Loose Graph Simulations

Alessio Mansutti[1], Marino Miculan[1], and Marco Peressotti[2⋆]

[1] Department of Mathemathics, Computer Science and Physics, University of Udine
alessio.mansutti@lsv.fr marino.miculan@uniud.it
[2] Department of Mathemathics, Computer Science, University of Southern Denmark
peressotti@imada.sdu.dk

Abstract We introduce *loose graph simulations* (LGS), a new notion about labelled graphs which subsumes in an intuitive and natural way *subgraph isomorphism* (SGI), *regular language pattern matching* (RLPM) and *graph simulation* (GS). Being an unification of all these notions, LGS allows us to express directly also problems which are "mixed" instances of previous ones, and hence which would not fit easily in any of them. After the definition and some examples, we show that the problem of finding loose graph simulations is NP-complete, we provide formal translation of SGI, RLPM, and GS into LGSs, and we give the representation of a problem which extends both SGI and RLPM.

1 Introduction

Graph pattern matching is the problem of finding patterns satisfying a specific property, inside a given graph. This problem arises naturally in many research fields: for instance, in computer science it is used in automatic system verification, network analysis and data mining [5, 15, 25, 28]; in computational biology it is applied to protein sequencing [24]; in cheminformatics it is used to study molecular systems and predict their evolution [1, 4]; in forensic science and social network analysis to profile users and their behaviours [8].

Given a so wide range of applications, many definitions of patterns have been proposed, each aiming to highlight different properties of a graph; for instance, these properties can be specified by another graph, by a formal language, by a logical predicate, etc. This situation has led to different notions of graph pattern matching, such as *subgraph isomorphism* (SGI), *regular language pattern matching* (RLPM) and *graph simulation* (GS). Each of these notions has been studied in depth, yielding similar but different theories, algorithms and tools.

A drawback of this situation is that it is difficult to deal with matching problems which do not fit directly in any of these variants. In fact, most often we need to search for patterns that can be seen as a compositions of multiple notions of graph pattern matching. An example is when we have to find a pattern which has to satisfy multiple notions of graph pattern matching at once; due to the lack

⋆ Supported by the CRC project, grant no. DFF–4005-00304 from the Danish Council for Independent Research, and by the Open Data Framework project at the University of Southern Denmark.

of proper tools, these notions can only be checked one by one with a worsening of the performances. Another example can be found in [8, 9], where extensions of RLPM and their application in network analysis and graph databases are discussed. A mixed problem between RLPM and SGI is presented in [2].

This situation would benefit from a more general notion of graph pattern matching, able to subsume naturally the more specific ones find in literature. This general notion would be a common ground to study specific problems and their relationships, as well as to develop common techniques for them. Moreover, a more general pattern matching notion would pave the way for more general algorithms, which would deal more efficiently with "mixed" problems.

To this end, in this paper we propose a new notion about labelled graphs, called *loose graph simulation* (LGS, Section 2). The semantics of its pattern queries allow us to check properties from different classical notions of pattern matching, at once and without cumbersome encodings. LGS queries have a natural graphical representation that simplifies the understanding of their semantic; moreover, they can be composed using a sound and complete algebra (Section 3). Various notions of graph pattern matching can be naturally reduced to LGSs, as we will formally prove in Sections 4 to 6; in particular, the encoding of subgraph isomorphism allows us to prove that computing LGSs is an NP-complete problem. Moreover, "mixed" matching problems can be easily represented as LGS queries; in fact, these problems can be obtained compositionally from simpler ones by means of the query algebra, as we will show in Section 7 where we solve a simplified version of the problem in [2]. Final conclusions and directions for further work (such as a distributed algorithm for computing LGSs) are in Section 8.

2 Hosts, Guests and Loose Graph Simulations

Loose graph simulations are a generalized notion pattern matching for certain labelled graphs. As often proposed in the literature, the structures that need to be checked for properties are called *hosts*, whereas the structures that represent said properties are called *guests*.

Before we formalise LGSs, let us fix some basic notions and notation.

Definition 1. *A* host graph *is a triple* (Σ, V, E) *consisting of a finite set of symbols* Σ *(also called* alphabet*), a set* V *of nodes and a set* $E \subseteq V \times \Sigma \times V$ *of edges. For an edge* $e = (v, l, v')$ *write* $s(e)$*,* $\sigma(e)$*, and* $t(e)$ *for its* source node v*,* label l*, and* target node v'*, respectively. For a vertex* v *write* $\mathrm{in}(v)$ *and* $\mathrm{out}(v)$ *for the sets* $\{e \mid t(e) = v\}$ *and* $\{e \mid s(e) = v\}$ *of its incoming and outgoing edges.*

When clear from the context, we refer host graphs simply as graphs and denote them and their components as H and as (Σ_H, V_H, E_H) (and variations thereof).

Definition 2. *A* guest $G = (\Sigma, V, E, \mathcal{M}, \mathcal{U}, \mathcal{E}, \mathcal{C})$ *is a (host) graph* (Σ, V, E) *additionally equipped with:*

- *three sets* $\mathcal{M}, \mathcal{U}, \mathcal{E} \subseteq V$*, called respectively* must*,* unique *and* exclusive *set.*
- *a* choice *function* $\mathcal{C} : V \to \mathcal{P}(\mathcal{P}(E))$*, s.t.* $\bigcup \mathcal{C}(v) = \mathrm{out}(v)$ *for each* $v \in V$*.*

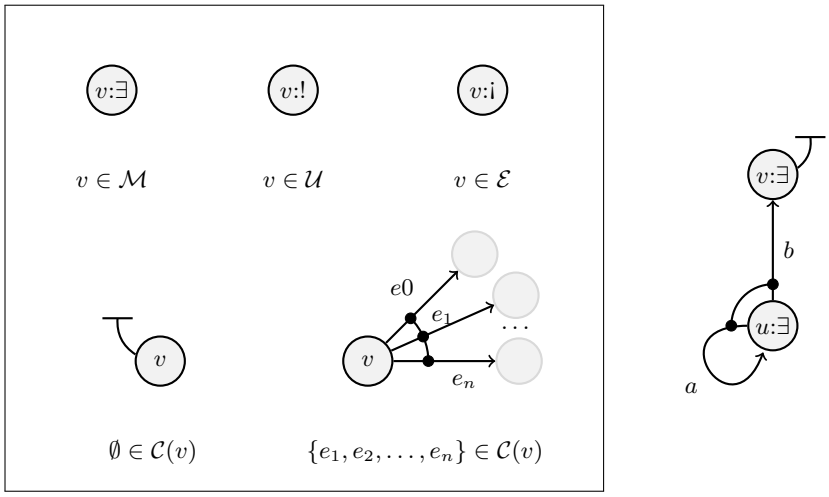

Figure 1: The guest graphic notation (left) and an example (right).

