Potential NP-completeness of an approximate tree compression problem

Clément Legrand-Duchesne

Univ Rennes, F-35000 Rennes, France

July 26, 2018

Abstract

A tree is said to be self-nested if all his subtrees of equal height are isomorphic. It has been shown that self-nested trees are a natural and relevant model for the structure of a plant. We are interested in approximating trees by self-nested trees. More precisely, given a tree T, we try to find the self-nested tree S that minimizes the edit distance between S and T. In this paper, we present our attempts to prove that this problem is NP-complete, as well as polynomial algorithms when T is of height 2.

This internship has been carried out from May 28, 2018 to July 27, 2018, in the MOSAIC team (MOrphogenesis Simulation and Analysis In siliCo) of the RDP laboratory (*Reproduction et Développement des Plantes*) of the ENS de Lyon, under the supervision of Christophe Godin.

Keywords: Graph Theory, NP-completeness, self-nested trees, edit distance, tree compression.

Contents

Introduction

The MOSAIC team develops mathematical models and tools to study the morphogenesis and growth of animals and plants. In order to do that, they design mathematical models and data structures able to describe efficiently and accurately the shapes and properties of the plants or animals.

For example, it is natural to represent the branching structure of a plant by a tree. In those trees representing plants, we can observe that some ramifications patterns tend to repeat themselves in many places. The MOSAIC team has thus designed a theoretical model of the structure of a plant based on this idea of repeated patterns: self-nested trees, trees in which all the subtrees of equal height are isomorphic.

Some biological or physical quantities (such as the flow of sap in the plant for example) can be computed on these trees. Depending on the species, the size of the trees representing the plants can be tremendous and lead to unreasonable computing time. They also developed a compression algorithm for trees, based on the same idea of patterns within the tree, by eliminating this redundancy. Moreover, by eliminating this redundancy, the different patterns in the tree and their organization are highlighted. For this reason, this compression helps to understand the structure of the plant and its developement.

Those compression algorithms return DAGs (Directed Acyclic Graph) and it is possible to show that the tree having the best compression factor are the self-nested ones. Therefore, it is beneficial to approximate trees by self-nested ones, in order to have an approximate yet compact representation of them. There are several ways to do this and the MOSAIC team has developed different algorithms doing this. However, the approximation given by those algorithms are not the best (either because these algorithms are heuristics or because some additional conditions are imposed on the self nested trees returned by the algorithm). In fact no efficient algorithm has been found to compute the nearest self-nested tree to a tree.

Therefore, proving that there exists no polynomial algorithm solving this problem would justify the use of the non optimal algorithms previously developed by the team. More precisely, the goal of this internship is to prove that this problem is NP-complete. NP-complete problems are the hardest problems of the NP class and under the assumption that the P and NP complexity classes are different, this would prove there exists no polynomial algorithm able to find the nearest self-nested tree to a tree in general.

The state of the art, along with definitions and notations will be presented and introduced in a first section. We will then give further details on the NP-complete theory and present the results found during the intership. Finally, we will explain how this results consist in an improvement of the state of the art and the consequences and applications.

