Potential NP-completeness of an approximate tree compression problem

Clément Legrand-Duchesne

Univ Rennes, F-35000 Rennes, France

August 23, 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

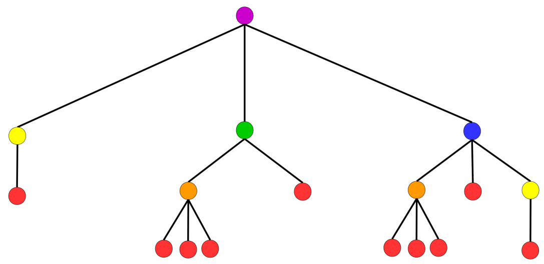
1.1 Self-nested trees and definitions

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 $T[v]$.

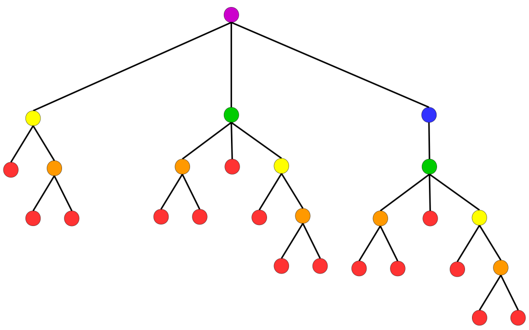
Two trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ are said to be isomorphic if there exists a bijection f from V_1 to V_2 such that for each pair of nodes (u, v) in V_1, $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$.

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

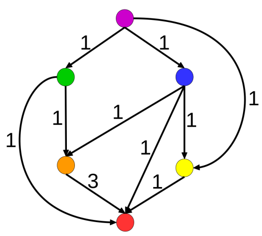
Figure 1 – Difference between regular trees and self nested trees

1.2 DAG compression

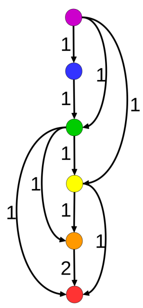
For $T = (V, E) \in \mathcal{T}$, let $\mathcal{Q}(T) = (V_\mathcal{Q}, T_\mathcal{Q})$ be the quotient graph of T by the equivalence relation $\equiv$. In concrete terms, $V_\mathcal{Q} = \{C(v) | v \in V\}$ is the quotient set of V by $\equiv$ and $E_\mathcal{Q} = \{(C(u), C(v)) | (u, v) \in T\}$.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

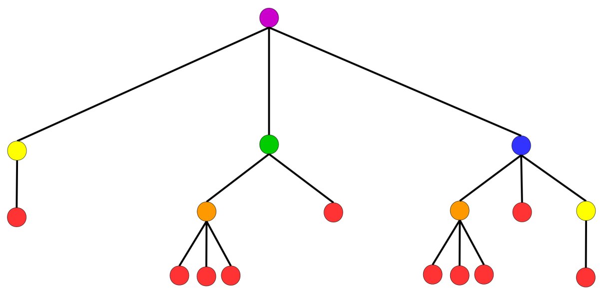
The trees whose quotient graph is a linear *DAG* (*DAG* for which there exists a hamiltonian path) are exactly the self-nested trees. Since the nodes of equal height in a self-nested tree are isomorphic to one anoter, there is a single vertex in the *DAG* for each height: the tree Subfigure 2b is the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

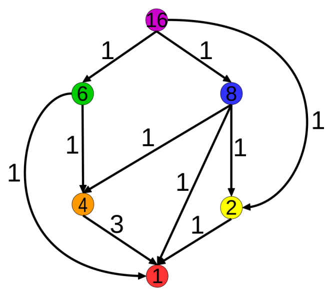
Self-nested trees have a linear quotient graph, as a result they are cheaper to store, but there are other advantages...

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

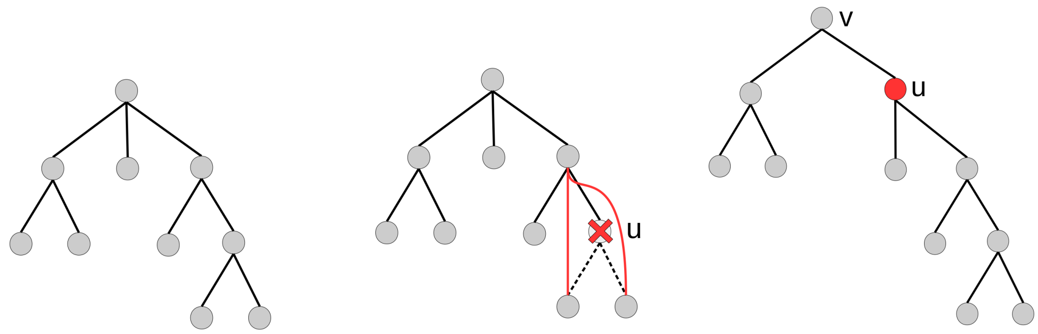
Figure 3 – Example of computation on the DAG

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. There are several ways to overcome this difficulty: one of them consists in restricting the set of instances to consider a set of trees in which the distance is easier to compute. Another one consists in modifying slightly the distance, for example by adding other constraints, in order to build a new edit distance easier to compute.

