-7_4.

S. Battiston, M. D'Errico, S. Gurciullo, and G. Caldarelli. Leveraging the network: a stress-test framework based on DebtRank. *ArXiv e-prints*, March 2015.

Fabian Beck, Michael Burch, Stephan Diehl, and Daniel Weiskopf. The state of the art in visualizing dynamic graphs. In *EuroVis - STARs*, pages 83–103. Eurographics Association, 2014. doi:10.2312/eurovisstar.20141174. URL http://doi.acm.org/10.2312/eurovisstar.20141174.

Chris Bennett, Jody Ryall, Leo Spalteholz, and Amy Gooch. The aesthetics of graph visualization. In *Proceedings of the Third Eurographics conference on Computational Aesthetics in Graphics, Visualization and Imaging*, Computational Aesthetics'07, pages 57–64, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. ISBN 978-3-905673-43-2. doi:10.2312/COMPAESTH/COMPAESTH07/057-064. URL http://dx.doi.org/10.2312/COMPAESTH/COMPAESTH07/057-064.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975. ISSN 0001-0782. doi:10.1145/361002.361007. URL http://doi.acm.org/10.1145/361002.361007.

J. Bertin. *Semiology of graphics*. Central Asia book series. University of Wisconsin Press, 1983. ISBN 9780299090609. URL https://books.google.fr/books?id=ruZQAAAAMAAJ. Translated from the french: "Sémiologie Graphique. Les diagrammes, les réseaux, les cartes" (1967).

E. Bertuzzo, S. Azaele, A. Maritan, M. Gatto, I. Rodriguez-Iturbe, and A. Rinaldo. On the space-time evolution of a cholera epidemic. *Water Resources Research*, 44(1):n/a–n/a, 2008. ISSN 1944-7973. doi:10.1029/2007WR006211. URL http://dx.doi.org/10.1029/2007WR006211.

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. doi:10.1098/rsif.2009.0204. URL http://rsif.royalsocietypublishing.org/content/7/43/321.abstract.

S. Bhattacharya, R. Black, L. Bourgeois, J. Clemens, A. Cravioto, J. L. Deen, Gordon Dougan, R. Glass, R. F. Grais, M. Greco, I. Gust, J. Holmgren, S. Kariuki, P.-H. Lambert, M. A. Liu, I. Longini, G. B. Nair, R. Norrby, G. J. V. Nossal, P. Ogra, P. Sansonetti, L. von Seidlein, F. Songane, A.-M. Svennerholm, D. Steele, and R. Walker. The cholera crisis in africa. *Science*, 324(5929):885–885, 2009. ISSN 0036-8075. doi:10.1126/science.1173890. URL http://science.sciencemag.org/content/324/5929/885.

T. Blascheck, K. Kurzhals, M. Raschke, M. Burch, D. Weiskopf, and T. Ertl. Visualization of eye tracking data: A taxonomy and survey. *Computer Graphics Forum*, pages n/a–n/a, 2017. ISSN 1467-8659. doi:10.1111/cgf.13079. URL http://dx.doi.org/10.1111/cgf.13079.

Michael L. Blinov, Jin Yang, James R. Faeder, and William S. Hlavacek. *Graph Theory for Rule-Based Modeling of Biochemical Networks*, pages 89–106. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-48839-2. doi:10.1007/11905455_5. URL http://dx.doi.org/10.1007/11905455_5.

Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008. doi:10.1088/1742-5468/2008/10/P10008. URL http://stacks.iop.org/1742-5468/2008/i=10/a=P10008.

B. Bollobás. *Random Graphs*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2001. ISBN 9780521797221. URL https://books.google.fr/books?id=o9WecWgilzYC.

Bela Bollobas. *Modern Graph Theory*, volume 184 of *Graduate Texts in Mathematics*. Springer-Verlag New York, 1 edition, 1998. ISBN 978-0-387-98488-9. doi:10.1007/978-1-4612-0619-4.

A. Bondy and U.S.R. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer London, 2011. ISBN 9781846289699. URL https://books.google.fr/books?id=HuDFMwZOwcsC.

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.

Ulrik Brandes. *Drawing on Physical Analogies*, pages 71–86. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. ISBN 978-3-540-44969-0. doi:10.1007/3-540-44969-8_4. URL https://doi.org/10.1007/3-540-44969-8_4.

Ulrik Brandes and Thomas Erlebach, editors. *Network Analysis - Methodological Foundations*, volume 3418 of *Theoretical Computer Science and General Issues*. Springer-Verlag Berlin Heidelberg, 1 edition, 2005. ISBN 978-3-540-24979-5. doi:10.1007/b106453.

Roger Brown. *Social Psychology*. New York: Free Press, 1965.

Ceren Budak, Divyakant Agrawal, and Amr El Abbadi. Limiting the spread of misinformation in social networks. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 665–674, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0632-4. doi:10.1145/1963405.1963499. URL http://doi.acm.org/10.1145/1963405.1963499.

Alberto Cairo. *The Functional Art: An introduction to information graphics and visualization*. Voices That Matter. Pearson Education, 2012. ISBN 9780133041361. URL https://books.google.com/books?id=xwjhh6Wu-VUC.

S. K. Card and J. Mackinlay. The structure of the information visualization design space. In *Proceedings of the 1997 IEEE Symposium on Information Visualization (InfoVis '97)*, INFOVIS '97, pages 92–, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-8189-6. URL http://dl.acm.org/citation.cfm?id=857188.857632.

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. ISBN 9781139443432. URL http://books.google.fr/books?id=4Ty5xP_KcpAC.

Dorwin Cartwright and Frank Harary. Structural balance: a generalization of Heider's theory. *Psychological Review*, 63:277–293, 1956. doi:10.1037/h0046049.

Meeyoung Cha, Alan Mislove, and Krishna P. Gummadi. A measurement-driven analysis of information propagation in the flickr social network. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 721–730, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-487-4. doi:10.1145/1526709.1526806. URL http://doi.acm.org/10.1145/1526709.1526806.

Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 199–208, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-495-9. doi:10.1145/1557019.1557047. URL http://doi.acm.org/10.1145/1557019.1557047.

Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 1029–1038, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0055-1. doi:10.1145/1835804.1835934. URL http://doi.acm.org/10.1145/1835804.1835934.

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 *Proceedings of the Eleventh SIAM International Conference on Data Mining, SDM 2011, April 28-30, 2011, Mesa, Arizona, USA*, pages 379–390, 2011. doi:10.1137/1.9781611972818.33. URL http://dx.doi.org/10.1137/1.9781611972818.33.

E. H. H. Chi, P. Barry, J. Riedl, and J. Konstan. A spreadsheet approach to information visualization. In *Proceedings of the 1997 IEEE Symposium on Information Visualization (InfoVis '97)*, INFOVIS '97, pages 17–, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-8189-6. URL http://dl.acm.org/citation.cfm?id=857188.857626.

Ed H. Chi. A taxonomy of visualization techniques using the data state reference model. In *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on*, pages 69–75, 2000. doi:10.1109/INFVIS.2000.885092.

Lily A. Chylek, Bin Hu, Michael L. Blinov, Thierry Emonet, James R. Faeder, Byron Goldstein, Ryan N. Gutenkunst, Jason M. Haugh, Tomasz Lipniacki, Richard G. Posner, Jin Yang, and William S. Hlavacek. Guidelines for visualizing and annotating rule-based models. *Mol. BioSyst.*, 7:2779–2795, 2011. doi:10.1039/C1MB05077J. URL http://dx.doi.org/10.1039/C1MB05077J.

Lily A. Chylek, Leonard A. Harris, James R Faeder, and William S. Hlavacek. Modeling for (physical) biologists: an introduction to the rule-based approach. *Physical Biology*, 12(4): 045007, 2015. URL http://stacks.iop.org/1478-3975/12/i=4/a=045007.

Uroš Čibej and Jurij Mihelič. *Search Strategies for Subgraph Isomorphism Algorithms*, pages 77–88. Springer International Publishing, Cham, 2014. ISBN 978-3-319-04126-1. doi:10.1007/978-3-319-04126-1_7. URL http://dx.doi.org/10.1007/978-3-319-04126-1_7.

Uroš Čibej and Jurij Mihelič. Improvements to ullmann's algorithm for the subgraph isomorphism problem. *International Journal of Pattern Recognition and Artificial Intelligence*, 29 (07):1550025, 2015. doi:10.1142/S0218001415500251. URL http://www.worldscientific.com/doi/abs/10.1142/S0218001415500251.

Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70:066111, Dec 2004. doi:10.1103/PhysRevE.70.066111. URL https://link.aps.org/doi/10.1103/PhysRevE.70.066111.

R. Cohen and S. Havlin. *Complex Networks: Structure, Robustness and Function*. Cambridge University Press, 2010. ISBN 9781139489270. URL https://books.google.fr/books?id=1ECLiFrKulIC.

Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM. doi:10.1145/800157.805047. URL http://doi.acm.org/10.1145/800157.805047.

D. G. Corneil and C. C. Gotlieb. An efficient algorithm for graph isomorphism. *J. ACM*, 17(1): 51–64, January 1970. ISSN 0004-5411. doi:10.1145/321556.321562. URL http://doi.acm.org/10.1145/321556.321562.

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.

Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. B)*, pages 193–242. MIT Press, Cambridge, MA, USA, 1990. ISBN 0-444-88074-7. URL http://dl.acm.org/citation.cfm?id=114891.114896.

Haskell B Curry. *Foundations of Mathematical Logic*. McGraw-Hill, 1963. URL https://www.amazon.com/Foundations-Mathematical-Logic-Haskell-Curry/dp/B0000CLQZI.

Peter T. Daniels and William Bright. *The World's Writing Systems*. Oxford University Press, 1996. ISBN 9780195079937. URL https://books.google.fr/books?id=621jAAAAMAAJ.

Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, and Jean Krivine. Rule-based modelling of cellular signalling. In L. Caires and V. Vasconcelos, editors, *CONCUR 2007 - Concurrency Theory*, volume 4703 of *LNCS*, pages 17–41. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-74406-1. doi:10.1007/978-3-540-74407-8_3. URL http://dx.doi.org/10.1007/978-3-540-74407-8_3.

Juan de Lara and Hans Vangheluwe. AToM^3: A tool for multi-formalism and meta-modelling. In *Fundamental approaches to software engineering. 5th international conference, FASE 2002. Held as part of the joint European conferences on theory and practice of software, ETAPS 2002, Grenoble, France, April 8–12, 2002. Proceedings*, pages 174–188. Berlin: Springer, 2002. ISBN 3-540-43353-8.

Vanessa Hoffmann De Quadros, Juan Carlos González-Avella, and José Roberto Iglesias. Credit risk in interbank networks. *Emerging Markets Finance and Trade*, 51(sup6):S27–S41, 2015. doi:10.1080/1540496X.2015.1080554. URL http://dx.doi.org/10.1080/1540496X.2015.1080554.

Eric J. Deeds, Jean Krivine, Jérôme Feret, Vincent Danos, and Walter Fontana. Combinatorial complexity and compositional drift in protein interaction networks. *PLOS ONE*, 7(3):1–14, 03 2012. doi:10.1371/journal.pone.0032032. URL https://doi.org/10.1371/journal.pone.0032032.

W. Didimo and F. Montecchiani. Fast layout computation of hierarchically clustered networks: Algorithmic advances and experimental analysis. In *Information Visualisation (IV), 2012 16th International Conference on*, pages 18–23, July 2012. doi:10.1109/IV.2012.14.

Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag Berlin Heidelberg, 4 edition, 2000. ISBN 978-3-642-14278-9.

P.S. Dodds and D.J. Watts. A generalized model of social and biological contagion. *Journal of Theoretical Biology*, 232(4):587 – 604, 2005. ISSN 0022-5193. doi:http://dx.doi.org/10.1016/j.jtbi.2004.09.006. URL http://www.sciencedirect.com/science/article/pii/S0022519304004515.