1 Context

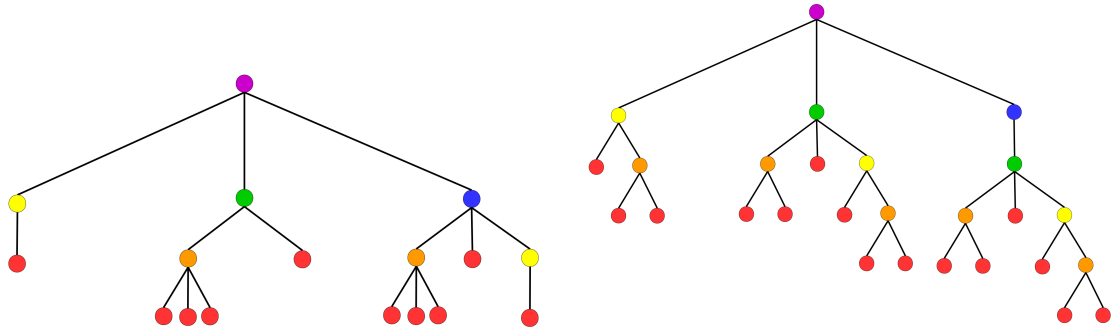
1.1 Self-nested trees and definitions

In this report, we will only consider rooted unordered trees, denoted by $\mathcal{T}$. Unordered trees are trees for which the order among the children of a same node is not significant. Let v be a node of T, the complete subtree of T rooted in v will be denoted by $T[v]$.

Two trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ are said to be isomorphic if there exists a bijection f from V_1 to V_2 such that for each pair of nodes (u, v) in V_1, $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$.

Let us consider the equivalence relation $\equiv$ on $\mathcal{T}$ defined by $T_1 \equiv T_2$ if and only if T_1 and T_2 are isomorphic. More generally, we will say that two nodes v_1 and v_2 of T , are equivalent if the subtrees $T[v_1]$ and $T[v_2]$ are isomorphic. We will denote the equivalence class of v by $C(v)$.

A self-nested tree is a tree whose subtrees of identical height are isomorphic to one another. An exemple of tree that is not self-nested is given Figure 1a, an example of self-nested tree is given Figure 1b. We will denote $\mathcal{S}$ the set of the self-nested trees.



(a) Example of regular tree

(b) Example of self-nested tree

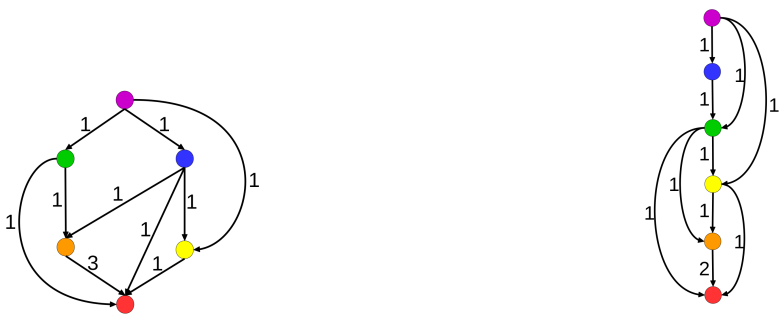
Figure 1 – Difference between regular trees and self nested trees

1.2 DAG compression

For $T = (V, E) \in \mathcal{T}$, let $\mathcal{Q}(T) = (V_\mathcal{Q}, T_\mathcal{Q})$ be the quotient graph of T by the equivalence relation $\equiv$. In concrete terms, $V_\mathcal{Q} = \{C(v) | v \in V\}$ is the quotient set of V by $\equiv$ and $E_\mathcal{Q} = \{(C(u), C(v)) | (u, v) \in T\}$. Let δ be the weighting function on $\mathcal{Q}(T)$ defined by

$$\delta(C(u), C(v)) = \#\{v' \in Child(u) | C(v) = C(v')\}$$

The subfigure 2a the quotient graph corresponding to the tree represented Subfigure 1a. The equivalent nodes are represented in the same color.



(a) DAG compression of the tree Subfigure 1a

(b) DAG compression of the tree Subfigure 1b

Figure 2 – Example of DAG compression

Let $T \in \mathcal{T}$, C. Godin and P. Ferraro showed in [1] that $\mathcal{Q}(T)$ is a weighted directed acyclic graph (*DAG*) and that for each *DAG* Q with a weighting function δ, there exists a single tree $T \in \mathcal{T}$ such as $\mathcal{Q}(T) = Q$.

