Scott : A method for representing graphs as rooted trees for graph canonization

Nicolas Bloyet, Pierre-François Marteau, Emmanuel Frenod

Scott : A method for representing graphs as rooted trees for graph canonization

Nicolas Bloyet[123], Pierre-François Marteau[1], and Emmanuel Frénod[23]

[1] IRISA, Université Bretagne Sud, Campus de Tohannic, 56000 Vannes, France
[2] LMBA, Université Bretagne Sud, Campus de Tohannic, 56000 Vannes, France
[3] See-d, Parc Innovation Bretagne Sud, 56000 Vannes, France
firstname.lastname@univ-ubs.fr

Abstract. Graphs increasingly stand out as an essential data structure in the field of data sciences. To study graphs, or sub-graphs, that characterize a set of observations, it is necessary to describe them formally, in order to characterize equivalence relations that make sense in the scope of the considered application domain. Hence we seek to define a canonical graph notation, so that two isomorphic (sub) graphs have the same canonical form. Such notation could subsequently be used to index and retrieve graphs or to embed them efficiently in some metric space. Sequential optimized algorithms solving this problem exist, but do not deal with labeled edges, a situation that occurs in important application domains such as chemistry. We present in this article a new algorithm based on graph rewriting that provides a general and complete solution to the graph canonization problem. Although not reported here, the formal proof of the validity of our algorithm has been established. This claim is clearly supported empirically by our experimentation on synthetic combinatorics as well as natural graphs. Furthermore, our algorithm supports distributed implementations, leading to efficient computing perspectives.

Keywords: graph canonization, graph isomorphism, graph rewriting, labeled graph

1 Problem statement and related works

We address in this article the canonization of general graphs, in particular labeled graphs. Graph structures are adapted to represent a population of interactive, distributed entities such as networks, molecules, social relations, etc.

A graph can be described by an enumeration of its elements contained in a set of vertices and a set of edges. However, such an enumeration is not unique, because one cannot presuppose the existence of an ordering relation defined on these two sets. Due to this non uniqueness, two identical graphs can have different representations, i.e. encodings. Beyond the obvious theoretical interest that it raises, being able to decide on the structural equality of graphs has a great application impact in practice. This leads to the definition of the so-called graph isomorphism (GI) problem.

Definition 1. ***Graph Isomorphism****: Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two graphs. An isomorphism $f : V_G \to V_H$ from G to H is a bijection from V_G to V_H preserving the edges in G and H, namely such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H .*

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

If f exists, then G and H are isomorphic, and we write $G \simeq H$. Determining the existence (or non-existence) of f is the most basic way to answer the GI problem. Determining an expression of f is a task at least as complex which also addresses the GI problem. Despite numerous works on the GI problem ([3] [25]), its complexity class remains poorly known ([21], [1]): it is not proved that this problem is **NP-complete**, although no polynomial algorithm solving the GI problem for the general case has been discovered yet. This problem has its own complexity class ([5], [13]), and recent work [2] seems to indicate that this problem is quasi-polynomial.

1.1 Canonical form of a graph

Another way to solve this problem is to compute, for any graph G, a canonical representative. This representative (usually a graph that is isomorphic to G) is unique for the entire isomorphism class associated to G [4].

Definition 2. *Canonization function*

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

As the graph canonization (GC) problem provides a solution to the GI problem, it is obviously at least as complex as the GI problem. In the general case, it is more expensive to calculate a canonical form than to test the existence of an isomorphism. Indeed, it is often possible to rule on isomorphism before the complete execution of the algorithm in charge of solving GC.

Despite this algorithmic overhead, the GC problem has many advantages. Since the canonical form of a graph G is unique, it only needs to be computed once: afterwards we can directly compare the canonical forms of candidate isomorphic graphs very efficiently. This additional complexity is therefore very rapidly profitable as long as we have to perform a lot of matching tests on a large population of (sub-) graphs.