Pedro Domingos and Matt Richardson. Mining the network value of customers. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 57–66, New York, NY, USA, 2001. ACM. ISBN 1-58113-391-X. doi:10.1145/502512.502525. URL http://doi.acm.org/10.1145/502512.502525.

Nan Du, Le Song, Manuel Gomez-Rodriguez, and Hongyuan Zha. Scalable influence estimation in continuous-time diffusion networks. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, NIPS'13, pages 3147–3155, USA, 2013. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999792.2999963.

F.S.L.G. Duarte, F. Sikansi, F.M. Fatore, S.G. Fadel, and F.V. Paulovich. Nmap: A novel neighborhood preservation space-filling algorithm. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):2063–2071, Dec 2014. ISSN 1077-2626. doi:10.1109/TVCG.2014.2346276.

Tim Dwyer, Kim Marriott, and Michael Wybrow. Topology preserving constrained graph layout. In IoannisG. Tollis and Maurizio Patrignani, editors, *Graph Drawing*, volume 5417 of *Lecture Notes in Computer Science*, pages 230–241. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-00218-2. doi:10.1007/978-3-642-00219-9_22. URL http://dx.doi.org/10.1007/978-3-642-00219-9_22.

Peter Eades. A Heuristic for Graph Drawing. *Congressus Numerantium*, 42:149–160, 1984.

V. Eberhard. *Zur morphologie der polyeder*. B.G. Teubner, 1891. URL https://books.google.fr/books?id=imb1z2f5yowC.

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*. World Scientific, 1997a.

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, 1997b.

Hartmut Ehrig, Michael Pfender, and Hans Jürgen Schneider. Graph-grammars: An algebraic approach. In 14^{th} *Annual Symp. on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 167–180, 1973.

Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1-3*. World Scientific, 1997c.

Paul Erdős and Alfréd Rényi. On random graphs i. *Publicationes Mathematicae (Debrecen)*, 6: 290–297, 1959.

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.

Claudia Ermel, Michael Rudolf, and Gabriele Taentzer. The AGG approach: Language and environment. 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 551–603. World Scientific, 1997.

Stephen Eubank, Hasan Guclu, V. S. Anil Kumar, Madhav V. Marathe, Aravind Srinivasan, Zoltan Toroczkai, and Nan Wang. Modelling disease outbreaks in realistic urban social networks. *Nature*, (6988):180–184, 2004. doi:10.1038/nature02541. URL http://www.nature.com/nature/journal/v429/n6988/full/nature02541.html.

James Faeder, Michael Blinov, and William Hlavacek. Rule-based modeling of biochemical systems with bionetgen. In I. V. Maly, editor, *Systems Biology*, volume 500 of *Methods in Molecular Biology*, pages 113–167. Humana Press, 2009. ISBN 978-1-934115-64-0. doi:10.1007/978-1-59745-525-1_5. URL http://dx.doi.org/10.1007/978-1-59745-525-1_5.

Jean-Daniel Fekete and Catherine Plaisant. Interactive information visualization of a million items. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, pages 117–124, 2002. doi:10.1109/INFVIS.2002.1173156.

Jean-Daniel Fekete, David Wang, Niem Dang, and Catherine Plaisant. Overlaying Graph Links on Treemaps. In IEEE, editor, *Information Visualization*, Seattle, United States, October 2003. IEEE. URL https://hal.inria.fr/hal-00875194.

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. ISBN 978-3-642-32210-5. doi:10.1007/978-3-642-32211-2_12. URL http://dx.doi.org/10.1007/978-3-642-32211-2_12.

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. doi:10.4204/EPTCS.159.3. URL http://dx.doi.org/10.4204/EPTCS.159.3.

Maribel Fernández, Hêiène Kirchner, and Bruno Pinaud. Strategic Port Graph Rewriting: An Interactive Modelling and Analysis Framework. Research report, Inria, January 2016a. URL https://hal.inria.fr/hal-01251871.

Maribel Fernández, Hélène Kirchner, Bruno Pinaud, and Jason Vallet. Labelled Graph Rewriting Meets Social Networks. In Dorel Lucanu, editor, *Rewriting Logic and Its Applications, WRLA 2016*, volume 9942 of *LNCS*, page 25, Eindhoven, Netherlands, April 2016b. Springer International Publishing Switzerland. doi:10.1007/978-3-319-44802-2_1. URL https://hal.inria.fr/hal-01347355.

Rudolf Fleischer and Colin Hirsch. *Graph Drawing and Its Applications*, pages 1–22. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. ISBN 978-3-540-44969-0. doi:10.1007/3-540-44969-8_1. URL https://doi.org/10.1007/3-540-44969-8_1.

Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3–5):75 – 174, 2010. ISSN 0370-1573. doi:http://dx.doi.org/10.1016/j.physrep.2009.11.002. URL http://www.sciencedirect.com/science/article/pii/S0370157309002841.

L. C. Freeman. *The Development of Social Network Analysis: A Study in the Sociology of Science*. Empirical Press, 2004.

Arne Frick, Andreas Ludwig, and Heiko Mehldau. A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration). In Roberto Tamassia and IoannisG. Tollis, editors, *Graph Drawing*, volume 894 of *Lecture Notes in Computer Science*, pages 388–403. Springer Berlin Heidelberg, 1995. ISBN 978-3-540-58950-1. doi:10.1007/3-540-58950-3_393. URL http://dx.doi.org/10.1007/3-540-58950-3_393.

Karinthy Frigyes. *Minden Másképpen Van*. Karinthy Frigyes works. Athenaeum Irodalmi és Nyomdai R.-T., 1929. URL https://www.antikvarium.hu/konyv/karinthy-frigyes-minden-maskeppen-van-275759.

Yaniv Frishman and Ayellet Tal. Multi-level graph layout on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1310–1319, November 2007a. ISSN 1077-2626. doi:10.1109/TVCG.2007.70580. URL http://dx.doi.org/10.1109/TVCG.2007.70580.

Yaniv Frishman and Ayellet Tal. Online dynamic graph drawing. In *Proceedings of the 9th Joint Eurographics / IEEE VGTC conference on Visualization*, EUROVIS'07, pages 75–82, Aire-la-Ville, Switzerland, Switzerland, 2007b. Eurographics Association. ISBN 978-3-905673-45-6. doi:10.2312/VisSym/EuroVis07/075-082. URL http://dx.doi.org/10.2312/VisSym/EuroVis07/075-082.

Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 21(11):1129–1164, November 1991. ISSN 0038-0644. doi:10.1002/spe.4380211102. URL http://dx.doi.org/10.1002/spe.4380211102.

Marco Gaertler. *Clustering*, pages 178–215. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31955-9. doi:10.1007/978-3-540-31955-9_8. URL https://doi.org/10.1007/978-3-540-31955-9_8.

A Ganesh, L. Massoulie, and D. Towsley. The effect of network topology on the spread of epidemics. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 2, pages 1455–1466 vol. 2, March 2005. doi:10.1109/INFCOM.2005.1498374.

Rubino Geiß, Gernot Veit Batz, Daniel Grund, Sebastian Hack, and Adam Szalkowski. GrGen: A Fast SPO-Based Graph Rewriting Tool. In *Proc. of ICGT*, volume 4178 of *LNCS*, pages 383–397. Springer, 2006.

Mohammad Ghoniem, Jean-Daniel Fekete, and Philippe Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. In *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*, pages 17–24, 2004. doi:10.1109/INFVIS.2004.1.

Mohammad Ghoniem, Jean-Daniel Fekete, and Philippe Castagliola. On the readability of graphs using node-link and matrix-based representations: A controlled experiment and statistical analysis. *Information Visualization*, 4(2):114–135, July 2005. ISSN 1473-8716. doi:10.1057/palgrave.ivs.9500092. URL http://dx.doi.org/10.1057/palgrave.ivs.9500092.

George Giakkoupis, Rachid Guerraoui, Arnaud Jégou, Anne-Marie Kermarrec, and Nupur Mittal. Privacy-Conscious Information Diffusion in Social Networks. In Yoram Moses and Matthieu Roy, editors, *DISC 2015*, volume LNCS 9363 of *29th International Symposium on Distributed Computing*, Tokyo, Japan, October 2015. Toshimitsu Masuzawa and Koichi Wada, Springer-Verlag Berlin Heidelberg. doi:10.1007/978-3-662-48653-5_32. URL https://hal.archives-ouvertes.fr/hal-01207162.

E. N. Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 12 1959. doi:10.1214/aoms/1177706098. URL https://doi.org/10.1214/aoms/1177706098.

Jacob Goldenberg, Barak Libai, and Eitan Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, 12(3):211–223, 2001. ISSN 0923-0645. doi:10.1023/A:1011122126881. URL http://dx.doi.org/10.1023/A%3A1011122126881.

Manuel Gomez-Rodriguez and Bernhard Schölkopf. Influence maximization in continuous time diffusion networks. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 313–320, New York, NY, USA, 2012. ACM. URL http://icml.cc/2012/papers/189.pdf.

Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 1019–1028, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0055-1. doi:10.1145/1835804.1835933. URL http://doi.acm.org/10.1145/1835804.1835933.

Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. *ACM Trans. Knowl. Discov. Data*, 5(4):21:1–21:37, February 2012. ISSN 1556-4681. doi:10.1145/2086737.2086741. URL http://doi.acm.org/10.1145/2086737.2086741.

Manuel Gomez-Rodriguez, Jure Leskovec, David Balduzzi, and Bernhard Schölkopf. Uncovering the structure and temporal dynamics of information propagation. *Network Science*, 2:26–65, 4 2014. ISSN 2050-1250. doi:10.1017/nws.2014.3. URL http://journals.cambridge.org/article_S2050124214000034.

R. Gove, N. Gramsky, R. Kirby, E. Sefer, A. Sopan, C. Dunne, B. Shneiderman, and M. Taieb-Maimon. Netvisia: Heat map and matrix visualization of dynamic social network statistics and content. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational*

Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on, pages 19–26, Oct 2011. doi:10.1109/PASSAT/SocialCom.2011.216.

Amit Goyal, Francesco Bonchi, and Laks V.S. Lakshmanan. Learning influence probabilities in social networks. In *Web Search and Data Mining, Proceedings of the Third ACM International Conference on*, WSDM '10, pages 241–250, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-889-6. doi:10.1145/1718487.1718518. URL http://doi.acm.org/10.1145/1718487.1718518.

M. Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83(6):1420, 1978. URL https://sociology.stanford.edu/publications/threshold-models-collective-behavior.

John Guare. *Six Degrees of Separation: A Play*. Vintage Series. Vintage Books, 1990. ISBN 9780679734819. URL https://books.google.fr/books?id=TJsZqfHCUskC.

Stephen J. Guastello. *Human Factors Engineering and Ergonomics: A Systems Approach, Second Edition*. Taylor & Francis, 2013. ISBN 9781466560093. URL https://books.google.fr/books?id=nyItAgAAQBAJ.

Annegret Habel, Jürgen Müller, and Detlef Plump. Double-pushout graph transformation revisited. *Mathematical Structures in Computer Science*, 11(5):637–688, 2001.

Stefan Hachul and Michael Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In János Pach, editor, *Graph Drawing*, volume 3383 of *Lecture Notes in Computer Science*, pages 285–295. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-24528-5. doi:10.1007/978-3-540-31843-9_29. URL http://dx.doi.org/10.1007/978-3-540-31843-9_29.

Stefan Hachul and Michael Jünger. Large-graph layout algorithms at work: An experimental study. *Journal of Graph Algorithms and Applications*, 11(2):345–369, 2007. doi:10.7155/jgaa.00150.

Russ Harmer, Vincent Danos, Jérôme Feret, Jean Krivine, and Walter Fontana. Intrinsic information carriers in combinatorial dynamical systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 20(3):037108, 2010. doi:10.1063/1.3491100. URL http://dx.doi.org/10.1063/1.3491100.

Mark Harrower and Cynthia A. Brewer. Colorbrewer.org: An online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1):27–37, 2003. doi:10.1179/000870403235002042. URL http://www.tandfonline.com/doi/abs/10.1179/000870403235002042.