The article [3] presents an additional, 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 allows 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

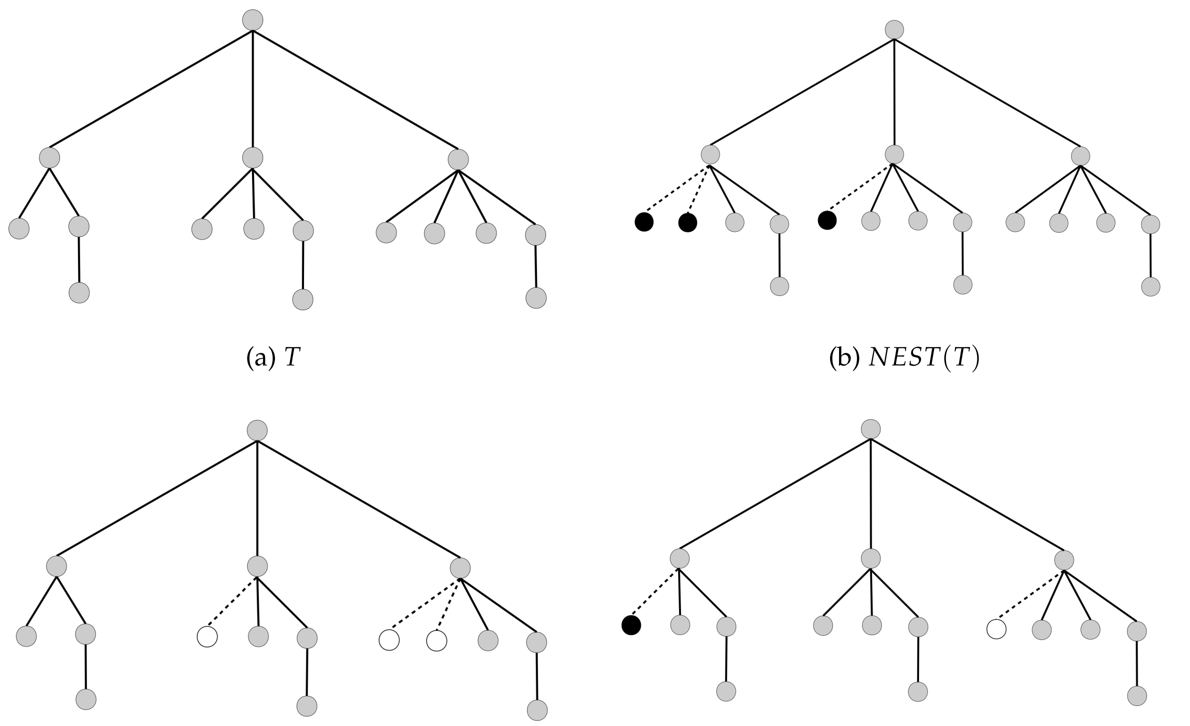

(a) T (b) $NEST(T)$ (c) $NeST(T)$ (d) $NST(T)$

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

NST problem is NP-complete). This result would justify the use of approximation algorithms such as the heuristic and the algorithm computing the NeST and the NEST.

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 Y. See the appendix for further reminder on the complexity theory.

I did not manage to prove that the NST problem is NP complete. We will also present in this section the different attempts we made and the reasons why they were relevant but not conclusive.

2.1 Belonging to the NP class

We aim to prove that the NST decision problem is NP-complete, this means it would belong to the NP class and be NP-hard. 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, both in polynomial time.

Figure 6 – edit operation height conservation

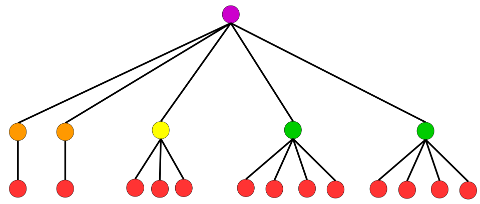


Figure 7 – example of tree of height 2

2.2 First approach

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. Indeed, in order to be able to prove the equivalence $\omega_A \in Y_A \Leftrightarrow \omega_B = f(\omega_A) \in Y_B$ (where A is the initial NP-complete problem and B is the problem we consider), the images of the reduction have often a very particular shape. Yet, the image of Ω_A by the reduction has to be big enough so that we are unable 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.