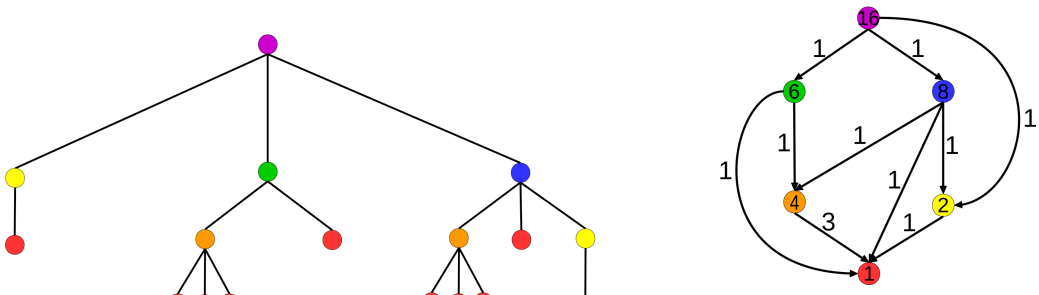
The trees whose quotient graph is a linear *DAG* (*DAG* for which there exists a hamiltonian path) are exactly the self-nested trees. Since the nodes of equal height in a self-nested tree are isomorphic to one anoter, there is a single vertex in the *DAG* for each height: the tree Subfigure 2b is the quotient graph of the self-nested tree represented Subfigure 1b. As a result, the number of vertices in the *DAG* is equal to the height of the initial tree.

1.3 Avantages of self-nested trees

Self-nested trees have a linear quotient graph, as a result they are cheaper to store, but there are other advantages...

Let $T \in \mathcal{T}$ and f be a recursive function ($f(u)$ depends only on the values $(f(u_i))_{i \in I}$ where $(u_i)_{i \in I}$ are the children of u). For example, f can be the height or the number of vertices of the subtree rooted in u. Since f is recursive, f takes the same value on all the subtrees of T isomorphic to one another, and computing f on T leads to make the same computation several times. Thus, computing $f(T)$ directly on the DAG corresponding to T is more efficient than computing it on T itself.

For example on Figure 3 we can compute the number of nodes directly on the DAG: the number of nodes under a leaf (red node) is equal to one, the number of nodes under the orange nodes in T is equal to three times the number of nodes under a leaf plus one, that is four. We can carry one the computation, at the end the number of recursive calls will be equal to the number of nodes in the DAG instead of the number of nodes in the tree (as it would have been if we had computed f on the tree). Thus, the computation of recursive functions is particularly efficient on self-nested trees because of the small size of their DAG.



(a) Example of tree

(b) Computation of the number of nodes directly on its DAG

Figure 3 – Example of computation on the DAG

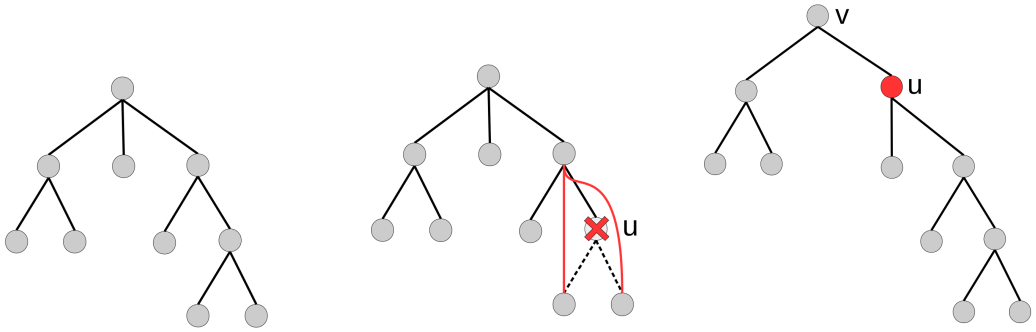
Self-nested trees are a particularly relevant model of the structure of a plant and are therefore usefull for the study of their growth. Indeed, meristems are the tissues responsible for the growth and some studies support the idea that the meristems of a plant go through successive developement stages during their lives [2]. Furthermore, the transitions from one stage to another occur in the same way for all the meristems of a same plant and are specific to the species. We can indeed observe that the way plants grow, the branching structure of their stems and branches seem to follow some regular pattern repeated in several places in the plant. Thus,

representing the structure of the plant by a self-nested tree whose nodes are the meristems and edges are the stems or branches of the plant is natural and appropriate.

