Canonical Forms for General Graphs Using Rooted Trees - Correctness and Complexity Study of the SCOTT Algorithm

Nicolas Bloyet, Pierre-François Marteau, Emmanuel Frénod

RESEARCH

Canonical Forms for General Graphs Using Rooted Trees - Correctness and Complexity Study of the SCOTT Algorithm

Nicolas Bloyet[1,2,3*†], Pierre-François Marteau[1] and Emmanuel Frénod[2,3]

*Correspondence: Nicolas DOT Bloyet AT see-d DOT fr
[1]IRISA, Université de Bretagne Sud, Campus de Tohannic, 56000 Vannes, France
Full list of author information is available at the end of the article
†Equal contributor

Abstract

Graph canonization (that solves also the graph isomorphism question) is an old problem that still attracts a lot of attention today, mainly because of the ubiquitous aspect of graph-based structures in computer science applications. This article present the proofs of correctness for the SCOTT algorithm, a graph canonization algorithm designed to provide a Canonical form for general graphs, namely graphs for which vertices and edges are coloured (labelled). These proofs ensure that the three canonical forms provided by SCOTT are valid, namely a canonical adjacency matrix, a canonical rooted tree (or DAG) and a canonical string. In addition, some crude lower and upper complexity bounds are presented and discussed. Finally some empirical evaluation is provided on a difficult synthetic benchmark with some comparison with the state of the art algorithms.

Keywords: graph canonization; graph isomorphism; rooted tree; SCOTT algorithm

1 Introduction

Graphs are particularly well suited for representing sets of inter-connected entities involving directed or bidirectional links that are potentially inhomogeneous (labelled or colored). Telecommunication or transportation networks, molecular structures or social relations are examples of systems that are quite well represented using graphs.

However, this data structure is difficult to handle mostly because the algorithmic complexity for classical data management operations, such as indexing and retrieval (including exact or approximate searching), sub-graph pattern mining, etc., is in general very high compared to string or tree structures.

In particular, characterizing uniquely a graph (up to an isomorphism) is challenging. This old problem, which is referred to as graph canonization, is still nowadays the object of active research in discrete mathematics and computer science. It appears, at least to our knowledge, that this problem is not natively solved by state of the art algorithms for general graphs, namely graphs for which vertices and edges are potentially labeled.

We address in this article the canonization of such general graphs. Our contribution is three folds:

- we provide the proof of validity for the SCOTT canonization algorithm that we have previously presented in [1],
- from this proof we extend the SCOTT algorithm to derive three possible canonization outputs: a string trace, a DAG tree and an adjacency matrix,

- we detail the complexity analysis for the SCOTT algorithm.

The applications are related to the speed-up of the searching operation in large graph database. For exact search (up to an isomorphism) the string trace or a hashing of this trace will be obviously quite efficient to process. For similarity or approximate search, string or tree editing distances are also more efficient than graph editing distances.

We also conjecture that canonical adjacency matrices will find application in machine learning approaches developed for graph processing, such as graph convolutional and attention networks models. By reducing the variability of the input data, the convergence of such models may be possibly improved as well as the the size of the training dataset.

In the second section of this article we formalize the canonization problem, while briefly presenting the state of the art in this domain including the SCOTT algorithm. SCOTT is built around two successive reversible constructions: i) the reversible transformation of any labelled graph into a labelled rooted tree which is the purpose of the fourth section, ii) the reversible canonization of any labeled trees (or DAG) that will be addressed in the third section. The validity proof of the SCOTT algorithm is decomposed into the validity of these two steps that are detailed into these two sections. The fifth section covers the complexity analysis for the SCOTT algorithm. The last section is decicated to concluding remarks and perspective issues.

2 Problem statement and state of the art

2.1 Graph isomorphism

A graph $G = (V_G, E_G) \in \mathbb{G}$ is a data structure composed with a set of vertices or vertices V_G that are connected through edges belonging to set E_G.

Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ two graphs: an *isomorphism* $f : V_G \to V_H$ from G to H is a bijection from the set of vertices G to the set of vertices H that preserves the edges of the two graphs.

If f exists, then G and H are isomorphic, and we note $G \simeq H$.

$$G \simeq H \iff \exists\, f, \forall\, u, v \in V_G,\ (u, v) \in E_G \Leftrightarrow (f(u), f(v)) \in E_H$$

in spite of the numerous works undertaken on the isomorphism of graphs problem [2, 3], its class of complexity remains unknown [4, 5]. Basically, if it is not yet proved that this problem is **NP-complete**, in another hand, no polynomial algorithm has yet been proposed for general graphs. Hence, this problem has its own class of complexity noted **GI** (Graph Isomorphism) [6, 7]. Recent work conjectures that this problem is quasi-polynomial [8], but once again, no definitive proof has been published yet.

2.2 Graph canonization

Computing for any graph G a canonical representative is another avenue to tackle the isomorphism problem. This representative is unique up to an isomorphic transformation, hence it represents the isomorphic equivalence class associated to G,

noted $[G]$[1]. As a result, two isomorphic graphs have a same canonization. The canonization function $Canon$ is thus a morphism from the set of graphs $\mathbb{G}$ to the quotient set $\mathbb{G}/\simeq$.

$$Canon : \mathbb{G} \to \mathbb{G}/\simeq$$
$$G \simeq H \iff Canon(G) = Canon(H)$$

The graph canonization problem (GC) is obviously at least as complex as the GI problem and in general it induces some computation overhead. However its solutions provide supplementary services which can be very useful when addressing indexing and retrieval of graphs in large datasets.

2.3 Existing graph canonization algorithms

Polynomial complexity algorithms solving the GC problem exist for a large number of restricted classes of graphs (bounded degree [9], planar graphs [10], etc.). However, for general graphs the best known algorithms implemented so far remain exponentially complex in the general case.

[11] details an exhaustive state of the art of these implementations. In addition the authors propose an evaluation benchmark for these algorithms on complex isomorphism cases built using the synthesis protocols defined in [12]. *saucy* [13, 14], *conauto* ([15]), *bliss* [16, 17] and *nauty/traces* [18, 19, 20] algorithms are thus compared in conditions that are close to the worse case situation.

Among these algorithms, only *bliss*, *nauty* and *traces* provide a canonical labeling for partially labelled graphs. Their operating principle is based on the search for a so-called *fair* coloration (cf. bibliography) of the vertices, which then allows to induce an order relation on the set of vertices using search trees (with backtracking), whose progressive pruning allows a drastic reduction of the space of possible solutions. Once such an order relation is available, it is then possible to canonically enumerate all the constituent elements of a graph in a canonical way.

Unfortunately, there is no proof yet that these approaches based on vertex colouring can provide a solution in the context of non-homogeneous edges, i.e. when the edges are also labelled (or coloured), other than by exploiting a deep rewriting of the graph. Typically, by duplicating the graph into as many edge modalities as necessary and then masking them respectively. This difficulty motivated the development of the Scott algorithmic approach for which we present validity proof in the following sections. Moreover, most of the algorithms forming the state of the art are based on a backtracking method, mentioned above. To our knowledge, none of them use graph re-writings, although this process may offer great advantages, such as the native processing of the colored edges.

[1]"is isomorphic to" relation is noted $\simeq$

2.4 The Scott algorithm

The Scott algorithm addresses the GC problem in three main steps:

Step. 1 Ordering of the vertices in levels, given a previously determined root (cf. section 3).

Step. 2 Rewriting, in a reversible way, edges that induce cycles (cf. section 4).

Step. 3 Canonical encoding of the resulting tree (cf. section 4).

Figure 1 illustrates on a simple example the three steps listed above. The following sections detail and formalize each of these steps, culminating in the final canonical form. The grammar used is described in §4.4. (Grammar refgrammar:newick)

The first two steps consists in converting any graph into a rooted tree namely a directed acyclic graph (DAG). We will show that this conversion is reversible up to an isomorhism, which means that in an isomorhism class, each graph will be associated to a unique graph representative. We prove this main result in section 4.

The third step consists in producing from the rooted tree a string trace that is unique up to an isomorphism. This second result is proved in section 3. Furthermore, the reversibility of this procedure is ensured.

3 Canonical representative for (rooted) trees

3.1 Definitions

Let Σ be a set of symbols (an alphabet). The set of strings constructed from the symbols of Σ is noted $\Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n$, with Σ^n being the set of strings of size n. We consider that the set Σ^* is ordered by the lexicographic order, which we note $(\Sigma^*, \leq)$.

For any graph $G = (V, E) \in \mathbb{G}$, let $e_G(\cdot, \cdot) : V \times V \to \mathbb{N}$ be the function that returns the value (possibly null) attached to the edge existing between two vertices i and j, which is nothing but the value A^G_{ij} where A^G is the adjacency matrix of G. Let $\mathbb{T} \subset \mathbb{G}$ be the subset of graphs such that all $t = (V, E) \in \mathbb{T}$ satisfies:

- t is acyclic and fully connected
- it exists a unique minimal length path between any two vertices in V.
- t has exactly $n - 1$ edges, with $n = |V|$

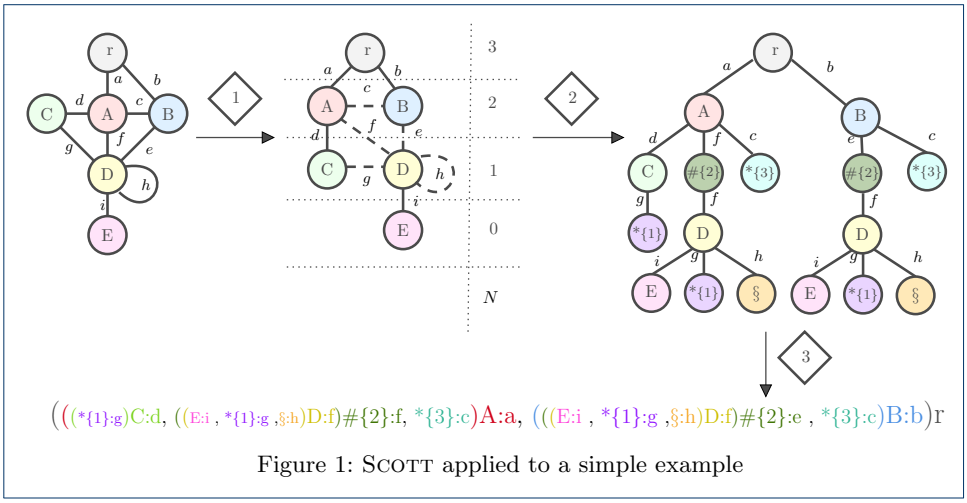


Figure 1: Scott applied to a simple example

More precisely, we are interested in planar rooted trees t_ρ in which one of the vertices, $\rho \in V$, is identified as a root vertex.

For such tree t_ρ, to each $\nu \in V$, one can affect a level N, that is related to the length of the unique path connecting it to the root vertex ρ. This length is denoted $\| \mu(\nu,\ \rho) \|$, with $\mu(\cdot,\ \cdot)$ the function evaluating the unique minimal length path existing between two vertices.