In order to understand better the difficulty of the problem, I tried to find a way to compute efficiently the NST in special cases. Indeed, as long as we find a polynomial algorithm solving the NST problem on the set of trees considered, we are ensured that the difficulty leading to the NP-hardness lies somewhere else. I also decided to start with very special cases where polynomial algorithms could be found and to add progressively some generality to identify the trees and conditions that make the problem difficult.

I first considered the trees of height 2. In the NEST and NeST algorithms, C. Godin, P. Ferraro and R. Azais consider only edit operations such that the height of any pre-existing node is unchanged (see Figure 6). For the sake of simplicity, I decided to first use these edit operations.

2.2.1 Trees of height 2 with height conservation

Let T be a tree of height 2 (see Figure 7). Let us denote $NST(T)$ by S and an optimal edit mapping between T and S by Φ. We will show in this section that it is possible to find Φ and S in polynomial time.

We can first notice that if there are any leaves of depth 1, they do not require any edit operations to obtain $NST(T)$. We can for this reason assume that every node of depth 1 is of height 1 (see Figure 7).

Let us denote the nodes of height 1 of T by $(u_i)_{1\leqslant i\leqslant N}$ and $(n_i)_{1\leqslant i\leqslant N}$ their respective number of children. We will suppose that the sequence $(n_i)_{1\leqslant i\leqslant N}$ is sorted by ascending order. Each u_i can be either deleted or mapped on a node of height 1 in S, and since S is self-nested, all the nodes of height 1 in S are isomorphic to one another. The complete subtree under each node of height 1 of S is characterized by its number of leaves. As a result, the edit mapping

Figure 8 – optimal cost for u_i in function of $\tilde{n}$

leading from T to S is characterized by the I_0 the set of indices of the nodes deleted and by $\tilde{n}$ the number of leaves under a node of height 1 in S. Thus, we will try to find I_0 and $\tilde{n}$ such that the corresponding self nested tree is a minimal distance from T.

Because of the constraint of Zhang, before deleting one of the nodes u_i of height 1 in T, it is necessary to delete all of his children except one. Therefore, deleting u_i costs n_i where n_i is the number of children of u_i.

If u_i is not deleted, than we have to adjust the number of children it has to $\tilde{n}$, which costs $|n_i - \tilde{n}|$. As a result, u_i has to be deleted if and only if n_i is closer to 0 than to $\tilde{n}$. Figure 8 shows f_i the minimum cost for each u in function of $\tilde{n}$.

We will denote by I_1 the complement of I_0 in the set of indices $[\![1; N]\!]$. I_1 is composed of the indices i such that $\Phi(u_i)$ is of height 1.

Remark. The distance between T and S is equal to ${}_{i \in I_0} n_i + {}_{i \in I_1} |n_i - \tilde{n}|$

By summing the functions f_i on every u_i, we obtain f the minimal cost of a mapping leading from T to a self nested tree of height 2, with N nodes of depth 1, in function of $\tilde{n}$ the number of children of the nodes of height 1. We can observe that the resulting function is a piecewise linear function and that every break in its differentiability has for abscissa one of the n_i or $2 * n_i$. Therefore, the minimum of this function is reached in one of those points. Since every f_i is increasing just before $2 * n_i$ and constant after and that the derivative of the other f_j are equal before and after $2 * n_i$, $2 * n_i$ cannot be a minimum of f.

Thus, by evaluating f on every n_i, it is possible to find the minimum of f and thereby find the NST of T.

It is possible to show that every $f(n_i)$ can be computed in constant time by using the value of $f(n_{i-1})$. $f(n_1)$ can be computed in linear time. For this reason, $NST(T)$ can be computed in $O(N * log(N))$ (it is necessary to order the $(n_i)_{1 \leqslant i \leqslant N}$ by ascending order) where N is the number of nodes of height 1 in T.

Since the NST (with only height preserving operations) can be computed in polynomial time on the trees of height 2, this proves that if the NST problem (with only height preserving operations) is NP-complete, the set of images of the reduction cannot be the trees of height 2, otherwise we would have found a way to solve any NP-complete problem in polynomial time, which is unreasonnable.

We can make two more remarks that will help us later.

Remark. For every $i \in I_0$, for every $j \in I_1$, $n_i < n_j$.
Intuitively, if $n_i < n_j$ it costs less to delete all the children of u_i than the one of u_j. See appendix for the proof.

Remark. Furthermore, $\tilde{n}$ is one of the medians of the $(n_i)_{i \in I_1}$.
Indeed, the median minimizes ${}_{i \in I_1} |n_i - \tilde{n}|$. See proof in appendix for further details.

We will keep in mind that the solution that we proposed here is a brute-force search, we didn't manage to find any strategy that would help us understand why computing the NST of a tree of height 2 seems to be easier than in the general case. The main reason that explains why this special case is easy to solve is that the trees of height 2 are simple enough to have a NST described by only 2 parameters.