C. G. Healey. Choosing effective colours for data visualization. In *Visualization '96. Proceedings.*, pages 263–270, Oct 1996. doi:10.1109/VISUAL.1996.568118.

Christopher G. Healey, Kellogg S. Booth, and James T. Enns. High-speed visual estimation using preattentive processing. Technical report, Vancouver, BC, Canada, Canada, 1993.

Fritz Heider. Attitudes and cognitive organization. *The Journal of Psychology*, 21(1):107–112, 1946. doi:10.1080/00223980.1946.9917275. URL http://dx.doi.org/10.1080/00223980.1946.9917275. PMID: 21010780.

N. Henry and J. D. Fekete. Matrixexplorer: a dual-representation system to explore social networks. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):677–684, Sept 2006. ISSN 1077-2626. doi:10.1109/TVCG.2006.160.

Nathalie Henry and Jean-Daniel Fekete. Matlink: Enhanced matrix visualization for analyzing social networks. In Cécilia Baranauskas, Philippe Palanque, Julio Abascal, and SimoneDinizJunqueira Barbosa, editors, *Human-Computer Interaction – INTERACT 2007*, volume 4663 of *Lecture Notes in Computer Science*, pages 288–302. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-74799-4. doi:10.1007/978-3-540-74800-7_24. URL http://dx.doi.org/10.1007/978-3-540-74800-7_24.

Nathalie Henry, Jean-Daniel. Fekete, and Michael J. McGuffin. Nodetrix: a hybrid visualization of social networks. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1302–1309, Nov 2007. ISSN 1077-2626. doi:10.1109/TVCG.2007.70582.

Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, January 2000. ISSN 1077-2626. doi:10.1109/2945.841119. URL http://dx.doi.org/10.1109/2945.841119.

H. Hethcote. The mathematics of infectious diseases. *SIAM Review*, 42(4):599–653, 2000. doi:10.1137/S0036144500371907. URL http://epubs.siam.org/doi/abs/10.1137/S0036144500371907. Cited 1051.

David Hilbert. Ueber die stetige Abbildung einer Line auf ein Flächenstück. *Mathematische Annalen*, 38(3):459–460, 1891. ISSN 0025-5831. doi:10.1007/BF01199431. URL http://dx.doi.org/10.1007/BF01199431.

Petter Holme and Beom Jun Kim. Growing scale-free networks with tunable clustering. *Phys. Rev. E*, 65:026107, Jan 2002. doi:10.1103/PhysRevE.65.026107. URL http://link.aps.org/doi/10.1103/PhysRevE.65.026107.

Mao Lin Huang and Quang Vinh Nguyen. A space efficient clustered visualization of large graphs. In *Image and Graphics, 2007. ICIG 2007. Fourth International Conference on*, pages 920–927, Aug 2007. doi:10.1109/ICIG.2007.10.

Weidong Huang, Peter Eades, Seok-Hee Hong, and Chun-Cheng Lin. Improving multiple aesthetics produces better graph drawings. *Journal of Visual Languages & Computing*, 24(4): 262 – 272, 2013. ISSN 1045-926X. doi:http://dx.doi.org/10.1016/j.jvlc.2011.12.002. URL http://www.sciencedirect.com/science/article/pii/S1045926X11000814.

Yannis Ioannides. Random graphs and social networks: An economics perspective. Discussion Papers Series, Department of Economics, Tufts University 0518, Department of Economics, Tufts University, 2005. URL http://EconPapers.repec.org/RePEc:tuf:tuftec:0518.

T. Itoh, C. Muelder, Kwan-Liu Ma, and J. Sese. A hybrid space-filling and force-directed layout method for visualizing multiple-category graphs. In *Visualization Symposium, 2009. PacificVis '09. IEEE Pacific*, pages 121–128, April 2009. doi:10.1109/PACIFICVIS.2009.4906846.

Matthew O. Jackson, David Martimort, Philippe Jehiel, Benny Moldovanu, Larry Samuelson, Dirk Bergemann, Juuso Valimaki, Ilya Segal, Narayana R. Kocherlakota, Per Krusell, Anthony Smith, Daron Acemoglu, Antonio Merlo, and Timothy Besley. *Advances in Economics and Econometrics: Theory and Applications, Ninth World Congress*, volume 1 of *Econometric Society Monographs*. Cambridge University Press, Oct 2006. ISBN 9780521692083. 462 pp.

Brian Johnson and Ben Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2Nd Conference on Visualization '91*, VIS '91, pages 284–291, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press. ISBN 0-8186-2245-8. URL http://dl.acm.org/citation.cfm?id=949607.949654.

Daniel A. Keim. Pixel-oriented visualization techniques for exploring very large data bases. *Journal of Computational and Graphical Statistics*, 5(1):58–77, 1996. doi:10.1080/10618600.1996.10474695. URL http://www.tandfonline.com/doi/abs/10.1080/10618600.1996.10474695.

Daniel A. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, January 2002. ISSN 1077-2626. doi:10.1109/2945.981847. URL http://dx.doi.org/10.1109/2945.981847.

N. Kejžar, Z. Nikoloski, and V. Batagelj. Probabilistic inductive classes of graphs. *ArXiv Mathematics e-prints*, December 2006. URL https://arxiv.org/abs/math/0612778.

N. Kejžar, Z. Nikoloski, and V. Batagelj. Probabilistic inductive classes of graphs. *The Journal of Mathematical Sociology*, 32(2):85–109, 2008. doi:10.1080/00222500801931586. URL http://dx.doi.org/10.1080/00222500801931586.

David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 137–146, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0. doi:10.1145/956750.956769. URL http://doi.acm.org/10.1145/956750.956769.

David Kempe, Jon Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In Luís Caires, GiuseppeF. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 1127–1138. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-27580-0. doi:10.1007/11523468_91. URL http://dx.doi.org/10.1007/11523468_91.

Richard Kennaway. On "on graph rewritings". *Theor. Comput. Sci.*, 52(1–2):37–58, 1987. doi:10.1016/0304-3975(87)90079-X. URL http://dx.doi.org/10.1016/0304-3975(87)90079-X.

N. Kerracher, J. Kennedy, and K. Chalmers. A task taxonomy for temporal graph visualisation. *Visualization and Computer Graphics, IEEE Transactions on*, 21(10):1160–1172, Oct 2015. ISSN 1077-2626. doi:10.1109/TVCG.2015.2424889.

Elias Khalil, Bistra Dilkina, and Le Song. Cuttingedge: Influence minimization in networks. In *Workshop on Frontiers of Network Analysis: Methods, Models, and Applications at NIPS*, 2013. URL http://snap.stanford.edu/networks2013/papers/netnips2013_submission_9.pdf.

Jinhyun Kim, HyukGeun Choi, Hansang Yun, and Byung-Ro Moon. Measuring source code similarity by finding similar subgraph with an incremental genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 925–932, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4206-3. doi:10.1145/2908812.2908870. URL http://doi.acm.org/10.1145/2908812.2908870.

Masahiro Kimura and Kazumi Saito. Tractable models for information diffusion in social networks. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Knowledge Discovery in Databases: PKDD 2006*, volume 4213 of *Lecture Notes in Computer Science*, pages 259–271. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-45374-1. doi:10.1007/11871637_27. URL http://dx.doi.org/10.1007/11871637_27.

Masahiro Kimura, Kazumi Saito, and Hiroshi Motoda. Solving the contamination minimization problem on networks for the linear threshold model. In Tu-Bao Ho and Zhi-Hua Zhou, editors, *PRICAI 2008: Trends in Artificial Intelligence*, volume 5351 of *Lecture Notes in Computer Science*, pages 977–984. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-89196-3. doi:10.1007/978-3-540-89197-0_94. URL http://dx.doi.org/10.1007/978-3-540-89197-0_94.

Masahiro Kimura, Kazumi Saito, and Hiroshi Motoda. Blocking links to minimize contamination spread in a social network. *ACM Trans. Knowl. Discov. Data*, 3(2):9:1–9:23, April 2009a. ISSN 1556-4681. doi:10.1145/1514888.1514892. URL http://doi.acm.org/10.1145/1514888.1514892.

Masahiro Kimura, Kazumi Saito, and Hiroshi Motoda. Efficient estimation of influence functions for sis model on social networks. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 2046–2051, San Francisco, CA, USA, 2009b. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1661445.1661772.

Hélène Kirchner. Rewriting strategies and strategic rewrite programs. In *Logic, Rewriting, and Concurrency (LRC 2015), Festschrift Symposium in Honor of José Meseguer*, Lecture Notes in Computer Science. Springer, 2015. URL https://hal.inria.fr/hal-01143486.

Konstantin Klemm and Víctor M. Eguíluz. Growing scale-free networks with small-world behavior. *Phys. Rev. E*, 65:057102, May 2002. doi:10.1103/PhysRevE.65.057102. URL http://link.aps.org/doi/10.1103/PhysRevE.65.057102.

Bryan Klimt and Yiming Yang. *Machine Learning: ECML 2004: 15th European Conference on Machine Learning, Pisa, Italy, September 20-24, 2004. Proceedings*, chapter The Enron Corpus: A New Dataset for Email Classification Research, pages 217–226. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-30115-8. doi:10.1007/978-3-540-30115-8_22. URL http://dx.doi.org/10.1007/978-3-540-30115-8_22.

Stephen G. Kobourov. Spring embedders and force directed graph drawing algorithms. *CoRR*, abs/1201.3011, 2012. URL http://arxiv.org/abs/1201.3011.

Robert Koenig. International groups battle cholera in zimbabwe. *Science*, 323(5916):860–861, 2009. ISSN 0036-8075. doi:10.1126/science.323.5916.860. URL http://science.sciencemag.org/content/323/5916/860.

Barbara König and Vitali Kozioura. AUGUR 2 – *a new version of a tool for the analysis of graph transformation systems.*, pages 201–210. Amsterdam: Elsevier, 2008. doi:10.1016/j.entcs.2008.04.042.

Christopher M. Kribs-Zaleta and Jorge X. Velasco-Hernández. A simple vaccination model with multiple endemic states. *Mathematical Biosciences*, 164(2):183 – 201, 2000. ISSN 0025-5564. doi:http://dx.doi.org/10.1016/S0025-5564(00)00003-1. URL http://www.sciencedirect.com/science/article/pii/S0025556400000031. Cited 147.

Marcelo Kuperman and Guillermo Abramson. Small world effect in an epidemiological model. *Phys. Rev. Lett.*, 86:2909–2912, Mar 2001. doi:10.1103/PhysRevLett.86.2909. URL http://link.aps.org/doi/10.1103/PhysRevLett.86.2909.

Yves Lafont. Interaction nets. In *Proc. of the* 17^{th} *ACM Symp. on Principles of Programming Languages (POPL'90)*, pages 95–108. ACM Press, 1990.

Steve Lawrence and C. Lee Giles. Searching the World Wide Web. *Science*, 280(5360):98–100, 1998. URL citeseer.nj.nec.com/lawrence98searching.html.

M.V. Lawson. *Finite Automata*. Taylor & Francis, 2003. ISBN 9781584882558. URL https://books.google.fr/books?id=MDQ_K7-z2AMC.

Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, and Nathalie Henry. Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization*, BELIV '06, pages 1–5, New York, NY, USA, 2006. ACM. ISBN 1-59593-562-2. doi:10.1145/1168149.1168168. URL http://doi.acm.org/10.1145/1168149.1168168.

Jinsoo Lee, Wook-Shin Han, Romans Kasperovics, and Jeong-Hoon Lee. An in-depth comparison of subgraph isomorphism algorithms in graph databases. *Proc. VLDB Endow.*, 6(2):133–144, December 2012. ISSN 2150-8097. doi:10.14778/2535568.2448946. URL http://dx.doi.org/10.14778/2535568.2448946.

Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *CoRR*, abs/0810.1355, 2008. URL http://arxiv.org/abs/0810.1355.

Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Predicting positive and negative links in online social networks. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 641–650, New York, NY, USA, 2010a. ACM. ISBN 978-1-60558-799-8. doi:10.1145/1772690.1772756. URL http://doi.acm.org/10.1145/1772690.1772756.

Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1361–1370, New York, NY, USA, 2010b. ACM. ISBN 978-1-60558-929-9. doi:10.1145/1753326.1753532. URL http://doi.acm.org/10.1145/1753326.1753532.

Angsheng Li and Linqing Tang. The complexity and approximability of minimum contamination problems. In *Proceedings of the 8th Annual Conference on Theory and Applications of Models of Computation*, TAMC'11, pages 298–307, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-20876-8. URL http://dl.acm.org/citation.cfm?id=2008967.2009009.

Fredrik Liljeros, Christofer R. Edling, and Luis A.Nunes Amaral. Sexual networks: implications for the transmission of sexually transmitted infections. *Microbes and Infection*, 5(2):189 – 196, 2003. ISSN 1286-4579. doi:http://dx.doi.org/10.1016/S1286-4579(02)00058-8. URL http://www.sciencedirect.com/science/article/pii/S1286457902000588.

Shuyang Lin, Qingbo Hu, Guan Wang, and PhilipS. Yu. Understanding community effects on information diffusion. In Tru Cao, Ee-Peng Lim, Zhi-Hua Zhou, Tu-Bao Ho, David Cheung,

and Hiroshi Motoda, editors, *Advances in Knowledge Discovery and Data Mining*, volume 9077 of *Lecture Notes in Computer Science*, pages 82–95. Springer International Publishing, 2015. ISBN 978-3-319-18037-3. doi:10.1007/978-3-319-18038-0_7. URL http://dx.doi.org/10.1007/978-3-319-18038-0_7.

Pedro G. Lind, Luciano R. da Silva, José S. Andrade, and Hans J. Herrmann. Spreading gossip in social networks. *Phys. Rev. E*, 76:036117, Sep 2007. doi:10.1103/PhysRevE.76.036117. URL http://link.aps.org/doi/10.1103/PhysRevE.76.036117.

Michael Löwe. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, 109:181–224, 1993.

Michael Löwe, Martin Korff, and Annika 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. ISBN 0-471-93567-0. URL http://dl.acm.org/citation.cfm?id=167817.167848.

J. Lu, X. Yu, and W. Wan. Visualization research of the tweet diffusion in the microblog network. In *2014 International Conference on Audio, Language and Image Processing*, pages 592–595, July 2014. doi:10.1109/ICALIP.2014.7009863.

Yao Lu, Kaizhu Huang, and Cheng-Lin Liu. A fast projected fixed-point algorithm for large graph matching. *Pattern Recogn.*, 60(C):971–982, December 2016. ISSN 0031-3203. doi:10.1016/j.patcog.2016.07.015. URL https://doi.org/10.1016/j.patcog.2016.07.015.

Chuan Luo, Kainan Cui, Xiaolong Zheng, and D. Zeng. Time critical disinformation influence minimization in online social networks. In *Intelligence and Security Informatics Conference (JISIC), 2014 IEEE Joint*, pages 68–74, Sept 2014. doi:10.1109/JISIC.2014.20.

Michael W. Macy. Chains of cooperation: Threshold effects in collective action. *American Sociological Review*, 56(6):730–747, December 1991. doi:10.2307/2096252. URL http://www.jstor.org/stable/2096252. ISSN: 0003-1224.

N. Madar, T. Kalisky, R. Cohen, D. Ben-Avraham, and S. Havlin. Immunization and epidemic dynamics in complex networks. *The European Physical Journal B - Condensed Matter and Complex Systems*, 38(2):269–276, 2004. ISSN 1434-6028. doi:10.1140/epjb/e2004-00119-8. URL http://dx.doi.org/10.1140/epjb/e2004-00119-8.

Benoît B. Mandelbrot. *Fractals: Form, Chance, and Dimension*. Freeman, 1977. URL https://books.google.fr/books?id=8DGFmgEACAAJ.

Ciaran McCreesh and Patrick Prosser. A parallel, backjumping subgraph isomorphism algorithm using supplemental graphs. In *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming*, CP'15, pages 295–312, Switzerland, 2015. Springer. ISBN 978-3-319-23218-8. doi:10.1007/978-3-319-23219-5_21. URL https://doi.org/10.1007/978-3-319-23219-5_21.

Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara, and Alessandro Provetti. Generalized louvain method for community detection in large networks. *CoRR*, abs/1108.1502, 2011. URL http://arxiv.org/abs/1108.1502.

Miriah Meyer, Michael Sedlmair, and Tamara Munzner. The four-level nested model revisited: Blocks and guidelines. In *Proceedings of the 2012 BELIV Workshop: Beyond Time and Errors*

- *Novel Evaluation Methods for Visualization*, BELIV '12, pages 11:1–11:6, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1791-7. doi:10.1145/2442576.2442587. URL http://doi.acm.org/10.1145/2442576.2442587.

Gergely Mezei, Sándor Juhász, and Tihamer Levendovszky. A distribution technique for graph rewriting and model transformation systems. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks, as part of the 25th IASTED International Multi-Conference on Applied Informatics, February 13-15 2007, Innsbruck, Austria*, pages 63–68, 2007.

Stanley Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.

Ugo Montanari. Separable graphs, planar graphs and web grammars. *Information and Control*, 16(3):243–267, 1970. doi:10.1016/S0019-9958(70)90135-X. URL https://doi.org/10.1016/S0019-9958(70)90135-X.

Cristopher Moore and M. E. J. Newman. Epidemics and percolation in small-world networks. *Phys. Rev. E*, 61:5678–5682, May 2000. doi:10.1103/PhysRevE.61.5678. URL http://link.aps.org/doi/10.1103/PhysRevE.61.5678.

Jacob L. Moreno. *Who Shall Survive: A New Approach to the Problem of Human Interrelations*. 1934. URL https://archive.org/details/whoshallsurviven00jlmo.

G. M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, IBM Ltd., Ottawa, Canada, 1966. URL http://domino.research.ibm.com/library/cyberdig.nsf/0/0dabf9473b9c86d48525779800566a39?OpenDocument.

Tomer Moscovich, Fanny Chevalier, Nathalie Henry, Emmanuel Pietriga, and Jean-Daniel Fekete. Topology-aware navigation in large networks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 2319–2328, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-246-7. doi:10.1145/1518701.1519056. URL http://doi.acm.org/10.1145/1518701.1519056.

Amal Moussa. *Contagion and Systemic Risk in Financial Networks*. PhD thesis, Columbia University, 2011.

Fangping Mu, Robert F. Williams, Clifford J. Unkefer, Pat J. Unkefer, James R. Faeder, and William S. Hlavacek. Carbon-fate maps for metabolic reactions. *Bioinformatics*, 23 (23):3193, 2007. doi:10.1093/bioinformatics/btm498. URL +http://dx.doi.org/10.1093/bioinformatics/btm498.

Chris Muelder and Kwan-Liu Ma. Rapid graph layout using space filling curves. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1301–1308, Nov 2008. ISSN 1077-2626. doi:10.1109/TVCG.2008.158.

Anders K. Munk, Mette S. Abildgaard, Andreas Birkbak, and Morten K. Petersen. (Re-)Appropriating Instagram for Social Research: Three Methods for Studying Obesogenic Environments. In *Proc. of the 7th 2016 Int. Conf. on Social Media & Society*, SMSociety '16, pages 19:1–19:10. ACM, 2016. ISBN 978-1-4503-3938-4. doi:10.1145/2930971.2930991. URL http://doi.acm.org/10.1145/2930971.2930991.

Tamara Munzner. A nested model for visualization design and validation. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):921–928, November 2009. ISSN 1077-2626. doi:10.1109/TVCG.2009.111. URL http://dx.doi.org/10.1109/TVCG.2009.111.

Tamara Munzner. *Visualization Analysis & Design*. A K Peters Visualization. CRC Press, 2014. ISBN 9781498759717.

John Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, 1951. URL http://jmvidal.cse.sc.edu/library/nash51a.pdf.

Kawa Nazemi, Matthias Breyer, and Arjan Kuijper. User-oriented graph visualization taxonomy: A data-oriented examination of visual features. In Masaaki Kurosu, editor, *Human Centered Design*, volume 6776 of *Lecture Notes in Computer Science*, pages 576–585. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-21752-4. doi:10.1007/978-3-642-21753-1_64. URL http://dx.doi.org/10.1007/978-3-642-21753-1_64.

M. E. J. Newman. Spread of epidemic disease on networks. *Phys. Rev. E*, 66:016128, Jul 2002. doi:10.1103/PhysRevE.66.016128. URL http://link.aps.org/doi/10.1103/PhysRevE.66.016128.

M. E. J. Newman. Communities, modules and large-scale structure in networks. *Nature Physics*, 8:25–31, January 2012. doi:10.1038/nphys2162.

Mark Newman, Albert-László Barabási, and Duncan J. Watts. *The structure and dynamics of networks*. Princeton Studies in Complexity. Princeton University Press, 2006.

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 International Conference on*, pages 525–532, Aug 2013. doi:10.1145/2492517.2492569.

Ulrich Nickel, Jörg Niere, and Albert Zündorf. The FUJABA environment. In *ICSE*, pages 742–745, 2000.

Sadegh Nobari, Xuesong Lu, Panagiotis Karras, and Stéphane Bressan. Fast random graph generation. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 331–342, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0528-0. doi:10.1145/1951365.1951406. URL http://doi.acm.org/10.1145/1951365.1951406.

Mancur Olson. *The Logic of Collective Action: Public Goods and the Theory of Groups, Second Printing with New Preface and Appendix*. American studies collection. Harvard University Press, 1965. ISBN 9780674537514. URL https://books.google.fr/books?id=jzTeOLtf7_wC.

A. James O'Malley. The analysis of social network data: an exciting frontier for statisticians. *Statistics in Medicine*, 32(4):539–555, 2013. doi:10.1002/sim.5630.

Sirinya On-at, Arnaud Quirin, André Péninou, Nadine Baptiste-Jessel, Marie-Françoise Canut, and Florence Sèdes. *A Parametric Study to Construct Time-Aware Social Profiles*, pages 21–50. Springer International Publishing, Cham, 2017. ISBN 978-3-319-53420-6. doi:10.1007/978-3-319-53420-6_2. URL https://doi.org/10.1007/978-3-319-53420-6_2.

J.-P. Onnela, J. Saram aki, J. Hyvönen, G. Szabó, D. Lazer, K. Kaski, J. Kertész, and Albert-László Barabási. Structure and tie strengths in mobile communication networks. *Proceedings of the National Academy of Sciences*, 104(18):7332–7336, 2007. doi:10.1073/pnas.0610245104. URL http://www.pnas.org/content/104/18/7332.abstract.

James S. Ormrod. *Smelser's Theory of Collective Behaviour*, pages 184–199. Palgrave Macmillan UK, London, 2014. ISBN 978-1-137-34817-3. doi:10.1057/9781137348173_7. URL https://doi.org/10.1057/9781137348173_7.

Romualdo Pastor-Satorras and Alessandro Vespignani. Epidemic spreading in scale-free networks. *Phys. Rev. Lett.*, 86:3200–3203, Apr 2001. doi:10.1103/PhysRevLett.86.3200. URL http://link.aps.org/doi/10.1103/PhysRevLett.86.3200.

G. Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36(1): 157–160, 1890. ISSN 0025-5831. doi:10.1007/BF01199438. URL http://dx.doi.org/10.1007/BF01199438.

John L. Pfaltz and Azriel Rosenfeld. Web grammars. In D. E. Walker and L. M. Norton, editors, *Proceedings of the 1st International Joint Conference on Artificial Intelligence, Washington, DC, USA, May 7-9, 1969*, pages 609–620. William Kaufmann, 1969. ISBN 0-934613-21-4. URL http://ijcai.org/Proceedings/69/Papers/054.pdf.