To facilitate further notations, we consider that the minimum level 0 does not corresponds to ρ but rather to its furthest leave(s). So level N stands for the difference between the maximum distance to the root vertex that is found in t_ρ and the minimum distance between the current vertex and the root vertex. Hence, the level takes values in the interval $[\ 0,\ N_G\ ]$, N_G designating the level of the root ρ, and thus the maximum level observed in G.

$$
\begin{aligned}
&lvl : V \to \mathbb{N},\\
&lvl(\nu)\ = \underset{\nu' \in V_G}{max} \| \mu(\nu',\ \rho) \| - \| \mu(\nu,\ \rho) \| = N_G - \| \mu(\nu,\ \rho) \|
\end{aligned}
$$

Let V_N be the set of vertices associated to level N, i.e. such that $\nu \in V_N \Leftrightarrow lvl(\nu) = N$. It is a remarkable property of trees that each vertex $\nu \in V_N$ is the only parent of a set of vertices of level $N-1$, which we call descendants. If we exclude all vertices of level higher than N, each vertex $\nu \in V_N$ of a tree is thus itself the root of a sub-tree, noted t_ν. Let us note T_N the set of these rooted trees associated with a root located at level N and let $\Lambda_\nu \subseteq V$ be the set of descendants of a vertex $\nu \in V$,

$$
\lambda \in \Lambda_\nu \ \Leftrightarrow \begin{cases} lvl(\lambda) = lvl(\nu) - 1 \\ e_G(\nu,\ \lambda) \neq 0 \end{cases} .
$$

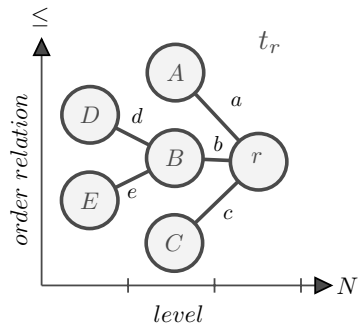
Theorem 1 (Neveu) *Given an order relation defined on the set of trees, any rooted tree is canonically encodable.*

It has been proved that a planar embedding of a rooted tree admits a non-ambiguous (canonical) (*Neveu*) notation in the form of a sequence of words, so that two trees with the same notation are equal [21].

This notation, of which we give an example in Fig. 2, is however canonical only if the planar embedding (the two-dimensional representation) is itself canonical, and if there is in fact an order relation well defined for vertices belonging to a same level N (sibling vertices). We then detail a method for systematically defining this order relation in the case of trees with labelled (heterogeneous) vertices and edges. We also propose an alternative to the Neveu's notation, thanks to a trace function σ^T defined for any tree, that limits the redundancy into the trace.

Lemma 1 *Any vertex is encodable as a string of symbols.*

Proof The vertices of a graph G can be coloured, that is to say associated to a sequence of symbols whose different modalities are designated as *labels*. This coloration is characterized as an association between each element of V_G and a value



$$Neveu(t_r) = \{\ r,\ A:a,\ B:b,\ BD:d,\ BE:e,\ C:c\ \}$$

$$\sigma^T(t_r) = (A:a\ ,\ (D:d,\ E:e)\ B:b,\ C:c\ \ )r$$

Figure 2: Encoding of a rooted tree t_r by a sequence of symbols, according to the *Neveu*'s notation (extended to labelled edges), and by the proposed trace function σ^T (cf. Eq.14) An order relation is supposed to be set on vertices.

modality in this set of colors/labels, $L_G^{V_G} : V_G \to \Sigma^*$. In this case, we reinforce the definition of isomorphism previously introduced, since, in addition to preserving the edges, it is necessary to preserve also the labels. This definition also applies to any non-colored graph, which can be considered as a particular case (for which only one single label is attached to all the edges).

$$\forall\ G = (V_G,\ E_G),\ H = (V_H,\ E_H) \in \mathbb{G}$$

$$G \simeq H \Leftrightarrow \exists\ f : V_G \to V_H\ \ , \begin{cases} \forall\ \nu_1\ , \nu_2\ \in V_G\ ,\ e_G(\nu_1\ ,\ \nu_2) = e_H(f(\nu_1),\ f(\nu_2)) \\ \forall\ \nu \in V_G\ ,\ L_G^{V_G}(\nu) = L_H^{V_H}(f(\nu)) \end{cases}$$

We note that the first condition is always satisfied when f is a permutation involving two vertices (special case of automorphism).

$$f : V_G \to V_H\ \ , \forall\ \nu_1\ ,\ \nu_2 \in V_G\ \ , \begin{cases} f(\nu_1) = \nu_2 \\ f(\nu_2) = \nu_1 \end{cases} \Rightarrow\ e_G(\nu_1\ ,\ \nu_2) = e_H(f(\nu_1),\ f(\nu_2))$$

In other words, to permute two vertices ν_1 and ν_2 two-by-two without changing the class of isomorphism, it is necessary and sufficient that $L_G^{V_G}(\nu_1) = L_G^{V_G}(\nu_2)$. Two vertices fulfilling this condition can thus be ambiguous without impacting the isomorphism class of their graph.

In the framework of a canonical notation up to an isomorphism, a vertex of any graph G is thus sufficiently defined by its label (possibly null), that one can choose to represent without loss of generality as a sequence of symbols in Σ^*, a set endowed with a lexicographical order relation.

This encoding is given by a trace function $\sigma^{V_G} : V_G \to \Sigma^*$, and allows by transitivity to endow V_G with an order relation, because $(\Sigma^*, \leq)$ is lexicographically ordered. □

Corollary 1 *Any edge can be encoded as a string of symbols.*

Proof In a similar way, any edge in a graph G can be labeled, by exploiting the relation $L_G^{E_G} : E_G \to \Sigma^*$. An isomorphism defined for this kind of graph implies a supplementary condition on edge permutations, in the sense that, on the one hand their start and end vertices must be invariant, and on the other hand, the label relative to the edge must be unchanged. Here again, if we permute edges two-by-two, it is enough to have two vertices with the same label for them to be permuted in a way that does not affect the class of isomorphism of the graph.

Within the context of a canonical notation, an edge is thus sufficiently defined by its source target vertices, themselves represented by their respective symbol strings, as well as its own label (possibly null), which can be chosen to be represented with the same alphabet Σ without loss of generality. This encoding is given by a trace function $\sigma^{E_G} : E_G \to \Sigma^*$, and allows to endow E_G with an order relation. □

3.2 The trace function σ

In the same way that we have defined trace functions allowing to canonically encode the vertices and edges of a graph, we introduce a trace function defined for any tree, without presupposing an order relation on them.

Proposition 1 *Every tree can be encoded uniquely, up to an isomorphism, as a sequence of symbols produced by a $\sigma^T : \mathbb{T} \to \Sigma^*$ function.*

$$\exists\ \sigma^T : \mathbb{T} \to \Sigma^*,\ \forall\ t_1\ ,\ t_2\ \in \mathbb{T},\ \sigma^T(t_1) = \sigma^T(t_2) \Leftrightarrow t_1 \simeq t_2 \tag{1}$$

We also define the left inverse $\tilde{\sigma}^T$ of this trace function (also called a *section* in the theory of categories), associating to the trace produced by σ^T of a tree t the unique representative of its equivalence class $[t]$. The output space of this application is therefore $\mathbb{T}$ quotiented by the isomorphism relation $\simeq$, hence the impossibility to describe it as a fully inverse function.

$$\exists\ \tilde{\sigma}^T : \Sigma^* \to \mathbb{T}/\simeq,\ \tilde{\sigma}^T \circ \sigma^T : \mathbb{T} \to \mathbb{T}/\simeq,\ \sigma^T(t) \mapsto [t] \tag{2}$$

Proposition 2 (Corollary) *A tree $t \in \mathbb{T}$, which can be a subgraph of a graph $g \in \mathbb{G}$, can be reversibly assimilated to a single vertex associated to label $\sigma^T(t)$.*

As an immediate consequence of Proposition 1, if a trace function can encode a tree as a sequence of symbols without loss of information, and as these kind of sequences can be used to label vertices, it is possible to compress any (sub)tree into a single vertex labelled with its trace, even if this tree is itself a sub-graph of any graph. We can formalize this compression process by a function κ projecting any tree on the set V containing the structures homogeneous to the vertices of V_G.

$$(1),(2) \Rightarrow \exists\ \kappa : \mathbb{T} \to V,\ \exists\ \tilde{\kappa} : V \to \mathbb{T}/\simeq,\ \tilde{\kappa}\ \circ \kappa\ \simeq id_{\mathbb{G}} \tag{3}$$

We now seek to

- prove Prop. 1 et 2
- determine an expression for the function σ^T

Lemma 2 *The t_ν subtrees of a N level are lexicographically ordered.*

Proof We formulate for any level $N \in [\ 0, N_G\ ]$ the following $\mathcal{P}_N$ proposition, in which we assume true the Propositions 1 and 2. $\mathcal{P}_N$ Let T_N be the set of trees rooted on the vertices of a N level,

$$T_N = \bigcup_{\nu\ \in\ V_G\ |\ lvl(\nu)\ =\ N} t_\nu$$

We restrict the respective definition domains of σ^T and κ to the N level only, corresponding to $\mathcal{P}_N$.

$$\exists\ \sigma_N^T :\ T_N \to \Sigma^*,\ \tilde{\sigma}_N^T \circ \sigma_N^T \simeq id_{\mathbb{G}} \tag{4}$$

$$\exists\ \kappa_N : T_N \to V,\ \exists\ \tilde{\kappa_N} : V \to T_N/\simeq,\ \tilde{\kappa_N}\ \circ \kappa_N\ \simeq id_{\mathbb{G}} \tag{5}$$

This proposed trace function endows sibling trees of the same N level with a lexicographical order relation, so that the set noted K_N of vertices bearing as labels their respective traces is itself ordered.

$$K_N = \bigcup_{t\ \in\ T_N} \kappa_N(t),\ (K_N\ ,\ \leq) \tag{6}$$

For any level N, we then suppose that we are able to order all the sibling trees of this level, by using a trace function σ_N^T that provides for each of them a canonical encoding. This proposition is illustrated in Fig.3.

$\mathcal{P}_{N+1}$ Inductive Hypothesis (IH) : we suppose that $\mathcal{P}_N$ is true. Can we deduce that $(K_{N+1}\ ,\ \leq)$ is ordered? Can we also deduce a σ^T expression from it?

Any element of K_N is thus connected to a single element of V_{N+1} exclusively by an edge $e \in E_N$. One thus describes formally and unambiguously all the trees T_{N+1} by an enumeration of all existing triplets in $K_N \times E_N \times V_{N+1}$.

$$\left.\begin{array}{r} \mathcal{P}_N \Rightarrow (K_N\ ,\ \leq) \\ (E_N\ ,\ \leq) \\ (V_{N+1}\ ,\leq) \end{array}\right\} \Rightarrow\ (K_N \times E_N \times V_{N+1}\ ,\leq) \tag{7}$$

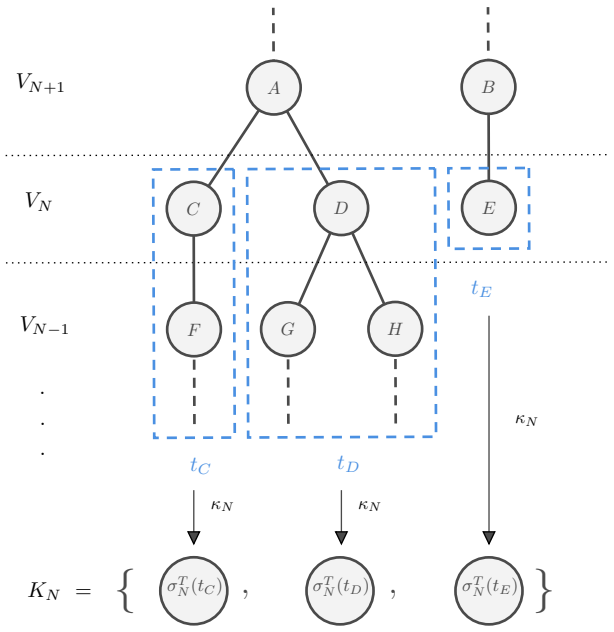


Figure 3: Proposition $\mathcal{P}_N : (K_N, \leq)$

$$\forall\, t_\nu \in T_{N+1} \;,\; \left(\bigcup_{\lambda \,\in\, \Lambda_\nu} \sigma^V(\lambda) \;\times\; \sigma^E(e(\nu\,,\lambda)) \;\times\; \sigma^V(\nu)\;,\; \leq \right) \tag{8}$$

$$\mathcal{P}_{N+1} \text{ is true} \tag{9}$$

Thus, if we are able to give a canonical encoding of a (sub)tree for a certain level N, then we can do the same for the next level $N+1$.