Figure 9 – Shape of S

2.2.2 Trees of height 2 without preserving the height of the nodes

The NST problem is solved easily on the trees of height 2 if we authorize only the edit operations preserving the height. It can either be due to the set of trees considered or to the restriction on the edit operations. To gain in generality, I decided to look at the problem without the restriction on the edit operations, in order to use the work done in the previous section. Once again, I couldn't find any global stategy that would give us a polynomial algorithm so I tried to find as many properties on the NST of T as I could, in order to reduce the space of possibilities that we have to explore.

Since the poofs are quite fastidious we will only present the results we found in this section, the poofs of the lemmas are in the appendix.

We will use the same notations as in the preceding paragraph. We will denote by I_k the set of indices i of $[\![1;N]\!]$ such that $\Phi(u_i)$ is of height k. Let us denote by $\tilde{n}^k$ the number of children of the nodes of height k in S.

It is possible to show that S has the shape depicted Figure 9 and that S is characterized by $(I_k)_{0\leqslant k\leqslant H-1}$ and $(\tilde{n}^k)_{1\leqslant k\leqslant H-1}$ where H is the height of S. Therefore, the goal is to find the families $(I_k)_{0\leqslant k\leqslant H-1}$ and $(\tilde{n}^k)_{1\leqslant k\leqslant H-1}$ suh that the corresponding self nested trees is at minimal distance from T. The remarks of the previous section can be adapted.

Lemma 2.1. *For every $k \geqslant 0$, for every $i \in I_k$ and $j \in I_{k+1}$, $n_i < n_j$.*

Lemma 2.2. *The distance between T and S is equal to*

$$D(S,T) = \mathop{}\limits_{i\in I_0} n_i + \mathop{}\limits_{k=1}^{H}\left(\mathop{}\limits_{i\in I_k} |n_i - \tilde{n}^k| + |I_k| \mathop{}\limits_{l=1}^{k-1}(\tilde{n}^l)\right)$$

which is also equal to

$$D(S,T) = \mathop{}\limits_{i\in I_0} n_i + \mathop{}\limits_{k=1}^{H}\left(\mathop{}\limits_{i\in I_k} |n_i - \tilde{n}^k| + \tilde{n}^k \mathop{}\limits_{l=k+1}^{H}(|I_l|)\right)$$

Proof. Let $i \in I_0$. u_i is deleted, which costs n_i (as explained in the previous section).
Let $i \in I_k$ with $k \geqslant 1$. $\Phi(u_i)$ is of height k. To obtain the complete subtree under $\Phi(u_i)$ in S from the subtree under u_i in T, it is necessary to adjust the number of children of u_i to $\tilde{n}^k$ which costs $\left|n_i - \tilde{n}^k\right|$. It is also necessary to add ${}_{l=1}^{k-1}\tilde{n}^l$ nodes under one of the children of u_i, so that the height of u_i increases to k. ■

For every partition $(I_k)_{0\leqslant k\leqslant H-1}$ of $[\![1;N]\!]$, that verifies the lemma 2.1, it is possible to compute in $O(N)$ the family $(\tilde{n}^k)_{1\leqslant k\leqslant H-1}$ such that the distance between the corresponding self nested tree and T is minimal. It is also possible to show that

Lemma 2.3. *For every $k \geqslant 1$,*

$$|I_k| > \mathop{}\limits_{l=k+1}^{H-1}|I_l|$$

Lemma 2.4. *For every $k \geqslant 1$,*

$$\tilde{n}^k > \mathop{}\limits_{l=1}^{k-1}\tilde{n}^l$$

Corollary 2.5. *The height H of S is bounded by $min(log_2(N+1), log_2(n_N+1))$.*

Proof. The lemma 2.3 leads to the following inequalities:

$$\begin{array}{rcl} |I_k| & \geqslant & {}_{l=k+1}^{H} |I_l| + 1 \\ & \geqslant & |I_{k+1}| + {}_{l=k+2}^{H} |I_l| + 1 \\ & \geqslant & 2({}_{l=k+2}^{H} |I_l| + 1) \\ & \geqslant & 2^{H-k-1}(I_H + 1) \\ & \geqslant & 2^{H-k} \end{array}$$

As a consequence, $N \geqslant {}_{k=1}^{H} |I_k| \geqslant {}_{k=1}^{H} 2^{H-k} \geqslant 2^H - 1$, and $H \leqslant \lfloor \log_2(N+1) \rfloor$.
The same reasonning can be applied to the lemma 2.4 to show that $H \leqslant \lfloor \log_2(\tilde{n}^H + 1) \rfloor$, and we will notice that $\tilde{n}^H \leqslant n_N$, which leads to the result. ∎

By enumerating the partitions $(I_k)_{0 \leqslant k \leqslant H-1}$ that verify the lemmas 2.1 and 2.3 and computing the optimal family $(\tilde{n}^k)_{1 \leqslant k \leqslant H-1}$ and the distance to the corresponding self nested tree each time, we will find the NST of T. The number of partitions $(I_k)_{0 \leqslant k \leqslant H-1}$ of $[\![1; N]\!]$ that verify the lemmas 2.1 and 2.3 is bounded by $\frac{\binom{N+H}{H}}{2^H}$.