Bruno Pinaud, Dubois Jonathan, and Guy Melançon. Porgy: Interactive and visual reasoning with graph rewriting systems. In *Conf. on Visual Analytics Science and Technology (VAST), 2011 IEEE (Poster Abstract)*, pages 293–294, Providence, United States, October 2011. IEEE. doi:10.1109/VAST.2011.6102480. URL http://hal.inria.fr/inria-00617547.

Bruno Pinaud, Guy Melançon, and Jonathan Dubois. Porgy: A visual graph rewriting environment for complex systems. *Computer Graphics Forum*, 31(3):1265–1274, 2012. doi:10.1111/j.1467-8659.2012.03119.x. URL http://hal.inria.fr/hal-00682550.

Pedro C. Pinto, Patrick Thiran, and Martin Vetterli. Locating the source of diffusion in large-scale networks. *Phys. Rev. Lett.*, 109:068702, Aug 2012. doi:10.1103/PhysRevLett.109.068702. URL http://link.aps.org/doi/10.1103/PhysRevLett.109.068702.

D. Plump. The graph programming language gp. In Symeon Bozapalidis and George Rahonis, editors, *Algebraic Informatics*, volume 5725 of *Lecture Notes in Computer Science*, pages 99–122. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-03563-0. doi:10.1007/978-3-642-03564-7_6.

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.

Detlef Plump. The design of GP 2. In S. Escobar, editor, *Proc. 10^{th} Int. Workshop on Reduction Strategies in Rewriting and Programming, WRS 2011, Novi Sad, Serbia, 29 May 2011.*, volume 82 of *EPTCS*, pages 1–16, 2011. doi:10.4204/EPTCS.82.1. URL http://dx.doi.org/10.4204/EPTCS.82.1.

Mathias Pohl, Markus Schmitt, and Stephan Diehl. Comparing the readability of graph layouts using eyetracking and task-oriented analysis. In *Proceedings of the Fifth Eurographics Conference on Computational Aesthetics in Graphics, Visualization and Imaging*, Computational Aesthetics'09, pages 49–56, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association. ISBN 978-3-905674-17-0. doi:10.2312/COMPAESTH/COMPAESTH09/049-056. URL http://dx.doi.org/10.2312/COMPAESTH/COMPAESTH09/049-056,http://de.evo-art.org/index.php?title=Comparing_the_Readability_of_Graph_Layouts_using_Eyetracking_and_Task-oriented_Analysis.

Helen Purchase. *Which aesthetic has the greatest effect on human understanding?*, pages 248–261. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. ISBN 978-3-540-69674-2. doi:10.1007/3-540-63938-1_67. URL https://doi.org/10.1007/3-540-63938-1_67.

Helen C. Purchase. *Experimental Human-Computer Interaction: A Practical Guide With Visual Examples*. Cambridge University Press, New York, NY, USA, 2012. URL http://eprints.gla.ac.uk/78680/.

Helen C. Purchase, Eve Hoggan, and Carsten Görg. How important is the "mental map"?: an empirical investigation of a dynamic graph layout algorithm. In *Proceedings of the 14th international conference on Graph drawing*, GD'06, pages 184–195, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-70903-9. URL http://dl.acm.org/citation.cfm?id=1758612.1758633.

X. Que, F. Checconi, F. Petrini, and J. A. Gunnels. Scalable community detection with the louvain algorithm. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 28–37, May 2015. doi:10.1109/IPDPS.2015.59.

Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76:036106, Sep 2007. doi:10.1103/PhysRevE.76.036106. URL https://link.aps.org/doi/10.1103/PhysRevE.76.036106.

Jean-Claude Raoult. On graph rewritings. *Theor. Comput. Sci.*, 32:1–24, 1984. doi:10.1016/0304-3975(84)90021-5. URL http://dx.doi.org/10.1016/0304-3975(84)90021-5.

S. Redner. How popular is your paper? an empirical study of the citation distribution. *The European Physical Journal B - Condensed Matter and Complex Systems*, 4(2):131–134, 1998. ISSN 1434-6036. doi:10.1007/s100510050359. URL http://dx.doi.org/10.1007/s100510050359.

Arend Rensink. The GROOVE Simulator: A Tool for State Space Generation. In JohnL. Pfaltz, Manfred Nagl, and Boris Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance*, volume 3062 of *Lecture Notes in Computer Science*, pages 479–485. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22120-3. doi:10.1007/978-3-540-25959-6_40.

Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 61–70, New York, NY, USA, 2002. ACM. ISBN 1-58113-567-X. doi:10.1145/775047.775057. URL http://doi.acm.org/10.1145/775047.775057.

George Robertson, Danyel Fisher, Bongshin Lee, John Stasko, and Roland Fernandez. Effectiveness of animation in trend visualization. In *IEEE TVCG (InfoVis 2008)*, January 2008. URL https://www.microsoft.com/en-us/research/publication/effectiveness-of-animation-in-trend-visualization/.

Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008. doi:10.1073/pnas.0706851105. URL http://www.pnas.org/content/105/4/1118.abstract.

Tarik Roukny, Stefano Battiston, and Joseph E. Stiglitz. Interconnectedness as a source of uncertainty in systemic risk. *Journal of Financial Stability*, 2016. ISSN 1572-3089. doi:http://dx.doi.org/10.1016/j.jfs.2016.12.003. URL http://www.sciencedirect.com/science/article/pii/S1572308916302200.

G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

Sébastien Rufiange, Michael J. McGuffin, and Christopher P. Fuhrman. Treematrix: A hybrid visualization of compound graphs. *Computer Graphics Forum*, 31(1):89–101, 2012. ISSN 1467-8659. doi:10.1111/j.1467-8659.2011.02087.x. URL http://dx.doi.org/10.1111/j.1467-8659.2011.02087.x.

Kazumi Saito, Masahiro Kimura, Kouzou Ohara, and Hiroshi Motoda. Efficient estimation of cumulative influence for multiple activation information diffusion model with continuous time delay. In Byoung-Tak Zhang and MehmetA. Orgun, editors, *PRICAI 2010: Trends in Artificial Intelligence*, volume 6230 of *Lecture Notes in Computer Science*, pages 244–255. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-15245-0. doi:10.1007/978-3-642-15246-7_24. URL http://dx.doi.org/10.1007/978-3-642-15246-7_24.

Kazumi Saito, Kouzou Ohara, Yuki Yamagishi, Masahiro Kimura, and Hiroshi Motoda. Learning diffusion probability based on node attributes in social networks. In Marzena Kryszkiewicz, Henryk Rybinski, Andrzej Skowron, and ZbigniewW. Raś, editors, *Foundations of Intelligent Systems*, volume 6804 of *Lecture Notes in Computer Science*, pages 153–162. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-21915-3. doi:10.1007/978-3-642-21916-0_18. URL http://dx.doi.org/10.1007/978-3-642-21916-0_18.

Arnaud Sallaberry, Faraz Zaidi, and Guy Melançon. Model for generating artificial social networks having community structures with small-world and scale-free properties. *Social Network Analysis and Mining*, 3(3):597–609, 2013. ISSN 1869-5469. doi:10.1007/s13278-013-0105-0. URL http://dx.doi.org/10.1007/s13278-013-0105-0.

Joris Sansen, Romain Bourqui, Bruno Pinaud, and Helen Purchase. Edge visual encodings in matrix-based diagrams. In *Proc. of the 19th Int. Conf. on Information Visualisation*, IV '15, 2015. URL https://hal.archives-ouvertes.fr/hal-01189166.

Hans Jürgen Schneider. Chomsky-Systeme für partielle Ordnungen. Technical Report 3, Universität, Erlangen, 1970.

Tobias Schreck, Daniel Keim, and Florian Mansmann. Regular treemap layouts for visual analysis of hierarchical data. In *In Spring Conference on Computer Graphics (SCCG'2006), April 20-22, Casta Papiernicka, Slovak Republic. ACM Siggraph*, pages 184–191, 2006.

Andy Schürr, Andreas J. Winter, and Albert Zündorf. The PROGRES Approach: Language and Environment. 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 479–546. World Scientific, 1997.

John Scott and Peter J. Carrington. *The SAGE Handbook of Social Network Analysis*. SAGE, 2011. ISBN 1446250113, 9781446250112.

D. Shah and T. Zaman. Rumors in a network: Who's the culprit? *Information Theory, IEEE Transactions on*, 57(8):5163–5181, Aug 2011. ISSN 0018-9448. doi:10.1109/TIT.2011.2158885.

Paul Shannon, Andrew Markiel, Owen Ozier, Nitin Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: A software environment for integrated models of biomolecular interaction networks. 13:2498–504, 12 2003.

Daniel Sheldon, Bistra N. Dilkina, Adam N. Elmachtoub, Ryan Finseth, Ashish Sabharwal, Jon Conrad, Carla P. Gomes, David B. Shmoys, William Allen, Ole Amundsen, and William Vaughan. Maximizing the spread of cascades using network design. *CoRR*, abs/1203.3514, 2012. URL http://arxiv.org/abs/1203.3514.

Lei Shi, Nan Cao, Shixia Liu, Weihong Qian, Li Tan, Guodong Wang, Jimeng Sun, and Ching-Yung Lin. Himap: Adaptive visualization of large-scale online social networks. In *IEEE Pacific Visualization Symposium*, pages 41–48, 2009. doi:10.1109/PACIFICVIS.2009.4906836.

B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343, 1996. doi:10.1109/VL.1996.545307.

Ben Shneiderman. Extreme visualization: Squeezing a billion records into a million pixels. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 3–12, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-102-6. doi:10.1145/1376616.1376618. URL http://doi.acm.org/10.1145/1376616.1376618.

N.J. Smelser. *Theory of collective behavior*. International library of sociology and social reconstruction. Free Press of Glencoe, 1962. URL https://books.google.fr/books?id=0uo9AAAAYAAJ.

K.L. Smith, S. Moriarty, K. Kenney, and G. Barbatsis. *Handbook of Visual Communication: Theory, Methods, and Media*. Routledge Communication Series. Taylor & Francis, 2004. ISBN 9781135636524. URL https://books.google.fr/books?id=ikmM_irMjKUC.

Sooel Son and Vitaly Shmatikov. *The Hitchhiker's Guide to DNS Cache Poisoning*, pages 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-16161-2. doi:10.1007/978-3-642-16161-2_27. URL https://doi.org/10.1007/978-3-642-16161-2_27.

John Stasko and Eugene Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Proceedings of the IEEE Symposium on Information Vizualization 2000*, INFOVIS '00, pages 57–, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0804-9. URL http://dl.acm.org/citation.cfm?id=857190.857683.

K. Stein, R. Wegener, and C. Schlieder. Pixel-oriented visualization of change in social networks. In *Advances in Social Networks Analysis and Mining (ASONAM), 2010 International Conference on*, pages 233–240, Aug 2010. doi:10.1109/ASONAM.2010.18.

Lucas Stich, Gerald Golla, and Alexandros Nanopoulos. Modelling the spread of negative word-of-mouth in online social networks. *Journal of Decision Systems*, 23(2):203–221, 2014. doi:10.1080/12460125.2014.886494. URL http://dx.doi.org/10.1080/12460125.2014.886494.

A. N. Strahler. Quantitative analysis of watershed geomorphology. *Transactions, American Geophysical Union*, 38:913–920, 1957. doi:10.1029/TR038i006p00913.

Gabriele Taentzer, Enrico Biermann, Dénes Bisztray, Bernd Bohnet, Iovka Boneva, Artur Boronat, Leif Geiger, Rubino Geiß, Ákos Horváth, Ole Kniemeyer, Tom Mens, Benjamin Ness, Detlef Plump, and Tamás Vajk. Generation of sierpinski triangles: A case study for graph transformation tools. In *Applications of Graph Transformations with Industrial Relevance,*

Third International Symposium, AGTIVE 2007, Kassel, Germany, October 10-12, 2007, Revised Selected and Invited Papers, pages 514–539, 2007. doi:10.1007/978-3-540-89020-1_35. URL http://dx.doi.org/10.1007/978-3-540-89020-1_35.