In various application such as chemistry, one seeks to identify substructures (sub-graphs) that have a minimal statistical importance in a population of graphs. These substructures may be refereed to as fragments or patterns among others naming, and convey some domain-dependent knowledge (e.g. the benzenic cycle in chemistry). It is important to identify and uniquely characterize

[4] equivalence class of the equivalence relation "is isomorphic to"

(up to an isomophism) rapidly these substructures to either index and retrieve data represented by graphs or design relevant graph embeddings approaches [4] suited for subsequent machine learning (in particular deep learning) techniques.

For very specific domains, heuristics have been developed to achieve dedicated canonization of graphs structures. This is the case for the *SMILES* notation [24] or *InChI* [9] commonly used in organic chemistry, which are intended to provide unique entries for molecules. The main limitation of these type of graph canonization lies in the invocation of chemical invariants, namely domain specific knowledge (that is external to the graph notation), to remove all notational ambiguities. These heuristics can only canonize whole and valid chemical compounds, and are thus not applicable in the general case.

1.2 Existing canonization algorithms

There is a significant gap between the theoretical and practical handling of the GI/GC problems: if polynomial complexity algorithms exist for a large number of restricted classes of graphs (bounded degree [15], planar graphs [10], etc.), and if it is theoretically known that the general case is likely to be processed in quasi-polynomial time ([2]), the best implemented algorithms so far solve these problems for partially labeled graphs with an exponential complexity.

A complete state of the art of these implementations is proposed in [18], as well as a benchmark on complex isomorphism cases built using [8]. The algorithms Saucy ([7], [6]), Conauto ([14]), Bliss ([11], [12]) and Nauty/Traces ([16], [20], [17]) are evaluated in a context that is close to the worst possible case.

We are interested here in solving the GC problem in the most general case, i.e. for graphs for which the vertices and the edges are labeled. Among the existing algorithms, only Bliss, Nauty and Traces achieve a canonical form for partially labeled graphs (for which only the vertices are labeled). They handle the GC problem by finding an *equitable* coloring of vertices (no two adjacent vertices have the same color, and the numbers of vertices in any two color classes differ by at most one)[5], from which they induce an ordering on the set of vertices with the help of search trees (using a backtracking process). A progressive pruning allows for a drastic reduction of the solution space. Once this ordering is obtained, it is straightforwardly possible to enumerate canonically all the elements of a graph.

However, when the edges are labeled, the above-mentioned coloration procedure is not defined, and it is not possible to apply these algorithms other than by carrying out a deep rewriting such as to transform labeled graphs into graphs with unlabeled edges.

1.3 Proposed algorithm (SCOTT)

To overcome the limitation of the existing algorithm that solve the GC problem, we propose the following algorithm, Scott (Structure Canonization using

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

Ordered Tree Translation), that is based on graph rewriting. To our knowledge, such approach to the GC problem has not been proposed yet.

Scott implements a way to transform, in a reversible manner, an arbitrary graph to a tree, a class of graph for which a canonical symbolic encoding can be easily obtained. The order relation necessary to ensure the canonicity of the process is directly connected to the image tree. This encoding will go beyond a simple signature or hashing value: it will ensure the reversibility of the process allowing to get back to the original graph (up to an isomorphism). Hence, in addition to solving the GC problem for arbitrary graphs, it will also offer an effective way of encoding graphs.

Furthermore, this algorithm can be executed in a distributed way, allowing for a horizontal scalability, contrarily to the algorithms of the state of the art: although demonstrating a very good execution time due to their optimized implementation, these algorithms remain sequential.

2 Canonical notation of rooted trees

We detail hereinafter some theoretical properties on graphs, particularly trees, to introduce the hypothesis and mechanisms that we use in the next section to develop our algorithm.

2.1 Definitions and properties

Let Σ be an alphabet, namely a set of symbols, and Σ^* the set of symbolic sequences of any length, ordered by the lexicographic order, denoted $(\Sigma^*, \leq)$.

We define trace functions that associate a label (a sequence of symbols) to any element of a graph $G = (V_G,\ E_G) \in \mathbb{G}$, such as $\sigma^V : V_G \rightarrow \Sigma^*$ and $\sigma^E : E_G \rightarrow \Sigma^*$.