Since $\frac{\binom{N+H}{H}}{2^H}$ is a $O(N^H)$ and that $H \leqslant log_2(N) + 1$, the number of partitions that have to be considered is in the range of $N^{log_2(N)}$, which is unfortunetely not polynomial.

I was not able to find any better compexity than this one, yet a few remarks can be made. First of all, the complexity is expressed in function of N the number of nodes of height 1 and not in function of the size of the tree and in practice, the trees that have a NST of height close to the boundary $log_2(n)$ are very big in comparison of N. Secondly, the proofs are far more complicated when all the edit operations are considered than when only the height conserving operations are authorized. In the following, we will also prefer to use the restricted operations whenever it is possible.

This approach is not very conclusive, indeed, the fact that the algorithms presented are based on a brute force search, makes it difficult to spot the exact difficulty in the NST problem, apart from the tremendous number of parameters that describe the NST of a tree.

2.3 Second approach

We will present in this section a different approach of the problem, by studying analogies of the NST problem with NP complete problem and by pointing out similarities between the proofs of NP completeness of the state of the art. It appears that many diffrent proofs of NP completeness rely on the same general patterns (such as the ideas of *variables*, *widgets* and *constraints*). Since the first approach was not conclusive, we tried to adapt those ideas to the NST problem. In this section we will present these concepts through two different reductions and present a family of trees that illustrates the concepts of *variables* and *widgets* in the NST problem.

Let us begin with the proof of the reduction from 3-SAT to INDEP-SET (presented in [6] for example). As a brief reminder:

Definition 2.6 (3-SAT)**.**
Input: Φ a formula in conjonctive normal form where each clause is composed of three litterals.
Output: Yes if and only if there exists an interpretation that satifies Φ.

Figure 10 – Reduction 3-SAT to INDEP-SET: example for the formula

Definition 2.7 (INDEP-SET)**.**
Input: $G = (V, E)$ an undirected graph and k an integer.
Output: Yes if and only if there exists a set of vertices $V' \subset V$ of size k such that for every pair of vertices (u, v) of V', $(u, v) \notin E$.

In the reduction proposed by Garey and Johnson [**?**], the image instance of INDEP-SET is built in two steps: the first one consists in defining triangles that correspond to each clause of the initial instance of 3-SAT (represented in blue on the example Figure 10); the second one consists in adding edges that link the nodes of the triangles that are labeled with the negation of the same variable of 3-SAT (represented in red on the example Figure 10). One way of interpreting this division in two steps is to consider the triangle *widgets* as *variables* that can take three different values (the *variable* term here should not be confused with the variable of the 3-SAT problem) and to consider the edges linking the *widgets* as *constraints*.

These three different possible values are the labels of the three nodes of the triangle. The value taken is the label of the node that belongs to the set of vertices of V' the INDEP-SET problem. For example, Figure 10, the first widget can take the values x, y or z.

The edges linking two *widgets* A and B forbid the two coresponding *variables* to take simultaneously some given values. Indeed, we can see Figure 10 that the edge $(x, \neg x)$ between the first and the second *widgets* forbids the first *widget* to take the value x if the second takes the value $\neg x$ at the same time.

We will now consider the reduction from VERTEX-COVER to HAMILTONIAN-PATH proposed by Cormen, Leiserson, Rivest and Stein [**?**]. As a reminder:

Definition 2.8 (VERTEX-COVER)**.**
Input: $G = (V, E)$ a undirected graph and k an integer.
Output: Yes if and only if there exists set ofvertices $V' \subset V$ of size k such that for every edge $(u, v) \in E$, $u \in V'$ or $v \in V'$.

Definition 2.9 (HAMILTONIAN-CYCLE)**.**
Input: $G = (V, E)$ an undirected graph.
Output: Yes if and only if G is hamiltonian, in other words, if there exists a cycle on G visiting every vertex one and only one time.

In this reduction, the image instance is also built in two steps: the first one consist in defining small pieces of graph composed of twelve nodes (Figure 11), and the second one consists in linking this pieces together through additionnal edges and vertices.

Once again, the small *widgets* of the first part can be interpreted as variables. Indeed, there are two and only two ways of visiting one *widget* during the traversal of the complete graph: in one time or two times. For this reason, the *widget* can be seen as a *variable* that can take two values, one or two.