Shaojie Tang, Jing Yuan, Xufei Mao, Xiang-Yang Li, Wei Chen, and Guojun Dai. Relationship classification in large scale online social networks and its impact on information propagation. In *INFOCOM, 2011 Proceedings IEEE*, pages 2291–2299, April 2011. doi:10.1109/INFCOM.2011.5935046.

Edward R. Tufte. *Envisioning Information*. Envisioning Information. Graphics Press, 1990. URL https://books.google.fr/books?id=r21HAAAAMAAJ.

Barbara Tversky, Julie Bauer Morrison, and Mireille Betrancourt. Animation: Can it facilitate? *International Journal of Human-Computer Studies*, 57(4):247–262, October 2002. ISSN 1071-5819. doi:10.1006/ijhc.2002.1017. URL http://dx.doi.org/10.1006/ijhc.2002.1017.

J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, January 1976. ISSN 0004-5411. doi:10.1145/321921.321925. URL http://doi.acm.org/10.1145/321921.321925.

Julian R. Ullmann. Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. *J. Exp. Algorithmics*, 15:1.6:1.1–1.6:1.64, February 2011. ISSN 1084-6654. doi:10.1145/1671970.1921702. URL http://doi.acm.org/10.1145/1671970.1921702.

Jason Vallet, Hélène Kirchner, Bruno Pinaud, and Guy Melançon. A visual analytics approach to compare propagation models in social networks. In Arend Rensink and Eduardo Zambon, editors, *Proceedings Graphs as Models, London, UK, 11-12 April 2015*, volume 181 of *Electronic Proceedings in Theoretical Computer Science*, pages 65–79. Open Publishing Association, 2015a. doi:10.4204/EPTCS.181.5.

Jason Vallet, Bruno Pinaud, and Guy Melançon. Une approche de visualisation analytique pour comparer les modèles de propagation dans les réseaux sociaux. In Thomas Tamisier Jérôme Darmont, Benoît Otjacques, editor, *Extraction et Gestion de Connaissances (EGC 2015)*, volume RNTI-E-28, pages 365–376, Luxembourg, Luxembourg, January 2015b. URL https://hal.archives-ouvertes.fr/hal-01112592. Prix du meilleur article académique (http://www.egc.asso.fr/Manifestations_dEGC/5-FR-Prix_EGC).

Jason Vallet, Guy Melançon, and Bruno Pinaud. JASPER: Just A new Space-filling and Pixel-oriented layout for large graph ovERview. In *Conference on Visualization and Data Analysis (VDA 2016)*, volume 2016 of *Electronic Imaging*, pages 1–10, San-Francisco, CA, United States, February 2016. URL https://hal.archives-ouvertes.fr/hal-01302430.

Frank van Ham and Jarke J. van Wijk. Beamtrees: Compact visualization of large hierarchies. *Information Visualization*, 2(1):31–39, 2003. doi:10.1057/palgrave.ivs.9500036. URL http://dx.doi.org/10.1057/palgrave.ivs.9500036.

Frank van Ham, Hans-Jörg Schulz, and Joan M. Dimicco. *Honeycomb: Visual Analysis of Large Scale Social Networks*, pages 429–442. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-03658-3. doi:10.1007/978-3-642-03658-3_47. URL http://dx.doi.org/10.1007/978-3-642-03658-3_47.

Jarke J. Van Wijk and Huub van de Wetering. Cushion treemaps: Visualization of hierarchical information. In *Proceedings of the 1999 IEEE Symposium on Information Visualization*,

INFOVIS '99, pages 73–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0431-0. URL http://dl.acm.org/citation.cfm?id=857189.857663.

Dániel Varró and András Balogh. The model transformation language of the VIATRA2 framework. *Sci. Comput. Program.*, 68(3):214–234, 2007. ISSN 0167-6423. doi:10.1016/j.scico.2007.05.004.

Luca Vismara, Giuseppe Di Battista, Ashim Garg, Giuseppe Liotta, Roberto Tamassia, and Francesco Vargiu. Experimental studies on graph drawing algorithms. *Software: Practice and Experience*, 30(11):1235–1284, 2000. ISSN 1097-024X. doi:10.1002/1097-024X(200009)30:11<1235::AID-SPE339>3.0.CO;2-B. URL http://dx.doi.org/10.1002/1097-024X(200009)30:11<1235::AID-SPE339>3.0.CO;2-B.

Tatiana von Landesberger, Arjan Kuijper, Tobias Schreck, Jörn Kohlhammer, Jarke J. van Wijk, Jean-Daniel Fekete, and Dieter W. Fellner. Visual analysis of large graphs: State-of-the-art and future research challenges. *Computer Graphics Forum*, 30(6):1719–1749, 2011. ISSN 1467-8659. doi:10.1111/j.1467-8659.2011.01898.x. URL http://dx.doi.org/10.1111/j.1467-8659.2011.01898.x.

Christopher P Wadsworth. *Semantics and Pragmatics of the Lambda Calculus*. PhD thesis, 1971.

B. Wang, Y. Sun, C. Tang, and Y. Liu. A visualization toolkit for online social network propagation and influence analysis with content features. In *2014 International Conference on Orange Technologies*, pages 129–132, Sept 2014. doi:10.1109/ICOT.2014.6956616.

Chi Wang, Wei Chen, and Yajun Wang. Scalable influence maximization for independent cascade model in large-scale social networks. *Data Mining and Knowledge Discovery*, 25(3):545–576, 2012. ISSN 1384-5810. doi:10.1007/s10618-012-0262-1. URL http://dx.doi.org/10.1007/s10618-012-0262-1.

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. ISSN 1434-6028. doi:10.1140/epjb/e2006-00389-0. URL http://dx.doi.org/10.1140/epjb/e2006-00389-0.

Matthew O. Ward, Georges Grinstein, and Daniel Keim. *Interactive Data Visualization: Foundation, Techniques and Applications*. A K Peters book. CRC Press, 2010. ISBN 9781568814735.

C. Ware. *Information Visualization: Perception for Design*. Interactive Technologies. Elsevier Science, 2012. ISBN 9780123814654. URL https://books.google.fr/books?id=UpYCSS6snnAC.

Duncan J. Watts. A simple model of global cascades on random networks. *Proceedings of the National Academy of Sciences*, 99(9):5766–5771, 2002. doi:10.1073/pnas.082090499. URL http://www.pnas.org/content/99/9/5766.abstract.

Duncan J. Watts and Steven H. Strogatz. Collective dynamics of ̀small-world ́networks. *Nature*, 393:440–442, 1998. ISSN 0028-0836. doi:10.1038/30918. URL http://dx.doi.org/10.1038/30918.

S. Wen, M. S. Haghighi, C. Chen, Y. Xiang, W. Zhou, and W. Jia. A sword with two edges: Propagation studies on both positive and negative information in online social networks. *IEEE Transactions on Computers*, 64(3):640–653, March 2015. ISSN 0018-9340. doi:10.1109/TC.2013.2295802.

Robert West, Hristo S. Paskov, Jure Leskovec, and Christopher Potts. Exploiting social network structure for person-to-person sentiment analysis. *CoRR*, abs/1409.2450, 2014. URL http://arxiv.org/abs/1409.2450.

James D. Westaby, Danielle L. Pfaff, and Nicholas Redding. Psychology and social networks: a dynamic network theory perspective. *The American psychologist*, 69(3):269–284, 2014. doi:10.1037/a0036106.

Ladd Wheeler. Toward a theory of behavioral contagion. 73:179–192, 03 1966. doi:10.1037/h0023023.

R.J. Wilson. *Four Colors Suffice: How the Map Problem was Solved*. Princeton Paperbacks. Princeton University Press, 2002. ISBN 9780691120232. URL https://books.google.fr/books?id=b11saMg_8FMC.

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 International Conference on*, pages 960–965, 2012. doi:10.1109/ICDM.2012.40.

C. Woods, G. Emberling, E. Teeter, and University of Chicago. Oriental Institute. *Visible Language: Inventions of Writing in the Ancient Middle East and Beyond*. Oriental Institute Museum publications. Oriental Institute of the University of Chicago, 2010. ISBN 9781885923769. URL https://books.google.fr/books?id=Iyi0cQAACAAJ.

Hongchao Yang, Chongjun Wang, and Junyuan Xie. Maximizing influence spread in a new propagation model. In Tianrui Li, HungSon Nguyen, Guoyin Wang, Jerzy Grzymala-Busse, Ryszard Janicki, AboulElla Hassanien, and Hong Yu, editors, *Rough Sets and Knowledge Technology*, volume 7414 of *Lecture Notes in Computer Science*, pages 292–301. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-31899-3. doi:10.1007/978-3-642-31900-6_37. URL http://dx.doi.org/10.1007/978-3-642-31900-6_37.

Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *CoRR*, abs/1205.6233, 2012. URL http://arxiv.org/abs/1205.6233.

Jin Yang, Michael I. Monine, James R. Faeder, and William S. Hlavacek. Kinetic monte carlo method for rule-based modeling of biochemical networks. *Phys. Rev. E*, 78:031910, Sep 2008. doi:10.1103/PhysRevE.78.031910. URL https://link.aps.org/doi/10.1103/PhysRevE.78.031910.

Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982. doi:10.1137/0211059. URL https://doi.org/10.1137/0211059.

H Peyton Young. The diffusion of innovations in social networks. Economics Working Paper Archive, The Johns Hopkins University,Department of Economics 437, The Johns Hopkins University,Department of Economics, May 2000. URL http://ideas.repec.org/p/jhu/papers/437.html.

W.W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977. URL http://www1.ind.ku.dk/complexLearning/zachary1977.pdf.

Jeffrey M Zacks and Barbara Tversky. Structuring information interfaces for procedural learning. *Journal of Experimental Psychology: Applied*, 9(2):88–100, 2003.

Ke-Bing Zhang, Kang Zhang, and Mehmet A. Orgun. *Grammar-Based Layout for a Visual Programming Language Generation System*, pages 106–108. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. ISBN 978-3-540-46037-4. doi:10.1007/3-540-46037-3_13. URL https://doi.org/10.1007/3-540-46037-3_13.

Changwei Zhao, Zhiyong Zhang, Hanman Li, and Shiyang Zhao. A social network information propagation model considering different types of social relationships. In Jeng-Shyang Pan, Pavel Krömer, and Václav Snášel, editors, *Genetic and Evolutionary Computing*, volume 238 of *Advances in Intelligent Systems and Computing*, pages 275–282. Springer International Publishing, 2014. ISBN 978-3-319-01795-2. doi:10.1007/978-3-319-01796-9_29. URL http://dx.doi.org/10.1007/978-3-319-01796-9_29.

Shengdong Zhao, M. J. McGuffin, and M. H. Chignell. Elastic hierarchies: combining treemaps and node-link diagrams. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, pages 57–64, Oct 2005. doi:10.1109/INFVIS.2005.1532129.