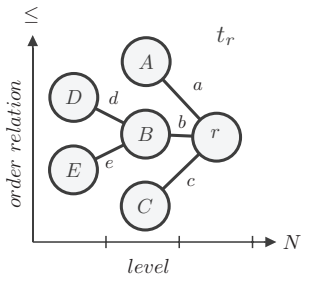
We note $\mathbb{T} \subset \mathbb{G}$, the set of trees. $t = (V_t\ ,\ E_t) \in \mathbb{T}$ iff t verifies the following properties: i) t is fully connected and acyclic, ii) t has exactly $n - 1$ edges, where $n = |V_t|$, iii) For any pairs of vertices, there exists a unique path of minimal length connecting them.

More specifically, we are interested in the rooted planar trees t_ρ for which one of the vertices $\rho \in V_t$ is identified as a root. We can associate with each vertex $\nu \in V_t$ of this tree a level N characterizing its minimal distance (without differentiating the different linkage modalities, i.e. edge label, if any) to ρ, the level 0 designating the leaves that are farthest from the root. Each vertex ν is thus exclusively connected with a single vertex of directly higher level noted $\rho_\nu \in V_t$ and the set of vertices of level directly lower to ν is noted $\Lambda_\nu \subset V_t$ which is *de facto* the set of children vertices of vertex ν.

Each vertex ν is thus itself the root of a subtree that we denote $t_\nu = (V_{t_\nu}, E_{t_\nu})$, with $V_{t_\nu} \subseteq V_t$ and $E_{t_\nu} \subseteq E_t$.

Theorem 1 (Neveu [19]). *If a fully ordered relation is defined on the set of rooted trees, then any rooted tree admits a canonical notation.*

It is proved [19] that a planar representation of a rooted tree admits an unambiguous (canonical) notation in the form of a sequence of words, so that two trees with the same notation are equal. This notation for which we give an example in



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

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

Fig. 1: Encoding of a rooted tree t_r using a sequence of symbols, according to the (edge-extended) *Neveu*'s notation, and by the proposed alternative trace function σ^T (2).

Fig. 1, on the other hand, is canonical only if the representation itself is canonical, and therefore there should necessarily exist an ordering between vertices of the same level (sibling trees). We detail below the systematic construction of this ordering in the framework of labeled trees. We also propose an alternative notation (to the Neveu's one) through a tree-defined trace function σ^T to avoid too much redundancy.

2.2 Ordering on the set of labelled trees

To be able to provide for any tree a canonical notation similar to the one illustrated above, we need to provide an ordering on the set of trees. In other words, for all level N, all the subtrees associated with the vertices of level N needs to be ordered, with respect to both edges and vertices labels.

Proposition 1. *The subtrees t_ν that belong to a level N are lexicographically ordered.*

Proof. We make the assumption that a vertex (respectively an edge) is fully identified by its label, so that a permutation of two vertices (or edges) holding the same label will not modify the tree in question, which is true in a trivial way for an encoding of the same type as the one proposed by Neveu [19], because the trace produced by an in depth or in width traversal of the tree will be identical after such a permutation.

Vertices, as well as edges, can thus be expressed in the form of sequences of symbols, produced by traces function $\sigma^V : V_t \to \Sigma^*$ and $\sigma^E : E_t \to \Sigma^*$. Such

symbolic sequences are ordered by the lexicographic order relation, but what we have to order at a level N is not a set of detached vertices, but subtrees rooted on those vertices, otherwise our ordering would just ignore underneath levels. We thus seek a recursive trace function for rooted (sub)trees $\sigma^T : \mathbb{T} \to \Sigma^*$. Let Π be the concatenation operator taking two arguments: a subset ϵ of a set $\mathscr{E}$ ordered by a total order relation $\leq$, and a concatenation symbol in Σ, as defined by Eq. 1.