Roughly speaking, a guest is graph whose:

- nodes are decorated with usage constraints telling whether they must appear in the host, if their occurrence should be unique, and whether their occurrences can also be occurrences of other nodes or are exclusive;
- edges are grouped into possible "choices of sets of ongoing edges" for any given source node to be considered by a simulation.

The semantics of the three sets $\mathcal{M}$, $\mathcal{U}$, $\mathcal{E}$ and the choice function $\mathcal{C}$ will be presented formally in the definition of loose graph simulations (Definition 5). We adopt the convention of denoting guests as G (and variations thereof) and writing $(\Sigma_G, V_G, E_G, \mathcal{M}, \mathcal{U}, \mathcal{E}, \mathcal{C})$ for the components of the guest G.

Guests can be conveniently represented using the graphical notation shown in Figure 1 (a formal algebra is discussed in Section 3). A node belonging to the must, unique or exclusive set is decorated with the symbols $\exists$, ! and ¡, respectively. Choice sets are represented by arcs with dots places on the intersection with an edge that belong the the given choice set. The empty empty choice set ($\emptyset \in \mathcal{C}(v)$) is represented by the "corked edge" (⊤).

Example 1. Figure 1 shows the graphical representation of a guest with two nodes u and v. The must set is $\{u, v\}$, the unique and exclusive sets are both empty, and the choice function takes u to $\{\{u, v\}\}$ and v to $\{\emptyset\}$.

Akin to graph simulations (Definition 11), a loose one is a suitable subgraph of the product graph of guest and host that is coherent with the additional information prescribing node and edge usage.

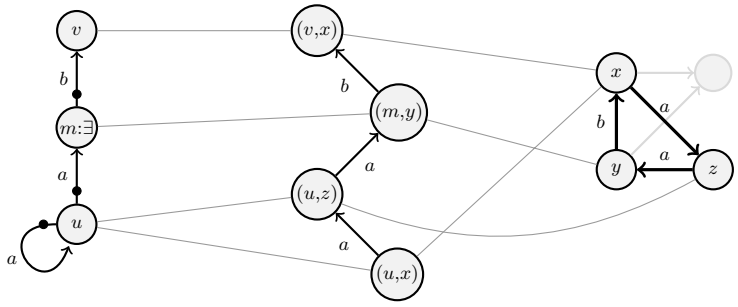

Figure 2: An LGS (center) between a guest (left) and a host (right).

Definition 3. *Let* $G_1 = (\Sigma_1, V_1, E_1)$ *and* $G_2 = (\Sigma_2, V_2, E_2)$ *be two graphs. Their* tensor product graph *is a graph defined as*

$$G_1 \times G_2 \triangleq (\Sigma_1 \cap \Sigma_2,\ V_1 \times V_2,\ E^{\times})$$
$$E^{\times} \triangleq \{((u,u'),a,(v,v')) \mid (u,a,v) \in E_1 \wedge (u',a,v') \in E_2\}$$

Definition 4. *For* $M = (\Sigma, V, E)$ *a graph over a set of labels* Σ, $\mathbb{P}_M$ *is the set of all paths in* M*:*

$$\mathbb{P}_M \triangleq \bigcup_{n\in\mathbb{N}} \{(e_0,\dots,e_n) \in E^n \mid \forall i \in \{1,\dots,n\}\ s(e_i) = t(e_{i-1})\}$$

The source ($s\colon \mathbb{P}_M \to V$)*, target* ($t\colon \mathbb{P}_M \to V$)*, and label* ($\sigma\colon \mathbb{P}_M \to \Sigma^+$) *functions are extended accordingly:*

$$s((e_0,\dots,e_n)) \triangleq s(e_0) \quad t((e_0,\dots,e_n)) \triangleq t(e_n) \quad \sigma((e_0,\dots,e_n)) \triangleq \sigma(e_0)\dots\sigma(e_n).$$

Let $v, v' \in V$*.* $\mathbb{P}_M(v,v')$ *denote the set of all paths starting from* v *and ending in* v'*,* i.e. $\mathbb{P}_M(v,v') \triangleq \{\rho \in \mathbb{P}_M \mid s(\rho) = v \wedge t(\rho) = v'\}$.

Note how paths are represented by sequences of edges, since a sequence of host's vertices can express the existence of multiple paths.

Definition 5. *A* loose graph simulation *(LGS for short) of* G *in* H *is a subgraph* $(\Sigma_G \cap \Sigma_H, V^{G\to H}, E^{G\to H})$ *of* $G \times H$ *subject to the following conditions:*

(LGS1) *vertices of* G *in the must set occur in* $V^{G\to H}$*:*

$$\forall u \in \mathcal{M}\ \exists u' \in V_H :\ (u,u') \in V^{G\to H};$$

(LGS2) *vertices in the unique set are assigned to at most one vertex of* H*:*

$$\forall u \in \mathcal{U}\ \forall u', v' \in V_H :\ (u,u') \in V^{G\to H} \wedge (u,v') \in V^{G\to H} \implies u' = v';$$

(LGS3) *vertices of H assigned to a vertex in the exclusive set cannot be assigned to other vertices:*

$$\forall u \in \mathcal{E}\ \forall v \in V_G\ \forall u' \in V_H :\ (u,u') \in V^{G\to H} \wedge (v,u') \in V^{G\to H} \implies u = v;$$

(LGS4) *for $(u,v) \in V^{G\to H}$, there is a set in $\mathcal{C}(u)$ s.t. each of its elements is related to an edge with source v and only such edges occur in $E^{G\to H}$:*

$$\begin{aligned}
&\forall (u,u') \in V^{G\to H}\ \exists \gamma \in \mathcal{C}(u)\ \forall (u,a,v) \in \gamma\ \exists v' \in V_H \\
&\qquad ((u,u'),a,(v,v')) \in E^{G\to H}, \\
&\forall ((u,u'),a,(v,v')) \in E^{G\to H}\ \exists \gamma \in \mathcal{C}(u)\ \big((u,a,v) \in \gamma \wedge \\
&\qquad \forall (u,b,w) \in \gamma\ \exists w' \in V_H\ ((u,u'),b,(w,w')) \in E^{G\to H}\big);
\end{aligned}$$

(LGS5) *the simulation preserves the connectivity w.r.t. nodes marked as must: for each $(u,u') \in V^{G\to H}$ and $v \in \mathcal{M}$ if $\mathbb{P}_G(u,v) \neq \emptyset$ then there exists $v' \in V_H$ such that $\mathbb{P}_{(\Sigma_G \cap \Sigma_H, V^{G\to H}, E^{G\to H})}((u,u'),(v,v')) \neq \emptyset$.*

The domain of all LGSs for G and H is denoted as $\mathbb{S}^{G\to H}$.

