

A Method for Representing Graphs as Rooted Trees for Graph Canonization

SCOTT : Structure Canonisation using Ordered-Tree Translation

Nicolas Bloyet^{1, 2, 3} Pierre-François Marteau¹ Emmanuel Frénod^{2, 3}

¹IRISA - Université Bretagne Sud - Vannes, France

²LMBA - Université Bretagne Sud - Vannes, France

³See-d - Parc d'Innovation Bretagne Sud - Vannes, France

Complex Networks 2019, Lisbon, December 2019

Outline

Outline

We present SCOTT, a method to handle **Graph Canonization** (hence Graph Isomorphism) on **fully labeled graphs (edges and vertices)**.

The method produces :

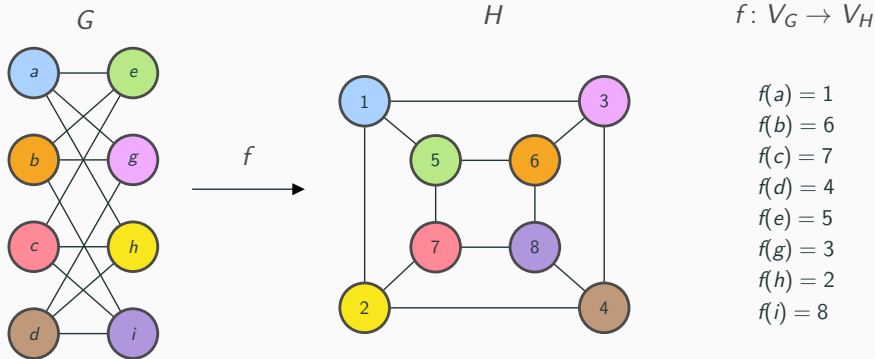
- a string representation of the input graph (trace), unique for an isomorphism class, which can be used to identify any (sub)structure (hash)
- a canonical adjacency matrix derived from that trace, which can be used to standardize (up to an isomorphism) any graph computing

We provide an open-source Python implementation.

Graph Isomorphism

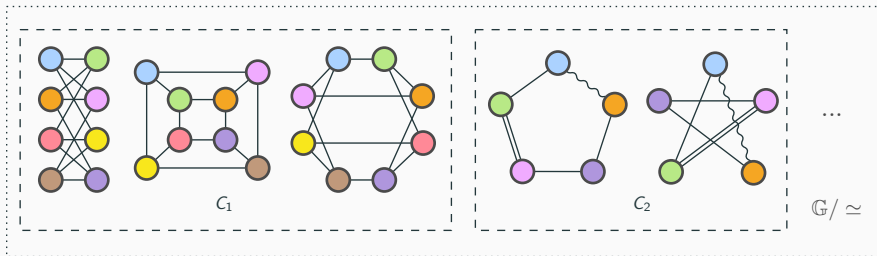
Graph Isomorphism problem

Let $G \in \mathbb{G}$ and $H \in \mathbb{G}$ be two graphs. They are said to be isomorphic if there exists a bijection between their respective vertices sets preserving edges.



Isomorphism class

Isomorphism is an equivalence relation, and so partitions the ensemble of graphs into equivalence classes, where all graphs belonging to a class represent the same structure.



In many graph-related tasks and applications, we want to consider graphs belonging to the **same isomorphism class as equals**.

Problem Complexity

While determining if two graphs are isomorphic seems trivial for small graphs, it is actually a problem which remains unresolved in polynomial time (very costly).

In some applications where we have to make a lot of isomorphism testing, we would prefer a **re-usable method**.

Graph Canonization

Graph Canonization Problem

The graph canonization is a related problem, consisting in finding for a graph a canonical representant, usually a graph, unique for its isomorphism class.

Two graphs are isomorphic if and only if they have the same canonical representant.

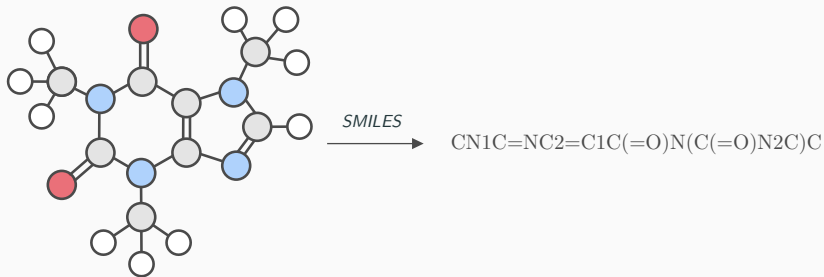
$$G \simeq H \iff \text{Canon}(G) = \text{Canon}(H)$$

This problem is **at least as difficult**¹ as graph isomorphism, as it answers to it explicitly, but provides a **re-usable result**.