$$\Pi : \mathscr{E} \times \Sigma \to \Sigma^*$$

$$\forall \epsilon \subseteq \mathscr{E},\ \forall \cdot \in \Sigma,\ \Pi(\epsilon,\ \cdot) \ = \bigoplus_{e_i \ \in \ \epsilon \ \mid \ e_{i-1} \ \leq \ e_i}^{\cdot} e_i \ = e_1 \cdot \ e_2 \ \cdot \ \ldots \ \cdot \ e_{|\epsilon|} \tag{1}$$

$$\sigma^T : \mathbb{T} \to \Sigma^*$$

$$\sigma^T(t_\nu) = \Pi \left(\bigcup_{\lambda \ \in \ \Lambda_\nu} \left\{ \sigma^T(\lambda) \ : \ \sigma^E(\lambda,\ \nu) \right\},\ \cdot \right) \cdot \ \sigma^V(\nu) \tag{2}$$

At some point vertices are leaves, which means that $\Lambda_\nu = \emptyset$. In this specific case $\sigma^T(t_\nu) \equiv \sigma^V(\nu)$, making those leaves ordered, allowing recursively to transmit this ordering property to the above levels.

We illustrate in Fig. 1 the encoding given by the above σ^T function when applied to the example. For convenience and to be consistent with Neveu's notation we use the comma "**,**" as a concatenation symbol between several trees (instead of the usual dot symbol used in the definition above, Eq. 2).

Proposition 2. *The trace function σ^T gives a canonical representation for each tree $t \in \mathbb{T}$.*

Proof. The trace σ^T being defined, we are able to order the subtrees of any level N, and so to apply *Neveu*'s notation for any rooted tree. But, we can notice that the above definition of σ^T is applicable to each rooted tree on its main root ρ. If two tree traces are equal, then all of its subsequent elements are equal as well, which means these sets of elements can be permuted without any impact on the isomorphism class of the tree. Thus, all members of an isomomorphism class will have the same canonical representation through σ^T. Finally, we determine the identity of the root ρ as the node leading to the minimum trace. A tree $t \in \mathbb{T}$ is thus canonically encoded by its minimum encoding through the trace function σ^T. □

Proposition 3. *A tree $t \in \mathbb{T}$ can be compressed into a single vertex of label $\sigma^T(t)$.*

As our trace function can encode without any loss a tree under the form of a symbol sequence, and since those sequences can be used as vertex labels in a graph, there is no loss of information when compressing a (sub-)tree into a single vertex holding this label, even in the context of an unrestricted graph.

3 Proposition: Scott algorithm

Scott addresses the canonization of unrestricted graphs, namely fully labeled graphs. We do not provide here the formal proof of validity of the algorithm that will be detailed elsewhere. However, the canonical trace that is produced by Scott is empirically assessed and applied in the scope of some benchmarking (see next section), as well as supported by results known from the bibliography.

3.1 Steps summary

Scott relies in three main steps consisting in:

Step. 1 Organize the vertices by levels according to the identification of a root.
Step. 2 Rewrite, in a reversible way, edges inducing cycles in a acyclic way.
Step. 3 Get a canonical encoding of the resulting tree.

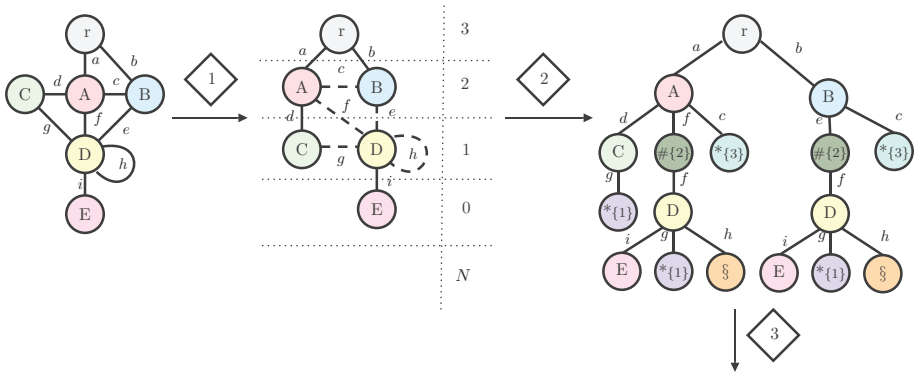


Fig. 2: Scott steps applied to a simple example

We aim to associate for all connected graph a tree representative of its class of isomorphism. The interest of using a reference graph, here a tree, as a canonical representative, lies in its encoding properties. While numerous degrees of freedom (dof) exist in an unrestricted graph to achieve a canonical encoding, much less dof exist in a tree, hence a greater efficiency is expected.