List of Figures

1.1 Example of visual representation in the context of art and writing. 1
1.2 Information visualisation at its best and worst 2
1.3 Visual representations of graphs (or networks) carrying a specific meaning. . . . 4

2.1 Some examples of graphs with different numbers of nodes and edges. 9
2.2 A few examples of different graphs with particular topologies. 10
2.3 Examples of directed graphs using directed, mixed and reciprocal edges (arrows). 11
2.4 Additional examples of directed graphs with a structural peculiarity. 12
2.5 Examples of subgraphs extracted from a simple undirected graph. 12
2.6 Example of labelled subgraph representing tagged content on Instagram 14
2.7 Example of labelled subgraph representing flying routes between the airports of Australia and New-Zealand 15
2.8 Example of labelled port graph representing protein-protein interactions (from Andrei et al. (2011a)) 17
2.9 Examples of labelled port graphs (from Danos et al. (2007) and Deeds et al. (2012)) 18
2.10 Example of a directed labelled port graph representing a partial flying map/schedule between different European cities (from Tufte (1990)) 19
2.11 Examples of port graph rewrite rules from Andrei (2008) 23
2.12 Example of rewrite rules to use for random multiple graph generation. 25
2.13 Example of conditional rewrite rules to use for simple random graph generation. 27
2.14 Example of derivation tree obtained after successive applications of Rule 2.12a . 29
2.15 Examples of port graphs generated using Rules 2.12a and 2.12b 38

3.1 Example of an inductive class of graphs $\mathscr{I} = (B, R1, R2)$ (from Kejžar et al. (2006)) 44
3.2 Example of random graph construction (from Kejžar et al. (2006)) 45
3.3 Examples of networks obtained using the construction model described in Watts and Strogatz (1998) 49
3.4 Rewrite rules used for generating a simple directed ring network 50
3.5 Different steps of completion for a regular ring network generation 51
3.6 Rewrite rule *Create Jump X* to generate a regular lattice on a simple ring 52
3.7 Different steps of completion for a regular ring lattice generation 53
3.8 Rewrite rule *Perform Rewiring 1* to rewire edges between nodes at distance 1 . . 54
3.9 Rewrite rules *Leave Unchanged X* and *Next Node* 55
3.10 Rewrite rule *Perform Rewiring X* to rewire edges between nodes at distance *X* . 56
3.11 Rewrite rule *Clean Edge* to remove any *Temporary* edge from the final network . 57
3.12 Different layouts of an achieved small-world network generation. 58
3.13 Rules used for generating and attaching nodes to the social network 63

3.14 Example of resulting graph after applying Strategy 6 64
3.15 Rules generating additional connections between individuals 65
3.16 Example of resulting graph after applying Strategy 7 67
3.17 Layouts of all possible triadic configurations 68
3.18 Rules generating additional connections based on triad configurations 69
3.19 Example of resulting graph after applying Strategy 8 71

4.1 Rules used to express the Independent Cascade model (**IC**) 85
4.2 Rules used to express the Linear Threshold model (**LT**) 90
4.3 Derivation tree and detailed representation of the graph being rewritten 93
4.4 Analysis of a specific property evolution along a branch of the derivation tree . . 94
4.5 Rules used to express the Riposte model (**RP**) 99
4.6 Rules used to express the Riposte with Linear Threshold model (**RP-LT**) 105

5.1 Representations of the Karate Club network from (Zachary, 1977) 114
5.2 Examples of hybrid visualisations combining matrix and node-link representations. 115
5.3 The nested four-level model for visualization design and evaluation proposed in Munzner (2009) 117
5.4 Example of small-multiple representation used for information visualisation . . . 118
5.5 Threats and validation for each of the four levels of the nested model 119
5.6 Examples of representations maximising the use of space to display information. 121
5.7 Examples of visualisations using pixel-oriented representations. 122
5.8 Workflow showing how operations are pipelined in JASPER 123
5.9 Intermediate resulting representations obtained on an example during Phase I . . 124
5.10 Example of a coarsening process on a simple graph divided in 10 clusters. 125
5.11 Intermediate resulting representations obtained on an example during Phase II . 127
5.12 Successive developments of the N-order space-filling curve from Morton (1966) . 127
5.13 First iterations of space division on a graph with 16 nodes 128
5.14 Nodes' ordering and placement along the space-filling curve. 129
5.15 Application of JASPER on a graph generated with Wang et al. (2006)'s model . . 133
5.16 Application of JASPER on the DBLP graph (Leskovec et al., 2008) 134
5.17 Application of JASPER on the Youtube graph (Yang and Leskovec, 2012) 135
5.18 Application of JASPER on the LiveJournal graph (Yang and Leskovec, 2012) . . . 136
5.19 Small-multiple view of a propagation phenomenon on a medium graph. 137
5.20 Example of the four different kind of representations used during the experiment 141
5.21 Evaluation graphical user interface 143
5.22 Results of the user experimentation 145

B.1 Measures of the computation times needed to perform different number of rule applications. 194

C.1 Evolution of the Characteristic Path Length and Clustering Coefficient on two small world models 196

List of Algorithms

1 Ordered application of two rules . . . 37
2 Equi-probabilistic application of two rules . . . 37

3 Strategy: Ring creation – Form a simple ring with n nodes . . . 51
4 Strategy: Weaving the mesh – Insert "shortcut" edges between neighbours . . . 53
5 Strategy: Rewiring the edges – Rewire the edges on the regular ring lattice . . . 57
6 *Node generation*: Creating a directed acyclic graph of size N . . . 64
7 *Edge generation*: addition of $|E'|$ edges if possible. . . . 66
8 *Community generation*: creating edges to strengthen communities . . . 70

9 **IC** propagation . . . 85
10 **LT** propagation . . . 91
11 **IC** propagation using activation rounds . . . 94
12 **RP** dissemination . . . 100
13 **RP-LT** dissemination . . . 106

Appendix A

Author's publications

Book chapters

TULIP 5. David Auber, Romain Bourqui, Maylis Delest, Jonathan Dubois, Antoine Lambert, Patrick Mary, Morgan Mathiaut, Guy Melançon, Bruno Pinaud, Benjamin Renoust, and Jason Vallet.
To be published. Encyclopedia of Social Network Analysis and Mining. Reda Alhajj, and Jon Rokne (Eds.), 2nd edition, Springer-Verlag New York. ISBN: 978-1-4939-7130-5. 2018.
https://www.springer.com/gp/book/9781493971305

International conferences

Semantic social networks: a new approach to scaling digital ethnography. Alberto Cottica, Amelia Hassoun, Jason Vallet, and Guy Melançon.
Pre-print version. 4th International Conference on Internet Science (INSCI 2017), Thessaloniki, Greece. LNCS, volume XXXX, Springer International Publishing. 2017.
https://zenodo.org/record/832464#.Wct0Khdx3mF

JASPER: Just A new Space-filling and Pixel-oriented layout for large graph ovERview. Jason Vallet, Bruno Pinaud, and Guy Melançon.
Conference on Visualization and Data Analysis (VDA 2016), San-Francisco, CA, United States. IS&T Electronic Imaging, volume 1, pp.1–10, 2016.
https://hal.archives-ouvertes.fr/hal-01302430

International workshops

Labelled Graph Rewriting Meets Social Networks. Maribel Fernández, Hélène Kirchner, Bruno Pinaud, and Jason Vallet.
Workshop on Rewriting Logic and Its Applications (WRLA 2016), Eindhoven, Netherlands. LNCS, volume 9942, page 1–25, Springer International Publishing Switzerland. 2016.
https://hal.archives-ouvertes.fr/hal-01347355

A Visual Analytics Approach to Compare Propagation Models in Social Networks. Jason Vallet, Hélène Kirchner, Bruno Pinaud, and Guy Melançon.
Graphs as Models, London, United Kingdom. Electronical Proceedings in Theoretical

Computer Science (EPTCS), volume 181. 2015.
https://hal.archives-ouvertes.fr/hal-01150667

Other international events

Studying propagation dynamics in networks through rule-based modeling. Jason Vallet, Bruno Pinaud, and Guy Melançon.
Poster communication. Visual Analytics Science and Technology (IEEE VAST), Paris, France. Poster Electronic Proceedings VAST2014. 2014.
https://hal.archives-ouvertes.fr/hal-01112569

Propagation Dynamics in Social Networks Through Rule-Based Modeling Jason Vallet, Bruno Pinaud, Guy Melançon, and Hélène Kirchner.
Abstract only communication. 1st European Conference on Social Network (EUSN), Barcelona, Spain. 2014.
https://hal.archives-ouvertes.fr/hal-01073628

Domestic journals and conferences

Une approche de visualisation analytique pour comparer les modèles de propagation dans les réseaux sociaux. Jason Vallet, Bruno Pinaud, and Guy Melançon.
Extraction et Gestion de Connaissances (EGC 2015), Luxembourg, Luxembourg. Revue des Nouvelles Technologies de l'Information (RNTI), RNTI-E-28, pp.365–376. 2015.
https://hal.archives-ouvertes.fr/hal-01112592
[**Best academic paper award**].

Domestic workshops

PORGY : a Visual Analytics Platform for System Modelling and Analysis Based on Graph Rewriting. Bruno Pinaud, Oana Andrei, Maribel Fernández, Hélène Kirchner, Guy Melançon, and Jason Vallet.
Atelier Visualisation d'informations, interaction et fouille de données, Conférence EGC 2017. 2017.
https://hal.archives-ouvertes.fr/hal-01461513

Un modèle de génération de graphes "petit monde" imitant les réseaux sociaux . Jason Vallet, Bruno Pinaud, and Guy Melançon.
7ème conférence sur les Modèles et l'Analyse des Réseaux : Approches Mathématiques et Informatiques (MARAMI), Cergy-Pontoise, France. 2016.
https://hal.archives-ouvertes.fr/hal-01416524

JASPER: Visualisation orientée pixel de grands graphes. Jason Vallet, Bruno Pinaud, and Guy Melançon.
Ateliers Visualisation d'informations, Interaction, et Fouille de données (VIF 2016), Reims, France. 2016.
https://hal.archives-ouvertes.fr/hal-01302434

Modélisation par règles de propagation au sein de réseaux. Jason Vallet, Bruno Pinaud, and Guy Melançon.

Journées communes aux Groupes de Travail EGC et AFIHM – Big Data Mining and Visualization. Lille, France. 2014.
https://hal.archives-ouvertes.fr/hal-01112575

Research reports

Porgy Strategy Language: User Manual. Maribel Fernández, Hélène Kirchner, Bruno Pinaud, and Jason Vallet.
Research report. Université de Bordeaux, LaBRI; Inria Bordeaux Sud-Ouest; King's College London. 2017.
https://hal.archives-ouvertes.fr/hal-01566525

TULIP 4. David Auber, Romain Bourqui, Maylis Delest, Antoine Lambert, Patrick Mary, Guy Melançon, Bruno Pinaud, Benjamin Renoust, and Jason Vallet.
Research report. LaBRI - Laboratoire Bordelais de Recherche en Informatique. 2016.
https://hal.archives-ouvertes.fr/hal-01359308

Labelled Graph Strategic Rewriting for Social Networks. Maribel Fernández, Hélène Kirchner, Bruno Pinaud, and Jason Vallet.
Research report. Université de Bordeaux; Inria; King's College London. 2016.
https://hal.archives-ouvertes.fr/hal-01429893

Other publications

PORGY: a Visual Analytics Platform for System Modelling and Analysis Based on Graph Rewriting. Bruno Pinaud, Oana Andrei, Maribel Fernández, Hélène Kirchner, Guy Melançon, and Jason Vallet.
Tool demonstration. Extraction et Gestion de Connaissances (EGC 2017), Grenoble, France. Revue des Nouvelles Technologies de l'Information RNTI-E-33 pp.473–476. 2017.
https://hal.archives-ouvertes.fr/hal-01450630

Submissions under review

Labelled Graph Strategic Rewriting for Social Networks. Maribel Fernández, Hélène Kirchner, Bruno Pinaud, and Jason Vallet.
Currently under review. Submitted to the Journal of Logical and Algebraic Methods in Programming (JLAMP) in November 2016; major revision submitted in September 2017.

Appendix B

Porgy computation times

This supplementary material is added to the thesis to give an idea of the computation times needed by Porgy to perform graph rewriting operations. While the issue of scalability of our solution is mentioned a few times in this document, we wish to show why our solution is not yet ready to perform extended series of transformations on larger graphs.

Overall, and although Porgy has been and is still used to perform rewrite operations on a variety of models, its incursion in the territory of social networks has not been without challenges. While we know that social networks come in very different sizes and shapes, from the smallest ones (e.g., 34 individuals in Zachary (1977)) to very large ones (e.g. 64M nodes and 1G edges in one of the datasets studied in Yang and Leskovec (2012)), the popularity of online social networks has however produced expectations of large networks as soon as the expression "social network" is mentioned (e.g., Facebook which has recently reached two billion users[1]). 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, using the generative model presented in Chapter 3, Section 3.2, graphs of 100 nodes and 500 edges have been created in less than two minutes on a standard workstation. Multiple benchmarks have already been performed on our rewriting platform to identify bottlenecks and critical operations, and, while several upgrades have been deployed, the results are still far from impressive at the moment. It is nonetheless important to note that Porgy was not originally designed to address such requirements, and therefore further improvements are needed to start tackling graphs with tens of thousands of elements in a fair amount of time.