As already mentioned at the end of Definition 2, the definition of LGS attributes a semantics for the must, unique, exclusive sets and the choice function. Regarding the *unique set*, Condition LGS2 requires that every vertex of the guest in this set to be mapped by at most one element of the host. Similarly, Condition LGS3 requires the vertices of the host paired in the LGS with a node of the *exclusive set* to be only paired with that node. Condition LGS4 defines the semantics of the choice function: given a pair of vertices $(u,u') \in V^{G\to H}$, it requires to select at least one set from $\mathcal{C}(u)$. The edges of these selected sets (and only these edges, as stated by the second part of the condition) must be paired in the LGS to edges in H with source u'. This condition can be seen as a generalization of the second condition of *graph simulations* (Definition 11) that requires all outgoing edges from u to be in relation with outgoing edges of u'.

Conditions LGS1 and LGS5 define the semantics of the must set: the first condition imposes that every vertex in this set must appear in the LGS, while the second condition requires that, for each $(u,u') \in V^{G\to H}$, each vertex in the must set reachable in the guest from u is also reachable in the LGS, with a path starting from (u,u').

Example 2. Figure 2 shows a guest and its loose graph simulation over a host. In this example $\mathcal{M} = \{m\}$ and $\mathcal{U} = \mathcal{E} = \emptyset$. Moreover, the choice function is *linear*, *i.e.* $\mathcal{C} = \lambda x.\{\{e\} \mid e \in \mathrm{out}(x)\} \cup \{\emptyset \mid \mathrm{out}(x) = \emptyset\}$. LGSs of this guest represents paths $(e_0, e_1, \ldots, e_n)$ of arbitrary length in the host such that $\forall i < n\ \sigma(e_i) = a$ and $\sigma(e_n) = b$. The guest is therefore similar to the regular language $a^\star b$ and a LGS identifies paths in the host labelled with words in this language.

Proposition 1. *Let G be a guest with choice function $\mathcal{C}$ defined as $\lambda x.\{\mathrm{out}(x)\}$, let H be a host and let $S = (\Sigma_G \cap \Sigma_H, V^{G\to H}, E^{G\to H})$ be a subgraph of $G \times H$. If S satisfies Condition* LGS4 *then it also satisfies Condition* LGS5.

Proof. Under the additional hypothesis that for all $v \in V_G$ $\mathcal{C}(v) = \{\text{out}(v)\}$, if $(u, u') \in V^{G \to H}$ then Condition LGS4 requires that for all $(u, a, v) \in \text{out}(u)$ there exists v' such that $(v, v') \in V^{G \to H}$ and $((u, u'), a, (v, v')) \in E^{G \to H}$. Coinductively, since the same will hold for every of those pair (v, v'), it follows that whenever there is a path in G from u to a node $m \in \mathcal{M}$ in the must set, then there must be a path in S from (u, u') to a pair of vertices (m, w), where $w \in V_H$. Hence, Condition LGS5 is satisifed.

3 An algebra for guests

Guests are used to specify the patterns to look for inside a host; hence they should be easy to construct and to understand. To this end, besides the graphical notation described in Section 2, in this section we introduce an algebra for guests which allows us to construct them in a compositional way.

Definition 6. *Let* `emp` *be the* empty guest*. A guest with only one vertex and no edges is a* unary guest *and is denoted as*

$$p_{\mathcal{A}} \triangleq (\emptyset, \{p\}, \emptyset, \{p \mid \exists \in \mathcal{A}\}, \{p \mid ! \in \mathcal{A}\}, \{p \mid \text{¡} \in \mathcal{A}\}, \{p \to \{\emptyset \mid \varnothing \in \mathcal{A}\}\})$$

where p is the only vertex and $\mathcal{A} \subseteq \{\exists, !, \text{¡}, \varnothing\}$. For α a name, P and Q unary guests, the arrow operator *from P to Q α is defined as*

$$P \xrightarrow{\alpha} Q \triangleq (\{\alpha\}, \{p, q\}, \{(p, \alpha, q)\}, \mathcal{M}_P \cup \mathcal{M}_Q, \mathcal{U}_P \cup \mathcal{U}_Q, \mathcal{E}_P \cup \mathcal{E}_Q, \mathcal{C}^{\to})$$

$$\mathcal{C}^{\to} \triangleq \lambda x. \begin{cases} c_P \cup \{\{(p, \alpha, q)\}\} \cup c_Q & \text{if } p = q \wedge x = p \\ c_P \cup \{\{(p, \alpha, q)\}\} & \text{if } p \neq q \wedge x = p \\ c_Q & \text{if } p \neq q \wedge x = q \end{cases}$$

The empty guest, all unary guests and all guests constructed with only the arrow operator are also called elementary guests*.*

For example, a node p with only a self loop labelled α can be expressed with the term $p \xrightarrow{\alpha} p$. Besides the elementary guests, the algebra is completed by introducing two binary operators used to combine guests.

Definition 7. *Let G_1 and G_2 be two guests. Their* addition *is the guest:*

$$G_1 \oplus G_2 \triangleq (\Sigma_1 \cup \Sigma_2, V_1 \cup V_2, E_1 \cup E_2, \mathcal{M}_1 \cup \mathcal{M}_2, \mathcal{U}_1 \cup \mathcal{U}_2, \mathcal{E}_1 \cup \mathcal{E}_2, \mathcal{C}^{\oplus})$$

where the choice function $\mathcal{C}^{\oplus}$ is defined as

$$\mathcal{C}^{\oplus} \triangleq \lambda x. \begin{cases} \mathcal{C}_1(x) \cup \mathcal{C}_2(x) & \text{if } x \in V_1 \wedge x \in V_2 \\ \mathcal{C}_1(x) & \text{if } x \in V_1 \\ \mathcal{C}_2(x) & \text{if } x \in V_2 \end{cases}$$

The multiplication *of G_1 and G_2 is the guest:*

$$G_1 \otimes G_2 \triangleq (\Sigma_1 \cup \Sigma_2, V_1 \cup V_2, E_1 \cup E_2, \mathcal{M}_1 \cup \mathcal{M}_2, \mathcal{U}_1 \cup \mathcal{U}_2, \mathcal{E}_1 \cup \mathcal{E}_2, \mathcal{C}^{\otimes})$$

where the choice function $\mathcal{C}^{\otimes}$ is defined as follows

$$\mathcal{C}^{\otimes} \triangleq \lambda x. \begin{cases} \{\gamma_1 \cup \gamma_2 \mid \gamma_1 \in \mathcal{C}_1(x) \wedge \gamma_2 \in \mathcal{C}_2(x)\} & \text{if } x \in V_1 \wedge x \in V_2 \\ \mathcal{C}_1(x) & \text{if } x \in V_1 \\ \mathcal{C}_2(x) & \text{if } x \in V_2 \end{cases}$$

Notice how addition and multiplication operators differs only by the definition of the choice function for vertices of both G_1 and G_2. In the case of addition, the resulting choice function is the union of the two choice function $\mathcal{C}_1$ and $\mathcal{C}_2$, whereas for the multiplication, given a vertex $v \in V_1 \cap V_2$, every set of $\mathcal{C}^{\otimes}(v)$ is the union of a set in $\mathcal{C}_1(v)$ and one in $\mathcal{C}_2(v)$.