Root selection and levelling (step 1): the first step consists in finding a root $\rho \in V_G$. Its selection is based on a heuristics whose aim is to minimize the size of the trace produced in the last step. The root is thus determined subsequently, among a pool of several candidates selected according to some invariant characteristics (label, degree, cumulative neighbor's degrees, etc.). In the worst case all vertices can be candidates to be the root vertex. Once the root ρ has

been selected, all other vertices ν are organized following their respective minimal distance N, refer to as level N, with ρ. The farthest vertex to ρ is associated to level 0. After that step, each vertex is exclusively connected with vertices of levels $[\![N-1,\ N+1]\!]$.

Cycles rewriting (step 2): this step aims at transforming an unrestricted connected graph into a tree by rewriting all cycles, without any loss of information. A graph rewriting can be expressed by graphs morphisms [23] [22], referred to as productions, that consist in mapping a subgraph identified as a left hand side (LHS) to a right hand side (RHS) subgraph. On a graph whose vertices are ordered by levels, only three configurations of edges lead to cycles, resolved respectively by associated productions :

- p_s: self-bound, an edge from a vertex to itself (e.g edge labelled h on Fig. 2)
- p_c: co-bound, an edge from a vertex to a distinct vertex of the same level (e.g edges labelled c and g on Fig. 2)
- p_i: in-bound, several edges from vertices of the same level to a common descendant (e.g edges labelled f and e on Fig. 2)

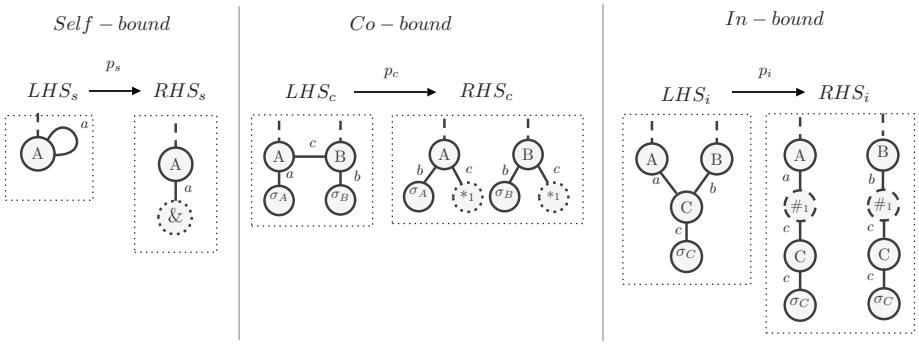


Fig. 3: Graphs morphisms used to rewrite cycles in any graph.

As pictured in figure 3, vertices of inferior level in the productions are trees, which have been compressed into single vertices. This is possible because levels are processed increasingly, so when applying productions at level N, all vertices of level $N-1$ have been already processed, and so are subtree roots. Within the processing of a level, productions are grouped and executed by type: self-bounds, then co-bounds and finally in-bounds. Finally, in the case of co-bound and in-bound productions, additional vertices that are created are given a common "anchor" label used to pair them together. This label is derived from tree traces of the underlying levels (extended Neveu's notation). According to the property stating that two subtrees with the same canonical encoding can be permuted without any impact on the isomorphism class, these labels are sufficient to ensure

that possible ambiguous cases (two identical productions applied to two vertices having the same anchor label) will lead to the same representative tree. To avoid side effects, anchors created at level N need to be computed initially during the processing of level N.