$\mathcal{P}_0$ Initialization: This proposition is trivial at the leaf level, because trees without descendants are directly assimilated to vertices (vertex E in Fig. 3), and can therefore be encoded as such by σ^V.

$$\Lambda_\nu = \varnothing \Leftrightarrow \sigma^T(t_\nu) \equiv \sigma^V(\nu) \tag{10}$$

$$\mathcal{P}_0 \text{ est vraie} \tag{11}$$

$$(9),\ (11) \Rightarrow \mathcal{P}_N \text{ est vraie} \tag{12}$$

□

We have just proved that if the sets V_G and E_G are ordered by an order relation, then it is possible to order any set of trees, which, together with Neveu's theorem, is sufficient to validate Prop. 1. Beyond the existence of this order relation, we finally try to characterize this trace function, according to the sets involved in the recurrence, namely K_N, E_N and V_{N+1}. This trace function σ^T would be recursive, and based on the vertex and edge encoding functions.

Lemma 3 *The trace function σ^T provides a canonical representation of any tree $t \in \mathbb{T}$.*

Proof Let Γ be the concatenation operator taking two arguments: a subset ϵ of a set $\mathscr{E}$ ordered by a total order relation $\leq$, and an arbitrary concatenation symbol $\cdot \in \Sigma$, as defined in Eq. 13.

$$\begin{aligned} &\Gamma : \mathscr{E} \times \Sigma \to \Sigma^* \\ &\forall \epsilon \subseteq \mathscr{E},\ \forall \cdot \in \Sigma,\ \Gamma(\epsilon,\ \cdot) \ = \left(\bigoplus_{e_i \ \in \ \epsilon \ \mid \ e_{i-1} \ \leq \ e_i}^{\cdot} e_i \right) = e_1 \cdot \ e_2 \ \cdot \ \ldots \ \cdot \ e_{|\epsilon|} \end{aligned} \tag{13}$$

$$\begin{aligned} &\sigma^T : \mathbb{T} \to \Sigma^* \\ &\sigma^T(t_\nu) = \Gamma \left(\bigcup_{\lambda \ \in \ \Lambda_\nu} \left\{ \sigma^T(\lambda) \ : \ \sigma^E(\lambda,\ \nu) \right\}, \cdot \right) \cdot \ \sigma^V(\nu) \end{aligned} \tag{14}$$

This trace function allows us to canonically encode all $\mathbb{T}$ trees. Indeed, two of these traces are unique if and only if all their constituent elements are rigorously equal, which means that a permutation of these sets of elements would have no impact on the isomorphism class of the considered tree, which validates σ^T as being a solution of Prop. 1 and 2. Thus, $(\mathbb{T}, \leq)$. □

We illustrate in Fig. 2 the encoding associated with the example graph by the trace function σ^T. For convenience and to stay close to the Neveu's notation, we use the comma "**,**" as the tree concatenation symbol, instead of the dot used in the above definition in Eq. 14.

4 Transforming any general graph into a rooted tree (steps 1 and 2 of the Scott algorithm)

In this section, we seek to associate to any graph $G \in \mathbb{G}$ a representative of its class of isomorphism in the form of rooted tree $t \in \mathbb{T}$, in order to be able to apply subsequently for any graph G the notation we have introduced in section 3.

Formally, we are looking for an application $f : \mathbb{G} \to \mathbb{T}$ such that f is injective on $\mathbb{G}/\simeq$. We propose a solution to this problem, which is the essence of the Scott (Structure Canonisation using Ordered Tree Translation) algorithm.

4.1 vertices ordering according to a root vertex (step 1 of the Scott algorithm)

The first step is to order all vertices g according to their minimum distance (called level) with a ρ vertex designated as root. It is assumed for the moment that this root is unambiguously known.

The objective is to to define an ordering according to which the cycles will be sequentially rewritten in a deterministic way. Indeed, by ordering the vertices by

level, the last "stage" of the graph will necessarily be connected to the vertices of the lower stage exclusively, otherwise by definition, this path would not be minimal.

The function $lvl : V \to \mathbb{N}$ introduced in the previous section, can be reused here as is, since even though the minimum path $mu(\nu, \rho)$ is not unique, its length $Vert\mu(\nu, \rho)$ is unique. Recall that this function is defined as:

$$\forall \nu \in V,\ lvl(\nu) \ = \max_{\nu' \in V} \parallel \mu(\nu',\ \rho) \parallel \ - \parallel \mu(\nu,\ \rho) \parallel$$

We illustrate the application of this ordering function on a simple graph example in Fig. 4.

4.2 Transforming a graph into a tree (step 2 of the Scott algorithm)

Given the previous ordering, let us consider the context of level N. We make the hypothesis that the lower level $N-1$ has been previously processed, i.e. each of these vertices $\nu \in V_{N-1}$ is the root of a tree. Hence, without loss of information, each $\nu \in V_{N-1}$ can be compressed into a simple vertex labeled by σ_ν, thus could be considered as a leaf in the N level context. (Prop. 2)

All the descendants of a vertex ν can therefore be summarized using a single $\sigma^T(\nu)$ labelled vertex.

The objective is then to treat each level N recursively, with the idea of editing the corresponding subgraphs in each context (level), in order to successively eliminate any occurrence of cycle and any multiple path between two vertices on the entire graph.

4.2.1 Definition of production rules

These editions of graphs can be formalized using the notion of productions [22]. Among the different existing formalisms, we will use the one based on the double push-out (DPO), in which we define a *production* $p : L \xleftarrow{l} K \xrightarrow{r} R$ as a pair of graph morphisms $l : K \to L$ and $r : K \to R$, both being injective. These three graphs L, R and K represent respectively the left part (LHS), the right part (RHS) and the interface of this production. The K interface represents the elements that are not affected by the production, while the $L - K$ represents the elements (vertices and edges) that will be removed and the $R - K$ represents the elements that will

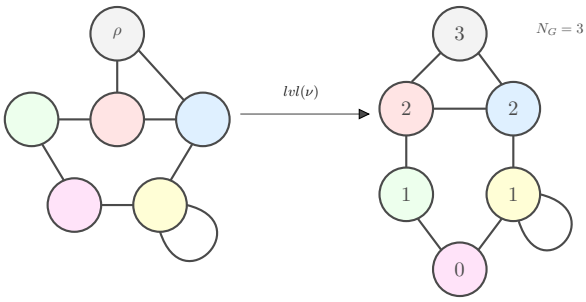


Figure 4: Ordering the vertex by the level function according to a root vertex.

be added. The implementation of one of these productions on a graph G is done by means of a bijective function called matching $m : V_G \to V_{LHS}$ associated with a system of assertions and logical predicates S_m. This matching affects to any vertex $\nu \in V_G$ the "role" (its corresponding vertex in V_{LHS}) that it will take in the application of this production as explain in the following sunbsections.

4.2.2 Matrix notation

We can reformulate this definition using a boolean logical form, more precisely by representing each of the graphs by :

- an adjacency matrix $|V_G| \times |V_G|$, $G^E_{i,j} = \begin{cases} 1 & if\ (i,\ j)\ are\ connected \\ 0 & otherwise \end{cases}$
- a matrix of vertices $|V_G| \times 1$, $G^V_i = \begin{cases} 1 & if\ i\ exists \\ 0 & otherwise \end{cases}$

The vertices matrix may seem redundant and optional at this point, but it is important to remember that vertices can be deleted, hence the need to keep track of the set of vertices.

These two matrices could be grouped together in a same tensor, but we prefer to keep them separated for clarity. It also allows to define all the following matrices for both edges and vertices. Logical algebra is used to describe the matrices l and r, that correspond respectively to deletion and addition matrices[2].

$$l = L \wedge \overline{R} = \overline{R} \wedge L \tag{15}$$
$$r = R \wedge \overline{L} = \overline{L} \wedge R \tag{16}$$

These matrices describe dynamically a production, and furthermore, they allow to express it as a deterministic and reversible matrix operations:

$$R = r \vee \left(\overline{l} \wedge L\right) \tag{17}$$
$$L = l \vee (\overline{r} \wedge R) \tag{18}$$
$$K = L \wedge R \tag{19}$$

4.2.3 Extension to labelled graphs

In the previous definition, there is no mention yet of labeled edges. As we are dealing with the general case that involves non-homogeneous (i.e. colored) edges and vertices, thus carrying labels, we have to extend this formalism.

Instead of boolean matrices only carrying an (in)existence information, we can augment into symbols-valued matrices through the use of encoding functions σ^{V_G} and σ^{E_G} respectively associated with the labelling functions $L^{V_G}_G$ and $L^{E_G}_G$ (cf. Lemma 1), to describe a fully-labelled graph :

- an adjacency matrix $|V_G| \times |V_G|$, $G^E_{i,j} = \sigma^{E_G}(e_G(i,j))$
- a matrix of vertices $|V_G| \times 1$, $G^V_i = \sigma^{V_G}(i)$

[2]each of these matrices is defined for both edges and vertices

The vertices matrix becomes here of primary importance, as it can express the vertices labelling.

Given $\mathscr{L} = \{a, b, \dots\}$ the set of labels applicable to any vertex or edge of a graph, and $\overline{\mathscr{L}} = \{\overline{a}, \overline{b}, \dots\}$ the set of these conjugated labels, let us define the following system of axioms, derived from propositional logic, where "?" designates a voluntarily indeterminate form because it will not be used afterwards:

$$\wedge : (\mathscr{L} \cup \overline{\mathscr{L}} \cup \{0,1\}) \times (\mathscr{L} \cup \overline{\mathscr{L}} \cup \{0,1\}) \rightarrow (\mathscr{L} \cup \overline{\mathscr{L}} \cup \{0,1\}), \forall\, a, b \in \mathscr{L}\ ,$$

$$\begin{aligned} a \wedge 0 &= 0 \qquad & 0 \wedge 0 &= 0 \\ a \wedge 1 &= a \qquad & 0 \wedge 1 &= 0 \\ a \wedge \overline{a} &= 0 \qquad & 1 \wedge 0 &= 0 \\ a \wedge b &= ? \qquad & 1 \wedge 1 &= 1 \end{aligned}$$

$$\vee : (\mathscr{L} \cup \overline{\mathscr{L}} \cup \{0,1\}) \times (\mathscr{L} \cup \overline{\mathscr{L}} \cup \{0,1\}) \rightarrow (\mathscr{L} \cup \overline{\mathscr{L}} \cup \{0,1\}), \forall\, a, b \in \mathscr{L}\ ,$$

$$\begin{aligned} a \vee 0 &= a \qquad & 0 \vee 0 &= 0 \\ a \vee 1 &= ? \qquad & 0 \vee 1 &= 1 \\ a \vee \overline{a} &= ? \qquad & 1 \vee 0 &= 1 \\ a \vee b &= ? \qquad & 1 \vee 1 &= 1 \end{aligned}$$

4.2.4 Cycle invariants in a rooted tree

We seek to rewrite our initial graph, now organized in levels, into an acyclic graph, more precisely a rooted tree. We are looking here for invariants characterizing cycles on such a graph, in order to define a set of productions to rewrite them.

A cycle c of size n in a graph $G = (V_G,\ E_G)$ is a path containing n distinct edges $e_c =< e_1,\ e_2,\ \dots,\ e_n >, \forall\, e \in e_c,\ e \in E_G$ for generating a sequence of vertices whose start vertex and end vertex are identical $\nu_c =< \nu_1,\ \nu_2,\ \dots,\ \nu_1 > \forall\, \nu \in \nu_c,\ \nu \in V_G$.

If G is organized in levels, then this sequence of vertices in turn generates a sequence of levels $N_c =< N_1,\ N_2\ ,\ \dots,\ N_1 >$, with $\forall\ i\ \in [\ 1,\ n-1\ ]\ ,\ N_{c,\ i+1} \in \{N_{c,\ i} + 1, N_{c,\ i}, N_{c,\ i} - 1\}$

We illustrate in Fig. 5 the generation of such sequences on elementary cycles organized in levels, according to a single root (a single vertex on the upper level).