Proposition 2. *The set of all guests with addition or multiplication is a commutative monoid, these two operations are idempotent and the multiplication is distributive over the addition.*

As we will see in the next section, this algebra can be used to represent cleanly loose graph simulations' guests and can be used as a tool to build hybrid queries w.r.t. this notions. Furthermore, a notion of normal form can be easily defined for the syntactical terms of this algebra.

Definition 8. *A guest syntactical term is considered in* normal form *iff it is an addition of one or more subterms, where each subterm is a multiplication of elementary guests.*

Example 3. The term $q_{\{\exists,\varnothing\}} \oplus (p_{\{\exists\}} \xrightarrow{a} p \otimes p \xrightarrow{b} q)$ is in normal form and represent the guest

$$(\{a,b\}, \{p,q\}, \{(p,a,p),(p,b,q)\}, \{p,q\}, \emptyset, \emptyset, \{p \mapsto \{\{(p,a,p),(p,b,q)\}\}, q \mapsto \{\emptyset\}\})$$

also shown in Figure 1 (right).

From the definitions of the algebraic operators we obtain the following result.

Proposition 3. *For $G = (\Sigma, V, E, \mathcal{M}, \mathcal{U}, \mathcal{E}, \mathcal{C})$ a guest, its normal form is:*

$$\bigoplus_{v \in V} v_{\{\exists \mid v \in \mathcal{M}\} \cup \{! \mid v \in \mathcal{U}\} \cup \{\mathrm{i} \mid v \in \mathcal{E}\} \cup \{\varnothing \mid \emptyset \in \mathcal{C}(v)\}} \oplus \bigoplus_{\substack{v \in V \\ \gamma \in \mathcal{C}(v)}} \left(\bigotimes_{e \in \gamma} \left(s(e) \xrightarrow{\sigma(e)} t(e) \right) \right)$$

In order to simplify the exposition, we end the definition of the guests' theory by introducing their renaming. Let V and E be a set of vertices and a set of edges respectively. We define their renaming as follows:

$$V[p/q] = \begin{cases} V & \text{if } p \notin V \\ (V \setminus \{p\}) \cup \{q\} & \text{otherwise} \end{cases}$$

$$E[p/q] = \left\{ (u,a,v) \;\middle|\; \begin{array}{l} (u',a,v') \in E \\ (u' \neq p \implies u = u') \wedge (u' = p \implies u = q) \\ (v' \neq p \implies v = v') \wedge (v' = p \implies v = q) \end{array} \right\}$$

Let $G = (\Sigma, V, E, \mathcal{M}, \mathcal{U}, \mathcal{E}, \mathcal{C})$ be a guest. We define the *renaming* of $p \in V$ w.r.t. a fresh name $q \notin V$ as follows:

$$G[p/q] = (\Sigma, V[p/q], E[p/q], \mathcal{M}[p/q], \mathcal{U}[p/q], \mathcal{E}[p/q], \mathcal{C}_{p,q})$$

where

$$\mathcal{C}_{p,q} = \lambda x. \begin{cases} \{S[p/q] \mid S \in \mathcal{C}(x)\} & \text{if } x \neq p \wedge x \neq q \\ \{S[p/q] \mid S \in \mathcal{C}(p)\} & \text{if } x = q \end{cases}$$

4 The LGS problem is NP-complete

In this section we analyse the complexity of computing LGSs by studying their emptiness problem. Without loss of generality, we will now consider only guests and hosts with the same Σ. In the following, let $G = (\Sigma_G, V_G, E_G, \mathcal{M}, \mathcal{U}, \mathcal{E}, \mathcal{C})$ and $H = (\Sigma_H, V_H, E_H)$ be a guest and a host respectively.

Definition 9. *The* emptiness problem for LGSs *between G and H consists in checking whether $\mathbb{S}^{G \to H} = \emptyset$.*

Proposition 4. *Computing LGSs, as well as their emptiness problem, is in NP.*

Proof. Let $S = (\Sigma, V^{G \to H}, E^{G \to H})$ be a subgraph of $G \times H$. We will now prove that there exists a polynomial algorithm w.r.t. the size of G and H that checks whether S satisfy all the conditions of Definition 5. The satisfiability checking of Condition LGS1 is in $\mathcal{O}(\mathcal{M} \times V^{G \to H})$ since it is sufficient for every vertex in the must set $\mathcal{M}$ to check whether a vertex of the host paired with it exists. For similar reasons, Conditions LGS2 and LGS3 can also be checked in polynomial time. Moreover, to check Condition LGS4 it is sufficient to check, for each $(u, v) \in V^{G \to H}$, whether there is $\gamma \in \mathcal{C}(v)$ s.t. $\gamma \subseteq \pi_1 \circ \mathrm{out}((u, v))$ and if for all $u' \in \pi_1 \circ \mathrm{out}((u, v))$ there exists $\gamma \in \mathcal{C}(v)$ s.t. $u' \in \gamma \subseteq \pi_1 \circ \mathrm{out}((u, v))$. This can be done by a naive algorithm in $\mathcal{O}(V_H \times E_G \times (V_G \times E_H + \mathcal{C} \times E_G^2))$. Lastly, checking whether S satisfies Condition LGS5 requires the evaluation of the reachability relation of G and S and therefore can be computed in $\mathcal{O}(V_G^3 \times V_H^3)$ using the Floyd-Warshall Algorithm [11]. Since every condition can be checked in polynomial time we can conclude that the LGS problem is in NP. □

4.1 NP-hardness: Subgraph Isomorphisms via LGSs

We will now show the NP-hardness of the emptiness problem for LGSs by reducing the emptiness problem for subgraph isomorphism to it. The subgraph isomorphism problem requires to check whether a subgraph of a graph (host) and isomorphic to a second graph (query) exists. Application of this problem can be found in network analysis [15], bioinformatics and chemoinformatics [1, 4].