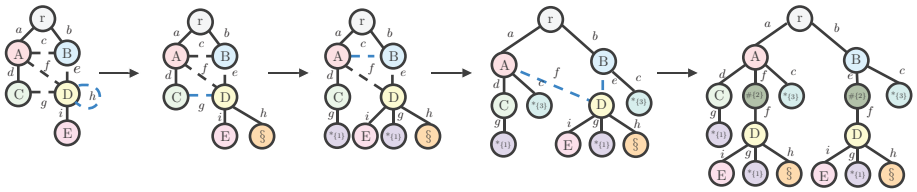


Fig. 4: Sequence of productions applied to a toy example.

We detail in Fig. 4 the actual ordering of productions applied to process the toy example case.

Canonical encoding (step 3): the final step consists in applying the trace function σ^T to the tree produced by the end of the previous steps. This tree is characteristic of the isomorphism class of the input graph, and so is its trace (extended Neveu's notation) as well. Notice that since the anchor labels are symbolic sequences only used as markers, they can be ordered once all anchors have been processed and then be replaced by a unique index to keep the trace short. Note also, that for an indexing application, hashing the trace can be convenient to get small and fixed-size symbolic sequences as indexes.

3.2 Time complexity

The temporal complexity of the Scott algorithm is closely related to the nature of the graphs in input: beyond the size of the graph, the number and type of edges to be rewritten are the main factors to study. In the case of co-bounds, two nodes are simply created, but in the case of in-bounds, it is all the descending nodes from this edge that are duplicated, hence an exponential behavior if such a case is observed at each level. Very regular graphs can add some complexity as well, as virtually each node can be a root candidate. On the other hand, if the input graph is initially close to a tree, corresponding to the best case for Scott, the most expensive operation of the algorithm will be the sorting of the subtrees, resulting in a quasi-linear behavior.

4 Application

We present below assesment metrics obtained on concrete cases, either synthetic or from the application domain of chemistry, and therefore of varying complexity. Calculations are performed, unless otherwise specified, on a dedicated computer with 8 cores (Intel Atom C2750 @ 2.40GHz) and 16 GB of memory.

4.1 Shrunken multipedes graphs

As a first experiment, we evaluate Scott on complex combinatorial graphs produced in the scope of a benchmark [18] dedicate to isomorphism decision. 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 are built from the *"Cai, Fürer and Immerman"* graphs.

It must be stressed that this family of graphs is specifically designed to present non-trivial cases of isomorphisms (or non-isomorphisms). Consequently their features are quite far from the ideal case for Scott, namely a graph that would be nearly acyclic. In particular, the number of edges is large compared to the number of vertices (with an average factor of 7.8), meaning that a lot of productions applications are necessaries.We compute 157 graphs up to 500 vertices, grouped into 79 isomorphism classes, each being correctly assigned a unique canonical trace. We compare the performance of our algorithm against state of the art algorithms that provide a canonical form, namely Traces, Nauty and Bliss. We present in fig. 5 (a) the time necessary to compute the canonical trace of one graph, in function of its vertice count. As there are several order of magnitudes represented on the computing time, a log-axis is used.

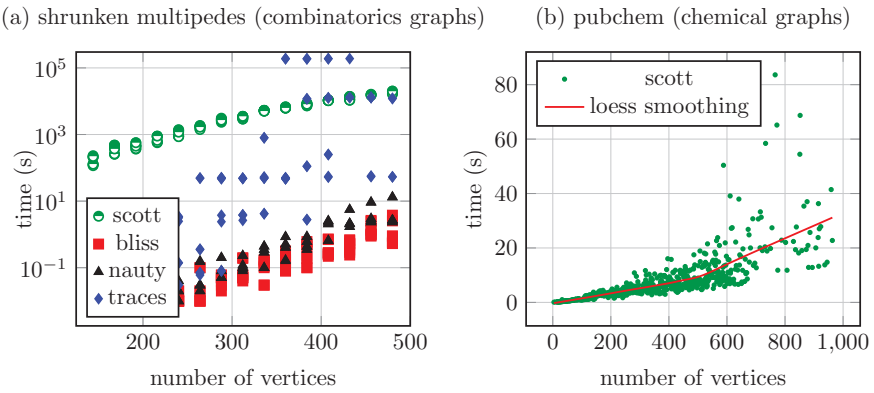


Fig. 5: Elapsed time towards number of vertices for graphs of several origins.