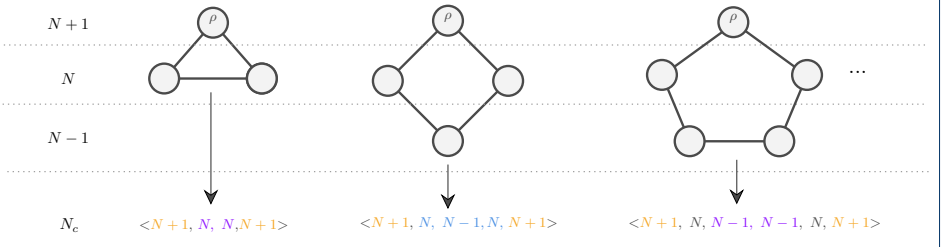


Figure 5: Organization of elementary cycles in graphs organized in levels

We observe a regularity in the sequences generated from the roots. These are all in the form $< N_\rho,\ N_\rho - 1,\ [\ N' < N_\rho - 1\ ]^*\ ,\ N_\rho - 1,\ N_\rho >$. If we continue these examples to more important cycles, we highlight that the edges making these graphs cyclic are characterized by the sequences $< N,\ N >$ and $< N,\ N-1, ..., N-1,\ N >$, in other words the edges allowing an alternative path to the one passing through the root between two vertices of the same level.

We thus rediscover graphically a property specific to rooted trees, namely the existence and uniqueness of a minimal path between any two vertices: as a consequence, if t_ρ is a rooted tree, then there is no path connecting two vertices of level N passing through vertices of level lower or equal to N. We deduce the characteristic constraints of a rooted tree. On such a tree t_ρ, any vertex ν_N belonging to the N level is connected exclusively to :

- a single parent at level $N + 1$
- any number of descendants at level $N - 1$

We can express these constraints by prohibiting the existence of the following sequences at lower levels:

- $< N,\ N >$
- $< N,\ N - 1, ..., N - 1,\ N >$

This ensures the uniqueness of a minimal path connecting any two vertices, a consequence of the absence of a cycle.

Notice that if level $N-1$ and subsequent lower levels have been process to remove the edges at the origin of cycle, then at level N, only the three left cases shown in Fig. 5 may occur.

4.2.5 Triplet of production solution

Following the discussion on cycle invariants in graphs organized in levels, we deduce that on a graph whose vertices have been ordered in levels, only three edge configurations (schematized in Fig. 6 by their minimal examples) result in the presence of cycles, which can be solved respectively by as many associated productions applied iteratively from the lower levels to the upper levels:

p_c co-bound, an edge from a vertex to a separate vertex belonging to the same N level (e.g. the c and g edges in Fig.6)

p_i in-bound, two vertices of the same N level connected to a common descendant at the $N - 1$ level (e.g. the f and e edges in Fig. 6)

p_s self-bound, an edge of a vertex towards itself (e.g. the h edge in Fig. 6)

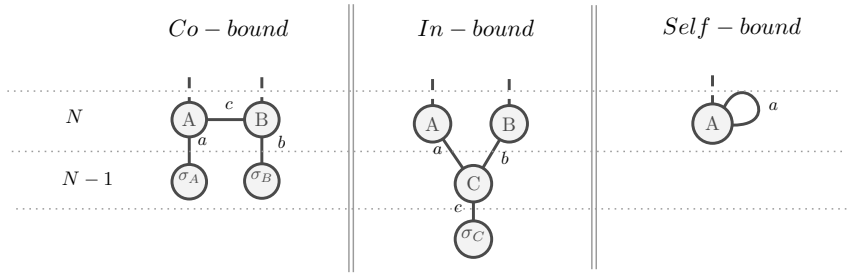


Figure 6: The different configurations resulting in cycles in a graph organized in levels.

These three productions p_c, p_i and p_s are detailed below. We give for each of them their representative matrices LHS (L^N, L^E) and RHS (R^N, R^E), and as an indication, we calculate in the first case the deletion matrices l^N and l^E, as well as the addition matrices r^N and r^E.

Notice that while the edge modalities (i.e. L^E and R^E values) are the same in LHS and RHS, some specific vertices modalities (called *magnets*) are created during the process to mark new inserted vertices as solving a cycle present in the original graph (such new vertices do not exist in the original graph). This mechanism prohibits any collision with other original graphs that may initially present a same structure, and furthermore ensures reversibility.

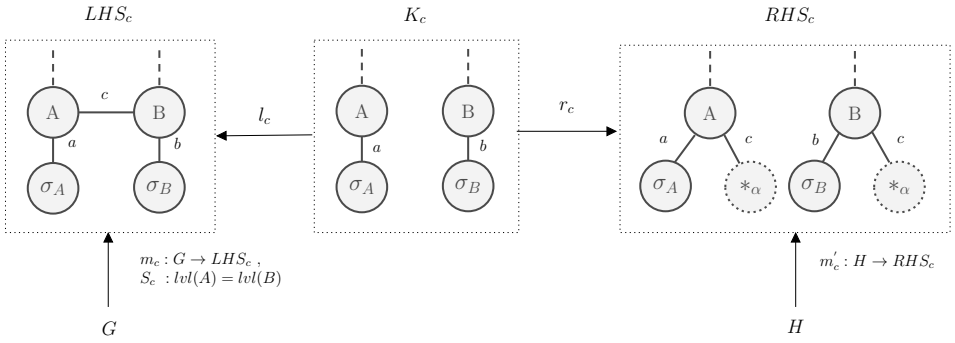
Those modalities can be fully expressed by including additional vertices labels. This way, the output graph has two independent labelling on its vertices, which can however be merged following an arbitrary pattern (cf. Grammar 1) and reserved symbols:

$*$ **(*virtual*)** vertex created by a *co-bound*
$\#$ **(*mirror*)** vertex created by an *in-bound*
$\&$ **(*self*)** vertex created by a *self-bound*

Co-bound (p_c):

In the context of a rooted tree, vertices A and B have a common ancestor, which may be ρ itself. Suppose that the paths $\mu(A,\ \rho)$ and $\mu(B, \rho)$ are direct (neglecting the intermediate vertices), i.e. we consider that the upper level is only the root vertex ρ.

Double Push-Out (DPO) Diagram associated with the "cobound" production :



Vertices and adjacency matrices of LHS_c and RHS_c :

$$L^V = \begin{bmatrix} A \\ B \\ \sigma_A \\ \sigma_B \\ 0 \\ 0 \end{bmatrix},\ L^E = \begin{bmatrix} 0 & \boldsymbol{c} & \boldsymbol{a} & 0 & 0 & 0 \\ \boldsymbol{c} & 0 & 0 & \boldsymbol{b} & 0 & 0 \\ \boldsymbol{a} & 0 & 0 & 0 & 0 & 0 \\ 0 & \boldsymbol{b} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},\quad R^V = \begin{bmatrix} A \\ B \\ \sigma_A \\ \sigma_B \\ *_\alpha \\ *_\alpha \end{bmatrix},\ R^E = \begin{bmatrix} 0 & 0 & \boldsymbol{a} & 0 & \boldsymbol{c} & 0 \\ 0 & 0 & 0 & \boldsymbol{b} & 0 & \boldsymbol{c} \\ \boldsymbol{a} & 0 & 0 & 0 & 0 & 0 \\ 0 & \boldsymbol{b} & 0 & 0 & 0 & 0 \\ \boldsymbol{c} & 0 & 0 & 0 & 0 & 0 \\ 0 & \boldsymbol{c} & 0 & 0 & 0 & 0 \end{bmatrix}$$

Deletion matrices l^V and l^E :

$$\overline{R^V} \wedge L^V \ {}^{l^V} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \overline{R^E} \wedge L^E \ {}^{l^E} = \begin{bmatrix} 0 & \boldsymbol{c} & 0 & 0 & 0 & 0 \\ \boldsymbol{c} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Addition matrices r^V and r^E :

$$\overset{r^V}{\overline{L^V} \wedge R^V} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ *_\alpha \\ *_\alpha \end{bmatrix} \qquad \overset{r^E}{\overline{L^E} \wedge R^E} = \begin{bmatrix} 0 & 0 & 0 & 0 & \boldsymbol{c} & 0 \\ 0 & 0 & 0 & 0 & 0 & \boldsymbol{c} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \boldsymbol{c} & 0 & 0 & 0 & 0 & 0 \\ 0 & \boldsymbol{c} & 0 & 0 & 0 & 0 \end{bmatrix}$$

with $*_\alpha = virtual : \sigma^T(t_A) \cdot \sigma^T(t_B)$, a vertex label made of a *prefix* "*virtual*" (here the symbol " $*$ ") stating its special status of not being in the input graph, and a *magnet* computed from an invariant of the re-wrote edge, $\sigma^T(t_A) \cdot \sigma^T(t_B)$. This magnet can be compressed into symbolic shortcuts, here α.

In LHS_c, there is therefore a *chain* (sequence of connected vertices) such as $< \rho, \ \ldots\ , A, B, \ \ldots\ , \rho >$, forming a cycle. Hence, LHS_c is not a tree, just like any graph $g \in \mathbb{G}$, satisfying at least once the matching relationship m_c.

On the other hand, RHS_c is a tree. More specifically, the application of this p_c production removes the elementary cycles present at the $lvl(A)$ level.

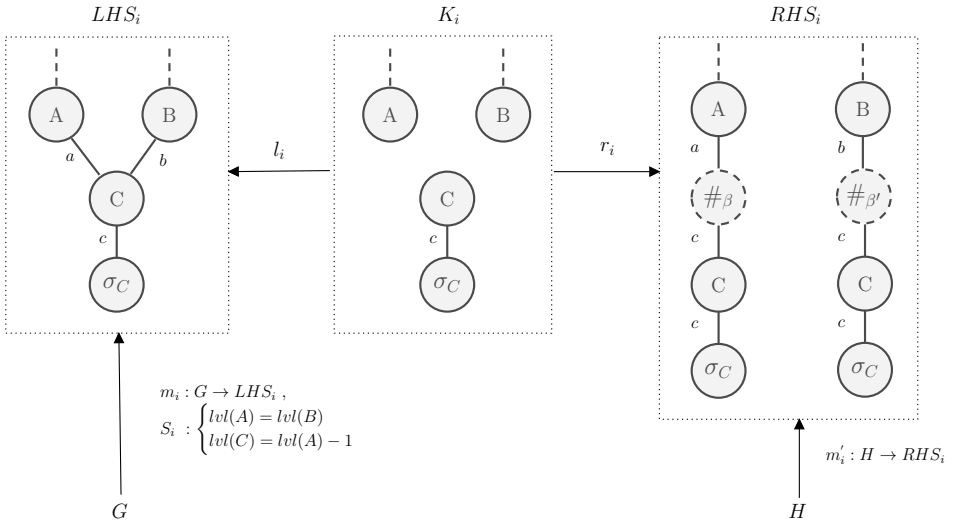
In-bound (p_i):

Here again, we are in the same cyclic graph configuration as in the previous example, because there is a string $< \rho, \ \ldots\ , A, C, B, \ \ldots\ , r >$. Furthermore, if $a = b$, there is not a single minimal path $\mu(C, \rho)$.

As in the previous production, LHS_i is not a tree, like any $g \in \mathbb{G}$ graph that satisfies the m_i matching relationship at least once.

On the other hand, RHS_i is a tree. More specifically, the application of this p_i production removes the non-elementary cycles present at the $lvl(A)$ level, and removes the non-unicity of the paths leading to ρ (and thus any other vertex).