Definition 10. *Let $H = (\Sigma, V_H, E_H)$ and $Q = (\Sigma, V_Q, E_Q)$ be two graphs called* host *and* query *respectively. There exists a subgraph of H isomorphic to Q whenever there exists a pair of injections $\phi : V_Q \hookrightarrow V_H$ and $\eta : E_Q \hookrightarrow E_H$ s.t. for each edge $e \in E_Q$*

$$\sigma(e) = \sigma \circ \eta(e) \qquad \phi \circ s(e) = s \circ \eta(e) \qquad \phi \circ t(e) = t \circ \eta(e)$$

The subgraph isomorphism problem, as well as the emptiness problem associated to it, is shown to be NP-complete by Cook [6]. Its complexity and its importance makes it one of the most studied problem and multiple algorithmic solution where derived for it [4, 7, 27]. We will now show that the emptiness problem for subgraph isomorphism can be solved using LGSs.

Proposition 5. *Let $H = (\Sigma, V_H, E_H)$ and $Q = (\Sigma, V_Q, E_Q)$ be a host and a query for subgraph isomorphism respectively. Moreover, let*

$$G = \bigoplus_{v \in V_Q} v_{\{\exists!\mathrm{i}\} \cup \{\varnothing \mid \mathrm{out}(v) = \emptyset\}} \oplus \left(\bigotimes_{e \in E_Q} \left(s(e) \xrightarrow{\sigma(e)} t(e) \right) \right)$$

There exists a subgraph of H isomorphic to Q iff there exists a LGS of G in H, i.e. $\mathbb{S}^{G \to H} \neq \emptyset$.

Proof. From the definition of G, its must, unique and exclusive sets, as well as its choice function, are $\mathcal{M} = \mathcal{U} = \mathcal{E} = V_Q$ and $\mathcal{C} = \lambda x.\{\mathrm{out}(x)\}$ respectively. Suppose $\phi : V_Q \hookrightarrow V_H$ and $\eta : E_Q \hookrightarrow E_H$ be two injections as in Definition 10. Then the graph $S = (\Sigma, V^{G \to H}, E^{G \to H})$ where $V^{G \to H} \triangleq \{(u, u') \mid u' = \phi(u)\}$ and $E^{G \to H} \triangleq \{((u, u'), a, (v, v')) \mid (u', a, v') = \eta((u, a, v))\}$ form a LGS for G. Indeed, it satisfy Conditions LGS1 to LGS3, since ϕ is an injection. Moreover, since $\eta : E_Q \hookrightarrow E_H$ is also an injection and for each edge $e \in E_Q$ it holds that $\sigma(e) = \sigma \circ \eta(e)$, $\phi \circ s(e) = s \circ \eta(e)$ and $\phi \circ t(e) = t \circ \eta(e)$, S must be such that for each $(u, u') \in V^{G \to H}$ and for each $(u, a, v) \in \mathrm{out}(u)$ there exists v' such that $(v, v') \in V^{G \to H}$ and $((u, u'), a, (v, v')) \in E^{G \to H}$. It follows that S is a subgraph of $G \times H$ and Condition LGS4 is satisfied, since $\mathcal{C}(u) = \{\mathrm{out}(u)\}$. Moreover the satisfaction of Condition LGS5 follows from Proposition 1. S is therefore a LGS of G in H. Suppose that there is a LGS $S = (\Sigma, V^{G \to H}, E^{G \to H})$. Let ϕ s.t. $\phi(u) = u' \iff (u, u') \in V^{G \to H}$ and η s.t. $\eta((u, a, v)) = (u', a, v') \iff ((u, u'), a, (v, v')) \in E^{G \to H}$. Since $\mathcal{M} = \mathcal{U} = \mathcal{E} = V_Q$ and S is a LGS, it holds that ϕ is an injection defined on the domain V_Q. Moreover η is also an injection, since $\mathcal{C} = \lambda x.\{\mathrm{out}(x)\}$ and S satisfies Condition LGS4, and together with the hypothesis that S is a subgraph of $G \times H$ it must also hold that for each edge $e \in E_Q$ $\sigma(e) = \sigma \circ \eta(e)$, $\phi \circ s(e) = s \circ \eta(e)$ and $\phi \circ t(e) = t \circ \eta(e)$. There exists therefore a subgraph of H isomorphic to Q. □

Note how the translation from subgraph isomorphism's queries to guest for LGSs defined in Proposition 5 is *structure-preserving*. Indeed, an example of this can be seen in Figure 3. This property is important since it makes defining LGSs' guests to solve the subgraph isomorphism problem as intuitive as the

Figure 3: A possible query for subgraph isomorphism (on the left) and its translation to a guest for LGSs (on the right).

respective queries for it. This is also the case for other notions commonly used in the graphs' pattern matching community. Moreover, since the translated guest are as intuitive as the original query, this property strengthens the idea of using guests and LGSs to represent and compute hybrid queries w.r.t. these notions.

From Proposition 4 and Proposition 5 follows that:

Theorem 1. *The emptiness problem for LGSs is NP-complete.*

5 Graph Simulations are Loose Graph Simulations

Graph simulations are relations between graphs used extensively by social networks companies to perform user analysis [8]. They also can be applied to bioinformatics and urban planning [10]. The *graph simulation problem* requires to check whether a portion of a graph (host) *simulates* another graph (query).

Definition 11. *Let $H = (\Sigma, V_H, E_H)$ and $Q = (\Sigma, V_Q, E_Q)$ be two graphs called* host *and* query*, respectively. There exists a* graph simulation *of Q in H iff there is a relation $\mathcal{R} \subseteq V_Q \times V_H$ such that:*

- *for each node $u \in V_Q$ there exists a node $v \in V_H$ such that $(u, v) \in \mathcal{R}$;*
- *for each pair $(u, v) \in \mathcal{R}$ and for each edge $e \in \mathrm{out}(u)$ there exists an edge $e' \in \mathrm{out}(v)$ such that $\sigma(e) = \sigma(e')$ and $(t(e), t(e')) \in \mathcal{R}$.*

Checking whether a graph simulation exists between two graphs can be done in polynomial time [3,13]. We will now show how to reduce the emptiness problem for graph simulations to the emptiness problem for LGSs.

Proposition 6. *Let $H = (\Sigma, V_H, E_H)$ and $Q = (\Sigma, V_Q, E_Q)$ be a host and a query for graph simulation respectively. Moreover, let*

$$G = \bigoplus_{v \in V_Q} v_{\{\exists\} \cup \{\varnothing \mid \mathrm{out}(v) = \emptyset\}} \oplus \bigotimes_{e \in E_Q} s(e) \xrightarrow{\sigma(e)} t(e)$$

There is a graph simulation of Q in H iff $\mathbb{S}^{G \to H} \neq \emptyset$.