We note that, while the best algorithms on this task demonstrate very good performance, Scott is quite comparable with some of them beyond 380 vertices: it is therefore reasonably usable in practice, even for graphs produced by a combinatorial process, that are very far from conditions generally encountered in practice. We also note that for a given number of vertices, the variability of the processing time is lower for Scott, while it can reach several orders of magnitude for other algorithms.

4.2 Molecular graphs

We finally evaluate Scott on molecular graphs. Molecules are not complex graphs, because the number of vertices rarely exceeds 500, and, above all, are naturally quite close to trees. Indeed, cycles and branches exist in such graphs and need to be processed, but the number of edges is generally close to the number of vertices, and it is in any case of the same order. It is therefore a category of graphs easily addressed by our algorithm.

We extract from *PubChem* a large set of molecules up to 1000 atoms (vertices). We randomly sample the molecules formed by up to 500 vertices, so that we do not have to process tens of millions of molecules. For more than 500 atoms, the molecules become rare, and it is no longer necessary to sample them. By the end of this sampling process, we hold a total of 4733 molecules ranging from 4 to 963 atoms.

We do not distinguish in the parsing *cis* and *trans* bonds, so we expect to observe trace collisions for moelcules which are stereoisomerisms, just as the classic Smiles notation. We would consider stereo-bonds as new edges modalities to obtain the behavior of Isomeric Smiles. We finally obtain 4490 unique traces through Scott execution, the same number of unique Smiles given in the Pubchem request, attesting an identical behavior and none unexpected collision, but without being restricted to whole and valid molecules.

In Fig. 5 b), we plot the median elapsed time obtained according to the number of vertices considered. A loess smoothing (Locally Weighted Least Squares Regression) is thus achieved that shows a piecewise linear characteristic for this type of graphs.

5 Discussion and Conclusion

Considerable work remains to be done at this stage of the implementation to optimize our algorithm, which is not yet as fast with the best algorithms of the state of the art. However, the proposed solution remains the only one to deal natively with colored edges, and shows good possibilities for parallelization. As it stands, our algorithm reaches, on some complex cases, the same level of performance than state of the art approaches such as *traces* beyond a certain number of vertices. Strikingly, our algorithm shows a very small variance of the elapsed time compared to other algorithms. This is probably because our algorithm does not integrate yet any optimized pruning strategy, which is the case for most of the state of the art algorithms.

If the temporal complexity remains problematic for synthetic graphs resulting from combinatorial methods, with an exponential execution time compared to the number of vertices (just like the state of the art algorithms), it is much more efficient on simpler graphs having a number of neighboring edges of the same order of magnitude than the number of vertices. For such category of graphs, we observe that the elapsed processing time for our algorithm follows a quasi linear shape, which makes it quite effective in practice.

The main objective that is addressed by our algorithm is to formally identify using a sequence of symbols, that is unique within an isomorphism class, (sub)graphs of reasonable size, ensuring that any subsequent comparison between subgraphs is trivial. Beyond adopting an approach based on graph morphisms, rather than a backtracking principle, the main difference from the state of the art solutions is that it can handle any kind of graphs since it is able to natively process colored edges as well as colored vertices, which is essential for addressing some application domain such as organic chemistry.

Perspectives: Backtracking-based algorithms, which represent the vast majority of the state of the art approaches on this problem, have over the years been able to develop heuristics that can progressively prune the space of possible solutions, greatly reducing their algorithmic complexity. Our implementation is young and does not develop any similar heuristics, but it is very likely that such heuristics exists. In particular, one can easily prune the number and the diversity of fast invariants to evaluate, which allows to limit the potential roots to consider (it is even possible to eliminate some of these roots even before the complete computation of their associated tree). In the same way, it is likely that an implementation of the algorithm with a more effective programming language (compiled), like those used for implementing the state of the art algorithms (C/C++) can bring a significant performance gain compared to the Python that we are using at this stage.