Double Push-Out (DPO) Diagram associated with the "inbound" production :



Vertices and adjacency matrices of LHS_i :

$$L^N = \begin{bmatrix} A \\ B \\ C \\ \sigma_C \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \; L^E = \begin{bmatrix} 0 & 0 & \boldsymbol{a} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \boldsymbol{b} & 0 & 0 & 0 & 0 & 0 \\ \boldsymbol{a} & \boldsymbol{b} & 0 & \boldsymbol{c} & 0 & 0 & 0 & 0 \\ 0 & 0 & \boldsymbol{c} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Vertices and adjacency matrices of RHS_i :

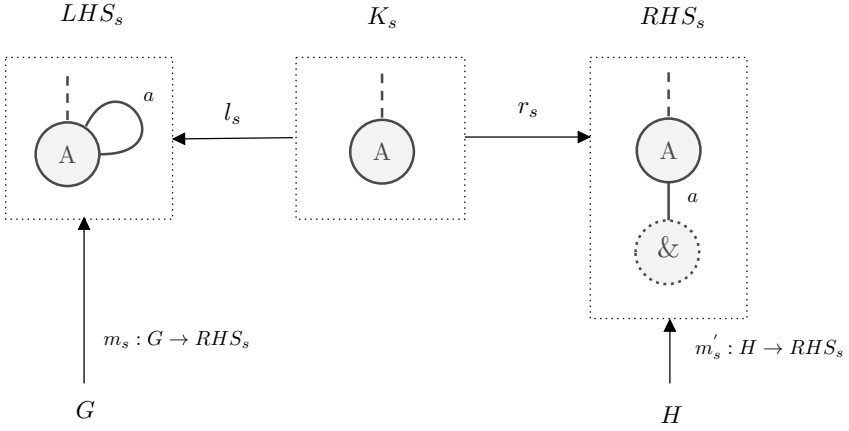
$$R^N = \begin{bmatrix} A \\ B \\ C \\ \sigma_C \\ \#_\beta \\ \#_{\beta'} \\ C \\ \sigma_C \end{bmatrix}, \; R^E = \begin{bmatrix} 0 & 0 & 0 & 0 & \boldsymbol{a} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \boldsymbol{b} & 0 & 0 \\ 0 & 0 & 0 & \boldsymbol{c} & a & 0 & 0 & 0 \\ 0 & 0 & \boldsymbol{c} & 0 & 0 & 0 & 0 & 0 \\ \boldsymbol{a} & 0 & a & 0 & 0 & 0 & 0 & 0 \\ 0 & \boldsymbol{b} & 0 & 0 & 0 & 0 & \boldsymbol{b} & 0 \\ 0 & 0 & 0 & 0 & 0 & \boldsymbol{b} & 0 & \boldsymbol{c} \\ 0 & 0 & 0 & 0 & 0 & 0 & \boldsymbol{c} & 0 \end{bmatrix}$$

with $\#_\beta = mirror : \sigma^T(t_C) \cdot \sigma^T(t_B)$, a vertex label made of a *prefix* "*mirror*" (here the symbol "#") stating its special status of not being in the input graph, and having to be distinct from nodes created by cobound productions, and again a *magnet* computed from an invariant of the re-wrote edges, $\sigma^T(t_C)$, suffixed by the "′" symbol on the lexicographic inferior branches. This magnet can be compressed into symbolic shortcuts, here β.

Self-bound (p_s) :

This configuration is trivial, and allows any graph to be rewritten as a graph without self-bound links.

Double Push-Out (DPO) Diagram associated with the "selfbound" production :



Vertices and adjacency matrices of LHS_s and RHS_s :

$$L^N = \begin{bmatrix} A \\ 0 \end{bmatrix}, \; L^E = \begin{bmatrix} \boldsymbol{a} & 0 \\ 0 & 0 \end{bmatrix}, \; R^N = \begin{bmatrix} A \\ \& \end{bmatrix}, \; R^E = \begin{bmatrix} 0 & \boldsymbol{a} \\ \boldsymbol{a} & 0 \end{bmatrix}$$

The described productions satisfy all (17) (18) and (19), so they can be described dynamically with the addition r and deletion l matrices, and applied in the context of a graph, when a matching $m : G \to LHS_m$ is available.

4.2.6 Ordering the productions

The productions we have just described allow to rewrite any kind of cycle in a graph. Their application is however carried out according to a precise ordering, by level and by category, forming a general morphism f_G such as:

$$f_G \ = f_{N_G} \circ \ \ldots \ \circ f_0 \tag{20}$$

$$\forall \ N \in [\ 0,\ N_G\ ],\ f_N = p_{i,}^N \circ \ p_c^N \circ \ p_s^N \tag{21}$$

$$\forall \ x \in \{i,\ c,\ s\},\ k = \Omega\left(\left\{p_x^N\right\}\right),\ f_x^N \ = p_{x_k}^N \circ \ldots \circ p_{x_1}^N \tag{22}$$

with $\Omega(\cdot)$ the cardinal of a set, and $\left\{p_k^N\right\}$ the set of individual productions composing the p_x^n morphism associated with level N and a production category x. These $\{p_x^N\}$ sets are constructed by determining all possible antecedents to the matching functions associated with k productions, for a given level N.

We use in further steps the following notations :

- Let $G_\rho = (V_G,\ E_G)$, be a graph in which a vertex $\rho \in V$ is identified as a root.
- We note $V_N \subseteq V_G,\ \nu \in V_N \ \Leftrightarrow \ lvl(\nu) = N$, the set of vertices in G_ρ associated to level N.
- Let $W_N \ \supseteq V_N\ ,\ W_N = \bigcup_{k=0}^N V_k\ ,\ \nu \in W_N \ \Leftrightarrow lvl(\nu) \leq N$, be the set of the vertices in a graph G associated to a level lower or equal to N.
- Let $G_{\rho,\ N} = (W_N\ ,\ E_N \in (W_N\ \times W_N)) \subseteq G_\rho$ be the sub-graph of G_ρ composed only with the vertices W_N associated to level N and the edges E_N linking exclusively pairs of vertices in W_N.
- Finally, we note $G_\rho^N = f_N(G_\rho)$ the graph that results from the processing at level N of graph G_ρ using $f_N \ \circ \ \ldots \ \circ \ f_0$ function. We lighten the syntax of a fully processed subgraph at level N, $G_{\rho,N}^N = G_{\rho,\ N}^*$.

We then construct the set of productions p_x^N by applying as many times as possible the matching function m_x associated with this category of production x in the context of $G_{\rho,N}$.

$$\left\{p_k^N\right\} = arg\left(G_{\rho,N} \rightarrow m_k : LHS_k\right) \tag{23}$$

These productions can be characterized by the trace of the trees associated with the vertices to which they apply ($\{\sigma_A\}$ for the self-bounds, $\{\sigma_A,\ \sigma_B\}$ for the co-bounds and the in-bounds) as well as the edge modalities that are involved, which induces an ordering on these productions.

$$(\mathbb{T},\ \leq) \ \Rightarrow \left(\left\{p_k^N\right\},\ \leq\right) \tag{24}$$

We illustrate in Fig.7 an example of production ordering on a simple example.

The application of these productions according to this ordering ensures that the algorithm results in a tree that depends only on the class of isomorphism of the input graph, which we prove below.

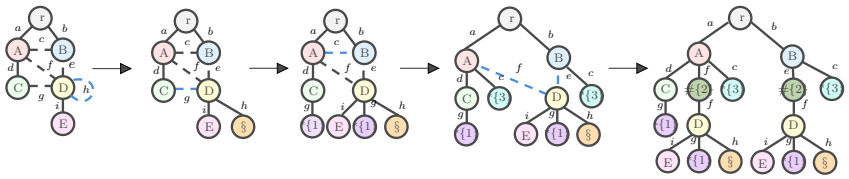
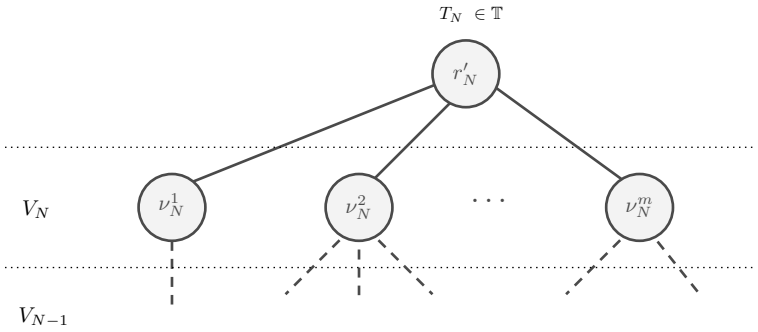


Figure 7: Production ordering on a simple example

4.2.7 Image set of f_G

Lemma 4 *The general morphism f leads to a tree.* $\forall\ g\ \in \mathbb{G},\ f(g) \in \mathbb{T}$

Proof We make the following proposal:



$\mathcal{P}_N$

In the context of $G^*_{\rho,N}$, sub-graph up to level level N, where it is no longer possible to apply any p_s^N, p_i^N or p_c^N output, if a ρ'_N vertex exists that connects to all vertices $\nu_N \in V_N$, then the resulting graph T_N is a tree whose root is ρ'_N.

$$T_N = \left(W_N \cup \rho'_N\ ,\quad E_N\ \cup \bigcup_{\nu_i \in V_N} (\rho'_N\ , \nu_i) \right),\ T_N \in \mathbb{T}$$

In other words, we claim that for a rank $N \in [\ 0,\ N_G\ ]$, applying all possible productions on $G_{\rho,N}$ results in the fact that all $\nu_N \in V_N$ becomes trees. We prove this claim by induction on the level N.

$\mathcal{P}_0$

vertices belonging to V_0 are leaves.

If a ρ'_0 vertex connects these leaves, the resulting graph is by definition a tree whose root is ρ'_0.

$$\mathcal{P}_0 \text{ is true} \tag{25}$$

$\mathcal{P}_{N+1}$

Suppose that $\mathcal{P}_N$ is true. (induction hypothesis)

$P_N \Leftrightarrow \forall \nu_N \in V_N$, ν_N is the root of a tree (potentially null) composed of all its descendants from the lower levels. Any ν_N can therefore be replaced without loss

of information by $\sigma^T(t_{\nu_N})$ according to the previous section (section 3), and thus become leaves.

Once this transformation is done, we can go back to the context of the vertices V_{N+1} and apply each of the productions as many times as we observe matches, to obtain $G^*_{\rho,\ N+1}$.

$$\begin{aligned}
&\nexists\ m_c : G^*_{\rho,\ N+1} \to LHS_c \ \Leftrightarrow \nexists\ <\rho'_{N+1}\ ,\ A,\ B,\ \rho'_{N+1}> \\
&\nexists\ m_i : G^*_{\rho,\ N+1} \to LHS_i \ \Leftrightarrow \nexists\ <\rho'_{N+1}\ ,\ A,\ C,\ B,\ \rho'_{N+1}> \\
&\nexists\ m_s : G^*_{\rho,\ N+1} \to LHS_s \ \Leftrightarrow \nexists\ <\rho'_{N+1}\ ,\ A,\ A,\ \rho'_{N+1}>
\end{aligned}$$

There is thus no cycle in $G^*_{\rho,\ N+1}$, which can be seen by construction, because in all the described productions, if a vertex ρ' is the direct parent of vertices A and B (or simply A in the case of self-bound), then this vertex ρ' is the root of a tree, hence:

$$\mathcal{P}_{N+1} \text{ is true} \tag{26}$$

$$(25), (26)\ \Rightarrow\ \mathcal{P}_N \text{ is true for all } N \tag{27}$$

Finally, for $N = lvl(\rho),\ \rho'_N = \rho$, and thus $T_N\ = G_\rho$.

□

4.2.8 Preservation of isomorphic properties by f_G

Lemma 5 *The tree produced by f is unique for any class of isomorphism.*

Notations and recall.

Recall that the productions are organized:

- by levels : $f_G\ = f_{N_G} \circ\ \ldots\ \circ f_0$
- by type inside a level : $f_N = p^N_{i,} \circ\ p^N_c \circ\ p^N_s$
- according to an order relation by level and given type: $p^N_x\ = p^N_{x_k} \circ \ldots \circ p^N_{x_1}$

With $\forall\ p_{x_i} \in p^N_x\ ,\ p_{x_i} : L \xleftarrow{l} K \xrightarrow{r} R$ designating a (bijective) production to be applied on the G graph, provided with an associated matching function $m_{x_i} : V_G \to LHS_m$ satisfying a system of conditions S_x. The respective sets of all these elements on a graph G are noted as $\{p_G\}$ and $\{m_G\}$.