Since it is quite complicated and not particularly relevant here, we won't detail how the

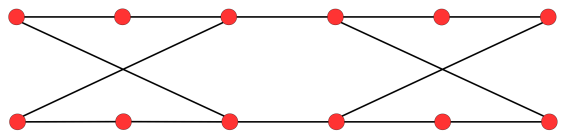


Figure 11 – Hamiltonian widget

small pieces of graph are linked to one another, but the edges and vertices added to connect the *widgets* together can likewise be seen as *constraints*.

In both cases, several remarks can be made: first of all, without the *constraints*, the *widgets* are totally independant: the value taken by te *variable* corresponding one of them has no influence on any of the values taken by the others. Secondly, the *widget* used in the reduction is specific to the second problem. Indeed, it is quite easy to reduce several different NP complete to a problem A by using the same *widgets* and link them differently. Finally, the *constraints* highly reflects the shape of the instance of the first problem and is a sort of translation of the difficulty of the instance of the first problem into the difficulties of the second problem.

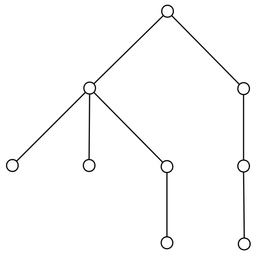
I choose to start the reduction from the INDEP-SET problem because of its simplicity and of the clarity of its *constraints* and *variables*. Indeed, each vertex of the graph can be considered as a binary *variable* taking the value 1 if the vertex belongs to V' and 0 otherwise. There are two types of *constraints*, the first one is that if there is an edge between the vertices x and y, the corresponding *variables* cannot take both the value 1; the second one is that only k variables can take the value 1.

The aim is to find pieces of trees that could be easily combined together and that could be the *widgets* of the reduction we are trying to find. These *widgets* have to be used in only two possible ways, in order to act as the *variables* of the INDEP-SET problem. In our case, this means that there are two and only two different self-nested trees at equal and minimal distance from the *widget*. The *widgets* also have to be easily combined and in a way that the choice of use of each *widget* is indepedent from the way we use the others (we have seen that without the *constraints*, the *widgets* are totally independent).

In order to be independant, the *widgets* have to be placed at different heights: indeed, if two *widgets* are at the same height, their variations to obtain the nearest self-nested tree will be the same, so they couldn't represent different *variables*. A solution to this issue is to search widgets through their DAG compression and to pile them up. A second advantage of this method is that, the size of the corresponding tree grows exponentionally, whereas the size of the DAG grows linearly with the number of *widgets*, so if we rephrase the NST problem to give in Input a DAG instead of a tree, the construction has still chances to be polynomial.

A piece of DAG that respects this properties is shown Figure 12a (its DAG compression is shown Figure 12b). The corresponding tree is at equal distance of the self nested trees Figures 13a and 13b, and their DAG compression is shown Figures 14a and 14b. To set the ideas, we will say arbitrarily that the *variable* corresponding to a *widget* takes the value 1 if in the NST considered the *widget* was transformed into Figure 14a and 0 if it was transformed into Figure 14b.

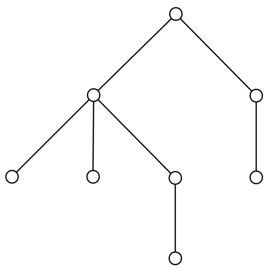
These *widgets* can be pilled up as shown Figure 15a. We will denote the tree obtained by pilling up n *widgets* by T_n. The *widgets* in T_n are independent from one another: it is possible to show that T_n is equidistant from every tree having the DAG compression of Figure 15b where every A_i is either the DAG piece Figure 14a or Figure 14b. We will denote by $\mathcal{S}_n$ the set of these