The families of graphs Scott can be adressed to is also subject to further works. In particular, each production used in the tree transformation can be derived into several productions defined on directed graphs. Therefore, the algorithm could be extended to directed graphs, but this point requires additional experimentations.

Finally, although not reported here, the formal proof of the validity of the algorithm has been obtained to support the main claims we have stated (stop condition, completeness, adequation, reversibility, invariance to isomorphism). These claims are perfectly empirically supported by our experimentation.

References

1. Vikraman Arvind and Piyush P Kurur. Graph isomorphism is in spp. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 743–750. IEEE, 2002.
2. László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697. ACM, 2016.
3. László Babai and Eugene M Luks. Canonical labeling of graphs. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 171–183. ACM, 1983.
4. Nicolas Bloyet, Pierre-François Marteau, and Emmanuel Frénod. Étude lexicographique de sous-graphes pour l'élaboration de modèles structures à activité–cas de la chimie organique. In *Extraction et Gestion des Connaissances: Actes de la conférence EGC'2019*, volume 79, page 3. BoD-Books on Demand, 2019.

5. Kellog S Booth and Charles J Colbourn. *Problems polynomially equivalent to graph isomorphism*. Computer Science Department, Univ., 1979.
6. Paolo Codenotti, Hadi Katebi, Karem A Sakallah, and Igor L Markov. Conflict analysis and branching heuristics in the search for graph automorphisms. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 907–914. IEEE, 2013.
7. Paul T Darga, Karem A Sakallah, and Igor L Markov. Faster symmetry discovery using sparsity of symmetries. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pages 149–154. IEEE, 2008.
8. Yuri Gurevich and Saharon Shelah. On finite rigid structures. *The Journal of Symbolic Logic*, 61(2):549–562, 1996.
9. Stephen R Heller and Alan D McNaught. The iupac international chemical identifier (inchi). *Chemistry International*, 31(1):7, 2009.
10. John E Hopcroft and Jin-Kue Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 172–184. ACM, 1974.
11. Tommi Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 135–149. SIAM, 2007.
12. Tommi Junttila and Petteri Kaski. Conflict propagation and component recursion for canonical labeling. In *International Conference on Theory and Practice of Algorithms in (Computer) Systems*, pages 151–162. Springer, 2011.
13. Johannes Kobler, Uwe Schöning, and Jacobo Torán. *The graph isomorphism problem: its structural complexity*. Springer Science & Business Media, 2012.
14. José Luis López-Presa, Antonio Fernández Anta, and Luis Núñez Chiroque. Conauto-2.0: Fast isomorphism testing and automorphism group computation. *arXiv preprint arXiv:1108.1060*, 2011.
15. Eugene M Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of computer and system sciences*, 25(1):42–65, 1982.
16. Brendan D McKay et al. Practical graph isomorphism. 1981.
17. Brendan D McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014.
18. Daniel Neuen and Pascal Schweitzer. Benchmark graphs for practical graph isomorphism. *arXiv preprint arXiv:1705.03686*, 2017.
19. Jacques Neveu. Arbres et processus de galton-watson. In *Annales de l'IHP Probabilités et statistiques*, volume 22, pages 199–207, 1986.
20. Adolfo Piperno. Search space contraction in canonical labeling of graphs. *arXiv preprint arXiv:0804.4881*, 2008.
21. Uwe Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988.
22. Pedro Pablo Perez Velasco. *Matrix Graph Grammars*. 2008.
23. Pedro Pablo Perez Velasco and Juan de Lara. Matrix Approach to Graph Transformation: Matching and Sequences. *Lecture Notes in Computer Science*, 4178:122, 2006.
24. David Weininger. 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. Viktor N Zemlyachenko, Nickolay M Korneenko, and Regina I Tyshkevich. Graph isomorphism problem. *Journal of Soviet Mathematics*, 29(4):1426–1481, 1985.