Figure 4: A possible query for *graph simulation* (on the left) and its translation in a guest for loose graph simulations (on the right).

Proof. From definition of G, its must, unique and exclusive sets, as well as its choice function, are $\mathcal{M} = V_Q$, $\mathcal{U} = \mathcal{E} = \emptyset$ and $\mathcal{C} = \lambda x.\{\text{out}(x)\}$ respectively. Let $\mathcal{R}$ be a graph simulations. The graph $S = (\Sigma, V^{G\to H}, E^{G\to H})$ where $V^{G\to H} = \mathcal{R}$ and $E^{G\to H} = \{((u,u'),a,(v,v')) \mid (u,u'),(v,v') \in R, (u,a,v) \in E_Q, (u',a,v') \in E_H\}$. is a loose graph simulations for G. $\mathcal{U} = \mathcal{E} = \emptyset$ makes Conditions LGS2 and LGS3 always true, whereas the first condition of Definition 11, that requires all vertices of V_Q to appear in the first projection of $\mathcal{R}$, makes Condition LGS1 satisfied. The second condition of Definition 11 requires that, given a pair $(u,v) \in R$, every edge of $\text{out}(u)$ is associated with one edge of $\text{out}(v)$ with the same label and with targets paired in $\mathcal{R}$. Condition LGS4 is therefore satisfied. Lastly, the satisfaction of Condition LGS5 follows from Proposition 1. S id therefore a loose graph simulation of G in H. Suppose there exists a LGS $S = (\Sigma, V^{G\to H}, E^{G\to H})$. Then $V^{G\to H}$ is a graph simulation. The definition of must set $\mathcal{M} = V_Q$ ensures that each vertex of V_Q must appear in the first projection of $V^{G\to H}$: the first condition of Definition 11 is satisfied. Moreover, the definition of the choice function $\mathcal{C} = \lambda x.\{\text{out}(x)\}$ and Condition LGS4 implies that for each $(u,u') \in V^{G\to H}$ and for all $(u,a,v) \in \text{out}(u)$ there exists v' such that $((u,u'),a,(v,v')) \in E^{G\to H}$ and, since S is a subgraph of $G \times H$, $(v,v') \in V^{G\to H}$. Thus, the second condition of Definition 11 holds and $V^{G\to H}$ is a graph simulation. □

Example 4. Figure 4 shows a query for graph simulations and the equivalent guest for loose graph simulations. As already seen in Section 4.1, the translation preserve the structure of the graph.

6 Regular languages pattern matching

Regular languages defines finite sequences of characters (called *words* or *strings*) from a finite alphabet Σ [14]. Although widely used in text pattern matching, they are also used in graph pattern matching [2,20]. In this section we will restrict ourselves to ϵ-free regular languages, *i.e.* regular languages without the empty word ϵ [29]. This restriction is quite common in the pattern matching setting, since the empty word is matched by any text or graph and therefore it doesn't represent a meaningful pattern.

Definition 12. *Let Σ be an alphabet. $\emptyset$ is a* ϵ-free regular language*. For each $a \in \Sigma$, $\{a\}$ is a* ϵ-free regular language*. If A and B are* ϵ-free regular language*,*

so are the following:

$$A \cdot B \triangleq \{vw \mid v \in P \wedge w \in Q\} \qquad A \mid B \triangleq A \cup B \qquad A^+ \triangleq \bigcup_{n\in\mathbb{N}} A^{n+1}.$$

In [29] it is shown that every regular language without the *empty* string ϵ can be expressed with the operations defined for *ϵ-free regular languages*. We will now introduce the pattern matching problem for non-empty ϵ-free regular languages. In the following let $H = (\Sigma, V_H, E_H)$ and $\mathcal{L}$ be respectively a host and a ϵ-free regular language such that $\mathcal{L} \neq \emptyset$.

Definition 13. *The Emptiness problem for Regular Language Pattern Matching (RLPM) consist in checking if there is a path $\rho \in \mathbb{P}_H$ such that $\sigma(\rho) \in \mathcal{L}$.*

To solve this problem using LGSs we will use the equivalence between regular languages and non-deterministic finite automata [26].

Definition 14. *An NFA is a 5-tuple, $N = (\Sigma, Q, \Delta, q_0, F)$ consisting of an alphabet Σ, a finite set of states Q, an* initial *state q_0, a set of* accepting *(or* final*) states $F \subseteq Q$ and a* transition *function $\Delta : Q \times \Sigma \to \mathcal{P}(Q)$.*

Let $w = a_0, a_1, \ldots, a_n$ be a word in Σ^*. The NFA N accepts w if there is a sequence of states $r_0, r_1, \ldots, r_{n+1}$ in Q s.t. $r_0 = q_0$, $r_{i+1} \in \Delta(r_i, a_i)$ for $i = 0, \ldots, n$, and $r_{n+1} \in F$. With $\mathcal{L}(N)$ we denote the set of words accepted by N, i.e. *its accepted language.*

Remark 1. Any non-empty *regular language* without ϵ can be translated to a *non-deterministic finite automaton* where the initial state does not have any incoming transitions and the only accepting state does not have any outgoing transitions. Indeed, let $N = (\Sigma, Q, \Delta, q_0, F)$ be a NFA capturing a non-empty *regular language* without ϵ, then:

- the initial state q_0 is not a final state. We can therefore construct an equivalente NFA where q_0 does not have any incoming transitions from N by adding a new initial state that q_0' that mimic the outgoing transitions of q_0;
- we add a new accepting state f and add all incoming transition from a state to a final state to the incoming transitions of f. Lastly, we update the set of accepting states to be $\{f\}$. As such, f is the only accepting state and does not have any outgoint transitions.

Formally, from N we therefore build $N' = (\Sigma, Q \cup \{q_0', f'\}, \Delta', q_0', \{f\})$ where:

- for all $a \in \Sigma$, $\Delta'(q_0', a) \triangleq \Delta(q_0, a)$ and $\Delta'(f, a) = \emptyset$;
- for all $q \in Q$ and $a \in \Sigma$, $\Delta'(q, a) \triangleq \Delta(q, a) \cup \{f \mid F \cap \Delta(q, a) \neq \emptyset\}$.

It is easy to show that $\mathcal{L}(N) = \mathcal{L}(N')$.