We recall that $G \simeq H$ if and only if there is at least one bijection from V_G into V_H preserving the structure and the labels :

$$G \simeq H \Leftrightarrow\ \exists\ \pi : V_G \to V_H\ ,\ \forall\ \nu_1, \nu_2 \in V_G\ ,\ \begin{array}{ll} & L^E_G(\nu_1,\ \nu_2) = L^E_H(\pi(\nu_1),\ \pi(\nu_2)) \\ \wedge & L^V_G(\nu_1) = L^V_H(\pi(\nu_1)) \\ \wedge & L^V_G(\nu_2) = L^V_H(\pi(\nu_2)) \end{array}$$

One notes such a bijection $\pi : V_G \leftrightarrow V_H$, but could use the syntaxic shortcut $\pi(V_G)\ =\ V_H$.

$$G \simeq H \Leftrightarrow \exists\ \pi : V_G \leftrightarrow V_H$$

We will deduce the contraposition of the previous proposition.

$$G \not\simeq H = \overline{G \simeq H}$$

$$G \not\simeq H \Leftrightarrow \forall\ \pi : V_G \to V_H\ ,\ \exists\ \nu_1, \nu_2 \in V_G\ ,\ \begin{array}{cc} & L_G^E(\nu_1,\ \nu_2) \neq L_H^E(\pi(\nu_1),\ \pi(\nu_2)) \\ \vee & L_G^V(\nu_1) \neq L_H^V(\pi(\nu_1)) \\ \vee & L_G^V(\nu_2) \neq L_H^V(\pi(\nu_2)) \end{array}$$

Similarly we note $\pi : V_G \nleftrightarrow V_H\ ,\ \pi(V_G) \neq V_H$.

$$G \not\simeq H \Leftrightarrow \forall\ \pi : V_G \nleftrightarrow V_H$$

Proof

$$\begin{cases} G \simeq H \Leftrightarrow f_G(G) \simeq f_H(H) \\ G \not\simeq H \Leftrightarrow f_G(G) \not\simeq f_H(H) \\ \overline{G \simeq H}\ =\ G \not\simeq H \end{cases} \Leftrightarrow \begin{cases} G \simeq H \Rightarrow f_G(G) \simeq f_H(H) & (\mathcal{P}_A) \\ G \not\simeq H \Rightarrow f_G(G) \not\simeq f_H(H) & (\mathcal{P}_B) \end{cases}$$

The problem is refomulated into the form of two proposals

$$(1)\ :\quad G \simeq H \Leftrightarrow \forall\ g = (V_g,\ E_g) \subseteq G,\ \exists\ h = (V_h,\ E_h)\ \subseteq H\ ,\ \exists\ \pi : V_g \leftrightarrow V_h$$

$$\overline{(1)}\ :\quad G \not\simeq H \Leftrightarrow \exists\ g = (V_g,\ E_g) \subseteq G,\ \forall\ h = (V_h,\ E_h)\ \subseteq H\ ,\ \forall\ \pi : V_g \nleftrightarrow V_h$$

We recall that $\{m_G\}$ designates the set of matching functions associated with productions that can be applied on G, in other words generated by G, such as :

$$\{m_G\} = \bigcup \begin{array}{l} m_c : V_G \to V_{LHS_c}\ ,\ S_c \\ m_i : V_G \to V_{LHS_i}\ ,\ S_i \\ m_s : V_G \to V_{LHS_s}\ ,\ S_s \end{array}$$

For each of these matching functions $m^j \in \{m_G\}$, a single sub-graph $g^j \subseteq G$ is selected. Formally, $m^j(V_{g^j}) = V_{LHS^j}$, $p^j(LHS^j) = RHS^j$.

$$(2)\ :\quad \exists\ \pi : V_G \leftrightarrow V_H \underset{g\ \subseteq\ G,\ h\ \subseteq\ H}{\overset{(1)}{\Longrightarrow}} \forall\ m^j \in \{m_G\},\ \exists\ m^{j'} \in \{m_H\}\ ,\ h^{j'} = \pi\left(g^j\right)$$

$$\overline{(2)}:\quad \exists\ m^j \in \{m_G\},\ \forall\ m^{j'} \in \{m_H\}\ ,\ h^{j'} \neq \pi\left(g^j\right) \underset{g\ \subseteq\ G,\ h\ \subseteq\ H}{\overset{\overline{(1)}}{\Longrightarrow}} \forall\ \pi : V_G \nleftrightarrow V_H$$

$$(3) \ : \ \{m_G\} = \{m_H\} \ \Leftrightarrow \forall \ m \ (m \ \in \ \{m_G\} \ \Leftrightarrow \ m \ \in \{m_H\})$$
$$\overline{(3)} \ : \ \{m_G\} \neq \{m_H\} \ \Leftrightarrow \exists \ m \in \{m_G\}, \ m \notin \{m_H\} \vee \exists \ m \in \{m_H\}, \ m \notin \{m_G\}$$

$$(4) \ : \quad G \simeq H \ \overset{(2),\ (3)}{\Longrightarrow} \ \{m_G\} = \{m_H\}$$
$$\overline{(4)} : \quad \{m_G\} \neq \{m_H\} \ \overset{\overline{(2)},\ \overline{(3)}}{\Longrightarrow} \ G \not\simeq H$$

The sets of matching functions are therefore equal if $G \simeq H$. However, it is not possible to conclude that these sets are equal only if $G \simeq H$. It is also not possible to say that if $G \not\simeq H$, then $\{m_G\} \neq \{m_H\}$ although the converse is true.

Furthermore, this is not sufficient to prove $(\mathcal{P}_A)$ either, because with f_1 and f_2 two applications, in general $f_1 \circ f_2 \neq f_2 \circ f_1$. Beyond the equality of these sets, we are now going to prove that they can be applied accordingly to an ordering leading to a result that is invariant to the class of isomorphism of the graph.

The productions forming a f function are ordered in a non-strict order:

$$(5) \ : \quad \begin{cases} ([\ 0, \ N_G \ ], \ <) \implies (\{f_G\}, \ <) \\ (\{s, \ c, \ i\}, \ <) \implies \forall \ f_N \ \in f_G \ (\{f_N\}, \ <) \\ \forall \ p^j \ \in p_x^N, \ \left(\bigcup_{\nu \ \in \ g^j} \sigma^T(t_\nu), \ \leq\right) \implies \left(\{p_x^N\}, \leq\right) \end{cases} \implies \left(\bigcup_{m^j \ \in \ \{m_G\}} m^j \ , \leq \right)$$

A set $\{m_G\}$ is thus generating an ordered sequence of K matching functions,

$$\{m_G\} \ \equiv < m_G^1 \ , \ \ldots, \ m_G^K > \ , \ \forall \ k \ \in [\ 1, \ K-1 \ ], \ m_{G,}^k \leq m_G^{k+1}$$

However, since this order is not strict, there may be equalities, noted $m^a \equiv m^b$:

$$(6) \ : \quad m^a, \ m^b \ \in \{m_G\}, \ m^a \equiv m^b \Leftrightarrow m^a \leq m^b \ \wedge \ m^b \leq m^a$$
$$m^a \equiv m^b \Leftrightarrow \bigcup_{\nu \ \in \ g^a} \sigma^T(t_\nu) \ = \ \bigcup_{\nu \ \in \ g^b} \sigma^T(t_\nu) \Leftrightarrow g^a \simeq g^b$$

This set of productions is stable, as no production application can add or remove any other production than the one being processed (as it is not possible to apply any matching function in an RHS, each of which is acyclic).

$$(7) \ : \quad G \simeq H \ \overset{(4),\ (5),\ (6)}{\Longrightarrow} \ \forall \ k \ \in [\ 1, \ K \ ] \ , \ m_G^k \equiv m_H^k$$
$$G \simeq H \implies \forall \ k \ \in [\ 1, \ K \ ] \ , \ g^k \simeq h^k$$

We note $\overset{k}{G}$ the graph G to which we applied the first k productions.

We study the conditions of an isomorphism between $\overset{k}{G}$ and $\overset{k}{H}$:

$$\forall \ k \ \in [\ 1, \ K \ ] \ , \ \overset{k}{G} \simeq \overset{k}{H} \ \Leftrightarrow p_G^{k-1} \left(\overset{k-1}{G} \right) \simeq p_H^{k-1} \left(\overset{k-1}{H} \right)$$

We recall that the productions are deterministic bijections. We underline here their injection properties:

$$\forall\ p_G \in \{p_G\},\ \forall\ G_1,\ G_2 \in \mathbb{G},\ p_G(G_1) \simeq p_G(G_2) \implies G_1 \simeq G_2$$
$$\forall\ p_H \in \{p_H\},\ \forall\ H_1,\ H_2 \in \mathbb{G},\ p_H(H_1) \simeq p_H(H_2) \implies H_1 \simeq H_2$$

Because of these properties, if two production applications lead to the same result, then the operands (the intermediate graphs) and the operators (the productions) are respectively isomorphic and identical:

$$p_G^{k-1}\left(\overset{k-1}{G}\right) \simeq p_H^{k-1}\left(\overset{k-1}{H}\right) \Leftrightarrow \left(p_G^{k-1} = p_H^{k-1}\right)\ \wedge \left(\overset{k-1}{G} \simeq \overset{k-1}{H}\right)$$

We rewrite with the proposal$P_k \Leftrightarrow \overset{k}{G} \simeq \overset{k}{H}$:

$$(8)\ :\quad P_k \Leftrightarrow \left(m_G^{k-1} \equiv m_H^{k-1}\right) \wedge P_{k-1}$$

In particular we identify,

$$P_K \Leftrightarrow f_G(G) \simeq f_H(H)$$

Supposing that $G \simeq H$, one can prove $(\mathcal{P}_A)$ by induction.
It is obvious that $G \simeq H \implies \overset{0}{G} \simeq \overset{0}{H}$ since $G \simeq H \equiv \overset{0}{G} \simeq \overset{0}{H}$.

$$\begin{cases} G \simeq H \overset{(7)}{\implies} \forall\ k\ \in [\ 1,\ K\ ]\ ,\ m_G^k \equiv m_H^k \\ G \simeq H \equiv \overset{0}{G} \simeq \overset{0}{H} \end{cases} \implies P_1\ is\ true$$

$$(9)\ :\quad \begin{cases} G \simeq H \overset{(7)}{\implies} \forall\ k\ \in [\ 1,\ K\ ]\ ,\ m_G^k \equiv m_H^k \\ P_1 \end{cases} \overset{(8)}{\implies} \forall\ k\ \in [\ 2,\ K\ ] P_k\ is\ true$$

$$G \simeq H \overset{(9)}{\implies} P_K\ is\ true$$

$$(\mathcal{P}_A)\ :\quad G \simeq H \implies f_G(G) \simeq f_H(H)$$

Let us assume that $G \not\simeq H$, we can prove $(\mathcal{P}_B)$ by directly invalidating the initialization P_1. By hypothesis, $G \not\simeq H \equiv \overset{0}{G} \not\simeq \overset{0}{H}$.

$$G \not\simeq H \equiv \overset{0}{G} \not\simeq \overset{0}{H} \implies P_1\ is\ false$$

$$(10) \ : \quad \bar{P_1} \overset{(8)}{\Longrightarrow} \forall\ k\ \in [\ 2,\ K\ ],\ P_k\ is\ false$$
$$G \ncong H \overset{(10)}{\Longrightarrow} P_K\ is\ false$$

$$(\mathcal{P}_B) \ : \quad G \ncong H \Longrightarrow f_G(G) \ncong f_H(H)$$

□

Furthermore, $\overset{K}{G} \in \mathbb{T},\ \overset{K}{H}\ \in \mathbb{T}$ according to the lemma 4, and the traces for these two trees by σ^T will be identical according to the lemma 3.

$$\sigma^T\left(\overset{K}{G}\right) = \sigma^T\left(\overset{K}{H}\right)$$

4.3 Root designation and encoding (step 3 of Scott algorithm)

We address this section by defining a trace function applicable to any graph with an identified root.

$$\sigma^{G_\rho} : G_\rho \to \Sigma^*,$$
$$\sigma^{G_\rho} :\ G_\rho \mapsto \sigma^T(f(G_\rho))$$

Until now, it was assumed that the ρ root was known. To designate among all the vertices which one is the root vertex, we can simply take the one generating a minimum trace, because any set with a total order admits a minimum element. Any graph is thus encoded by its minimum encoding. However several vertices may lead to the minimum trace due for instance to some symmetry in the graph. Hence the choice for the root vertex is not unique in general. If this is the case, we just select arbitrarily one of the potential root vertices leading to the minimal trace, since Scott will provide exactly the same trace for all of these vertices.

$$\rho_G = \underset{\nu \in V_G}{arg\ min}\left(\sigma^{G_\rho}(G_\nu)\right)$$

Since the identification of the ρ_G root for any ρ_G graph is now an invariant up to an isomorphism, the f morphism is bijective (up to an isomorphism), which until now was based on the assumption that ρ was identified. We derive an injective trace function applicable to any graph, usable as a hash function.

$$\sigma^G : G \to \Sigma^*,$$
$$\sigma^G :\ G \mapsto \sigma^T(f(G_{\rho_G}))$$

4.4 Grammar

The grammar that described the syntax of the trace σ^T proposed by Scott is given below.