Unfortunately, a real plant has few chances to have exactly the structure of a self-nested tree. Indeed, environmental conditions (such as light and nutrition for example) also impact the growth of the plant. This is one of the reasons why we are interested in approximating trees by self-nested ones.

1.4 Edit distance

Let us consider the two following edit operations: the deletion and addition of a node. Deleting a node u in a tree means making the children of u children of the father of u before removing u from the tree (Figure 4b). Adding a node is the complementary operation, thus adding a node u under v means making a subset of the children of v become children of u before placing u under v (Figure 4c).



(a) Initial tree (b) Deleting a node (c) Adding a node

Figure 4 – Example of edit operations

Let $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ be two trees.

An edit sequence is a succession of edit operations. We can define the cost of an edit sequence by the number of edit operations that were done.

An edit mapping between T_1 and T_2 is a function $\Phi : V_1' \to V_2'$ where $V_1' \subseteq V_1$ and $V_2' \subseteq V_2$, such as Φ is a bijection and preserves ancestrality among the nodes (in other words, $u \in V_1'$ is an ancestor of $v \in V_1'$ if and only if $\Phi(u)$ is an ancestor of $\Phi(v)$).

Given an edit mapping Φ between T_1 and T_2, there exists a natural edit sequence leading from T_1 to T_2: every node belonging to $V_1 \setminus V_1'$ is deleted and every node in $V_2 \setminus V'2$ is added. Thus, we can define the cost of an edit mapping Φ as the number of nodes that were added or deleted: $\#(V_1 \setminus V_1') + \#(V_2 \setminus V_2')$.

More precisely, K.Zhang showed in [3] that for every edit mapping between T_1 and T_2 there exists a edit sequence of equal cost. He also showed that for every edit sequence of k operations leading from T_1 to T_2, there exists an edit mapping of cost less than k.

It is also possible to define the edit distance between the trees T_1 and T_2 as the minimal cost of an edit mapping between the two trees (the properties of symmetry, identity of indescernibles and triangle inequality are satisfied). We will notice that the minimal cost of an edit mapping between T_1 and T_2 is also the minimal cost of an edit sequence leading from T_1 to T_2

(result from the preceding properties shown in [3]).

In [4] K. Zhang, R. Statman and D. Shahsha show that the problem of computing the edit distance between two trees is NP-complete. Consequently, it is convenient to add constraints to the edit mapping to build a new edit distance easier to compute.

The article [3] presents a sufficient and not very constraining condition on the edit mapping along with a polynomial algorithm to compute the new distance. The condition on the edit mapping is the following: for all $u, v, w \in V_1'$, the least common ancester of u and v is a proper ancester of w if and only if the least common ancestor of $\Phi(u)$ and $\Phi(v)$ is a proper ancestor of $\Phi(w)$. In other words, Φ preserves the sibling relation among the nodes: a node can be deleted only if all his descendant are deleted as well or if all his siblings have been deleted; during the addition of a new node u under v, all the children of v have to become children of v or none of them.

In this article, we will use this edit distance with its new constraint.

1.5 Approximation by self-nested trees

As we explained before, a real plant has few chances to have an exact self-nested structure. Therefore, approximating the trees of the structure of different plants by self-nested trees could allow us to find the theorical growth model of each species and thereby lead to a better understanding of the evolution of the meristems.