Proposition 7. *Let $N = (Q, \Sigma, \Delta, q_0, \{f\})$ be a NFA where the initial state q_0 does not have any incoming transitions and the only final state f does not have any outgoing ones. Let $H = (\Sigma, V_H, E_H)$ be a host. Let*

$$G = q_{0\{\exists\}} \oplus f_{\{\exists,\varnothing\}} \oplus \bigoplus_{\substack{q\in Q,\ a\in\Sigma \\ q'\in\Delta(q,a)}} \left(q \xrightarrow{a} q'\right)$$

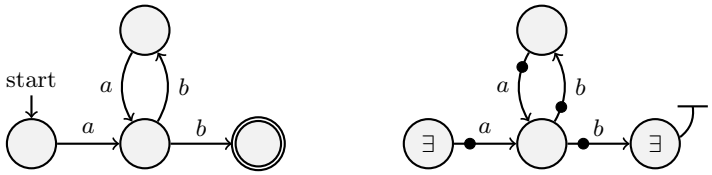

Figure 5: A possible query for *regular languages*, represented by the NFA on the left, and its translation in a guest for loose graph simulations (on the right). The language accepted by the NFA is $(ab)^+$.

There exists a path $\rho \in \mathbb{P}_H$ in H s.t. $\sigma(\rho)$ is accepted by N iff there exists a loose graph simulation of G in H, i.e. $\mathbb{S}^{G \to H} \neq \emptyset$.

Proof. If there exists $(e_0, \ldots, e_n) \in \mathbb{P}_H$ s.t. $\sigma(\rho)$ is accepted by N then from definition of acceptance condition of NFAs there must exists a sequence

$$(p_0, s(e_0)) \xrightarrow{\sigma(e_0)} (p_1, s(e_1)) \xrightarrow{\sigma(e_1)} \ldots \xrightarrow{\sigma(e_{n-1})} (p_n, s(e_n)) \xrightarrow{\sigma(e_n)} (p_{n+1}, t(e_n))$$

such that $p_0 = q_0$ and $p_{n+1} = f$; for all $i \in \{1, \ldots, n\}$ $t(e_{i-1}) = s(e_i)$; for all $i \in \{0, \ldots, n\}$ $p_{i+1} \in \Delta(p_i)$. It is easy to show that the graph S represented by this sequence is a loose graph simulation in $\mathbb{S}^{G \to H}$. Since G is constructed from N by preserving the transition relation Δ, S is a subgraph of $G \times H$. Conditions LGS1 to LGS3 trivially holds since $p_0 = q_0$, $p_n = f$ and $\mathcal{U} = \mathcal{E} = \emptyset$. From definition of $\mathcal{C}$, we have that for all $i \in \{0, \ldots, n\}$ $\{(p_i, \sigma(e_i), p_{i+1})\} \in \mathcal{C}(p_i)$ and therefore Condition LGS4 holds. Condition LGS5 is also verified since the path obtained by projecting the graph to its first component is a path from q_0 to f. Lastly, the space required by G is polynomial w.r.t. the size of N. If there exists a loose graph simulation of G in H, then Condition LGS5 ensures that there must exists a path $\rho = (e_0, \ldots, e_n)$ in it such that $\pi_1 \circ s(\rho) = q_0$ and $\pi_1 \circ t(\rho) = f$. Since a LGS in $\mathbb{S}^{G \to H}$ is a subgraph of the product $G \times H$, then by definition of E the path ρ must be coherent with Δ, *i.e.* $\forall i \in \{0, \ldots, n\}$ $\pi_1 \circ t(e_i) \in \Delta \circ \pi_1 \circ s(e_i)$. Thus, the path $\pi_2(\rho)$ obtained by projecting ρ to its second component, *i.e.* $\pi_2(\rho) = ((\pi_2 \circ s(e_0), \sigma(e_0), \pi_2 \circ t(e_0)), \ldots, (\pi_2 \circ s(e_n), \sigma(e_n), \pi_2 \circ t(e_n)))$, is such that $\sigma(\pi_2(\rho))$ is accepted by the automaton N. □

Example 5. Figure 5 shows the result of the translation of a NFA (left) accepting the regular language $(\{a\} \cdot \{b\})^+$. As already seen in the previous section, the resulting guest (right) preserve the structure of the NFA.

7 Subgraph isomorphism with regular path expressions

Many approaches found in literature define hybrid notions of similarities between graphs w.r.t. more known ones such as graph simulations, subgraph isomorphism

and RLPM [2,9]. In this section we will see how to use LGSs to solve these types or problems by studying a problem similar to the one in [2]. In this problem, called Subgraph isomorphism with regular languages (RL-SGI), queries are graphs where each edge is decorated with a regular language.

Definition 15. *A graph decorated with regular languages is a tuple* $(\Sigma, V, E, \mathcal{L})$ *consisting of an alphabet* Σ*, a set* V *of nodes, a set* $E \subseteq V \times V$ *of edges and a labelling function* $\mathcal{L} : E \to RE_\Sigma$ *decorating each edge with a non empty* ϵ*-free regular language over* Σ*.*

Definition 16. *Let* $H = (\Sigma, V_H, E_H)$ *be a host and let* $Q = (\Sigma, V_Q, E_Q, \mathcal{L})$ *be a graph decorated with regular languages. There is a subgraph of* H *isomorphic to a partial unfolding of* Q *w.r.t.* $\mathcal{L}$ *iff there is a pair of injections* $\phi : V_Q \hookrightarrow V_H$ *and* $\eta : E_Q \hookrightarrow \mathbb{P}_H$ *s.t. for each* $e \in E_Q$ $\phi \circ s(e) = s \circ \eta(e)$*,* $\phi \circ t(e) = t \circ \eta(e)$*, and* $\sigma \circ \eta(e) \in \mathcal{L}(e)$*. Vertexes of paths in* $\eta(E_Q)$ *cannot appear in* $\phi(V_Q)$ *except for their source and target,* i.e.*:* $\forall (e_0, \ldots, e_n) \in \eta(E_Q)\ \forall i \in \{1, \ldots, n\}\ s(e_i) \notin \phi(V_Q)$*.*

RL-SGI can be seen as a hybrid notion between subgraph isomorphism and RLPM. We will now show how to solve this problem with loose graph simulations by defining a proper translation from its queries to guests.