¹and actually more difficult in many cases

Graph Canonization Problem

For example, SMILES notation [Weininger, 1988] maps any molecular graph to a canonical string encoding.



Once the canonical representant is computed, subsequent tasks like comparisons are trivial.

State of the Art

State of the Art

Isomorphism testing

- conauto [López-Presa et al., 2011]
- saucy [Darga et al., 2008]

Canonization

- nauty [McKay et al., 1981, McKay and Piperno, 2014]
- bliss [Junttila and Kaski, 2007]
- traces [Piperno, 2008, McKay and Piperno, 2014]

None of them is able to **natively** deal with **labeled edges** otherwise than rewriting the graph in an edge-unlabeled way, due to their utilization of equitable vertex-coloration, undefined for edge-labeled graphs.

Scott

Key Idea

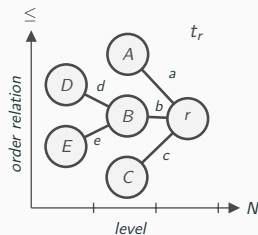
We propose here an algorithm based on graph rewriting instead of equitable coloration, to handle natively edges-labeled graphs. Scott execution follows three main steps, illustrated below.

- Levelling of vertices, according to an elected root
- Re-writing of cycles without information loss
- Exploitation of the resulting tree (as trace or as matrix)

Tree encoding

It is known [Neveu, 1986] that we can formally encode a tree as a string of symbols.

We propose a canonic notation $\sigma^T : \mathbb{T} \rightarrow \Sigma^*$, encoding any tree into a string. We can thus compress any tree t into a unique vertex of label $\sigma^T(t)$.



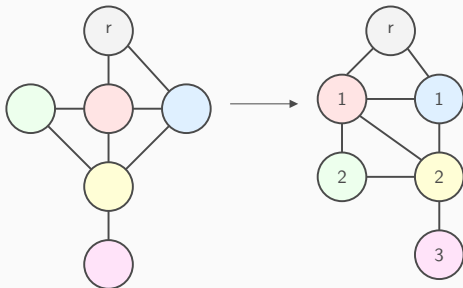
$$\text{Neveu}(t_r) = \{ r, A : a, B : b, BD : d, BE : e, C : c \}$$

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

Vertices levelling

The first step consists in ordering each vertex in levels, according to their minimum distance with a vertex identified as **root**.

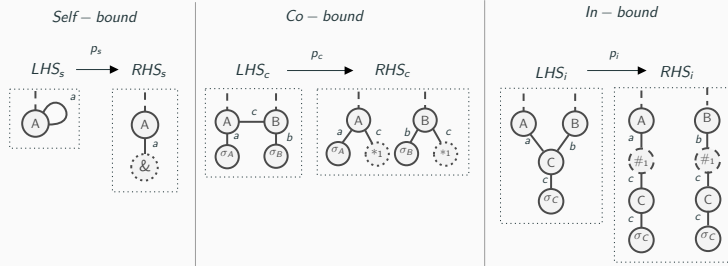
In the best case, the root identity is obvious (label, degree, etc.), otherwise it is determined *a posteriori*.



Cycle rewriting

We want to remove any form of cycle in the levelled graph, without any loss of information.

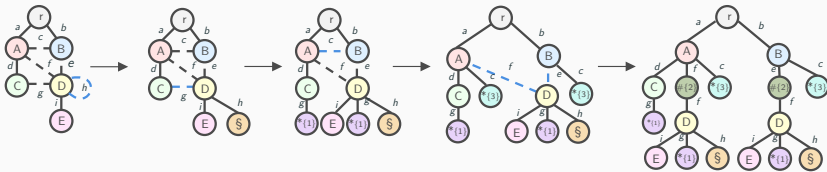
Tree \rightarrow Vertex compression : if N first levels are computed, then any cycle encountered at level $N + 1$ belongs to one of the three following cases.