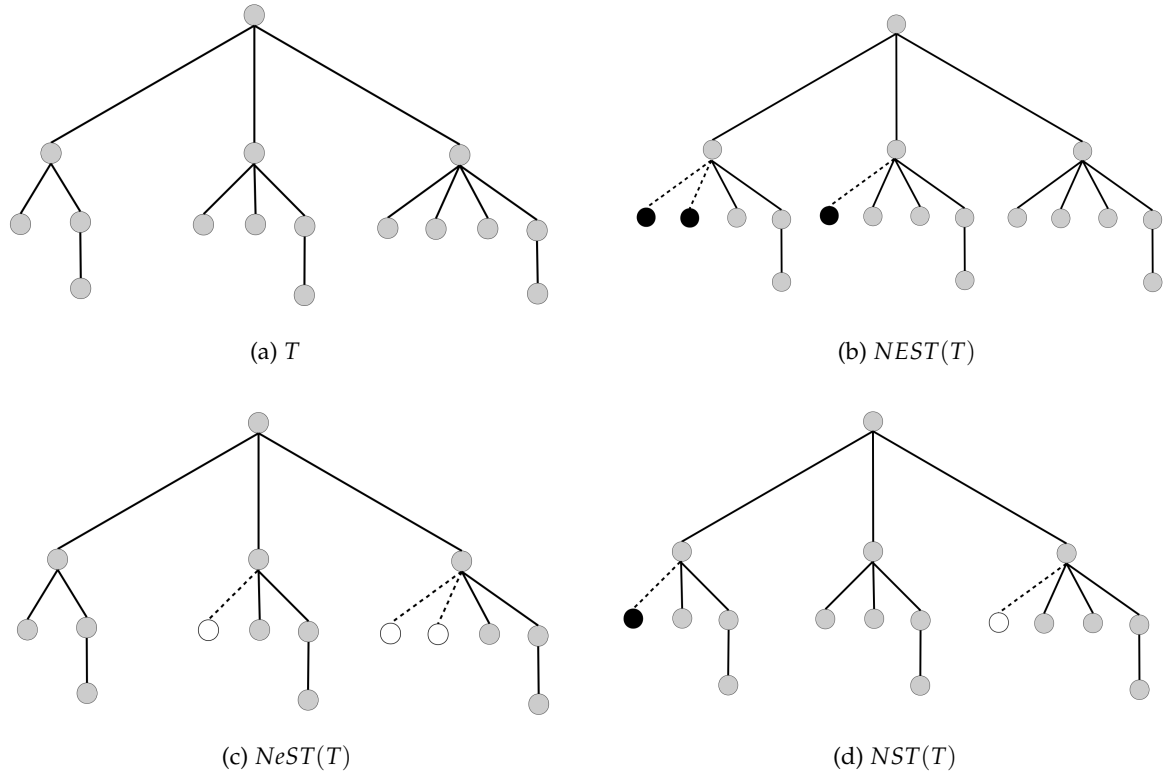
If the goal is to compute the value taken by a recursive function f on a tree T, approximating T by a self-nested tree S allow us to compute very effectively $f(S)$, which can, under certain conditions, be an approximative value of $f(T)$.

Several meanings can be given to the term "approximating a tree by a self-nested one". For example, C. Godin and P. Ferraro have proposed a polynomial algorithm [1] to compute the NEST (Nearest Embedding Self-nested Tree) of a tree T. The NEST of T is the self-nested tree S that minimize the edit distance between S and T and that embeds T. It can also be described as the minimal self-nested tree embedding T and it can be obtained only by doing adding operations on T. For example, the NEST of T (Figure 5a) is given in Figure 5b. The black nodes are the added ones. T is at distance 3 from its NEST.

R. Azais then presented an algorithm [5] computing in polynomial time the NeST (Nearest embedded Self-nested Tree) of a tree T. The NeST of T is the self-nested tree S that minimize the edit distance between T and S and is embedded in T. It is the maximal self-nested tree embedded in T and it can be obtained only by deleting nodes from T. For example, the NeST of T (Figure 5a) is given in Figure 5c. The white nodes are the ones that are deleted. T is at distance 3 from its NeST.