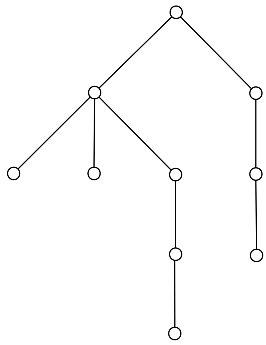
(a) fig:widget

(b) fig:DAGwidget

Figure 12 – The widget

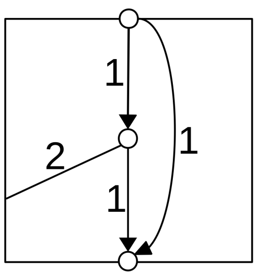


(a) fig:NSTwidget1

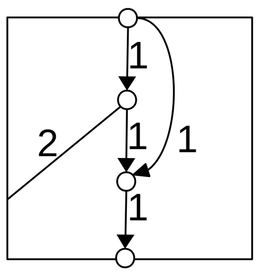


(b) fig:NSTwidget2

Figure 13 – two possible changes

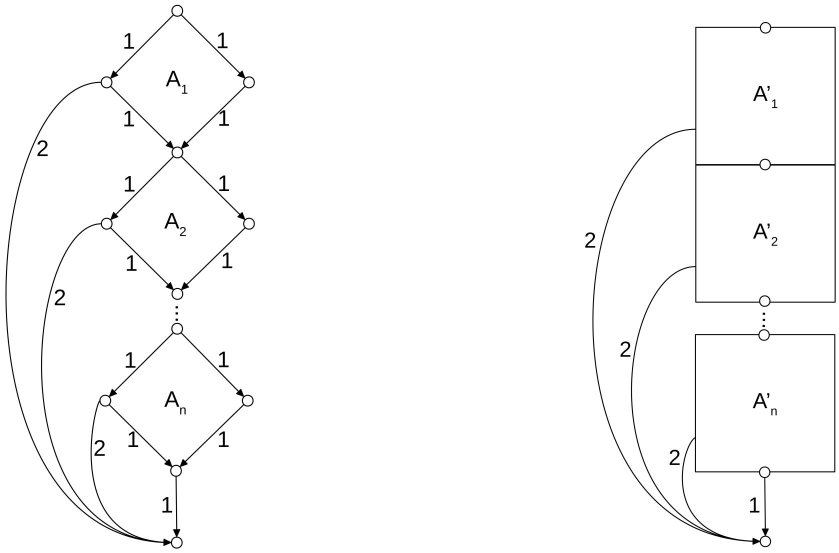


(a) fig:DAGwidget1



(b) fig:DAGwidget1

Figure 14 – their DAG compression



(a) fig:empilement (b) fig:NSTempilement

Figure 15 – pilling up the *widgets*

self nested trees. There exists no tree closer to T_n than these ones.

To summarize, T_n has exactly 2^n different NST and each one of this self nested trees corresponds to an assignment of n variables in $\{0;1\}$. As explained before, this could correspond to the first step of a reduction of the NST problem. In order to complete the reduction, we have to add *constraints*. Adding *constraints* means modifing T_n in function of the instance of INDEP-SET, in order to obtain a tree T closer to the self nested trees that correspond to valid assignments of *variables* for the instance of INDEP-SET. Given an instance $\mathcal{I}$ of INDEP-SET, we will denote by $\mathcal{S}_\mathcal{I}$ the subset of $\mathcal{S}_n$ constituted of the trees that correspond to valid assignments of *variables* in $\mathcal{I}$.

Let start with the first type of *constraint*: if there is an edge between u and v in the graph $G = (V, E)$ of $\mathcal{I}$, the *variables* represented by u and v cannot be both assigned to the value 1. Given two vertices u and v, if there is an edge between u and v, we have to modify T_n to obtain T such that T is closer to all the self nested trees of $\mathcal{S}_n$ where at least one of the *widgets* corresponding to u and v was modified into Figure 14b, than the rest of $\mathcal{S}_n$. By doing this modifications the NST of T will be the trees that correspond to a valid assignment of *variables*.

The goal is to add successively this modifications for every edge in E, and to reduce at every step the distance to the self nested trees of $\mathcal{S}_\mathcal{I}$. By doing this modifications for every edge in E, we should have applied all the *constraints* of the first type of the INDEP-SET problem and we would still have to find how to translate the second one (the fact that only k *variables* can take the value 1).

It appears that the modification for one edge has to reduce equally the distance to all the trees of $\mathcal{S}_\mathcal{I}$, so that the set of nearest self nested trees of T is exactly $\mathcal{S}_\mathcal{I}$. Otherwise, it is not possible to prove the implication $\omega_{\text{NST}} = f(\omega_{\text{INDEP-SET}}) \in Y_{\text{NST}} \Rightarrow \omega_{\text{INDEP-SET}} \in Y_{\text{INDEP-SET}}$

Unfortunately, the only modifications we found did not respect this property. For this reason, we were not able to finish the proof with this method.

(a) fig:constraint

(b) fig:DAGconstraint

Figure 16

2.4 Last approach

Since the second approch was not conclusive, I tried to approach the problem form the other side: instead of trying to find the *widgets*, we will this time search to adapt the concept of *constraint* to the NST problem.

As we said before, the *constraints* are generally a sort of translation from one problem to another of the difficulties. The issue is that we do not know precisely at this point where the difficulty truely lies in the NST problem.

The algorithms proposed by C. Godin, P. Ferraro and R. Azais [1, 5] to compute the NEST and the NeST rely on the fact that since only addition or deletion operation are made, it is possible to operate recursively on the DAG. More precisely, both algorithm start by doing operations on the nodes of low height, in order to make them all self-nested and isomorphic to one another, recuresively use the changed nodes of low height to make the ones of higher height alos isomorphic and self-nested. This method is possible because only one type of operations is allowed.