$\langle tree \rangle$ ::= [$\langle descendant\_list \rangle$] $\langle vertex \rangle$: $\langle edge\_modality \rangle$

$\langle descendant\_list \rangle$ ::= ($\langle tree \rangle$ { , $\langle tree \rangle$ })

$\langle vertex \rangle$ ::= $\langle vertex\_label \rangle$ | $\langle mirror\_vertex\_label \rangle$ | $\langle virtual\_vertex\_label \rangle$ | $\langle self\_vertex\_label \rangle$

$\langle vertex\_label \rangle$ ::= $\langle label \rangle$

$\langle mirror\_vertex\_label \rangle$ ::= # $\langle magnet \rangle$

$\langle virtual\_vertex\_label \rangle$ ::= * $\langle magnet \rangle$

$\langle self\_vertex\_label \rangle$ ::= §

$\langle magnet \rangle$ ::= {$\langle label \rangle$}

$\langle edge\_modality \rangle$ ::= $\langle label \rangle$

$\langle label \rangle$ ::= Σ^*

Grammar 1: Scott Grammar supporting σ^T, derived from [23]

5 Algorithmic complexity

In this section, we evaluate the temporal complexity of the Scott canonical encoding algorithm. We assimilate this complexity to the computational complexity required to evaluate the trace function σ^G, and its asymptotic behavior according to $n = \| V_G \|$.

We approximate this complexity through an upper bound $O\left(\sigma^G\right)$ corresponding to the worst possible case, and a lower bound $\Omega\left(\sigma^G\right)$ corresponding to the best possible case.

The overall complexity consists of the composition of the complexities of the three steps of the Scott algorithm :

- identification of the root ρ, noted φ_ρ
- morphism of $\mathbb{G}$ to $\mathbb{T}$ using f given the identification of ρ, noted φ_f
- encoding of the rooted tree, noted φ_t

$$\sigma^G \in \mathcal{O}(\varphi_\rho) \cdot [\mathcal{O}(\varphi_f) + \mathcal{O}(\varphi_t)]$$

with $\mathcal{O}(\varphi_\rho)$, $\mathcal{O}(\varphi_f)$ and $\mathcal{O}(\varphi_t)$ three complexity classes.

5.1 Complexity for identifying the root ρ, φ_ρ

Naively, one iterates on every vertex ν in G, which can be elected as the ρ root.

However, it is possible to restrict the set of vertices that can be candidates in this election. Thanks to a well-chosen function, we associate a computable score in $O(1)$ to each vertex, thus restricting the search domain to a number k_ρ of vertices, with $1 \leq k_\rho \leq n$. Hence:

$$\varphi_\rho \in \Omega(1) \quad \text{and} \quad \varphi_\rho \in O(n)$$

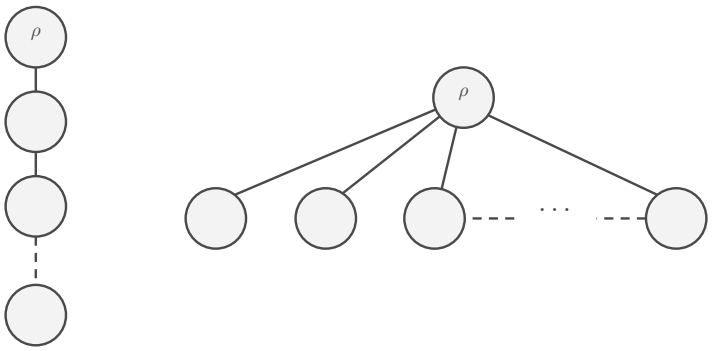


Figure 8: Best (left) and worst (right) cases for tree encoding

5.2 Complexity for encoding a rooted tree, φ_t

We can consider at this step of the algorithm that we have retained all the required information at any vertex level from the previous step. Thus, the encoding of a vertex and an edge is done in $O(1)$. Let t be the number of vertices of a tree, its encoding in a non canonical way is thus done in $O(t)$, to which we must add the complexity of the sorting operated for each set of descendants of a vertex. The best and worst case for this encoding (illustrated in Fig. 8) is thus directly linked to the sorting algorithm.

The complexity of a sort algorithm is typically in $O(t \cdot log(t))$ complexity, hence:

$$\varphi_t \in \Omega(t) \quad \text{and} \quad \varphi_t \in O(t + t \cdot log(t))$$

5.3 Morphism from $\mathbb{G}$ to $\mathbb{T}$ through f, φ_f

The application of morphisms leading to a tree representing a class of isomorphism affects the overall complexity of the algorithm in two ways. First, this morphism generates operations that have their own computing cost, but, in addition, it increases the size of the generated tree which will impact on the computing cost involved when producing the final trace.

In a deterministic way, Scott orders all the vertices by levels according to ρ. This φ_0 operation is carried out in exactly n iterations (using a breadth first search algorithm).

The number of applications of productions is not constant, and depends on the number of edges $e = ||E_G||$| compared to n, with $(n-1) \leq e\frac{n(n-1)}{2}$ for a simple graph, without self-bound which we omit for the moment.

5.3.1 Morphism algorithmic cost

The complexity of each of the productions necessary for implementing the morphism is relative to their location relatively to ρ. Indeed, the computation of *magnet* used to match two virtual vertices is performed from the trace of a sub-tree. In all cases, this *magnet* comes from the trace of two vertices, taken in the context of their exclusive descendants. Although one can keep the result once it is evaluated for a vertex, one can consider that such a trace is computed for all vertices in the worst case. Thus, each v vertex generates beforehand an operation in $O(w_v + w_v \cdot log(w_v))$

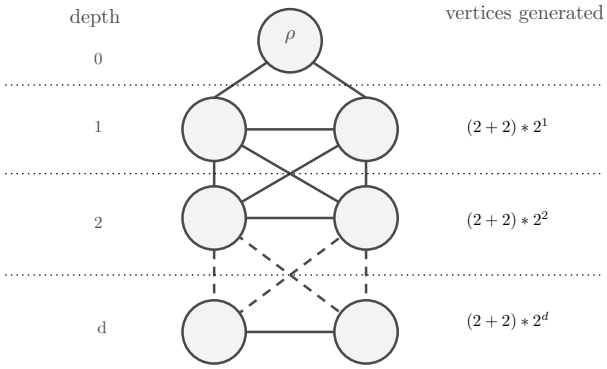


Figure 9: Graph maximising the size of the generated tree.

with w_v the number of descendants of this vertex. The following calculations will be performed in $O(1)$.

$$\varphi_{f_1} \in \Omega(1) \quad \text{and} \quad \varphi_{f_1} \in O\left(n * \frac{d_{max}}{2} + \frac{d_{max}}{2} \cdot log\left(\frac{d_{max}}{2}\right)\right)$$

We can express the number of edges separating a simple complete graph (without self-bound) from a tree, corresponding to the maximum number of productions to be applied, by $\frac{n(n-1)}{2} - (n-1) = \frac{n^2-3n+2}{2}$, to which we add a self-bound to be solved per vertex in the worst case, operation that is performed in $O(n)$. In the best case, if G is already a tree, there is no morphism to apply. Hence, we get:

$$\varphi_{f_2} \in \Omega(1) \quad \text{and} \quad \varphi_{f_2} \in O\left(n + \frac{n^2-3n+2}{2}\right) = O\left(\frac{n^2+n+2}{2}\right)$$

We obtain for sequence $\varphi_f = \varphi_{f_2} \circ \varphi_{f_1} \circ \varphi_{f_0}$:

$$\varphi_f \in \Omega(n) \quad \text{and} \quad \varphi_f \in O\left(n + n + \frac{n^2-3n+2}{2}\right) = O\left(\frac{n^2+n+2}{2}\right)$$

5.3.2 Size of the generated tree

Co-bound and in-bound productions do not generate the same number of additional vertices and edges. Indeed, while a production solving a co-bound will invariably generate two virtual vertices, an in-bound production will double the number of vertices succeeding the vertex concerned by the in-bound. Again, the worst possible case in terms of the size of the generated tree is not, as one might think, the graph with the most in-bounds, which would be a graph with two completely connected layers, but the graph having as many levels of two vertices as possible.

If each step will invariably generate two vertices for each co-bound, the vertices that are duplicated due to the in-bound will also be duplicated in the previous stages, hence the application of additional power for each step.

The number of d levels that can be obtained on this type of graph is $(n-1)/2$ (integer division). We can thus estimate the number of vertices generated in the

worst case at $\sum_{d=1}^{(n-1)/2} 8^d = \frac{2}{7}\left(2^{\frac{3n}{2}+\frac{1}{2}} - 4\right) \sim k^{2\cdot n}$, with $k \simeq 2$. The size n' of the trees generated in this step is thus framed as follows:

$$n \leq n' \leq n + k^{2\cdot n}\ ,\ k \simeq 2$$

5.4 Overall complexity

We recall the overall complexity of the σ^G algorithm as the following composition: $\sigma^G \in \mathcal{O}(\varphi_\rho) \cdot [\mathcal{O}(\varphi_f) \circ \mathcal{O}(\varphi_t)]$.

While some complexities are difficult to determine precisely, which leads to large upper bounds, there is however a very large difference in complexity between the simplest case corresponding to a graph that is directly in the form of a tree, and the most complex case corresponding to a graph too regular to reduce the number of candidate roots, and presenting a large number of in-bounds. This leads to our (crude) lower and upper bounds:

$$\sigma^G \in \Omega(n)$$

$$\sigma^G \in O\left(n \cdot \left(\left(n + k^{2n}\right) + \left(n + k^{2n}\right) \cdot log\left(\left(n + k^{2n}\right)\right) + \frac{n^2 + n + 2}{2}\right)\right) \sim O\left(k \cdot n^{2n}\right),\ k \simeq 2$$

5.5 Parallelisation

In spite of a significant temporal complexity, which is exponential in the worst case, our algorithm has parallelization possibilities allowing to gain up to an order of magnitude on multi-core architectures, or even two on multi-server cluster-type architectures.

The first parallelizing possibility is related to the root vertex identification. Once all the candidates have been identified (and if possible minimized), the evaluations of the traces associated with them are independent, and thus can be executed simultaneously on several processors. Indeed, the output of the algorithm is entirely dependent on the identification of the best root candidate, which is naturally well adapted to a distribution of the computations using functional programming.

The second avenue of parallelization consists in distributing the scheduling of the productions carried out for a given category and level. Indeed, the computation of these productions involves numerous evaluations of tree sub-graphs, whose impact is not negligible on the execution time of the algorithm. However, here again, the score (ordering) associated to a production is only a function of the elements directly involved in this production, i.e. the current level, which is not yet being processed, and the lower levels that have been already processed. Each production evaluation can therefore work on a local, fixed copy of these elements without any risk of side effects (since the processing has not started yet). Here again, an implementation using the functional paradigm allows to implement a parallelization at this stage of the algorithm. However, the application of the productions alters at each step the structure of the graph, hence, this sequence of transformations remains essentially sequential.