The NST (Nearest Self-nested Tree) of a tree T is the self-nested tree S that minimize the edit distance between T and S (authorizing this time both adding and deleting operations) [1]. For example, the NST of T (Figure 5a) is given in Figure 5d. The white node is the one that is deleted and the black node is the one that is added. T is at distance 2 from its NST.

No polynomial algorithm able to compute the NST of a tree has been found yet. Thus, the goal of the internship is to prove that there exists no such algorithm (more precisely that the NST problem is NP-complete). This result would legitimate the use of the NeST and NEST algorithms and of heuristics.



(a) T (b) $NEST(T)$

(c) $NeST(T)$ (d) $NST(T)$

Figure 5 – Different approximations of T by self-nested trees

2 Contribution

We assume that the reader is familiar with the complexity theory. In this article, we will denote the set of instances by Ω and the set of positive instances by S. See the appendix for further reminder on the compexity theory.

2.1 Proof of the belonging to the NP class

We aim to prove that the NST decision problem is NP-complete. It clearly belongs to the NP class. Indeed, for all tree $T = (V, E)$, for all integer k, given a potential solution S, it is possible to ascertain whether or not S is self-nested, and to compute the distance between S and T also in polynomial time.

2.2 Intuition guiding the researches

To help us build the reduction, a few conceptual remarks can be made, in order to guide the research. First of all, the reduction has no obligation to be surjective, as a matter of fact, it is quite often not surjective. The image of the reduction has to be small enough to know the general form of the solution of each instance in order to be able to prove the equivalence $\omega_A \in S_A \Leftrightarrow \omega_b = f(\omega_a) \in S_B$ (where A is the initial NP-complete problem and B is the problem we consider). Yet, if the image of Ω_A by the reduction has to be big enough so that we are enable to find a polynomial algorithm solving the problem on the set of images of the

reduction. Indeed, finding such an algorithm and a reduction would mean we would have solved the open question of the equality of the P and NP classes, which is unreasonable.

Furthermore, we can observe that generally, the construction of the image instance seems to be combination of two different things: a part that reflects only the number of variables in the instance of the first problem and a part reflecting the links and constraints that they have with one another. For example, in the proof of the reduction from 3-SAT to INDEP-SET [6], the first part consists in the triangles corresponding to each clause of the initial instance, and the second to the edges linking the nodes such as one is labeled with the negation of the other. In the proof of the reduction from VERTEX-COVER to HAMILTONIAN-CYCLE [6], the first part consists in the 12 nodes widgets corresponding to each edge of the initial graph, and the second part consists in the way they are linked to one another to form the image graph.

We can also observe that in the first part, a piece of the image instance corresponds to each variable of the initial instance. Moreover, this piece of the image instance is very restrictive and can be used in a solution only in a few (2 or 3 most of the time) different ways. For example in the proof of the reduction from VERTEX-COVER to HAMILTONIAN-CYCLE [6], the widgets can either be visited in one pass or in two.

Furthermore the way these pieces are used is completely independant until the second part corresponding to the constraints between the variables is added.

2.3 Trees of height 2 without variation of the height

The edit distance with constraint of K.Zhang being slightly difficult to manipulate, we will use a far more restrictive variant in this section: only leaves (and not internal nodes) can be added or deleted. More precisely, if u is an ancestor of v in T and $v \in V_1'$ then $u \in V_1'$.

In this section we present a polynomial algorithm computing the nearest self-nested tree of height 2 to a tree of height 2.

2.4 Trees of height 2 with variation of the height

In this section, we present a polynomial algorithm computing the nearest self-nested tree to a tree of height 2.

2.5 Family of bricks

In this section, we present a family of trees that could be used as the first part of the reduction to represent the variables of the initial instance.