Although we leave the complete break down of Porgy's performances for future work, we show in Figure B.1 a presentation of a few different computation times. The measures are performed on graphs of different sizes to grasp how our rewriting platform is able to handle growing sets of elements. To this end, we reuse the very simple random graph generative model presented in Chapter 2, the one introducing at the time the concepts of rules and strategies, and perform several simulations. We apply successively, on an initial graph containing a single node, Rules 2.12a and 2.12b on page 25 using Strategy 1 on page 37 and measure the intermediate times after a given total number of rule applications. Five different graphs of various sizes are thus considered with respectively 50, 100, 200, 500 and 1,000 nodes. The first measure is performed once all the nodes are generated (consequently creating a tree with $|N|$ nodes and $|N|-1$ edges). Afterwards, the computation time is measured once the Strategy has created $|N|$, $2|N|$, $5|N|$ and $10|N|$ new edges respectively for each graphs. Most of the computation times given in Figure B.1

[1] https://newsroom.fb.com/news/2017/06/two-billion-people-coming-together-on-facebook/

are the average values returned after ten simulations. The only exceptions are the five longest applications whose computation times exceeded 2,000 seconds and which have only been run a single time each. As one can see, we also did not completed the simulation for $|N| = 1,000$ to reach the ten thousands rule applications as the computation time was too long. However, if the tendency is similar to the results observed for the other simulations, performing all of the 11,000 rule applications (1,000 for creating the nodes and 10,000 for adding the edges) should take between 48,000 and 65,000 seconds (respectively 13 and 18 hours).

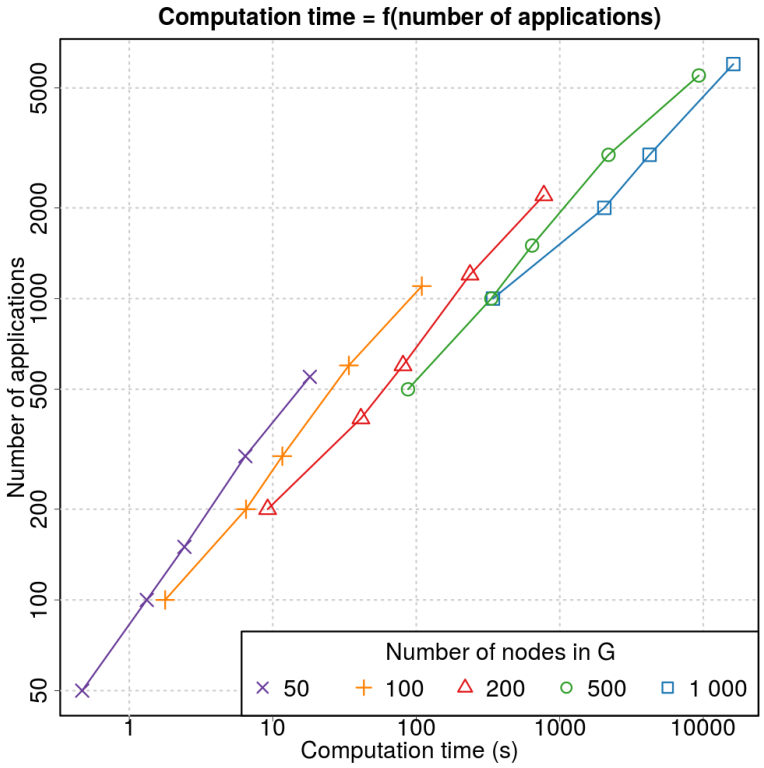


Figure B.1: Measures of the computation times needed to perform different number of rule applications.

Based on this last estimate, it is easy to understand why we cannot afford to use our graph rewriting technique to perform extended rewriting scenarios yet. Nonetheless, a few leads allowing some ameliorations performance-wise have already been identified and the implementation details are currently undergoing discussion. In any case, while we aim to widen our reach and plan to apply the graph rewriting formalism to other situations and models, these improvements are going to be essential if we want to consider working on larger graphs.

Appendix C

Small-world model analysis

In this appendix, we propose to take a closer look at the properties of the graphs created using the generative model presented in Chapter 3, Section 3.3 on page 61. As we have seen in Appendix B, PORGY is not well suited to work with larger graphs, however, an appropriate analysis cannot be simply performed on a graph with only a few hundreds elements. To address this problem, we propose instead to translate the transformations being performed in the generative model into another program, available as a plugin for TULIP, which does not present the restrictions currently existing in PORGY. We use this program to create several graphs larger than what our graph rewriting platform is currently able to handle in a satisfactory amount of time. By studying their properties with respect to the model parameters, we want to effectively validate the small-world characteristic announced for our generative model.

We first begin by precisely defining the two measures we use for our analysis: the **Characteristic Path Length** and the **Clustering coefficient**. Although these two metrics have already been presented in Chapter 2, Section 2.1 on page 8, we present here the exact formulas we use to compute the values to avoid any misconception.

Definition 50 (Characteristic Path Length (undirected graph))**.** *The* ***characteristic*** *(or* ***average*** *or* ***mean****)* ***path length*** *is defined as the average number of edges in the shortest path between any two nodes. In a graph $G = (N, E, \mathscr{E})$, the characteristic path length L is computed as follows:*

$$L = \frac{1}{|N| \times (|N| - 1)/2} \sum_{\forall a,b \in N} d_{a,b}$$

where $d_{a,b}$ is the distance between a to b (i.e., the minimal number of edges needed to link a to b).

Definition 51 (Clustering coefficient (undirected graph))**.** *For a given graph $G = (N, E, \mathscr{E})$, a node $n \in N$ has k_n neighbours, and, consequently, a maximum of $\frac{k_n(k_n-1)}{2}$ edges can exist between these nodes; this case only happens when all of n's neighbours, noted $N_G(n)$, are connected with every other neighbours. We call C_n the existing fraction of these edges between the k_n neighbours of n,*

$$C_n = \frac{1}{k_n(k_n - 1)/2} \sum_{\forall a,b \in N_G(n)} \Delta(a, b)$$

$$\text{with } \Delta(a, b) = \begin{cases} 1 & \text{if there is an edge } e \in E \text{ such that } \mathscr{E}(e) = (a, b) \\ 0 & \text{otherwise} \end{cases}$$

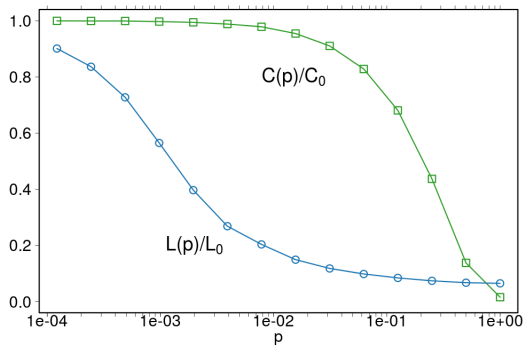
and we define the ***clustering coefficient*** C *as the average value of all the* C_n *computed for each node of the graph:*

$$C = \frac{1}{|N|} \sum_{\forall n \in N} C_n$$

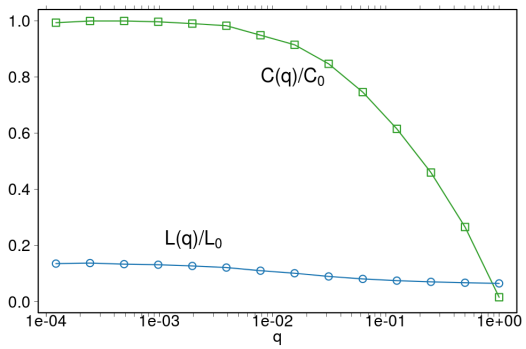
In social networks, these statistics have rather intuitive meanings. The characteristic path length L can thus be seen as the average number of connections in the shortest chain linking two people. As for C_n, its value reflects the extent to which neighbours of n are also connected to each other, while the clustering coefficient C measures how interconnected the average group of neighbours are.

As one can see, the definitions are given for undirected graphs whereas our generative model uses directed connections to indicate unilateral acknowledgement between nodes. This change is introduced here to easily compare our generated graphs to the ones initially proposed by Watts and Strogatz (1998). This small difference does not change much of our model, the only adjustment being that edges no longer have a direction. This still implies a few modifications should our model be adapted to cope with this specification. Thus, while the rules given in Figure 3.13 on page 63 (respectively in Fig. 3.18 on page 69) become virtually identical if the orientation is ignored and can be kept as such, Rule 3.15b on page 65 however has no longer any use and its application in Strategy 7 should simply be discarded (e.g., by using the `ppick` probability). Once these small changes have been taken care of, we can now create undirected graphs.

For our measures, we put ourselves in the very same conditions than for the original small-world network analysis in Watts and Strogatz (1998). We recall that in their generative model, the authors start with a regular graph (as defined in Figure 2.2d on page 10) of the appropriate size. It then sees each of its edges being rewired with a probability p; of course, if $p = 0$, the



(a) Evolution of L and C for the original small world generator; p gives the probability for each edge to be rewired.



(b) Evolution of L and C for our generative model; q indicates the probability for an edge to be created between two arbitrary nodes instead of being placed to link nodes with a common neighbour.

Figure C.1: Evolution of the Characteristic Path Length (L) and Clustering Coefficient (C) for the small world model introduced in Watts and Strogatz (1998) and our own generative model presented in Section 3.3 on page 61. Each plotted value is the average of 20 measures performed on graphs with 1,000 nodes and 5,000 edges. For each model, we define a ratio, p or q, defined as a parameter of the methods which we adjust when computing L and C. All the values are normalised using L_0 and C_0, the characteristic path length and clustering coefficient values computed on a regular graph of the same size.

graph stays regular. Using various values for p, the authors compare the characteristic path length ($L(p)$) and the clustering coefficient ($C(p)$) returned by the graphs and normalise the measures using the values obtained for an unchanged regular graph of identical size (respectively noted L_0 and C_0 for latter use) as shown in Figure C.1a.

We proceed similarly with our own graphs. While we do not have a probability p managing the rewiring, the generative model we propose must at some point decide whether each new edge must be created between *(a)* two randomly picked nodes or *(b)* between nodes sharing a common neighbour. We define a ratio q to steer the decision above by indicating which proportion of the edges is placed using solution *(a)*; thus, when $q = 0$, all the edges are set to connect nodes of the same neighbourhood, and when $q = 1$, all the edges are created to link a pair of arbitrary nodes. We then compute the characteristic path length ($L(q)$) and the clustering coefficient ($C(q)$) for the graphs created using our generative model with various values for q and, to be able to compare our results to those of Watts and Strogatz (1998), we also normalise the values for $L(q)$ and $C(q)$ using L_0 and C_0 respectively. The results are presented in Figure C.1b.

In the end, our model is effectively able to generate networks presenting small-world characteristics. Looking at the results, we can notice how our model has the advantage of conserving a low characteristic path length ($L(q)$) with any variation in q only bringing very little changes. This is not the case for the original small-world generator as $L(p)$ grows when p gets smaller. In both models however, the clustering coefficient decreases as p or q gets closer to 1. While $C(q)$ starts decreasing earlier than $C(p)$ for comparable values of p and q, it does so more slowly and progressively whereas the decrease of $C(p)$ is more abrupt.

Overall, these differences do not make one model better than the other. Watts and Strogatz (1998) can generate either regular, small-world and connected ER networks, and all the possible topologies in-between, depending of the given parameter value p. This modularity is however offset by the fact that each of these particular graphs can only be obtained when p is within small given intervals. Our model, on the other hand, lacks the capacity to create regular networks, but this is not what we expect from it. Small-world and connected ER networks can also be generated but the interval for q's value which will create these graphs is much easier to find.