Let illustrate on an example why this method does not seem to be possible in the case of the NST problem. Let us consider the tree Figure 1a. Its NEST is shown Figure **??**, its NeST is shown Figure **??**. Its NST is also its NEST. So far it does not seem particularly difficult, the issue is that in tree Figure **??** has for NST the tree **??**, where the subtrees of height 2 are isomorphic to the tree Figure **??** which is not the NST of the subtrees of heigth 2 (isomorphic to 1a).

The previous example highlights a sort of balance between two irreconcilable goals: to be close to the initial tree T, the subtrees of small height in the NST S have to be close to the trees of small height of T. But to make the higher nodes self-nested and isomorphic to one anoher, it will sometimes be necessary to add whole subtrees of smaller height. And therefore, the subtrees of small height in S also have to be of small size. This prevents us from using a recursion from the bottom of the DAG to the top or in the other direction. It prevents us from knowing where to start doing the modifications: if we start with the upper part of the DAG, we will need to know the exact size of the smaller subtrees to be able to choose properly which operations have to be done; if we start with the lower part of the DAG, we will have to know how many times the small subtrees will be completely added in order to choose a self-nested tree of small size yet close to the initial subtrees of small height.

The tree Figure 16a (that has the DAg compression shown Figure 16b) is simple tree that depicts perfectly this balance: the lower rhombus in the DAG needs to be modified in odrer to become self-netsed, as well as the upper rhombus, but the modifications made on any of them impacts the cost of the modifcation made on the other one.

This tree is at distance 12 from the self nested trees represented by their DAG Figures 17a, 17b and 17c and at distance 14 from the self nested tree Figure 17d. The 3 first trees are the NST of the tree Figure 16a. It is quite easy to see the two rhombus as *widgets* and assimilate them to *variables*, that can take the values 2 or 3, depending on which one of the 4 self nested tree we consider.

The equal distance to the 3 first self nested tree is particularly interessant because it is very

(a) fig:NST22 (b) fig:NST23 (c) fig:NST32 (d) fig:NST33

Figure 17 – 4 possibilites of variations

similar to the *constraint* of the INDEP-SET problem (Figure 18)! As a short reminder, the first type of *constraint* for the INDEP-SET problem is that if two vertices u and v are linked by an edge in G, the corresponding *variables* cannot be both set to 1.

We can see Figure 18 that the assignment of *variables* that corresponds to a valid independent set for the INDEP-SET problem, are exactly the assignments of *variables* that correspond to the NST of the tree Figure 16a

u	v	*constraint*
0	0	respected
0	1	respected
1	0	respected
1	1	violated

(a) *Constraint* of INDEP-SET

u	v	distance	*constraint*
2	2	12	respected
2	3	12	respected
3	2	12	respected
3	3	14	violated

(b) *Constraint* of NST

Figure 18 – Constraints

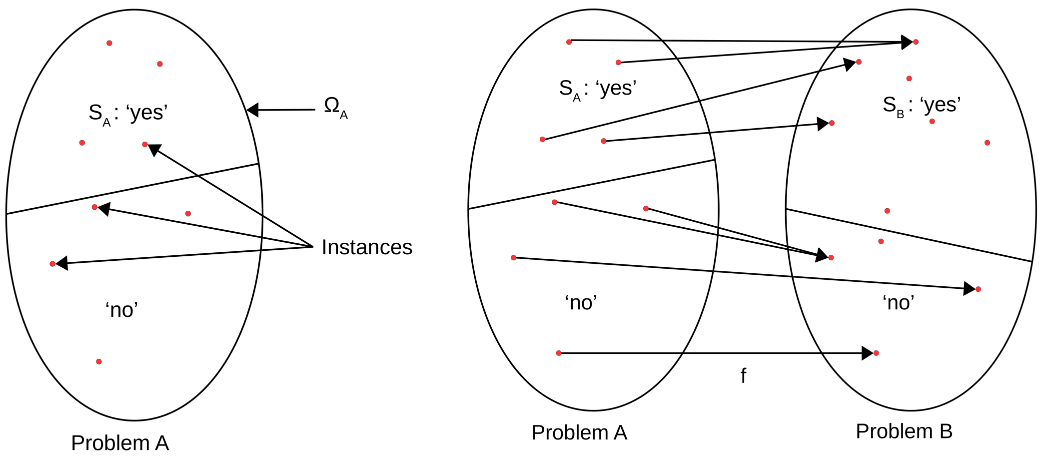
Conclusion

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 19a).

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 19b). 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. I also would like to thank Frédéric Vivien, who helped me to work on the first approach and gave me numerous advices on the way of approaching a problem of NP-completness.

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.