3 Validation and consequences

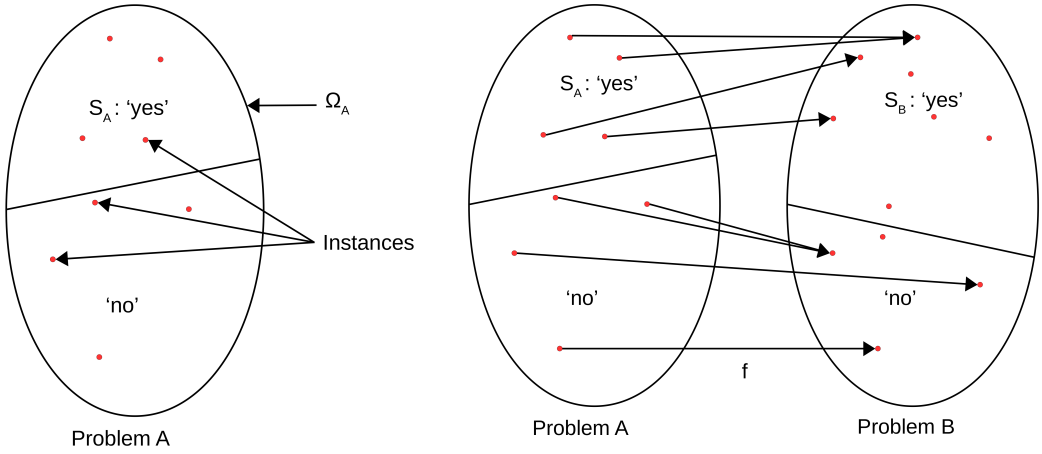
Conclusion

APPENDIX

Short reminder on the NP-completeness theory

A decision problem is a couple (Ω, S). Ω is described in the section INPUT and is a set of words called instances of the problem. S is described in the section OUTPUT and is a language included in Ω. S corresponds to the set of instances for which the answer to the problem is "yes" (figure 6a).

For example, the optimisation problem NST can be rephrased as the following decision problem:
INPUT: A tree T, an integer $k \in \mathbb{N}$
OUTPUT: Yes if and only if there exists a self-nested tree S such as the edit distancee between T ans S is less than k.
Here, each couple (T, k) is an instance, and Ω is the set of all the instances.



(a) Algorithmic problem

(b) Example of reduction

The complexity class P is the class of problems for which there exists a deterministic Turing machine deciding S in a time polynomial to the size of the instance. The complexity class NP consists of the problems for which S is accepted by a non-deterministic Turing machine in polynomial time.

A reduction from a problem A to a problem B is a function $f : \Omega_A \to \Omega_B$ such as for all instance ω_A of A, $\omega_A \in S_A \Leftrightarrow f(\omega_a) \in S_B$ (figure 6b). As a result, if there exists a polynomial reduction between A and B then B is "harder" than A. Indeed, if S_B is accepted (respectively decided) by a Turing machine in polynomial time, to accept (respectively decide) S_A in polynomial time, all there is to do is to apply the reduction to the instance of A and return the result computed by the Turing machine of the problem B.

An algorithmic problem is called NP-hard if there exists a polynomial reduction from each problem of the NP class to this one. The problems that are both NP-hard and belong to the NP class are said to be NP-complete.

Acknowledgments

I would like to thank Christophe Godin for the time he spent supervising and helping me, as well as him and Romain Azais for the many discussions we had together about NP-completeness and Self-nestd trees.

References

[1] C. Godin and P. Ferraro, "Quantifying the degree of self-nestedness of trees: Application to the structural analysis of plants," 2010.

[2] P. de Reffye, C.Edelin, J. Françon, M.Jaeger, and C. Puech, "Plant models faithful to botanical structure and development," 1988.

[3] K. Zhang, "A constrained edit distance between unordered labeled trees," 1996.

[4] K. Zhang, R. Statman, and D.Shasha, "On the editing distance between unorderd labeled trees," 1992.

[5] R. Azais, "Nearest embedded and embedding self-nested trees," 2017.

[6] O. Bournez, "Quelques problèmes np-complets." `http://www.enseignement.polytechnique.fr/informatique/INF423/uploads/Main/chap12-good.pdf`.