Proposition 8. *Let* $Q = (\Sigma, V_Q, E_Q, \mathcal{L})$ *be a query for RL-SGI. Let*

$$G = \bigoplus_{v \in V_Q} v_{\{\exists! i\}} \oplus \bigotimes_{e \in E_Q} G_e[q_e/s(e)][f_e/t(e)]$$

s.t. G_e *is the translation of the automaton* $N_e = (\Sigma, V_e, \delta_e, q_e, \{f_e\})$ *for* $\mathcal{L}(e)$*, as per Proposition 7 and where* q_e *and* f_e *are merged if* $s(e) = t(e)$*. For each host* $H = (\Sigma, V_H, E_H)$ *there exists a RL-SGI of* H *w.r.t.* Q *iff* $\mathbb{S}^{G \to H} \neq \emptyset$*.*

Proof (Proposition 8). By definition of G, the two following properties holds:

- V_Q is a subset of the vertices of G and $\mathcal{M} = \mathcal{U} = \mathcal{E} = V_Q$;
- let $\mathcal{C}$ the choice function of G and let $v \in V_Q$. Each set $\gamma \in \mathcal{C}(v)$ contains exactly one edge for every $e \in \mathrm{out}(v)$ of Q, that correspond to one possible transition from the initial state v in the automaton N_e.

Similarly to the proof of Proposition 5, Conditions LGS1 to LGS3 together with the first property ensures that each LGS over G correspond to an injection w.r.t the vertices of V_Q. Moreover, following the results in Proposition 7, Conditions LGS4 and LGS5 and the second property ensures that every LGS over G will contains, for each $e \in E_Q$ a path correspondent to a word in $\mathcal{L}(e)$, starting and ending with two vertices in $V_Q \times V_H$, whereas all other vertices of the path are in $(V_G \setminus V_Q) \times V_H$, where V_G is the set of all vertices of G. It follows that G can be used to verify the existence of a RL-SGI for Q, w.r.t. a host H, by checking if $\mathbb{S}^{G \to H} \neq \emptyset$. □

8 Conclusions and future work

In this paper we have introduced *loose graph simulations*, a notion of relation between graphs that can be used to check structural properties of labelled hosts. Loose graph simulations' guests can be represented using a simple graphical notation, but also compositionally by means of an algebra which is sound and complete. We have shown formally that computing LGSs is an NP-complete problem, where the NP-hardness is obtained via a trivial reduction of subgraph isomorphism to them. Moreover, we have shown that many other classical notions of graph pattern matching are naturally subsumed by LGSs. Loose graph simulations can therefore be seen as a simple common ground between multiple well-known notions of graph pattern matching and they can be used to define new hybrid fragments of these notions and develop common techniques for them.

An algorithm for computing LGSs in a decentralised fashion and inspired to the "distributed amalgamation" strategy is introduced in [16]. Roughly speaking, the host graph is distributed over processes; each process uses its partial view of the host to compute partial solutions to exchange with its peers. Distributed amalgamation guarantees all solutions are eventually found.

The same strategy is at the core of distributed algorithms for solving problems such as *bigraphical embeddings* and the distributed execution of bigraphical rewriting systems [17, 19, 22]. Bigraphs [12, 21, 23] have been proved to be quite effective for modelling, designing and prototyping distributed systems, such as *multi-agent systems* [18]. This similarity and the ability of LGS to subsume several graph problems suggests to investigate graph rewriting systems where redex occurrences are defined in terms of LGSs.

Another topic for further investigation is how to systematically minimise guests or combine sets of guests into single instances, while preserving the semantics of LGSs. A result in this direction would have a positive practical impact on applications based on LGSs.

Acknowledgements We thank the anonymous reviewers for their comments and Andrea Corradini for his insightful observations on a preliminary version of this work and for proposing the name "loose graph simulations".

References

1. J. Apostolakis, R. Körner, and J. Marialke. Embedded subgraph isomorphism and its applications in cheminformatics and metabolomics. In *1st German Conference in Chemoinformatics*, 2005.
2. P. Barceló, L. Libkin, and J. L. Reutter. Querying regular graph patterns. *ACM*, 61(1):8:1–8:54, 2014.
3. B. Bloom and R. Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Sci. Comput. Program.*, 24(3):189–220, 1995.
4. V. Bonnici, R. Giugno, A. Pulvirenti, D. E. Shasha, and A. Ferro. A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinformatics*, 14(S-7):S13, 2013.

5. D. Chakrabarti and C. Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv.*, 38:2, 2006.
6. S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158. ACM, 1971.
7. L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1367–1372, 2004.
8. W. Fan. Graph pattern matching revised for social network analysis. In *ICDT*, pages 8–21. ACM, 2012.
9. W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. Adding regular expressions to graph reachability and pattern queries. *FCS*, 6(3):313–338, 2012.
10. W. Fan, X. Wang, Y. Wu, and D. Deng. Distributed graph simulation: Impossibility and possibility. *PVLDB*, 7(12):1083–1094, 2014.
11. R. W. Floyd. Algorithm 97: Shortest path. *ACM*, 5(6):345, 1962.
12. D. Grohmann and M. Miculan. Directed bigraphs. In *Proc. MFPS*, volume 173 of *ENTCS*, pages 121–137. Elsevier, 2007.
13. M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, pages 453–462. IEEE Computer Society, 1995.
14. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley, 2003.
15. J. Lischka and H. Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *VISA*, pages 81–88. ACM, 2009.
16. A. Mansutti. Le simulazioni lasche: definizione, applicazioni e computazione distribuita. Master's thesis, University of Udine, 2016.
17. A. Mansutti, M. Miculan, and M. Peressotti. Distributed execution of bigraphical reactive systems. *ECEASST*, 71, 2014.
18. A. Mansutti, M. Miculan, and M. Peressotti. Multi-agent systems design and prototyping with bigraphical reactive systems. In K. Magoutis and P. R. Pietzuch, editors, *Proc. DAIS*, volume 8460 of *LNCS*, pages 201–208. Springer, 2014.
19. A. Mansutti, M. Miculan, and M. Peressotti. Towards distributed bigraphical reactive systems. In R. Echahed, A. Habel, and M. Mosbah, editors, *Proc. GCM*, page 45, 2014.
20. A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comput.*, 24(6):1235–1258, 1995.
21. M. Miculan and M. Peressotti. Bigraphs reloaded: a presheaf presentation. Technical Report UDMI/01/2013, Univ. of Udine, 2013.
22. M. Miculan and M. Peressotti. A CSP implementation of the bigraph embedding problem. *CoRR*, abs/1412.1042, 2014.
23. R. Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.
24. P. A. Pevzner. *Computational molecular biology - an algorithmic approach*. MIT Press, 2000.
25. G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.
26. K. Thompson. Regular expression search algorithm. *ACM*, 11(6):419–422, 1968.
27. J. R. Ullmann. An algorithm for subgraph isomorphism. *ACM*, 23(1):31–42, 1976.
28. X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724. IEEE Computer Society, 2002.
29. D. Ziadi. Regular expression for a language without empty word. *Theor. Comput. Sci.*, 163(1&2):309–315, 1996.