Cycle rewriting

To ensure the tree we are producing is representative of the isomorphism class, rewritings must be computed following a rigorous ordering, making the process deterministic for any graph encoding :

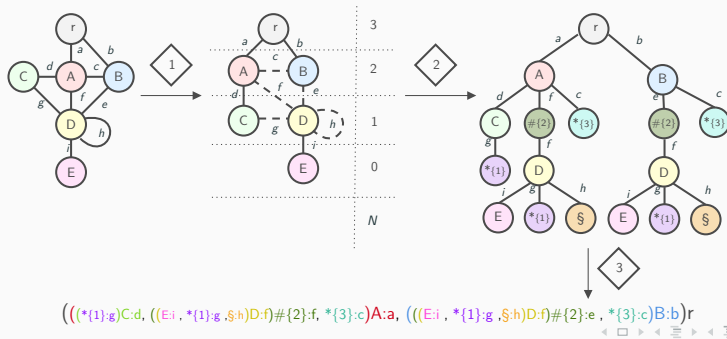
- by level
- by bound type
- by lexicographic order of the tree implied in the rewriting



Encoding by trace

We can finally encode the tree representative of the input graph's isomorphism class.

This trace can be used as an identifier, or we can use it to induce an order on the input graph's vertex set, to get a canonical adjacency matrix.



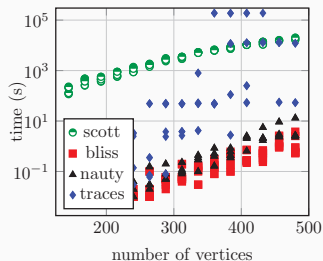
Implementation

Python library

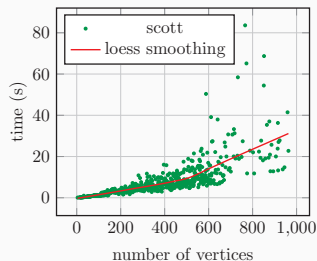
We provide a Python library² (MIT licence) implementing SCOTT.

Time complexity is exponential in worst cases³, but much more reasonable on simple graphs like molecules.

(a) shrunken multipedes (combinatorics graphs)



(b) pubchem (chemical graphs)



²github.com/theplatypus/scott

³a lot of vertices may be created

Conclusion

SCOTT: an algorithm that provides a canonical representant to **any fully labeled graph**

- canonical string representation
- canonical adjacency matrix

Python implementation available at github.com/theplatypus/scott.

Can be used for :




- graph indexing and retrieval
- graph fragmentation and embedding
- classification/regression for graph data in Graph Neural Networks

Thanks for your attention

`github.com/theplatypus/scott`

References





References i

-  Babai, L. (2016).
Graph isomorphism in quasipolynomial time.
In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697. ACM.
-  Babai, L. and Luks, E. M. (1983).
Canonical labeling of graphs.
In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 171–183. ACM.
-  Darga, P. T., Sakallah, K. A., and Markov, I. L. (2008).
Faster symmetry discovery using sparsity of symmetries.
In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pages 149–154. IEEE.





References ii

-  Harary, F. and Palmer, E. M. (2014).
Graphical enumeration.
Elsevier.
-  Junttila, T. and Kaski, P. (2007).
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.
-  López-Presa, J. L., Anta, A. F., and Chiroque, L. N. (2011).
Conauto-2.0: Fast isomorphism testing and automorphism group computation.
arXiv preprint arXiv:1108.1060.
-  McKay, B. D. et al. (1981).
Practical graph isomorphism.

References iii

-  McKay, B. D. and Piperno, A. (2014).
Practical graph isomorphism, ii.
Journal of Symbolic Computation, 60:94–112.
-  Neuen, D. and Schweitzer, P. (2017).
Benchmark graphs for practical graph isomorphism.
arXiv preprint arXiv:1705.03686.
-  Neveu, J. (1986).
Arbres et processus de galton-watson.
In *Annales de l'IHP Probabilités et statistiques*, volume 22, pages 199–207.
-  Olsen, G. (1990).
Gary olsen's interpretation of the “newick's 8: 45” tree format standard.
URL <http://evolution.genetics.washington.edu/phylip/newick doc. html>.

References iv

-  Piperno, A. (2008).
Search space contraction in canonical labeling of graphs.
arXiv preprint arXiv:0804.4881.
-  Velasco, P. P. P. (2008).
Matrix Graph Grammars.
-  Velasco, P. P. P. and de Lara, J. (2006).
Matrix Approach to Graph Transformation: Matching and Sequences.
Lecture Notes in Computer Science, 4178:122.
-  Weininger, D. (1988).
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.