An implementation of this parallelization can for example be done in the context of a *Map Reduce* framework. Work in this direction has been started on the *Spark* technology, allowing to divide the total computation time by the number of available processing cores.

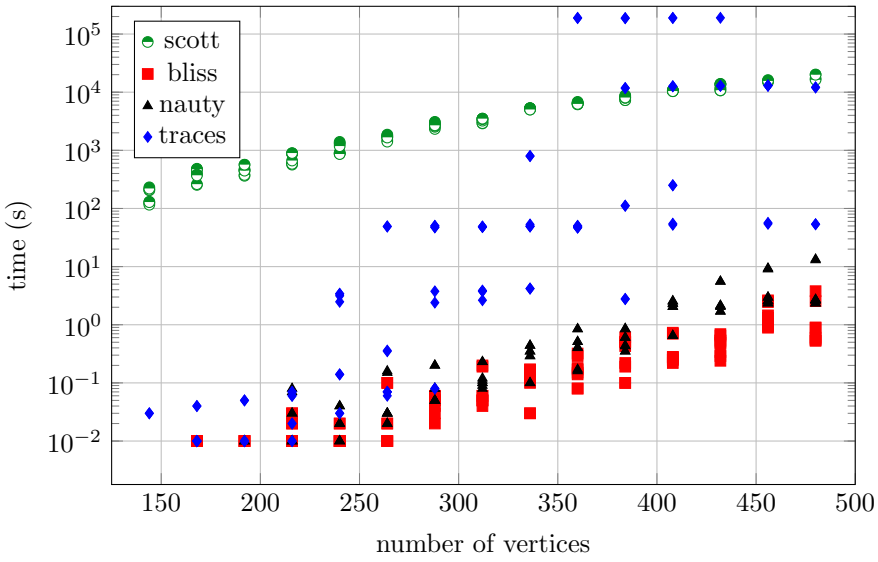


Figure 10: Benchmark on "shrunken multipedes" canonization (y-log scale)

5.6 Empirical evaluation

We empirically verify the effectiveness of the Scott algorithm, namely the correct detection of isomorphism in complex graphs collected from a synthetic data set, and estimate comparatively its empirical complexity while measuring the processing elapsed-time on a common hardware.

To that end, we evaluate Scott on complex combinatorial graphs produced in the scope of a benchmark [11] dedicate to isomorphism detection. Among the families of graphs used and made available in this benchmark, we are interested in the "shrunken multipedes" graphs, a class of graphs that is built from the *"Cai, Fürer and Immerman"* graphs.

This family of graphs is specially designed to present cases of non-trivial isomorphisms (or non-isomorphisms), and therefore with characteristics quite distant from the ideal cases mentioned above. In particular, the number of edges is large compared to the number of vertexs (by a factor of 7.8 on average). These graphs are, in practice, very far from trees.

We calculate the canonical form of 157 graphs having up to 500 vertexs, grouped correctly into 79 isomorphism classes. On this example, the sanity check performed on Scott gives 100% of accurracy: basically Scott produces no error (either false positive or false negative errors).

We then compare the performance of our algorithm with the state of the art algorithms that also propose a canonical form, namely Traces, Nauty and Bliss. We present in Fig. 10 the time necessary to calculate the canonical trace of a graph as a function of its number of vertexs. Since several orders of magnitude are represented on the first axis, a log scale is used.

We observe that while the best algorithms on this benchmark show a significantly better efficiency, Scott remains quite comparable with some of these algorithms when the the size of the processed graphs is above 380 vertexs. Scott is therefore

reasonably usable in practice, even for graphs resulting from combinatorial processes, which are far from optimal conditions. We also note that for a given number of vertexs, the variability of the processing time is much less important for Scott than for the other algorithms, for which a deviation of several orders of magnitude can be observed for several graphs having the same number of vertexs.

6 Conclusion

In this article we have presented the proofs of correctness for the Scott algorithm. We have shown in section 4 that the conversion of any general labelled graph into a rooted-tree (that corresponds two the two first steps of the Scott algorithm) is reversible up to an isomorhism. This means that inside an isomorhism class, each graph can be associated to a unique graph representative. This representative may take the form of a canonical adjacency matrix. Furthermore, we have shown in section 3 that the encoding of any rooted-tree into a string class is unique up to an isomorphism.

These results ensure that the three canonical forms provided by Scott for a general labelled graph (namely an adjacency matrix, a DAG or rooted-tree, and a string) are valid.

We have also presented some complexity bounds for the best and worse case situations. If the temporal complexity remains problematic for theoretical graphs resulting from combinatorial methods, with an exponential execution time with respect to the number of vertices (which is also the case for the reference baseline algorithms), it is much more tractable on simpler graphs with a number of edges close to the number of vertices. For this category of graphs, we rather observe in practice a linear trend, quite usable in common applications.

A important amount of work remains to be done on the implementation and optimization of this algorithm. However, as it stands, Scott remains the only algorithm to deal natively with colored vertices and edges, and offers good parallelization possibilities.

As a perspective, a generalization of Scott to handle oriented graphs, typically used in the representation of social relations or dependencies, could be quite easily considered by adapting the set of productions used to transform a graph into an equivalent tree. Since a rooted tree with non-oriented edges is equivalent to the same rooted tree with oriented edges, the trace function once the tree is obtained would be usable without any modification. However, the validity of this extension remains to be asserted.

Competing interests
The authors declare that they have no competing interests.

Acknowledgements
We thank Oleon and MiXScience companies for funding the thesis in which this work took place. Special thanks go to IRISA and LMBA laboratories teams for their discussions and feedback about this work.

Author's contributions
The algorithm was designed by NB and PFM, the proofs and complexity study involved all authors NB, PFM and EF, and finally the Python implementation was done by NB.

Availability of data and materials
A Python implementation of the Scott algorithm is available under MIT licence on a GitHub repository. Graphs used for isomorphism benchmark are available on `.dot` format.

Author details
[1]IRISA, Université de Bretagne Sud, Campus de Tohannic, 56000 Vannes, France. [2]LMBA, Université de Bretagne Sud, Campus de Tohannic, 56000 Vannes, France. [3]See-d , 6 bis, rue Henri Becquerel, 56000 Vannes, France.

References

1. Bloyet, N., Marteau, P.-F., Frenod, E.: Scott: A method for representing graphs as rooted trees for graph canonization. In: International Conference on Complex Networks and Their Applications, pp. 578–590 (2019). Springer, Cham
2. Babai, L., Luks, E.M.: Canonical labeling of graphs. In: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, pp. 171–183 (1983). ACM
3. Zemlyachenko, V.N., Korneenko, N.M., Tyshkevich, R.I.: Graph isomorphism problem. Journal of Soviet Mathematics **29**(4), 1426–1481 (1985)
4. Schöning, U.: Graph isomorphism is in the low hierarchy. Journal of Computer and System Sciences **37**(3), 312–323 (1988)
5. Arvind, V., Kurur, P.P.: Graph isomorphism is in spp. In: The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings., pp. 743–750 (2002). IEEE
6. Booth, K.S., Colbourn, C.J.: Problems Polynomially Equivalent to Graph Isomorphism. Computer Science Department, Univ., ??? (1979)
7. Kobler, J., Schöning, U., Torán, J.: The Graph Isomorphism Problem: Its Structural Complexity. Springer, ??? (2012)
8. Babai, L.: Graph isomorphism in quasipolynomial time. In: Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing, pp. 684–697 (2016). ACM
9. Luks, E.M.: Isomorphism of graphs of bounded valence can be tested in polynomial time. Journal of computer and system sciences **25**(1), 42–65 (1982)
10. Hopcroft, J.E., Wong, J.-K.: Linear time algorithm for isomorphism of planar graphs (preliminary report). In: Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, pp. 172–184 (1974). ACM
11. Neuen, D., Schweitzer, P.: Benchmark graphs for practical graph isomorphism. arXiv preprint arXiv:1705.03686 (2017)
12. Gurevich, Y., Shelah, S.: On finite rigid structures. The Journal of Symbolic Logic **61**(2), 549–562 (1996)
13. Darga, P.T., Sakallah, K.A., Markov, I.L.: Faster symmetry discovery using sparsity of symmetries. In: Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE, pp. 149–154 (2008). IEEE
14. Codenotti, P., Katebi, H., Sakallah, K.A., Markov, I.L.: Conflict analysis and branching heuristics in the search for graph automorphisms. In: 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, pp. 907–914 (2013). IEEE
15. López-Presa, J.L., Anta, A.F., Chiroque, L.N.: Conauto-2.0: Fast isomorphism testing and automorphism group computation. arXiv preprint arXiv:1108.1060 (2011)
16. Junttila, T., Kaski, P.: Engineering an efficient canonical labeling tool for large and sparse graphs. In: 2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX), pp. 135–149 (2007). SIAM
17. Junttila, T., Kaski, P.: Conflict propagation and component recursion for canonical labeling. In: International Conference on Theory and Practice of Algorithms in (Computer) Systems, pp. 151–162 (2011). Springer
18. McKay, B.D., et al.: Practical graph isomorphism (1981)
19. Piperno, A.: Search space contraction in canonical labeling of graphs. arXiv preprint arXiv:0804.4881 (2008)
20. McKay, B.D., Piperno, A.: Practical graph isomorphism, ii. Journal of Symbolic Computation **60**, 94–112 (2014)
21. Neveu, J.: Arbres et processus de galton-watson. In: Annales de l'IHP Probabilités et Statistiques, vol. 22, pp. 199–207 (1986)
22. Velasco, P.P.P.: Matrix Graph Grammars, p. 321 (2008). 0801.1245. http://arxiv.org/abs/0801.1245
23. Olsen, G.: Gary olsen's interpretation of the "newick's 8: 45" tree format standard. URL http://evolution. genetics. washington. edu/phylip/newick doc. html (1990)
24. Weininger, D.: Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. Journal of chemical information and computer sciences **28**(1), 31–36 (1988)
25. Heller, S.R., McNaught, A.D.: The iupac international chemical identifier (inchi). Chemistry International **31**(1), 7 (2009)
26. Diestel, R.: Graph theory. 2005. Grad. Texts in Math **101** (2005)
27. Babai, L., Kantor, W.M., Luks, E.M.: Computational complexity and the classification of finite simple groups. In: 24th Annual Symposium on Foundations of Computer Science (Sfcs 1983), pp. 162–171 (1983). IEEE
28. Inokuchi, A., Washio, T., Motoda, H.: An apriori-based algorithm for mining frequent substructures from graph data. In: European Conference on Principles of Data Mining and Knowledge Discovery, pp. 13–23 (2000). Springer
29. Yan, X., Han, J.: gspan: Graph-based substructure pattern mining. In: Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference On, pp. 721–724 (2002). IEEE
30. Cayley, A.: A theorem on trees. Quart. J. Math. **23**, 376–378 (1889)
31. Harary, F., Palmer, E.M.: Graphical Enumeration. Elsevier, ??? (2014)
32. Velasco, P.P.P., de Lara, J.: Matrix Approach to Graph Transformation: Matching and Sequences. Lecture Notes in Computer Science **4178**, 122 (2006). doi:10.1007/11841883_10
33. Ehrig, H., Hoffmann, K., Padberg, J.: Transformations of Petri nets. Electronic Notes in Theoretical Computer Science **148**(1 SPEC. ISS.), 151–172 (2006). doi:10.1016/j.entcs.2005.12.016
34. Vishwanathan, S.V.N., Schraudolph, N.N., Kondor, R., Borgwardt, K.M.: Graph kernels. Journal of Machine Learning Research **11**(Apr), 1201–1242 (2010)
35. Gaüzere, B., Brun, L., Villemin, D.: Two new graphs kernels in chemoinformatics. Pattern Recognition Letters **33**(15), 2038–2047 (2012)